



Universität Hamburg

DER FORSCHUNG | DER LEHRE | DER BILDUNG

an der Universität Hamburg eingereichte
kumulative Dissertation

Domain Operations for Multimodal Autonomous Systems

zur Erlangung des Doktorgrades
der Naturwissenschaften (Dr. rer. nat.)
dem Fachbereich Informatik
der Fakultät Mathematik, Informatik und Naturwissenschaften
der Universität Hamburg
vorgelegt von

Bernd Kast

2022

Day of oral defense: 12.06.2023

The following evaluators recommend the admission of the dissertation:

Prof. Dr. Jianwei Zhang
Department of Informatics
Universität Hamburg, Germany

Prof. Dr.-Ing. Matthias Riebisch
Department of Informatics
Universität Hamburg, Germany

Dr. Sebastian Albrecht
Siemens AG
Munich, Germany

Prof. Dr. Janick Edinger
Department of Informatics
Universität Hamburg, Germany

Prof. Dr. Tilo Böhmann
Department of Informatics
Universität Hamburg, Germany

Contents

I	Hierarchical Approach to Domain Operations for Multimodal Autonomous Systems	15
1	Introduction	17
1.1	Motivation and Objectives	17
1.2	Research Questions	19
1.3	Contributions and Outline	22
2	Set-Based Hierarchical Modeling Approach	25
2.1	Declarative Knowledge	25
2.1.1	Motivation	25
2.1.2	Theoretic Grounding	27
2.2	Procedural Knowledge	30
2.2.1	Motivation and Requirements	31
2.2.2	Theoretic Grounding	33
3	Hierarchical Planner	41
3.1	Motivation	41
3.2	Planning Algorithm	43
3.2.1	Hierarchical Factorization Algorithm	43
3.2.2	Planning Algorithm	51
3.3	Domains, Explainability and Introspection	56
4	Domain Optimizations	65
4.1	Motivation	65
4.2	Alignment of the Domain to the Task	68
4.3	Reasoning for Data Fusion	72
4.4	Model Harvesting from Unstructured Sources	78
4.5	Automatic Refinement and Abstraction of Operators	79
4.6	Detection of Model Inconsistencies	80
4.7	Data Driven Optimizations	81
4.8	Postprocessing for Parallel Execution	83
5	Conclusions	87

II Cumulative Part	91
6 Bridging the Gap Between Semantics and Control for Industry 4.0 and Autonomous Production	93
7 A hierarchical planner based on set-theoretic models: Towards automating the automation for autonomous systems	103
8 Domain Optimization for Hierarchical Planning Based on Set-Theory	117
9 Hierarchical Planner with Composable Action Models for Asynchronous Parallelization of Tasks and Motions	127
10 Automatic Domain Extension and Optimization based on Set-Theory	135
11 Configuration of Perception Systems via Planning Over Factor Graphs	163
12 Automatic Configuration of Perception Pipelines	173
13 Data-Driven Synthesis of Perception Pipelines via Hierarchical Planning	183
Bibliography	193

Abstract

The robotics market has increasingly gained momentum in recent years. Growing numbers of robots are leading to ever-lower hardware prices, which enable economical automation in additional fields of application. A limiting factor of today's automation systems is the setup time, integration, and programming costs, including developing and adapting control sequences to the respective task. In the past, industrial robots only relied on very few sensors, reducing the complexity and probability of failure and facilitating the implementation of control themes. However, due to their lack of self-adaptability, specialists must manually adjust those sensorless systems to new conditions each time the product or robotic cell changes. These automated systems are thus only economically viable compared to manual processes if high production volumes are targeted. Advances in lightweight robots, combined with increased use of force sensor technology, make new programming methods possible, reducing the time and cost for setup. For example, the complex calibration of poses can be performed by manually guiding the robot to the desired position instead of numerically specifying the positions. As a result, even trained employees who are not experts in robotics can adapt the system to the product as long as only minor changes are necessary. This ease of use shifted the limit for economical automation to a certain degree, but manual programming is still required for each product variant that should be manufactured. Thus, a paradigm change from programming to planning is necessary for more far-reaching improvements, allowing a batch size of one. The scientific community provides algorithms for flexible automation, which derives the required actions based on descriptions of the environment, machinery, and the desired goal without explicit programming. In contrast to automation, these autonomous systems can handle new tasks flexibly and are more robust against environmental changes due to the use of sensory information. Therefore, unlike classical approaches, which use expensive, specialized hardware and devices to reduce positioning uncertainty, these systems can handle unexpected events by utilizing cheap multimodal sensors, such as 2D and 3D cameras or the force sensors inherent to cooperative robots. Thus, the focus is on the software for autonomous decision-making, which is algorithmically more complex than predefined, programmed processes and requires additional modules for perception, data fusion, sensitive interactions, and motion generation. Different research lines deal with sub-problems of such an autonomous system and have already delivered several sophisticated algorithms that each come with their strengths and weaknesses, requirements, and parameterizations. So far, the core of robotics, integrating suitable components into a functioning system, has yet to be formalized and automated, especially if the system involves multiple sensory modes or modes of action.

In this dissertation, we propose a formal model based on set theory to describe all relevant components of autonomous systems. While the approach is primarily discussed based on industrially

relevant use cases, the methods can also be applied to arbitrary domains, such as households or agriculture. Our model combines declarative knowledge, which describes information and physical objects, and procedural knowledge, which deals with actions and modifications of declarative knowledge. Only with this combination, and based on the formal foundation, can we formulate specialized operations in these fields that autonomize robotics. These operations also include planning algorithms that perform actions in simulation or the real world and help us find a path to the specified goal. Our newly developed algorithms bring special features like compactness, loop detection, and completeness detection. Furthermore, these operators can also include approaches to calculate a hierarchical ordering within the model, which helps to factorize the specified planning task and structure the search space during planning. Combined, all those properties allow us to automatically compute a recursive decomposition of the planning problem, which is crucial to managing large, real-world problems that cover discrete and continuous properties. Since our primary goal is to reduce cost by broadly applying flexible, autonomous systems, we need a systematic approach to reuse components from different sources. Therefore, we have developed algorithms that detect and correct common errors in domains composed from multiple sources. By that, our automatic domain optimization algorithms allow for multimodal systems, which combine perception and sensor fusion steps of various modes with multimodal execution processes that include multiple tools and respective tool changes. In addition, our system is model-driven at its core, enabling easy engineering and debugging. For example, we can trace the processes involved, and due to the recursive factorization, we can help isolate and pinpoint errors quickly, facilitating troubleshooting. The automatic extension of formalized knowledge provides additional assistance to the engineer. Besides, we present operations that analyze and process existing information from known systems to extract additional knowledge for our planning algorithms, which includes harvesting non-formalized sources, such as program code or documentation, to generate usable, formalized models. The harvesting process involves data-driven approaches, which enable us to extract and learn models from recorded data. We can easily integrate these techniques, which have gained momentum in recent years, due to our set-based foundation, that shares principles with data-driven models. In both cases, instances or examples describe a more abstract symbol or object. The model-driven approach eases processing not only for engineers but also for algorithms. Parallel execution is an essential requirement for many industries that involve multiple machines for a single task. Since we aim at a general formalization of any industrial task, we have developed a novel approach to adapt the planning task, which is typically tailored to sequential processes, so that it can be combined with parallel execution and achieve optimal results.

Zusammenfassung

Der Robotikmarkt hat in den letzten Jahren zunehmend an Dynamik gewonnen. Die wachsende Zahl an Robotern führt zu immer niedrigeren Hardwarepreisen, die eine wirtschaftliche Automatisierung von neuen Aufgaben ermöglicht. Ein limitierender Faktor heutiger Automatisierungssysteme sind die Rüst-, Integrations- und Programmierkosten, welche die Entwicklung und Anpassung der Steuerungsabläufe an die jeweilige Aufgabe beinhalten. In der Vergangenheit wurden bei Industrierobotern kaum Sensoren wie Kameras, Kraft- oder Momentensensoren verwendet. Auf der einen Seite verringert dies die Komplexität und Ausfallwahrscheinlichkeit und erleichtert die Implementierung von Algorithmen. Auf der anderen Seite müssen Spezialisten das System jedes Mal, wenn sich das Produkt oder die Roboterzelle ändert, an neue Bedingungen anpassen. Diese zeitaufwendige und dadurch teure Aufgabe führt zu einer hohen minimalen Stückzahl, ab der die automatisierte Anlage im Vergleich zum manuellen Prozess erstmals wirtschaftlich rentabel ist. Fortschritte bei Leichtbaurobotern, kombiniert mit vermehrter Anwendung von Kraftsensorik, ermöglichen neue Programmierprozesse, die den Einrichtungsaufwand reduzieren. Die aufwändige Kalibrierung von Posen kann zum Beispiel durch manuelles Führen des Roboters an die gewünschte Position durchgeführt werden. Dadurch können auch geschulte Mitarbeiter, die keine Experten der Robotik sind, die Anlage zumindest bei geringfügigen Änderungen an das Produkt anpassen. Auf diese Weise können mehr Prozesse mit kürzerer Amortisationszeit wirtschaftlich automatisiert werden. Für weiterreichende Verbesserungen, die auch eine Losgröße von 1 zulassen, ist jedoch ein Paradigmenwechsel von der Programmierung zur Planung nötig. Die wissenschaftliche Gemeinschaft stellt Algorithmen für eine flexible Automatisierung zur Verfügung, die sich auf Sensordaten stützt und keine explizite Programmierung erfordert. Stattdessen wird nur das Ziel vorgegeben und die Planungsalgorithmen berechnen eine zielführende Handlungssequenz. Diese autonomen Systeme können im Gegensatz zur Automatisierung sehr flexibel mit neuen Aufgaben umgehen und sind robuster gegenüber Veränderungen in ihrer Umgebung. Daher kann auf teure, spezialisierte Hardware und Vorrichtungen verzichtet werden, die bei der klassischen Herangehensweise notwendig sind, um die Positionierungsunsicherheit zu reduzieren. Nur mit diesen teuren Speziallösungen sind unerwartete Ereignisse, die zu einem Ausfall führen würden, ohne weitreichenden Einsatz von Sensorik und Planung vermeidbar. Die autonome Entscheidungsfindung ist jedoch algorithmisch komplexer gegenüber vorgegebenen, programmierten Abläufen und erfordert zusätzliche Module für Wahrnehmung, Datenfusion, sensitive Interaktionen und Bewegungserzeugung. Verschiedene Forschungsstränge befassen sich mit Teilproblemen und brachten mehrere, ausgefeilte Algorithmen hervor, die ihre individuellen Stärken, Anforderungen, Parametrisierungen und Nachteile haben. Bisher war der Kern der Robotik, die Integration geeigneter Komponenten zu einem funktionierenden System, jedoch nicht formalisiert und automatisiert.

In dieser Dissertation wird ein formal fundiertes, auf der Mengenlehre basierendes Modell zur Beschreibung aller relevanten Komponenten autonomer Systeme vorgestellt. Während der Ansatz in erster Linie anhand industriell relevanter Anwendungsfälle diskutiert wird, kann er mit beliebigen Domänen umgehen und die Methoden auch zum Beispiel im Haushalt oder der Landwirtschaft anwenden. Das vorgestellte Modell vereint sowohl deklaratives Wissen, das Informationen und physikalische Objekte beschreibt, als auch prozedurales Wissen, bei dem es um Aktionen und Modifikationen des deklarativen Wissens geht. Nur mit dieser Kombination und aufgrund der formalen Fundierung können spezialisierte Operationen auf diesen Gebieten formuliert werden, die die Robotik autonomisieren. Diese Operationen umfassen zum einen Planungsalgorithmen, die Aktionen in der Simulation oder in der realen Welt ausführen, um einen Weg zum spezifizierten Ziel zu finden. Die in dieser Arbeit neu entwickelten Algorithmen bringen spezielle Eigenschaften wie Kompaktheit, Schleifen- und Vollständigkeitserkennung mit. Zum anderen können diese Operatoren auch Ansätze zur Berechnung einer Ordnung innerhalb des Modells umfassen, die helfen die spezifizierte Planungsaufgabe zu faktorisieren. Dies ermöglicht die automatische Berechnung einer rekursiven Zerteilung des Planungsproblems, das den Schlüssel zur Behandlung großer, realer Probleme darstellt, die sowohl diskrete als auch kontinuierliche Eigenschaften abdecken. Da das Hauptziel die Kostenreduzierung durch eine breite Anwendung flexibler, autonomer Systeme ist, wird ein systematischer Ansatz zur Wiederverwendung von Komponenten aus verschiedenen Quellen benötigt. Daher wurden in dieser Arbeit Algorithmen entwickelt, die häufige Fehler in solchen zusammengesetzten Domänen erkennen und korrigieren können. Hiermit werden auch die komplexeren Algorithmen und Handlungsabläufe multimodaler Systeme ermöglicht, bei denen die Stärken unterschiedlicher Sensorentypen, deren Datenfusion zu einem gemeinsamen Weltmodell, sowie die multimodale Ausführung durch verschiedene Werkzeug oder Greifern und ihrem Zusammenspiel zu tragen kommen. Zudem ist dieses System im Kern modellgetrieben und die Abläufe damit nachvollziehbar, wodurch die Fehlersuche erleichtert wird. Erleichtert kommt noch hinzu, dass mit der rekursiven Faktorisierung Fehler leicht auf ein Teilsystem eingegrenzt werden können.

Eine weitere Hilfestellung für den Ingenieur ergibt sich durch die automatische Erweiterung des formalisierten Wissens. Es werden Operationen vorgestellt, die vorhandene Informationen aus bestehenden Systemen analysieren und aufbereiten, um zusätzliches Wissen für die Planungsalgorithmen dieser Arbeit zu extrahieren. Dazu gehört das Sammeln aus nicht-formalisierter Quellen, wie Programmcode oder Dokumentation, um brauchbare, formalisierte Modelle zu generieren. Zusätzlich damit verbunden sind auch datengetriebene Ansätze, die es ermöglichen, aus aufgezeichneten Daten Modelle zu extrahieren und zu lernen. Diese in letzten Jahren an Fahrt aufnehmenden Techniken können aufgrund der mengenbasierten Grundlage, die auch als datengetriebenes Modell betrachtet werden kann, leicht integriert werden. Für viele industrielle Aufgaben, bei deren Bearbeitung meist mehrere Maschinen beteiligt sind, ist zudem eine parallele Ausführung eine Grundvoraussetzung. Da in dieser Arbeit eine allgemeine Formalisierung beliebiger industrieller Aufgaben angestrebt wird, wurde ein neuartiger systematischer Ansatz entwickelt, um Planung, die auf sequenzielle Prozesse ausgelegt ist, mit paralleler Ausführung zu kombinieren.

Acknowledgements

I want to express my gratitude and appreciation to all who have played a significant role in completing this dissertation.

My sincere thanks go to my supervisors, Prof. Dr. Jianwei Zhang and Dr. Georg von Wichert. Your support and provision of resources have been instrumental in enabling me to conduct this research. Your invaluable advice, insights into different conferences, and connections with partners and startups have enriched my understanding and opened doors to new opportunities.

I am also profoundly grateful to my advisor, Dr. Sebastian Albrecht, for his exceptional guidance, motivation, and insightful discussions throughout the entire research process. Your expertise and dedication have greatly influenced the quality and direction of this work, and I am truly fortunate to have had you as my advisor.

Sincere thanks go to Dr. Wendelin Feiten for his extraordinary vision, inspiration, discussions, and out-of-the-box thinking that have greatly enriched my research journey.

I would like to acknowledge colleagues of my research group whose presence has made the journey of pursuing this degree all the more fulfilling. The exchange of ideas, discussions, and support within our group have been invaluable, and I am grateful for the intellectual stimulation and encouragement that each of you provided.

I am sincerely grateful for the invaluable contributions of my fellow Ph.D. candidates, Philipp, Florian, and Vincent, with whom I collaborated on creating research demonstrators, attending fairs, and sharing memorable moments playing table soccer, making our journey even more enjoyable and fostering a strong bond of friendship.

A heartfelt appreciation is extended to the Siemens AG for the exceptional work environment, abundant resources, and access to state-of-the-art robotic laboratories, which have played a vital role in the success of this research.

Lastly, I extend my deepest gratitude to my family, especially my girlfriend Stephanie. Your unwavering support, understanding, and love have been my pillar of strength throughout this challenging journey. Your constant encouragement and belief in me have fueled my determination to overcome obstacles and pursue excellence.

To all those mentioned and the countless others who have contributed to my academic and personal development, I am indebted to you. Your support, encouragement, and belief in my abilities have been essential to completing this dissertation.

List of Publications

- **A hierarchical planner based on set-theoretic models: Towards automating the automation for autonomous systems**
B. Kast, V. Dietrich, S. Albrecht, W. Feiten, and J. Zhang
2019 International Conference on Informatics in Control, Automation, and Robotics
© SciTePress 2019
 - **Bridging the Gap Between Semantics and Control for Industry 4.0 and Autonomous Production**
B. Kast, S. Albrecht, W. Feiten, and J. Zhang
2019 IEEE/RSJ International Conference on Automation Science and Engineering
© IEEE 2019
 - **Der digitale Zwilling in der autonomen Robotik**
B. Kast, S. Albrecht, V. Dietrich, F. Wirnshofer, W. Feiten, and G. von Wichert
atp magazin
© Vulkan Verlag 2019
 - **Domain Optimization for Hierarchical Planning Based on Set-Theory**
B. Kast, V. Dietrich, S. Albrecht, W. Feiten, and J. Zhang
2020 International Conference on Informatics in Control, Automation, and Robotics
© SciTePress 2020
 - **Hierarchical Planner with Composable Action Models for Asynchronous Parallelization of Tasks and Motions**
B. Kast, P. S. Schmitt, S. Albrecht, W. Feiten, and J. Zhang
2020 IEEE International Conference on Robotic Computing
© IEEE 2020
 - **Automatic Domain Extension and Optimization based on Set-Theory**
B. Kast, V. Dietrich, S. Albrecht, G. von Wichert, W. Feiten, and J. Zhang
Lecture Notes in Electrical Engineering
© Springer 2020
-

- **Configuration of Perception Systems via Planning Over Factor Graphs**
V. Dietrich, **B. Kast**, P. S. Schmitt, S. Albrecht, M. Fiegert, W. Feiten, and M. Beetz
2018 International Conference on Robotics and Automation
© IEEE 2018
 - **Automatic Configuration of the Structure and Parameterization of Perception Pipelines**
V. Dietrich, **B. Kast**, M. Fiegert, S. Albrecht, and M. Beetz
2019 IEEE/RSJ International Conference on Advanced Robotics
© IEEE 2019
 - **Data-Driven Synthesis of Perception Pipelines via Hierarchical Planning**
V. Dietrich, **B. Kast**, S. Albrecht, and M. Beetz
2020 International Conference on Robotics in Alpe-Adria-Danube Region
© Springer 2020
 - **RTFM: Towards Understanding Source Code using Natural Language Processing**
M. Galanis, V. Dietrich, **B. Kast**, and M. Fiegert
2020 International Conference on Informatics in Control, Automation, and Robotics
© SciTePress 2020
-

List of Figures

1.1	Our approach can handle various domains, ranging from planning for complete plants over assembly planning to cell layout optimization [1].	23
2.1	The differently colored areas depict the base B_{Γ} (yellow rectangle) and concepts that have different intersections depending on their properties.	28
2.2	For autonomous systems, there are three dimensions to consider. Idealized approaches are covered by planning and control, while sophisticated and possibly non-deterministic simulations blend with real-world execution. Similarly, control behaves like planning once more and more alternative scenarios are considered for simulated future points in time.	31
2.3	Graph structure of an abstract pick operator (left) and a more detailed version (right) that has additional outputs to handle exceptions, as well as more detailed inputs and outputs.	34
2.4	The high-level operator may either be refined by a single operator or a set of operators that can form a network. In either case, the leaf nodes must be more detailed and have similar instances to the outputs of the coarse operator. Additionally, root nodes should be in the more detailed level for each input of the abstract operator [2].	39
3.1	The left robot approaches the box to grasp it. In parallel, the other robot grasps the screwdriver. During execution, both entities are handled in the same spatial proximity and can influence each other, while other robots in the factory are not considered.	44

-
- 3.2 Depiction of the hierarchical planning scheme. The spheres visualize different planning states, and the edges depict temporal relations for all non-vertical edges. The vertical edges are refinements to that state. The differently colored horizontal chains are successful plans on that level of abstraction. The topmost chain is the most abstract solution, successively refined by the rows below it. The white spheres are visited states that did not contribute to the solution, and the yellow spheres are lately added states. 47
- 3.3 Operator hierarchy within a domain. *DOIT* is the placeholder operator to configure the properties of the most abstract planning task. Successors of operators are available in the subdomain that refines this operator. Leaf nodes are operators used in the most refined domain, which executes the task in the real world. Operators can be used in multiple subdomains. The number of refinements is not the same for all operators in a domain such that no strictly hierarchical levels emerge. Those levels only emerge during planning based on the applied operations. The coloring is purely user-defined. 48
- 3.4 The upper, more abstract level defines the intermediate goals for the more refined level, which is depicted by matching colors. The red pair is the goal and the instance fulfilling this goal for the first step. The second step has two goals and respective goal instances. There, the refined level acts on the goal instance with additional detail instead of the instances of the coarse level and, therefore, forwards the detail calculated in that level. Further discussion can be found in chapter 7. 49
- 3.5 This illustration of [2] distinguishes different hierarchical levels by their color. No plan was found in the orange subdomain (hence the large number of planning states visited). Therefore, a new rough plan was searched in the black domain, which led to the new refinements of the green and dark blue nodes. 49
- 3.6 Each piece of information of an instance is depicted by a fragment with a color specific for its type in the rectangles. The more fractured instances hold more detail, which is reduced in this abstract domain as soon as the oval operators act on them. 51
- 3.7 The right planning state is a successor of the left planning state. The instances with green frames are available instances; the yellow framed instance is the goal, and the nodes marked in gray are the operator and instances touched in the last step that calculated the current state. We can see that the newly applied operator consumed instances that are no longer available to calculate the new instances. . . 52
-

-
- 3.8 Factories with multiple logistics and assembly robots require coordination of discrete and continuous actions, which are partly independent and thus factorizable. Therefore, this is a good field of application for our hierarchical planning approach. 53
- 3.9 This excerpt of the hierarchical planning state shows the abstract plan in the top row. Our planner is currently redefining the first step of this plan. However, the refined layer has no solution to this planning task. Therefore, we can observe the characteristic structure with multiple diamonds, each with its dead end. 54
- 3.10 Sada is an EU-funded project for which we applied the ideas of the set-based approach in the field of perception [3]. 56
- 3.11 This architecture of [1] allows our planner to handle domains with uncertainty to solve cell and perception configuration problems. Our model holds concepts about sensors and operators that implement perception and fusion algorithms. The goal state is represented by a maximal allowed variance. Our planner finds a sequence of data acquisition, extraction, and fusion algorithms to reach this goal. 57
- 3.12 This calibration, presented in [4], calibrates abstraction layers for a given task at hand such that our planner can rely on efficient and meaningful factorizations. . . 58
- 3.13 Even for a task with only two discrete steps, state-of-the-art planners tend to visit thousands of nodes, which prevents debugging and results in poor scalability. . . 59
- 3.14 With abstracted information, a more compact representation for interaction between humans and machines is available, which not only helps during debugging but also for cooperative or collaborative tasks. We can present the current progress, task, and relevant objects to a user, who is then prepared for the next actions of the autonomous system. 60
- 3.15 With our hierarchical approach, we can localize potential errors easily. For each required backtracking, we dump the planning state. We can then analyze the sub-task that was not refinable (top picture). In this sub-task, only the leaf nodes of the diamonds are of interest (bottom left picture). We can check which operations have already been applied, check the desired goals, and dump available instances to call operators that we expect to work but failed (bottom right picture). This process is supported by a graphical tool, which eases navigation, introspection, and debugging. 61
-


-
- 3.16 Our planning algorithm provides introspection capabilities to localize errors. To further investigate problems, we can replace hard-to-debug solvers, such as sampling-based planners, with algorithms that provide feedback even in case of failures. In this sequence of pictures, we can see an error case for a constraint-based motion generation. The robot tries to grasp the screwdriver but fails due to a poorly chosen start configuration. With this feedback, an engineer can adapt the planning task so that solutions exist. 62
- 4.1 For mobile robots, several components are integrated into an overall system, such as the mobile platform, lasers, manipulators, cameras, and grippers. The same applies to the software for localization and mapping or manipulation. 66
- 4.2 Different experts describe the same situation either in an object-centric manner or with a plant-centric view. In the left picture, the box is the focus of attention, and the gripper and robot are the only side characters. This perspective is suitable to describe abstract tasks. On the right side, the robot is in the center of attention and manipulates an object, which happens to be our box [5]. This view is appropriate for low-level task and motion planning and component development. 69
- 4.3 The solutions for the original and optimized domain for the assembly of the box and fixture with four screws start with the same initial steps (first row). After the box is localized and picked, and the lid is refined, we conduct the assembly, re-localize, and pick up the assembled objects. The difference starts after the first screw is picked up and inserted into the box-lid assembly. The original domain has to place the screwdriver to hand over the assembly, as the screws would be facing down otherwise, and place it on the table to reach the intermediate goal. In the optimized solution, we can pick up the next screws right away. Therefore, we can skip the additional handover and pick-up of the screwdriver. Additionally to these described steps, prior to any pick, screw, and assembly, the parts must be localized. The robots must move above the object and take a picture with the wrist-mounted camera [5]. 70
-

4.4	The object-centric a) and plant-focused representation b) describe the same physical phenomenon (relation between box and robot), which is represented by the green sphere in a higher dimension. All instances but some corner cases can be represented in both variants. These exceptions are empty lists, such as a robot with an empty gripper, for which all grasped objects are represented with a list, as depicted in c). If we project this empty robot onto the object plane, we need a <i>no object</i> instance in which the robot can be additional detail. Therefore, we extend the less expressive formulation as in d) and can then project the physical world to each of those representations and thus use them interchangeably to represent our intermediate planning goals [5].	71
4.5	Concepts of an Object with different units	73
4.6	Operator to convert between <i>g</i> and <i>kg</i>	73
4.7	Operator to convert between an object with <i>g</i> and <i>kg</i>	74
4.8	We can decompose the concept so that the concepts on their roles float freely. . .	75
4.9	The conversion operator can be applied to the extracted information of the concept.	75
4.10	With knowledge about the original roles, we can compose all available information to the new object with the converted units.	75
4.11	For both scenarios, each operator is assumed to take one time unit execution time. The baseline calculation (top) for the cost of a planning node ignores the later parallelization during execution and just sums up the execution times of all applied operators. Our heterogeneous state allows us to implement a dedicated per-instance cost calculation scheme (bottom), which reflects a later parallelization chapter 9. There, we calculate the earliest time we can generate a specific instance. The cost of a planning state is then determined by the latest generated instance instead of the sum of all applied operators.	84

I

Hierarchical Approach to Domain Operations for Multimodal Autonomous Systems

1



Introduction

1.1. Motivation and Objectives

In recent years, automation has become increasingly important. The number of industrial robots sold is a good indicator of the trend towards automation in industry, climbing with double-digit percentages. Especially Asian low-cost countries show a high growth rate of installed robots.

However, automation has also reached its limits in some cases. Western countries already lean towards high-tech production due to their labor costs to the point that some companies suffer from the excessive use of automation. Particularly in the case of small-scale production, the investment required to purchase and set up the plant often exceeds the advantages of a traditional production facility with a larger workforce. Let us assume that we want to produce bicycles. This market may be significant, but it is also fragmented. Therefore, we can only produce a small number of entirely identical vehicles until we have to change the type, size, or accessories. Automating this process requires the machines to be adjusted correctly for each lot. Every time the product or the environment changes, the entire system must be validated and possibly reprogrammed in a time-consuming manner. This adaptation requires qualified, trained, and experienced specialists, who are expensive and rare. In contrast, the less automated plant is very flexible due to its employees' intuition and abstract thinking, which allow them to derive all the necessary measures for production from the specifications provided. Current robot systems are programmed for a specific task with little variability in the sequence or parametrization of actions. Therefore, they work best in a controlled environment, providing speed, precision, and endurance. This specialization is why we only see small tasks automated at home, which even then require environmental adaptations, such as cleaning, so that the vacuum cleaning robot does not get stuck on obstacles of the wrong size. These restructurings and transformations into a controlled environment are cumbersome and prevent further application possibilities of automation. Nevertheless, the limiting factor is not necessarily the hardware. For example, surgical robots solve the most difficult tasks with incredible precision. However, manual control leaves all the decision-making and even motion generation

to humans. So, to expand automation's applicability and tackle new tasks, we must push the limits of today's decision-making and control algorithms. Traditional approaches to automation, such as linear programs, state machines, or behavior trees, tend to define the sequence of actions rather than describe the desired goal. For each action and all possible outcomes, the appropriate next operation to reach the goal has to be determined manually. This manual design requirement prevents a generalization of the control sequence or simple reuse of the modules involved in the system. We want autonomous systems that plan their actions on their own, based on an internal representation of the environment. Once the planning algorithm identifies a promising sequence of actions in simulation, we can apply this plan to the real system, compare the execution to the simulated process, and react to any deviations. In this way, we only need to specify the actions and their simulations once, while the system solves the tedious decision-making and parametrization processes automatically based on the goal specification and sensor inputs.

For the example of the bicycle factory, this means that not all robot movements must be programmed manually. When we automate this process, our system gets a model of the manufacturing plant, which describes all machines, materials, and bicycles in production. This model aggregates the declarative information relevant to the decision-making process for manufacturing. The bicycle's type, size, and accessories are examples of such characteristics, as they define the geometry required to calculate collision-free robot arm movements. This information is updated during execution by sensors of different modalities, such as cameras or force-torque sensors, considering their characteristics, such as noise and update rate. In addition, the decision-making process must include all possible actions and processes described by procedural knowledge. Based on this knowledge, a planning algorithm calculates possible solutions and executes them on the real plant.

On the one hand, systems in laboratories already implement such sophisticated algorithms, and even some of today's production systems would have been considered futuristic decades ago. On the other hand, there is still a large gap between experimental demonstrators and real industrial applications. The focus is now primarily on components and individual aspects of such a system rather than on a unifying theory for the integration and decision-making process, ranging from high-level planning to sub-symbolic motion generation. Therefore, the basis for the development of algorithms that target such far-reaching problems, which are both complex in the discrete world and complicated for parametrization in the continuous world, was missing. We already have algorithms for collision-free motions for complex kinematics in high-dimensional continuous spaces. We even have task and motion control algorithms that can handle some objects and their manipulation. However, their computational complexity generally scales exponentially with the size of the domain. Therefore, such algorithms cannot plan all the actions required to build our bike, while our approach specifically targets such large-scale hybrid planning problems.

Integrating different hardware and software components, the reasoning and planning to wire these components together, and implementing a process of decision-making and motion generation are the core of robotics. Computer scientists are developing great search or planning algorithms, the computer vision community has made a significant leap forward with neural networks, and even extremely sensitive tasks can be solved using the latest control theory. Nevertheless,

the integration of all these components is a highly manual task, which limits the applicability of advanced algorithms outside the labs and their demonstrators. Our goal is to adaptively apply different actions based on the catalog of possible operations defined by the current domain. Such a system can react flexibly to new scenarios, environments, or tasks. Moreover, the system is more likely to handle unforeseen situations successfully because it is more flexible and does not require the manual specification of responses for each exception. Imagine that our robot drops the wheel of our bicycle during assembly. In a conventional robotic cell, the position of each part is precisely defined and is ensured by mechanical means such as jigs and fixtures. If something falls, there is no sensor and therefore no way to correct the error. In a system that reacts flexibly to the environment and picks up parts from any position, a fallen wheel may be able to be fixed autonomously nevertheless.

The difficulty of planning for such a system lies in the hybrid and high-dimensional nature of the processes and the number of steps required to achieve the desired goals. Nevertheless, we need an automated decision-making process not only for a partial problem like motion planning or a few steps like task and motion planning but for the overall task, which may also include the logistics of delivering the parts to the assembly station and storing the finished wheel in a warehouse. Since the number of possibilities grows exponentially with the number of possible actions and length of the task, we need ways to counteract the so-called *curse of dimensionality*. When we as humans use natural language to describe the assembly of a bicycle, we explain it in a factorized and hierarchical way. We do not tell the other person to grab a screw and a screwdriver and insert it into the fifth hole but to mount the cup holder on the frame. It is also much easier to identify the root of a problem on the abstract level. Fewer alternatives must be checked, making it easy to see if and which details need additional checking. We need a similar ability to systematically troubleshoot and introspect for the factorized task, as more components introduce additional errors and problems into the domain.

Such a powerful system can then solve different problems in various domains. It is wider than the industrial context we use for the examples in this thesis. However, it offers very different challenges for our approach, ranging from discrete manufacturing, such as product assembly, to process industries, such as food processing, to the configuration of perceptual systems, which is a subproblem of optimizing cell configurations.

1.2. Research Questions

Autonomy is about decision-making in order to achieve a declaratively defined goal. We can do trivial tasks with a handbook that manually enumerates reactions to certain situations or events. Examples of such architectures are state machines and behavior trees. However, the manual definition of all possibilities is not feasible for problems with high variability, especially if continuous properties like geometries and collisions in three-dimensional space are involved. Reinforcement Learning tries to automatically derive an approximate mapping between situations and corresponding goal-directed actions from given or simulated data. However, it is not feasible to program such

a mapping manually, while the data-driven approach requires many test or simulation runs for generalization. A good simulation is rarely available for small lot sizes and unusual scenarios, and a good approximation based on previous test runs is impossible. Especially if the different tasks result in only a few similar runs, it is an excellent approach to try out actions in simulation and find a sequence that is likely to be successful in the real world. This execution of actions in simulation is our definition of *planning*. It can determine solutions for complex and challenging problems where pure heuristics fail. Additionally, we can bring it together with data-driven approaches once we collected enough data. Planning has already been studied in detail for various problems and sub-areas. Our research questions aim to analyze the difficulties these algorithms face in generalizing and dealing with the problems of real scenarios.

The combinatorial complexity of the discrete action space increases to unmanageable problem sizes even with a small number of parts and possible actions. Humans can, nevertheless, solve these problems quickly and efficiently. Even small children manage to assemble a dozen pieces to solve a puzzle. They locate and analyze the objects, pick them up, test, manipulate, and finally use them. If a successful plan for this puzzle consists of three steps per piece (locate, pick up, and insert) for a 36 piece puzzle, a naive approach would test 3^{36} combinations. If each combination were processed within one cycle of a 3 GHz processor, this approach would result in 579 days of computation time. This computation time is much longer than any three-year-old child would need for this type of task. They do not try to optimize for the whole task at once but identify sub-tasks to focus on. This factorization of the problem makes it possible to act on a single object, manipulate and localize it, without running into the curse of dimensionality. The downside of this approach is that we can neither guarantee nor target the optimality of the solution. However, this can be over-compensated because the problem would be unsolvable without this divide-and-conquer strategy. In [6] this problem-solving scheme was identified and mimicked for path planning algorithms that exploit spatial hierarchization. However, can we identify loops and dependent and independent sub-tasks as well as abstractions that simplify a domain, e.g., in temporal space? While this might lead to oversimplifications and biases, this scheme would enable us to solve significant problems in the real-world, which leads to the question:

Research Question 1: Can we provide an automatic and flexible online decomposition of planning tasks to solve large planning problems?

Combinatorial complexity is only one of many challenges in our example. We ignored the geometric constraints and physical properties by now. Even if we find a plan that is a feasible solution at the discrete level, its execution in the real-world will probably fail. Collisions, reachability problems, or deviations from the abstractly modeled properties can invalidate solutions that work on the abstract level. Since some actions can trigger irreversible events, such as broken parts, colliding robots, or dropped objects, the algorithms should be able to utilize simulations to sort out dead ends before testing them in the real-world. Moreover, even the best simulation is only an abstraction of the processes in the real world. Therefore, a system must contain a scheme to detect

and handle deviations during execution and adjust the plan accordingly. This behavior is similar to model predictive control (MPC) schemes, which calculate the control that optimizes some criterion for continuous problems based on a given model. In robotics, problems are hybrid and thus contain both discrete as well as continuous data. People can manage complex and complicated tasks, consider the effects of their actions, and react to external feedback. This raises the question:

Research Question 2: Can we integrate an MPC-like behavior into our planning algorithm so that deviations during planning or execution are detected by different sensors and handled automatically, such that hybrid problems are solved for real environments?

We try to imitate the divide-and-conquer strategy for arbitrary tasks so that autonomous systems can solve previously impossible hybrid problems like humans. This includes factorizing large problems into smaller sub-tasks that can be further divided recursively. For this, we need advanced planning algorithms to handle the complexity with such an approach. We do not want to write such a planner from scratch whenever we apply it to a new problem domain. Therefore, the task must be declaratively configurable by a domain and a problem description. This formalization of the process leads to models of the planning task that allow the development of generic and flexible planning algorithms. Profound literature about modeling languages and corresponding planning algorithms for discrete planning tasks exists. However:

Research Question 3: Can we define a modeling language that covers arbitrary discrete and continuous properties as well as actions, simulations, or executions from the discrete level to real-world execution for our generic planning algorithm?

Another critical point for the use and broad applicability is an easy setup and a high degree of explainability. Usually, apart from research demonstrators, systems are not set up by the developers themselves but by trained personnel. This means that several people and groups bring in components that are then integrated to form an autonomous system. As a result, the various parts are less perfectly matched than in systems designed by a single researcher. The composition of different modules during integration can lead to a loss of performance or even non-functional systems. Accordingly, introspection capabilities are essential for a complex system so that errors can be detected and solved quickly. Additionally, automatic domain optimization can reduce the time needed to set up the system and facilitate configuration. This is not only true for the development and commissioning phase but also during operation. Explainability is also crucial to an efficient operation for cases where even the robust MPC-like behavior reaches its limits or when a human-robot collaborative setup enhances the system's flexibility. Especially then, workers benefit from predictable actions that follow principles like human thinking [7]. This raises the question:

Research Question 4: Can we design a system that enables automatic optimization of the models and explainability for the developers?

1.3. Contributions and Outline

This dissertation proposes a hierarchical planning algorithm based on set-based models to solve significant hybrid planning problems with natively integrated execution and exception handling.

Every planner needs an appropriate description of the task and the environment in which it operates. These models and the respective planners can be tailored to the domain, enabling a higher degree of specialization with better performance but a smaller field of applicability. Examples of such combinations are motion planners that often need a geometric model, a kinematic description, and geometric start and goal positions. The implementation is optimized for robot systems without configurable sampling routines or validators of state transitions. If the model would allow for a generic description of these operations, these planners could also address areas similar in their problem structure, such as optimizing electrical networks, controlling chemical reactions, or task and motion control.

We designed a very generic model to cover a wide range of different domains and problems. In contrast to other description languages like OWL, we include not only declarative (descriptions and things) but also procedural knowledge (actions and modifications) as we follow the basic concept that *anything that is not used is useless*. Since only operations can use things and operations can only be defined with declarative knowledge, we necessarily need both types of knowledge in our model. This is also an essential requirement for automatic factorization and hierarchization during the planning phase and for all domain optimization algorithms proposed in this thesis. Another key feature that sets our approach apart from the PDDL family [8, 9, 10, 11] and the respective planners [12, 13, 14, 15] is the integration of arbitrary code. This not only allows us to integrate sophisticated simulations during the planning phase but also to handle deviations or exceptions natively during execution. This is due to the structural similarity between a refinement that leads to a different solution than its higher abstraction in the simulation, and a deviation between the simulated plan and the execution in the real-world.

We discuss our set-based model for procedural and declarative knowledge introduced with chapter 6 and the implications of the set-theoretical foundation and automatic hierarchizations in more detail in chapter 2.

Based on this model, we developed a planner that is discussed in detail in chapter 7. The key idea of this algorithm is to use an abstract representation of the domain to solve the given planning problem on a rough level and to refine it step by step later. Consider, for example, a logistics problem for which various packaged products must be shipped from A to B. Each crate has specific dimensions, mass, and handling requirements. We could start the planning with all the details and check every combination of crates packed with the means of transport, route, and schedule. It is easy to see that even handling the combinatorics between a dozen crates is not feasible.

In contrast, we look for a rough plan to factorize the overall task and then recursively refine and solve the resulting sub-tasks. We automatically abstract available instances to reduce the branching factor during the search. In this way, we can aggregate several of them and merge them into a single entity with a counter as soon as they no longer differ at the current abstraction level.

very different domains. The field of application ranges, as depicted in Figure 1.1, from factory automation, over assembly planning, to cell optimization. To ensure efficient re-usability, we ensure easy modularization and, even more importantly, composition by the automatic optimizations proposed in chapter 4 that are based on the set-theoretic principles of chapter 2. These optimizations further facilitate modeling and address various challenges in the setup of the domain. The first optimization algorithm presented in chapter 8 aligns the planning domain to compute abstract plans that are less restrictive for refinements to find better solutions faster. The basic idea of this approach is to detect and use homologies of intermediate goals so that the goals can be reached more efficiently with fewer steps. Let us consider the intermediate goal of a box with a lid. A robot holding a box with a lid in its grippers does not fulfill this goal since it is a robot, not a box. Still, it includes the desired instance. Relaxing the intermediate goal can, therefore, eliminate the extraction step for this instance, which involves placing the box on the table. The source of the automatically exploitable optimization potentials in the model is humans who designed the domains sub-optimally. This becomes even more relevant if the domain comprises models from different sources.

Automatic data fusion is another optimization scheme presented in this thesis that facilitates integration. Especially when sensors of different modalities measure an object, the information must be fused from the different sources into a single data point in the environmental model. Let us consider some information embedded in a different data structure, for example, *dimensions* in *mm*. We need to convert this information if a function takes over the data structure but needs other units, e.g., dimensions in *m*. The data fusion algorithm verifies that we can extract, convert, and merge the information with the first unit into the original instance without invalidating other information in the data structure.

A third optimization scheme published in chapter 9 is the parallelization of formerly sequential plans. These algorithms firstly extend the scheduler to optimize for later parallelization and, secondly, add an execution engine for the refinement process that identifies independent parallel strands and executes the actions as asynchronously as possible until the next synchronization lock is reached.

2

Set-Based Hierarchical Modeling Approach

In this chapter, we discuss the theoretical foundations of our models, which, as proposed in [16], include both declarative and procedural knowledge. As stated in the introduction, these representations form the basis for all algorithmic actions in the autonomous system. Both the planner in chapter 3 and the optimization methods of chapter 4 use the properties presented in this chapter. We will first discuss declarative knowledge and motivate the use of set-theory as its foundation. Then, we will discuss the closely related procedural knowledge and its representation, properties, and connection to declarative knowledge.

2.1. Declarative Knowledge

Declarative knowledge describes both abstract knowledge and concrete facts of the world, such as sensor measurements or a processed representation of objects derived from multimodal sensor data. Structuring them is especially useful because the actions discussed in section 2.2 are based on them. In this section, we explain our approach and its set-theoretic grounding and discuss the representations that enable the automated processing presented in the following chapters.

2.1.1. Motivation

When people speak, they exchange declarative knowledge. We can communicate facts and lessons in a way that others who have not had the same experiences can learn from them. We can also organize our activities, delegate tasks very compactly, and validate the results of our efforts, which is a decisive factor for human progress. However, if we want to leverage the potential of autonomous machines, we must also find ways to command robots. That said, the languages we speak and write represent a form of communication that is hardly standardized. There are attempts to analyze the information of spoken or written words and make it interpretable for computers. However, even if this should one day work perfectly, natural language is not a suitable and optimized format for the internal representation of declarative knowledge within autonomous machines. For this, we need

a model that can contain information similar to that of a language but in a structure that algorithms can quickly process. We need the link or a common form between the representation for humans and the formalized form for machines because the essential part of autonomy is the declarative definition of the task by humans, which is then solved by the machine.

Therefore, we aim to develop a formal definition of the information so that the autonomous machine can efficiently work on it while humans can read and write it. Only then can the autonomous machine benefit from human domain-specific experience and knowledge. In addition, the results can then be checked and adjusted since such an internal representation allows straightforward human interpretation. Another critical point is that nowadays, problem classes relevant to many fields of application in industry and private life can be solved by humans but not autonomously by robots. With a formalized representation, we can introduce problem-specific knowledge manually for automated solutions or develop algorithms that solve that problem class autonomously. We need a formalized representation of our knowledge with which the autonomous machine can work for all these advantages. It should be mathematically sound so that we can implement algorithms and validate operations based on it.

People generate knowledge by generalizing observations and experiences, which not only makes the world more accessible to us but also enables the simulation of actions and, thus, planning and weighing different options against each other.

Example 1:

Let us talk about a specific car, for example, *Bob's car*. We do not know anything concrete about this vehicle yet. However, we have an abstract understanding of the properties and abilities of this object and can answer some questions. We rely on assumptions gained from generalized experiences with our world. For example, we can tell whether we can use this vehicle, as we know that we can drive cars generally and therefore can also operate that specific instance. For some questions, the information given is too abstract and, therefore, not precise enough, so we need additional information. For example, to answer further questions, we might need to know whether the car has a trailer hitch or the weight of the car.

Our set-based models have the same objectives as spoken language, namely, to allow for abstraction and planning. However, knowledge and processes are described in a machine-readable and formalized form. Based on these models, we can develop novel hierarchical planning algorithms and domain optimizations while maintaining the ability to engage in introspection. Since our declarative model is based on abstracted information or data points, data-driven approaches can further fine-tune and optimize the models. Algorithms from the big-data domain can analyze given information and compile synthesized knowledge into our models. With them, the information can be reused and applied meaningfully to solve given tasks.

2.1.2. Theoretic Grounding

We call pieces of compiled information in our model *concepts*. They define the structures within our knowledge representation. Concepts allow us to define a partial ordering between *instances* that are the actual pieces of information of the physical or simulated system during execution or planning. We use these concepts at various points in this thesis. The definition of procedural knowledge that we discuss in section 2.2 is based on them, as well as the ordering and hierarchization of the domain and the validation of the planning results for the hierarchical planner presented in chapter 3. The concepts cover both discrete and continuous properties. They can contain concrete information about physical objects as well as abstract properties such as parameterizations, costs, or probabilities. The definition of our concepts is related to the formal concept analysis [17], which allows broad applicability in various domains. In section 3.3, we discuss several example problems, such as task and motion planning, root cause analysis, and optimization of cell setups. Moreover, our formalism can describe and derive different levels of abstraction and automatically infer relationships between these levels, which is essential for our hierarchical planning presented in chapter 3 and a difference from state-of-the-art modeling and planning algorithms. These representations belong to the same concept class Γ . Each concept class has a concept base B_Γ , defined by a set of instances. Concept bases are not necessarily finite, such as the set of all real numbers, set of locations, or set of masses. Other concept bases can be large but finite, such as the set of all instances of cars, people, or boxes. We, therefore, define the C concept as a subset of the concept base so that we can formally specify additional properties on it. These properties are named by *roles* r , which define further subsets on the concept base. These subsets are formed by intersections with other sets, which are also described by concepts. We represent these *composite concepts* by directed trees. The root node of these graphs defines the concept base B_Γ . Leaf nodes contain concepts that are defined by their concept base alone. Intermediate nodes represent a recursive application of composite concepts to define this role. Each concept on a new role forms a new intersection on the concept base set. These restrictions narrow down the volume for an instance of that concept. Therefore, with the additional information, which specifies a larger number of properties and thereby reduces ambiguities due to non-specified characteristics we get a more detailed instance, as highlighted in example 2. With this definition of declarative knowledge, we can formulate domain operations that modify and operate on our model algorithmically. For example, we can compute a partial order between two concepts that we call *more detailed than*. With the definition of concepts, we can algorithmically identify which concept is a subset of, and therefore, is more detailed than, another concept. We can calculate this relative order by recursively checking the following nodes' roles and (composite) concepts in the graph representation. If C_a has all the roles of C_b and the same concept base B_Γ , and the subsequent nodes are equal or more detailed than the concept on the respective nodes of C_b , then C_a is more detailed than C_b .

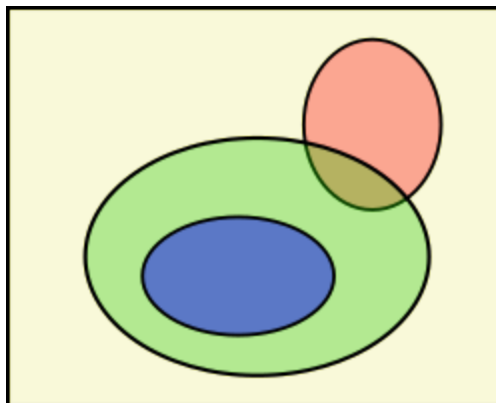
Example 2:

Figure 2.1.: The differently colored areas depict the base B_Γ (yellow rectangle) and concepts that have different intersections depending on their properties.

Consider the example of the pose of an object in Figure 2.1. The concept base B_Γ would be the set of all possible poses with different levels of abstractions (yellow rectangular). On an abstract level, this can be a discrete quantity such as *on the table* (green ellipse) or *in the gripper* (red ellipse). Of course, these sets can overlap, and we can use this information to evaluate prerequisites for actions and their simulations. We may already discard an assembly or drop if the object is not in the gripper, or a pick may not be feasible if the object is not on the table. For more detailed validations, such as collision checking, we need additional geometric information instead of just the semantic pose. In our example, we know the position within a tolerance of 5 cm (blue ellipse), which is entirely covered by the semantic position *on the table*. With a continuous pose, which defines the relative transformation and rotation between two frames, and a geometrical representation of the objects involved, we can compute distances and check for collisions. The discrete position describes several continuous poses; thus, both pieces of information can be valid simultaneously.

Our *domain* is composed of all knowledge that is necessary and available to solve a particular task. We can sort the concepts into a directed graph based on their partial ordering in such a domain. For each node of this subgraph, the descendants hold the more detailed concepts, and the ancestors hold the more abstract concepts. Each directed subgraph must have a single root node and a single leaf node. We can easily construct the root node that denotes the most abstract concept of that subgraph by eliminating all roles, which leaves the concept that contains all the elements of the concept base of the concepts of that particular subgraph. Since additional roles can only eliminate instances, all concepts in this subtree must be more detailed than this concept. We can calculate the most detailed concept when we merge all the concepts of this tree, which adds the missing roles to a newly created concept until each concept in the subgraph is a sub-tree of that concept. The properties of directed graphs in our hierarchical domain, such as the existence of these root and leaf nodes, are described and discussed in a mathematical context in [17]. We define *base types* as concepts with neither successors nor predecessors in the entire hierarchical

domain, which can only hold if they have no roles at all. Examples of such base types can be *real numbers*, *bool*, or *color*. This property, however, is domain-specific and can change with the context set by other concepts. We will briefly discuss how each of these concepts can be ordinary concepts in a different domain, which, for example, has successors in remark 1.

Since we use trees to represent our declarative knowledge, as opposed to graphs as in OWL [18], it can be represented as structs or classes in most programming languages. From this point of view, the directed graph, which has emerged from the partial order, is an inheritance hierarchy. All more detailed classes inherit from their ancestors. This hierarchy is automatically and dynamically updated as new concepts are added or modified, which is relevant when we solve a new task requiring newly modeled declarative knowledge or a composition of different domains. An essential part of our hierarchical factorization is checking whether an instance fulfills the requirements imposed by a second *goal* instance. The goal must be a goal region for continuous properties, as a planner cannot reach it with non-zero probability otherwise. Just consider goals, which are imposed on the real-world execution by a simulation phase. The real world will never exactly behave as the simulation predicted. So, the goal will never be reached to the last digit, despite the state being *close enough* to the simulation results for the given task. We must incorporate this *close enough* to our models. Thus, a concept not only defines which instances it represents and the minimal set of details it holds but also provides a relaxation for comparing those instances. By that, an instance describes a region rather than a point during comparison, which goes hand in hand with remark 1.

Remark 1:

A duality between concepts and instances is only resolved by the context of a specific domain. Each instance can define a concept and represent an instance in a different domain based on its limits of relevance in the given scenario.

As soon as an instance is an abstraction for which we have defined actions, as introduced in section 2.2, that instance defines a new concept based on dependent typing.

On the other hand, a concept is only a graph, which is declarative data. This data can be handled in a so-called *meta-domain*, where the set of all concepts defines a concept, which makes the concept of the common domain an instance in this meta-domain.

Let us look at an illustrative example: The concept that describes boxes in the assembly domain is an instance of the meta-planning domain. There, we have operators that can, for example, calculate the ordering relative to a second concept-instance.

Based on our declarative model, we can check algorithmically if an instance describes an area that overlaps with the region specified by a second goal instance. The first prerequisite we verify is that the instance belongs to a more detailed concept than the goal's concept. Then, we can check if the value on each leaf node of the instance is a subset of the respective leaf node value of the goal. We need this subset comparison at the leaf nodes to relax the goal area to the extent that a planner can reach it with a non-zero probability.

To illustrate this concept, we discuss example 2 in more detail. The concept of a *double* is a

basic type in our example domain. It represents arbitrary floating-point numbers. At some point, we must decide whether an instance is close enough when we use it in a composite concept that can be a goal, such as our pose. We know that in our domain, we can only perform calculations with machine precision, which is even more precise than necessary for real-world processes. So, we add a particular environment for which the instances also fulfill the goal defined by the concept. For example, the double concept has a smaller region for which instances compare equal to floats.

The goal of a planning task is defined by an instance, as described in the upcoming chapter 3. During the process, checking whether a given instance satisfies the goal instance is necessary. We call this instance *similar* to the goal instance. The definition of this property is comparable to the definition of *more-detailed than* on concepts. While the concept is more detailed iff **all** of the instances this set describes resides within the more abstract concept, an instance is similar iff **any** intersection between the two sets exists.

With this perspective, the new, data-driven generation of concepts is straightforward: Once a superset of instances can be identified, we can declare a new concept on that set. In this section, we have focused on models that can be relied upon because humans created them. Therefore, we had a top-down view, from abstract structures to more detailed ones. Instances can potentially be any data point with no apparent relationship to one or more concepts. This classification is similar to the exciting question of whether two concepts are the same despite differing concept bases or roles, which can happen for composed domains, for example. An algorithm that detects and reformulates such cases is discussed in chapter 4.

As mentioned above, we compile our models into classes in various generic programming languages. There, we represent the concept hierarchy and generate so-called *compare functions* that check whether one instance is similar to another. For our example 2, we must deal with symmetries of quaternions representing our orientation and epsilon environments in Cartesian space, which requires manually written arbitrary code for our compare functions, which is also part of our model. In our examples 1 and 2, the concepts are defined not only by the properties and their roles but also by values and relationships between different pieces of information. For instance, the box sits at a specific semantic position, like *on the table* for values in specific intervals of the x , y , and z coordinates. Therefore, our model implements dependent types that can be automatically composed and hierarchized to facilitate planning.

2.2. Procedural Knowledge

Our formal models include not only declarative knowledge, as discussed in the previous section, but also procedural knowledge that describes actions and operations that can be performed during planning or execution in the real-world. This section discusses our view of procedural knowledge, the characteristics of our model, and the differences compared to other representations.

2.2.1. Motivation and Requirements

In the last section, we learned how to use declarative knowledge to describe our system's task and environment. Additionally, the autonomous system must be able to influence its environment to be relevant. All our declarative knowledge would be useless without operations that can influence and modify it. Therefore, this section discusses procedural knowledge, which describes how the systems and the environment described by our declarative knowledge interact, are used or processed. Only with those two parts of knowledge we can discuss the planning algorithms that solve any tasks flexibly and autonomously, in the next section.

Remark 2:

Often, execution, planning, and process control are considered different approaches that work together but are still distinctive. For us, these terms only describe a spectrum featuring different views on the same algorithmic basis for which some transitions are more intuitive than others.

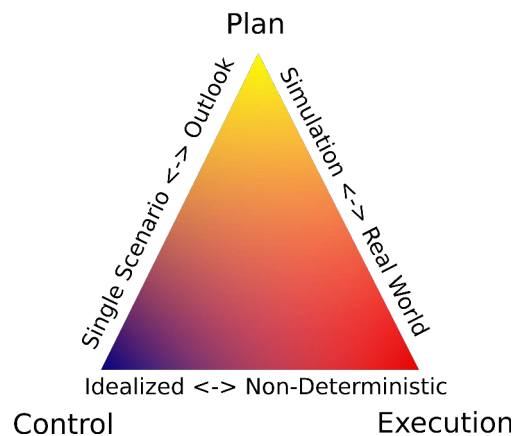


Figure 2.2.: For autonomous systems, there are three dimensions to consider. Idealized approaches are covered by planning and control, while sophisticated and possibly non-deterministic simulations blend with real-world execution. Similarly, control behaves like planning once more and more alternative scenarios are considered for simulated future points in time.

As stated in remark 3 and depicted in Figure 2.2, there is no hard border between planning and control. The same is valid for planning and execution as described in remark 4; thus, the three terms describe a continuity.

Ontologies, such as OWL, have a powerful type system. Nevertheless, they can only implicitly model actions, limiting their use as they require additional techniques and models to allow for planning, which requires a description of actions and their outcomes to calculate new states. This application of actions must be done both in simulation during the search phase as we want to find a path to a goal state as well as in the real world to validate the simulated plan and actually complete the task.

As mentioned in the introduction and in remark 2, planning and execution go hand in hand. There are always slight deviations between the simulated and the real-world due to sensor noise,

model inaccuracies, or simplified simulations. We have mechanisms to deal with this noise, as our declarative model specifies the allowed deviation for the given domain. Related to this is the similarity between instances, as described in the previous section. However, mismatches can quickly amount to more than *just noise* and require additional handling, which leads to an instance that is not similar to the expectation and, therefore, requires a different sequence or parameterization of actions to reach a goal state. To accommodate this, we implement an MPC-like behavior in our planning system, which blurs the line between planning and control as stated in remark 4. For procedural knowledge, this means that not only simulated effects but also real-world operations must be incorporated, such as the movement of an arm, the measurement of forces and distances, or the validation with optical sensors.

Remark 3:

For us, there is no difference between planning and control. Process control algorithms calculate controlled variables so that the difference between process variables and setpoints for a continuous system is minimized. Planners, conversely, search for a discrete sequence of actions such that a specified goal state is reached from some start state. During planning, actions are applied to the state in simulation to obtain new states that can be checked for goal achievement, in contrast to explicit formulas that determine the control variables for the process control. However, model predictive control (MPC) has blurred that line, as even for this control theoretic approach, the action induced by the control variables is simulated [19]. Any difference vanishes with the MPC algorithms for hybrid systems, which even consider discrete properties and discretized control variables. With that, *control variables* is just another term for *actions* or *operators*, and *state* and *process variable* become synonyms as well. The communities still focus on different aspects in their discussions, like the proof of maximal calculation times (real-time capability) or convergence rates to optimal solutions. Additionally, they commonly calculate commands or take feedback into account at different rates, but the boundary becomes blurred.

Our planning system is based on hierarchical abstractions to counteract the exponential increase of combinatorial complexity that comes with additional alternative actions, objects, or length of a successful plan, also known as *the curse of dimensionality*. This divide-and-conquer strategy is inspired by humans' natural approach to large problems. When we get a task, like moving the box of our example 2 to a different position, we consider actions on an abstract level first, like picking it up by hand or kicking it with our feet, and sorting out infeasible options immediately. That means we already know we do not need a bulldozer, a shovel, or a telephone to move this box. We can then consider the most promising candidates in more detail and estimate the required effort, damage to the goods, and other KPIs. In the end, we execute the planned sequence of actions, monitor any deviations, and replan if required. We follow the same algorithmic approach with our hierarchical planning and factorization scheme.

The previous section described the hierarchical order of our declarative knowledge, which is based on the physical instances and their abstractions. For procedural knowledge, we need a similar construct, which can then be used by the planner to automatically factorize the planning

problems and get rid of domain-specific factorization strategies such as [20]. Our actions are ultimately based on the operations required for execution in the real-world. All other operations are simplified versions, or in other words simulations, of these actions. These simulations can be synthesized by neglecting details, generalizing, and combining several steps into a single one. Our model must describe the partial ordering between actions on different levels of abstraction. Depending on the domain, there can be no real-world execution, but only a *most detailed level* of simulation for certain steps. We discuss examples of these domains, such as root cause analysis or cell optimization in section 3.3.

The need for real-world execution and arbitrarily detailed simulations requires embedding black-box code in the model of procedural knowledge. If we would only describe pre- and post-conditions with discrete variables, such as *object in gripper*, *object on the table*, we would never be able to cover the complexity of robotics with reachability issues, collisions, time optimality, acceleration limits, sensor noise, and other challenges. We must integrate multiple complex sub-systems to fulfill a given task. Due to the curse of dimensionality, the interaction with the real world can neither be fully covered by discrete properties nor discretized without the combinatorial explosion. We can use model-free, data-driven algorithms such as reinforcement learning for some sub-tasks. However, the system itself will never be model-free. There will always be some manually coded verification of the actions to prevent damage to the hardware through collisions, such as value ranges, continuities, or validated and enforced protocols. So, while the motion generation may be model-free, the overall system is not. Our model can be the melting pot of these upcoming data-driven algorithms and classical approaches to implement a powerful autonomous system.

We have only one requirement for the code to be included in one of our procedural models: The actions of the abstract layers must be stateless. Otherwise, we would not be able to use them during planning. Nevertheless, the real-world will never be stateless, and we can still handle it. However, there can be no stateless action that is more detailed than a stateful operation, which leads to the necessity to model all information needed to calculate the outputs. For the system, any missing input, such as communication with external components, would resemble an unobservable state and violate this essential requirement. As an example, we must model the seed for pseudo-random operators such that the internal state of the pseudo-random-number-generator is exposed to the planner. Additionally, we cannot refine the real-world manipulation of an object, as the internal state cannot be exposed to the planner.

2.2.2. Theoretic Grounding

We call the procedural knowledge of our model *operators*. They act on our declarative knowledge and functionally describe the changes we expect in our real or simulated world utilizing mappings between input and output instances. Since this mapping can take any form, it cannot be written as a reduced set of instructions. Therefore, we allow the implementation in a black box code as long as it has no internal states. Alternatives are mathematical formulas or modeled structures, such as data flow graphs. The sole purpose of our models is to automatically process the specified domain

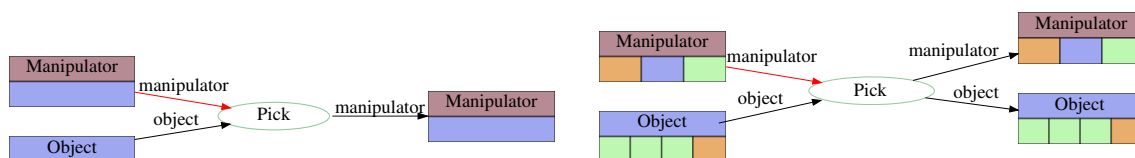


Figure 2.3.: Graph structure of an abstract pick operator (left) and a more detailed version (right) that has additional outputs to handle exceptions, as well as more detailed inputs and outputs.

knowledge to facilitate or solve tasks. Our key algorithm is the factorization by the hierarchical planning discussed in chapter 3. This planner needs additional information about the actions it can perform for planning and execution, which are handled similarly in our approach as remark 4 describes.

Remark 4:

Our hierarchical approach is based on different levels of abstraction that represent differently detailed simulations. With every introduced piece of detail, the task can be unsolvable despite valid plans on the more abstract level. Those abstractions can range from the purely symbolic information of the existence of an abstract object, such as a box and its processing, over a geometric level, which includes dimensions, grasp locations, and transportation paths, to physical simulation, including gravity and forces, to real-world execution. For us, the real world is just another refinement of our most advanced simulations. As with every refinement, something can go wrong and require replanning. Thus, in our planning scheme, error handling and replanning are natively incorporated, and we can include arbitrary constraints to the possible actions, even gravitational restrictions like [21], as long as we can simulate the outcome of the possible actions.

Operator Structure

The most important information about an operator for our planner is the data required to call it. In addition, the potential outputs are valuable information in case a backward search is to be performed. Therefore, we modeled our operators as directed, bipartite graphs. The first of the bipartite sets is used for the inputs and outputs described by our concepts, which form the declarative knowledge. The second set of the bipartite graph's node types are the function blocks that describe the mapping between the inputs and outputs.

Example 3:

Our example operator *pick* takes an object and a manipulator to attach the object to. The directed graph of this operator is depicted on the left-hand side of Figure 2.3. The more detailed version for real-world execution operates on more detailed instances and returns the object in case of a failed grasp attempt, as depicted on the right-hand side of Figure 2.3.

For a pick and place, the operator on the abstract level would receive an object and a manipulator as input and return a manipulator as shown in Figure 2.3. An operator can receive several instances

of the same type. Therefore, a distinctive role is required for each input and output. Note that we describe the operator with concepts as inputs and outputs. During planning, however, it is called with instances that fulfill these concepts.

Remark 5:

Operators are described with the help of concepts. However, according to our definition, concepts could also be defined by their impact on an operator.

It would not make sense to model the operator with an instance since this instance is unique, and the operator could be called with no other value. At this point, the duality of remark 1 comes into play again. The single and, therefore, static value could be incorporated into the operator without being exposed as an input. However, if the operator can handle other values on this input, all these inputs would form a set of *valid inputs* for this operator.

This automatic abstraction complies with our definition of concepts with the dependent type *applicable to the pick operator*. Indeed, we cannot tell whether an instance belongs to this set until we have seen it before or called the operator with that instance, but *boxes with all sides shorted than 5cm* might give a good guess and therefore result in the concept of probably graspable boxes. Nevertheless, we could find a superset to all these potentially applicable instances, for example *all objects lighter than 100 kg*, which is an upper bound for the hardware, and model this relaxation as our new concept, which we then use as an input type. When we choose the concept for our operator inputs, we thus model a superset of instances that provide the necessary information for output computation and hold feasible values.

In section 4.7, we discuss the data-driven generation of new concepts based on recorded data. As remark 5 suggests, there is a circular relationship between concepts and operators, which highlights that the model must describe both interdependent parts of knowledge to be self-contained.

Our model for procedural knowledge holds a set of important information for the planner, which we describe in the following:

Consumed Inputs

Our operators not only generate new information but may also have to modify their input. For our pick example, the instance of the robot arm will not be the same after moving it. The positions have changed, and when the pick is done, even a discrete property changes as the box is attached to the arm. To model cases in which changes are applied to an input instance without violating our model's requirement for functional operators, changes are not applied to the inputs themselves. Instead, operators return a modified copy of the unmodified inputs. However, during planning, this would lead to two conflicting instances describing the same real-world robot, which is infeasible. To maintain consistency, we must tell the planner that this specific input instance is invalidated as soon as the operator acts on it. We, therefore, mark that input as *consumed* in our model, which is depicted by the red arrow for the input in Figure 2.3. This way, the old instance can be removed from the state once the operator is successfully applied.

Outputs and Exceptions

All modeled outputs of our operators are **optional**, meaning that the maximum level of information that this operator can potentially calculate is modeled. During execution, only those instances that can be calculated with the given inputs are returned. If the operator is not applicable, even though the instances on the inputs match the correct concepts, the operator will not return anything. Some optional outputs will only contain information in case of an undesired event. These outputs, which would be exceptions in other programming languages, are treated the same way as all other outputs. Look at the example of the pick of example 3, which is executed in the real-world. Let us assume that closing the gripper fails for some reason. Typically, the operator "pick" only returns the robot, which internally holds the object. In case of a failure to grasp the object, which can easily happen during real-world execution, returning the original robot without any modification would result in a loss of information. We moved the arm for the pick and modified the robot instance. Additionally, the box would be gone because it is neither grasped and described by the robot nor exists as a standalone instance, as it had been consumed. We know that the box is still there and not in the robot's gripper. Therefore, the box's modeled output, which is usually not used, is filled with information about the box's state for this failed pick attempt, which allows the planner to deal with unexpected conditions and continue planning.

Wildcard Inputs

Our declarative knowledge is defined in a purely constructive manner. We can only model based on existing objects or information that defines our sets. We need to lift this restriction for operator inputs. Consider the pick task of example 3. We want to move the robot arm close to it to pick up the box without collision, which is usually computed by a motion planner that tests configurations or actions of the arm and checks if they can be connected and executed without collision. We cannot specify all the available empty space in which the robot arm can move freely. The only information we have is the position of all the objects in our scene that are known to us. Therefore, we can only tell that the movement is collision-free if we assume a closed world and rely on the completeness of our knowledge about the current situation. To model operators that rely on the assumption of a closed world, we can describe that any instance of the current domain that matches the specified concept should be used as input for that operator. Only with this non-constructive approach can we handle tasks such as path planning.

The next question related to the closed world assumption is the relevant size of the current domain. The basic idea is to factorize the problem so that we can handle it despite the curse of dimensionality. Consider the planning of an entire factory. If we check for collisions within a single cell in that scenario, the input asking for *any* object would include all the screws, cutters, and boxes in the factory, which is obviously neither feasible nor necessary. Therefore, we can specify the domain's extent relevant to the operator, such that the closed world assumption holds. In chapter 3, we discuss how to specify the different domains based on our model automatically.

Operator Entropy

During planning, our operators are applied on a set of instances. We assume that our operators are purely functional. Thus, if we provide the same inputs, the result of these executions is always the same. This is especially important because the plan we create on an abstract level will eventually be refined, resulting in a real-world execution. If success and failure are random and do not depend solely on the inputs, we cannot be sure we have found a feasible plan for this refinement. As a result, no hidden states are allowed within the operator's algorithm. With access to the implementation, internal states can easily be made explicit by additional inputs and outputs. This technique allows, for example, to implement a state machine that transparently loops back the resulting output state as a new input. Pseudo-random motion planners can be formulated deterministically by providing the seed as an explicit input. Our model facilitates the definition of operators that return different outputs on the same inputs only a few times. An example of a function that uses this is a grasp planner for a box that returns the four most practical grasps on an object. For this purpose, we model the operator's so-called *entropy*. During planning, we can apply this operator with an additional parameter that tells the implementation of the current number of calls. Since the entropy can also be infinite, e.g., for the random motion or state sampler in motion planning, the planner can decide how many attempts the operator should make with the same input.

Irreversible Operators

Another challenge is operators with internal states for which we have no access to the implementation. The most prominent example is an operator that interacts with the real-world. It has "internal states" and can fail, even though it worked with the same inputs just a few moments ago. Since we do not have access to this internal state, we cannot reset and try again in case of failure. As noted in remark 4, planning and execution follow the same mechanisms in our approach. The only differences are the level of detail in the "simulation" and the irreversibility of executions in the real-world. Our planner must respect that we cannot reapply operators with internal states, as their internal state cannot be reset. Thus, changes in the input, such as the consumption of inputs, are persistent. Therefore, our model has the property "irreversible" for every operator with an internal state, which our planner then uses to prevent inconsistent forks during our search. To allow for an MPC-like behavior of our planner, even in the case anything unexpected happens during the execution of an irreversible operator, the design rule to return all consumed inputs must be met, which means, for example, that a pick operation, which consumes the robot and a box, must return the robot and box even in case of some failure during execution. For sure, the returned state must represent the most recent modified state. In our example, in the case of an unsuccessful grasp, we would return both box and robot instead of the robot with the attached box, update the position of the robot and gripper to our knowledge, and possibly increase the uncertainty of the box position. That would allow us to restart the planning and, therefore, find a more suitable and updated solution to the current situation in an MPC-like fashion.

Operator Hierarchy

For our concepts, we have already discussed the hierarchical order that is immanently important for our planner in chapter 3. This concept hierarchy is a prerequisite for partially ordering the procedural knowledge modeled by the operators. Let us consider the pick scenario. On an abstract level, the coarse operator would solve the task of assembling two parts, such as a box and a lid, in just a single step. Like example 2 for declarative knowledge, we can find a refinement for this abstract process that also considers continuous properties. In the abstract version of this example, we only indicate whether an object is sitting on the table or is currently grasped. Depending on this state, we can then apply different operators. Objects lying on the table can be grasped, an object in the gripper can be assembled with another one on the table, and we can place objects in the gripper on the table again. It is easy to see that this abstract action can be refined with several operators. So, the partial ordering of operators is not a one-to-one relationship between two operators but a one-to-n for a single operator and a set of operators. We rely on their properties, inputs, outputs, and the declarative hierarchy to check whether a given set of operators is a feasible refinement. For a valid refinement of an operator, we need a network whose leaf nodes provide a subset so that a more detailed concept can be found for each output node of the coarse operator, as depicted in Figure 2.4. Remember that our outputs to operators must cover the most detailed instances that this operator can generate. This becomes important when verifying the valid hierarchical ordering of operators. Moreover, this is a necessary but insufficient requirement for the partial ordering of operators.

We observed a hierarchy between operators with the same inputs and outputs. The difference between these operators is the precision of the simulation they implement. It can vary, e.g., in different discretizations of time or space in the simulation or the relaxation and simplification of some physical processes. Most importantly, however, they differ in their computational complexity, which affects the runtime and the quality of the results. Therefore, it may be advisable to check the cheaper operator first, although the model does not differ in input or output types. To consider this in our model, we introduce so-called key performance indicators (KPIs). They describe the expected runtime of the operator once it is called with specific inputs. We also model the expected cost of the application and complete refinement of a particular operator, which becomes important for domains with several alternative paths that all lead to the same goal but differ in cost after execution on real hardware.

A good design of the hierarchical domain considers all possible decisions that can take place in the given domain. For all those decisions we need separate concepts with the respective instances. Our operators take those concepts so the planner can transparently decide what to do. For each refinement of the operators, a new dimension of a possible decision should be spanned so that we ultimately cover the whole decision space. We would start with the decision that rules out the largest chunk of the decision space on the highest level and proceed to more subtle decisions later.

Apart from the modeled input and output hierarchy and the ordering of the KPIs, an addi-

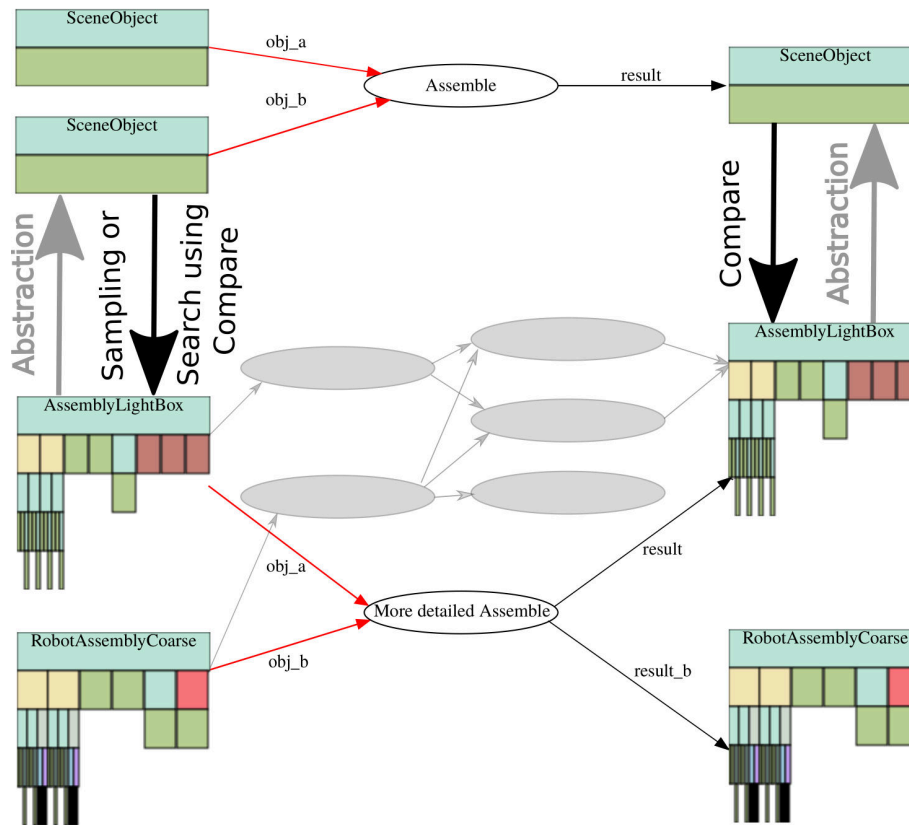


Figure 2.4.: The high-level operator may either be refined by a single operator or a set of operators that can form a network. In either case, the leaf nodes must be more detailed and have similar instances to the outputs of the coarse operator. Additionally, root nodes should be in the more detailed level for each input of the abstract operator [2].

tional requirement is a relaxation of the conditions in the coarser layers, which means that for any sequence in the more detailed layer, an appropriate solution on the coarser layer must exist, as otherwise possible solutions might be discarded based on too strict assumptions in more abstract layers. In particular, the result of the more detailed layer must be similar to the coarse layers outputs as shown in Figure 2.4. This requirement is a natural extension of our first prerequisite: finding more detailed output concepts on the more detailed layer. We cannot guarantee that this property will be fulfilled, but we can rely on proper modeling or validate it with test run data. However, we can check if the property is at least fulfilled on the meta-level, such that a valid path based on the input and output types exists, neglecting the implementation of the operators similar to approaches in discrete domains [22, 23, 24, 25].

Computation time and expected results of the operators may be domain-dependent. Therefore, the operator hierarchy, as opposed to the concept hierarchy, may not always be automatically computed from the models. We also allow humans to explicitly define the operator hierarchy for a specific domain to overcome this issue. For example, certain operators, such as additional lighting to take pictures, are not applicable in hazardous or explosive environments.

Remark 6:

We already stated that a concept can be defined by the behavior of the operators that act on them (cf. remark 5). Additionally, we discussed that the hierarchy of operators depends on the domain. The conclusion is that even between structurally similar concepts, a partial order can be determined only by the operators that use them. Therefore, the ordering of the concepts can depend on the domain and the operators available there.

Any operator with more detailed operators can span a new planning domain in our hierarchical factorization. These domains have specific characteristics that make different planning strategies advantageous. The operator in our model also defines this parameterization for the later planning process, such as search heuristics or scheduling. Furthermore, in the case of coarse operators that cannot be called only once, such as samplers for motion generation, the refined layer must preserve the decisions made on the more abstract layer. Therefore, we can specify which operators should be called with the same call count as the operators in the more abstract layer during refinement.

Remark 7:

Operators are graphs with relevant planning properties and a defined dataflow or blackbox that implements the behavior. This format is purely declarative and thus compliant with our concepts. Thus, we can implement operators that act on other operators described by concepts. We plan, schedule, or optimize the models with these meta-operators. Therefore, we can describe the whole system with its own structures and can let it plan and optimize itself after a bootstrapping step. An obvious point where we apply this functionality, in addition to the hierarchical planning discussed in chapter 3, is the automatic refinement of the operators with a modeled implementation discussed in section 4.5 or the automatic data fusion in section 4.3.

We use a bipartite graph structure to model the data flow, which describes which sub-operator inputs or outputs are fed to which input of a sub-operator in so-called plan-templates. For these operators, a connector node of the same bipartite type as the operator nodes connects each output to an input. This way, we can model sequences or loops of operators and introspect and manipulate them algorithmically. For example, this structure can model a scripted pick operation that simulates the arm's movement over the object before fine-localizing and grasping it. We can then automatically refine this so-called plan-template as described in section 4.5 and execute the same logic but replace the simulating operators with their executing pendants. This will reduce modeling effort for sub-problems that a fixed set of rules can solve without planning.

Similarly, we can automatically generate abstractions from a very detailed plan-template by replacing detailed sub-operators with abstracted versions, which are easier and cheaper to calculate.

3

Hierarchical Planner

Planning is about trying out actions until a given start state is transformed into the desired goal state. It is evident that an exhaustive brute-force search is very inefficient and time-consuming. This search type is only possible in some small scenarios due to time, cost, or technical reasons. Therefore, we usually rely on some heuristics for domains we have structural knowledge about, such as logistic use cases, manufacturing, or process industry. We must only apply brute force to less structured domains or new and unknown tasks. Our goal is to write a planner that can solve generic problems efficiently. In the previous chapter, we learned about the model, which is the prerequisite for an automatic operation on arbitrary domains. Here, we discuss the main features of the planner, cf. chapter 7. Additionally, we will analyze different ways to classify our algorithms relative to existing approaches, illustrate their introspection capabilities, and present examples of domains to which we have applied them so far.

3.1. Motivation

In the long term, we want systems that independently solve the tasks we define. Instead of a fine-grained, tedious, and time-consuming specification of every step the machine should do, we want to declaratively define tasks at a high level. If we compare the current situation in automation with the level of detail required for the task specification for differently experienced employees, we come to sobering assessments. In automation, we must specify tasks in great detail, similar to or even worse than working with trainees on their first day, as humans can at least bring in some common knowledge that they transfer from other domains. Only application-specific peculiarities, such as manufacturing steps or details in metal processing, must be explicitly mentioned. On the other hand, today's machines could even destroy themselves without detailed specifications for the operations. They are far from being able to transfer knowledge from previous tasks to newly specified ones. They cannot perform a task independently and derive all the necessary sub-steps on their own if it is only specified at a high level. This puts a higher burden on the humans that need

to specify every detail of the execution. Additionally, the program depends on the production plant and the task at hand. If we could only specify a high-level description of the task, any change to the product or plant would be decoupled from the other, and changes would be possible at a much faster pace than with today's interlinked systems. This is the only way to handle smaller lot sizes and automate other domains, such as housekeeping, car maintenance, and operations in remote and hazardous environments.

To achieve this, we need a system that can calculate and apply solutions to complex tasks where only the domain, the initial state, and the objectives are specified. To accommodate these requirements, we rely on model-based algorithms, which allow us to easily define different sub-domains with models, which can then be solved with arbitrary algorithms, which not only makes it easier to detect errors but also improves the theoretical scalability to larger problems compared to one or two-level planners for similar domains such as [26, 27]. Furthermore, our models are meaningful enough to describe themselves and our hierarchical planner.

This bootstrapping allows self-optimization and is only logical for approaches describing arbitrary autonomous systems since they are autonomous systems themselves. For model-free approaches, we have seen AutoML techniques incapable of describing themselves. In addition, we may not have enough data for the specific task because the simulation is not available to generate new data, and there is too little recorded data for rarely performed tasks. From this perspective, model-free approaches can only be part of the solution. Once we have collected enough data, we may switch to data-driven approaches for some sub-problems, thereby improving planning performance. This can be done seamlessly with heuristics that guide the planning process.

As soon as we deploy the algorithm to an actual system, it will naturally fail at some point due to unexpected events. The strengths of model-based algorithms come into play when we want to know the cause of the errors. We can determine which transitions should have worked but did not and which were allowed in the simulation but failed in the real world. We can combine different simulation levels for various sub-tasks to handle even difficult-to-model physical phenomena, such as contacts or fluids. The challenge with planning is the curse of dimensionality. Especially for multimodal systems, which have access to a variety of sensory options and multiple modes of interaction with the environment, the combination of discrete and continuous properties results in an exponential growth of computation time relative to the number of actions possible, the number of actuated or passive states, and thus the number of objects involved. Heuristics are not a generic solution either since they are very domain-specific. Therefore, we have developed a hierarchical planning algorithm that solves the problem on an abstract level and recursively decomposes it based on this solution. This plan on the coarse level can be seen as a kind of heuristic. However, it is very flexible and expressive due to the underlying planning to calculate it. In addition, this approach imitates the human way of solving such a task, allowing for explainability during debugging. Through our hierarchization and refinements, we can easily switch between the discrete and the continuous world, which are separated and handled with different algorithms in other approaches.

3.2. Planning Algorithm

Our planning algorithm is based on the formal models discussed in chapter 2. Because of their expressiveness, we can handle a variety of domains with different algorithms that operate on them. For this thesis, we solved problems related to task and motion planning, perceptual system configuration, root cause analysis in food production, and sensor data fusion. The hierarchical planning approach is divided into two different sub-algorithms. The main contribution is the hierarchical decomposition based on the set-theoretical foundation of chapter 2. The second contribution deals with the search algorithm and its data structures, which are optimized by the properties of our model. The basic idea behind all our algorithms is to reduce the number of states that must be visited to reach the given goal. There are two parameters to accomplish this. First, one can reduce the mantissa of the exponential function. For this purpose, we could introduce intermediate targets, which allow for reducing the number of states from n^l to $m * n^{l/m}$ in the ideal case. The other option is to reduce the base of the exponential function, which is determined by the number of combinations that the available actions can affect the currently available information. Our proposed method targets both options. It reduces the number of possible actions and the number of available instances by abstracting and factorizing while considering the current domain to minimize negative impacts on the solution.

3.2.1. Hierarchical Factorization Algorithm

The basic idea behind our planning algorithm is similar to that of hierarchical task networks (HTN) [28, 29, 30, 31], which operate on discrete state only. Extensions to these approaches already include geometric constraints [32, 33, 34, 35, 36]. Similar to this work, we first find a solution in a reduced domain and refine that plan until we consider enough detail to ensure its successful application in the real world. However, a significant difference between these well-known approaches is our strong type-system based on a set-theory for declarative knowledge. Based on this definition, we can automatically calculate refined derivations of the specified goals and flexibly factorize the complex overall task. Therefore, we have generic algorithms to flexibly factorize the task at hand and apply the planning algorithms recursively. Furthermore, our approach applies not only to discrete or specific continuous variables but also to a wide variety of heterogeneous data types and black-box operators that act on them. The task for our planner is specified by the domain containing the hierarchically ordered concepts and operators as described in chapter 2, a set of instances describing the state at the beginning, and desired goal instances. In this context, *domain* defines the limits of consideration as highlighted in remark 8.

Remark 8:

For us, a domain not only describes the setup for the planning task with all available information and actions. It also defines the horizon in various dimensions of consideration. Those aspects range from the level of detail over spatial boundaries to limits in the temporal dimension and its resolution. Especially the temporal and spacial boundaries are essential for operations that rely on the closed world assumption as they must assume knowledge about all *relevant* objects. The

domain also tells us the point beyond which we accept any more detailed effect as negligible noise that we treat online with backtracking and rescheduling once it is significant enough to affect our process.

As we can only consider a limited number of objects, the spatial dimensions of the domain, in which all relevant properties and objects are described, should be as small as possible to cover the current process. Additionally, some techniques require a closed world assumption, such as the collision checking of example 4, and therefore defined boundaries.

Example 4:

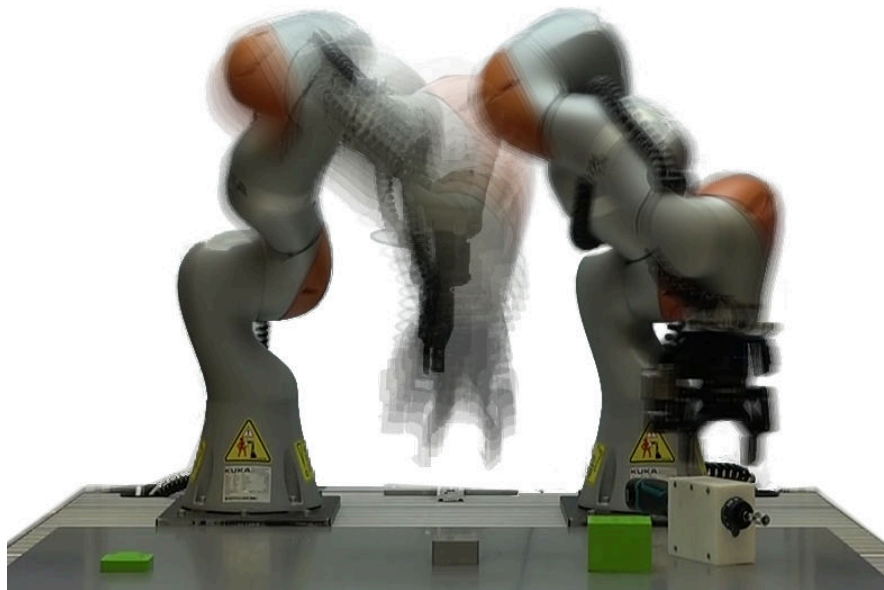


Figure 3.1.: The left robot approaches the box to grasp it. In parallel, the other robot grasps the screwdriver. During execution, both entities are handled in the same spatial proximity and can influence each other, while other robots in the factory are not considered.

In the case of a robotic cell, for example, a rigidly mounted dual arm setup with grippers and cameras for each manipulator as depicted in Figure 3.1 above, the temporal horizon starts at the current point in time and ends immediately after reaching the target.

For a less dynamic robot or a process with long time constants, even the time after reaching the target for the first time can be considered until the system reaches a standstill. The spatial boundaries are, in this example, necessary for collision avoidance. This domain of a robotic cell can be a sub-domain of a larger planning task, which conducts factory-wide scheduling and planning. This task, by itself, can be a sub-task of optimization across multiple factories and related logistics. A well-known example of such a setup is the manufacturing of cars. Production must be scheduled between various sites in various countries with specific demands, production capacity, transportation capabilities, and import tariffs. Each factory has multiple production lines for which maintenance, working shifts, and capabilities must be considered. Each line comprises several robotic cells that calculate purposeful actions, considering geometric and dynamic constraints.

As the domain is a declarative property, we can express it as an instance of a concept and calculate the partial ordering between several of them. As in the example 4, there may be more general domains that cover larger spaces and allow factory-wide planning or more detailed domains that only cover excerpts of the plant. A domain that considers only a single assembly cell is a sub-domain of our entire facility, which makes it a subset of the concept that describes the whole plant so that we can apply the *more-general-than* relationship from section 2.1. Theoretically, there is no obstacle to refining an area further and further and eventually reaching the quantum mechanical level, which is the most precise physical model we currently have for our world. However, if we look at the example 4 with the operation of a robot arm, there is a level of detail at which we can reasonably stop modeling further details. We could model the robot actuation as a single operator that performs point-to-point motion between different configurations and assume that there is linear interpolation with constant velocity between these configurations. A more refined version could also consider velocity, acceleration, or even inertial and Coriolis forces, motor currents, and gear ratios. From this example, we can see that at a certain point, we can simply assume that the underlying controller has a negligible deviation from the specified goal, which makes further modeled details unnecessary. This limit is domain-specific and thus defined by our domain model.

After the production in example 4, the goods must be delivered to the customer. This logistics use case of example 5 is a tangible example of the human-like approach of our hierarchical planner, which covers both discrete and continuous properties.

Example 5:

In our hierarchical logistics domain, we have operators on the most abstract level that can change the semantic location of the crate, which is encoded by the country and postal code of the current position. Different operators implement several modes of logistics, such as transport by ship, train, truck, plane, and van. Each operator calculates the outcomes and if the action is applicable given the provided inputs. The delivery van can only move within neighboring postal codes, while ships and airplanes can only travel to very few selected locations with the required infrastructure. However, the crate changes its positions similarly and only on a discrete level for all these different operators until we refine the process further.

Loading the truck and the van is quite similar on the middle level. We can do it by hand, using a forklift, or completing it with an advanced mobile robot. Only on the more detailed continuous level do we notice that the forklift is too tall to fit into the van. When we consider more detailed continuous and timing information, we might notice differences in the grasp and drop positions for the autonomous guided vehicle (AGV) that result in varying execution times because of differing trajectories for the arm and platform. As we calculate them, we might notice that those paths are significantly faster or slower than expected by the more abstract levels; thus, the truck might be loaded by the AGV while the van is packed manually.

At the most abstract level, our logistics domain covers long-distance transport. With several planners integrated into our hierarchical planning approach, we can calculate a feasible, quick, or cheap solution to this problem. If specific criteria are met, such as no continuous properties in the

current subproblem, and an introspective implementation of all operators in a specific formulation is satisfied, we can transform the task into a PDDL [8] domain and problem. We can then apply a planner from the rich set of well-tested and optimized solvers for such domains. Similarly, we have connected the hierarchical CHIMP planner [37], which is very efficient for some hierarchical forms of discrete planning. If the conversion is impossible or the domain does not benefit from one of these planners, we use our algorithm, described in subsection 3.2.2. This planner is optimized for models based on set theory and applicable to generic domains, making it less efficient than tailored algorithms for specific problems. Each sub-task is defined by an operator that we currently refine. The solution's successful application to the more abstract domain gives us hints, like heuristics, on which information and actions we should concentrate during our current search to reach some intermediate goals. The model of this operator holds information about which planner and parametrization of the planning algorithm are optimal for the current sub-domain. The domain contains all these properties for the most abstract layer, such as activated scheduling, which planning algorithm to use, and which operators to apply. In either case, the result of these algorithms is a plan that consists of a sequence of actions that operate on defined instances and return new instances. This plan is the basis for our factorization algorithm. Let us assume that the high-level plan of our example 5 is to transport the crate by truck and then by van. We can now factorize the overall task into two sub-problems, defined by the applied operators in the plan, their inputs, and outputs. These tasks can only be solved successively as we operate on the detailed outcomes of previous steps that are at least as refined as our current sub-task. For example, we need the expected location and time of arrival of the crate for the subsequent continuous actions. We cannot operate on the same inputs the more abstract operator used. Nevertheless, the two steps are combinatorically independent according to our set-based model, which assumes that the same abstract behavior for any action that operates on detailed instances can be expected if they are similar to the intermediate goals in the sense of our instance definition Figure 2.4. Thus, an example coarse plan for our example 5 can tell us that the crate may be moved to the goal from any detailed position that complies with the more abstract location description. At the same time, the arrival time does not influence subsequent steps as long as they are within the defined region or set of acceptance defined by the more abstract instance.

Each applied operator in the abstract plan defines the properties for the planning task in the subdomain. The input and output instances are used to calculate the available facts and goals for the sub-task, while the operator describes the configuration of the planning algorithm. For the first step of our example, we already decided which crate to use. There might be several of them on the shelf, and without the abstract layer, we would try to apply the following steps to each of them. We can limit the degree of combinatorics because we hold on to decisions made at the more abstract level. The algorithm that calculates the new planning domain based on the current sub-task is explained in detail in chapter 7. The rough idea of this algorithm is to provide only a subset of all available instances for the refined planning domain. We blacklist all instances that belong to any concept that is input to the operator that defines the current subdomain, except for instances that are similar to those actually used in the abstract layer. The goal of the sub-task is defined by the

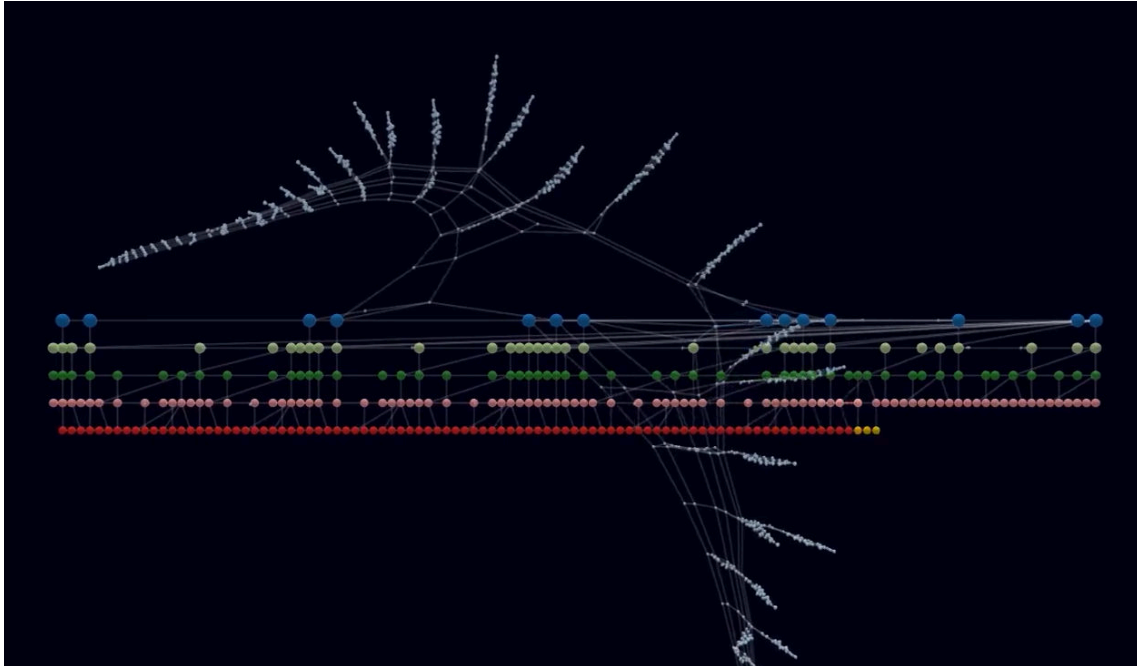


Figure 3.2.: Depiction of the hierarchical planning scheme. The spheres visualize different planning states, and the edges depict temporal relations for all non-vertical edges. The vertical edges are refinements to that state. The differently colored horizontal chains are successful plans on that level of abstraction. The topmost chain is the most abstract solution, successively refined by the rows below it. The white spheres are visited states that did not contribute to the solution, and the yellow spheres are lately added states.

outputs of the operator at the coarser level. With this approach, we define a new sub-task and can proceed wholly recursively as depicted in Figure 3.2. The recursion stops when no operator with more detailed sub-operators is applicable in the current plan. This is the case for our domains as soon as we apply the real-world execution operators, which move the robots, transport the goods, and manipulate the objects in the real world, ultimately solving the task. For simpler domains, the limit of detail can be at a simulation level, for example, because the factory layout optimization just provides a plan a human later refines and executes.

Since the refinements are defined purely by the domain’s (semi-)automatic hierarchization, we can easily combine models from different sources into a single domain. The hierarchical order results automatically from the information provided by the models. We have not defined strict layers in the sense that we could enumerate the number of levels and assign operators and concepts to those abstractions. The number of recursive refinements is determined only by the operators used during planning and their available refinements as depicted in Figure 3.3. During our planning process, we re-evaluate the operators with the new, possibly more detailed instances so that the current data is forwarded. In our example 5, we consider the calculation of approach positions for grasping based on the position of the objects. Therefore, this algorithm is relatively simple and has already been evaluated at the most abstract level in a rather coarse simulation. However, the object position input to this algorithm will change with more refined simulations or real-world

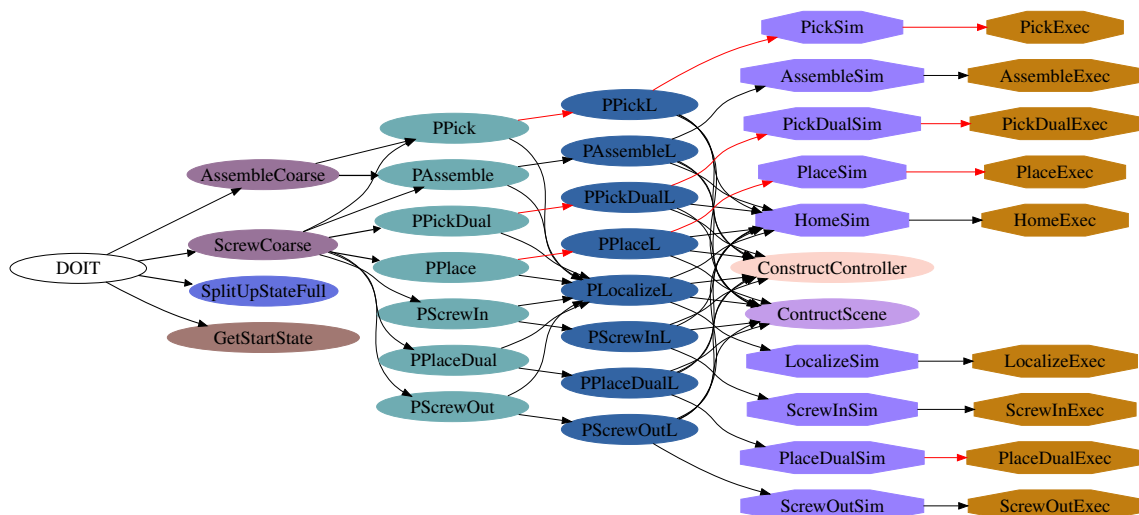


Figure 3.3.: Operator hierarchy within a domain. *DOIT* is the placeholder operator to configure the properties of the most abstract planning task. Successors of operators are available in the subdomain that refines this operator. Leaf nodes are operators used in the most refined domain, which executes the task in the real world. Operators can be used in multiple subdomains. The number of refinements is not the same for all operators in a domain such that no strictly hierarchical levels emerge. Those levels only emerge during planning based on the applied operations. The coloring is purely user-defined.

executions. Therefore, we must re-evaluate the operator on the updated instances to synchronize the output information as depicted in Figure 3.4.

When we abstracted our operators, we left out details, which affected not only our model’s declarative inputs and outputs but especially the procedural calculation. This can, therefore, lead to different conclusions or solutions. As mentioned above, we need to ensure that the abstracted operators provide a mapping between inputs and outputs that is a superset of the mapping in the refined layers. If we fail to honor this characteristic, the quality of our solution will deteriorate. Despite conducting a comprehensive search, we may not be able to find existing solutions or only provide a more expensive plan than necessary. The problem is similar to the construction of good heuristics. If they are imprecise, their usefulness is reduced. On the other hand, a strong heuristic can only handle a limited and smaller number of domains and is more challenging to construct. The abstract levels with their intermediate targets can be considered heuristics in our hierarchical approach. However, they are calculated with our planner, which is domain-independent and only influenced by the model. The domain-specific part is modular and thus as reusable as possible, easy to formulate, and so expressive that the *heuristic* may already include the evaluation of simulations. The approach, where the abstraction level is considered a heuristic, also suggests that these levels may be wrong in their estimation and thus may even fail and lead to dead ends. We can detect when an intermediate goal cannot be reached and look for another way. In other words, we update our heuristics during planning based on the feedback from more detailed levels, including the real-world execution. This *backtracking* is done by blacklisting the dead end in the abstract layer and restarting the planning algorithm on this layer afterward as depicted in Figure 3.5. This way, we find an alternative solution to the target, which may circumvent the problems not yet

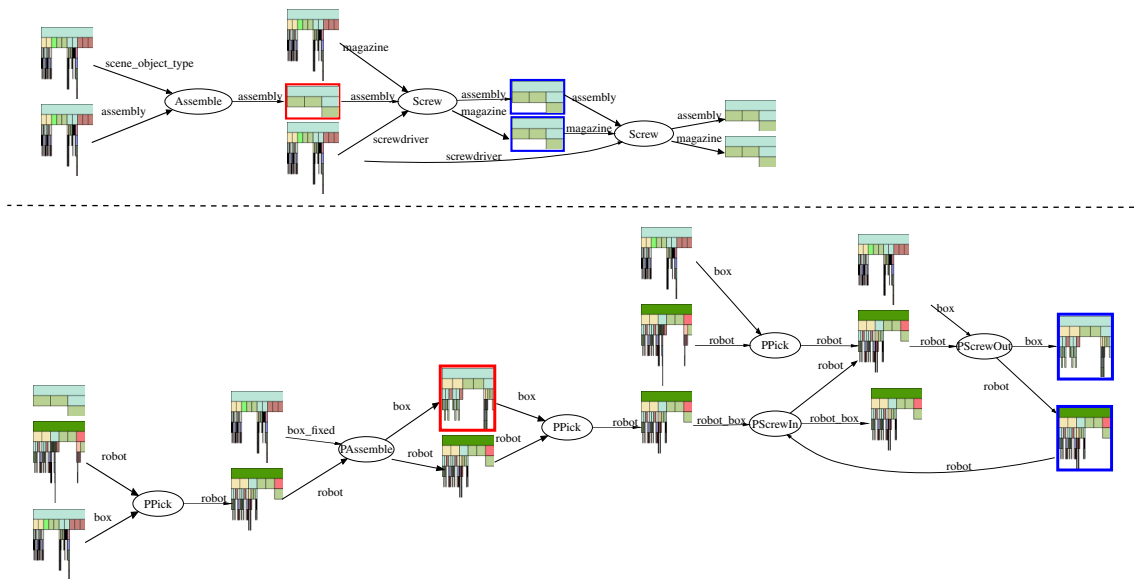


Figure 3.4.: The upper, more abstract level defines the intermediate goals for the more refined level, which is depicted by matching colors. The red pair is the goal and the instance fulfilling this goal for the first step. The second step has two goals and respective goal instances. There, the refined level acts on the goal instance with additional detail instead of the instances of the coarse level and, therefore, forwards the detail calculated in that level. Further discussion can be found in chapter 7.

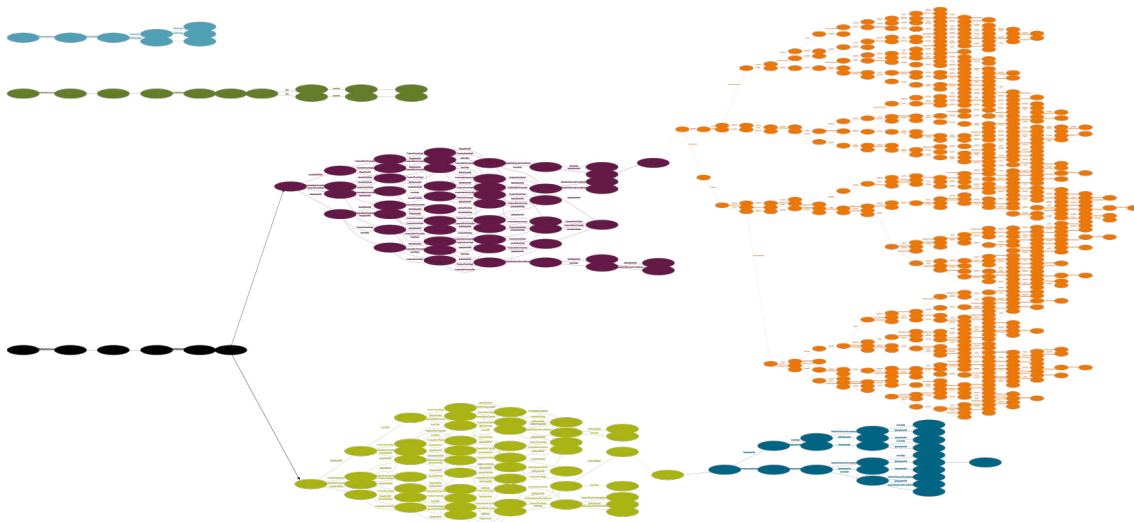


Figure 3.5.: This illustration of [2] distinguishes different hierarchical levels by their color. No plan was found in the orange subdomain (hence the large number of planning states visited). Therefore, a new rough plan was searched in the black domain, which led to the new refinements of the green and dark blue nodes.

considered at this abstraction level. In order to learn from the problems solved by this reaction in the long run, we need to extract knowledge from these situations, which can then be used in an updated model as discussed in chapter 4. During our backtracking, we treat stateful operators like real-world execution, with the same mechanism as purely simulated and stateless operators. Since stateful operators cannot simply be reset and resumed from another state, we must continue planning from the last state generated by such an irreversible operator. This approach obtains a behavior similar to MPC algorithms for a real-world operator. Suppose the perceived operation yields results that deviate from the expected outcome of our planning, for example, because the content of a drawer was not observable before, and positions and objects were only guessed. In that case, we backtrack and continue planning from the current state. We can solve problems like [38] once we model the expected outcomes more abstractly. When backtracking an irreversible operator, we not only limit the exploration of the current level to the state that resulted from the application of this operator but also derive all temporally following planning states on the more abstract levels from this state. Therefore, the backtracking could result in an unsolvable planning state, as the more abstract levels may not process the intermediate products of the more detailed levels, which can be avoided with suitable, coordinated modeling of the levels, as we create them automatically in section 4.2. When calculating new planning states, whether based on backtracking or the regular planning progress, the changes of the instances during planning, the automatic abstractions that take place, and the relationships between the instances and the concepts to which they belong become relevant. The initial state is the reference for the abstract level with which we start our hierarchical planning and the basis for the finest level we see during planning. Thus, the instances that live in this state belong to concepts that are detailed enough for the operators of execution. The abstract operators do not need this level of detail and, therefore, neglect most of the information of these instances. Nevertheless, our declarative knowledge also ensures that they can extract the relevant information from the more detailed instances. However, the output will lose all details because the abstract operator cannot make any statement about this specific information as depicted in Figure 3.6.

Therefore, the specified goal for our hierarchical planning must be an instance of an abstract concept so that it can be calculated and achieved by the highest-level operators, which means for our example 5 that the goal only tells that there must be a box in this house. Neither the geometric location, accuracy to the centimeter, nor the exact arrival time will be specified for the overall task as they are irrelevant to begin with. Alternatively, even the first level would have to consider this additional information. It would have to achieve it precisely, eliminating every chance of a hierarchization and factorization of the task. As the planning with refined levels progresses, the previous, more abstract levels define the objectives. As a result, the goals also become more and more detailed. The operators also take additional information into account and calculate more details for subsequent processes during planning, which means that less information from the instances of the initial state is ignored. We use two orthogonal measures to ensure that our planning fulfills the task in the real world as desired. First, we define the requirements for the target state with a set of target instances. The final state must be a subset of the target these instances describe,

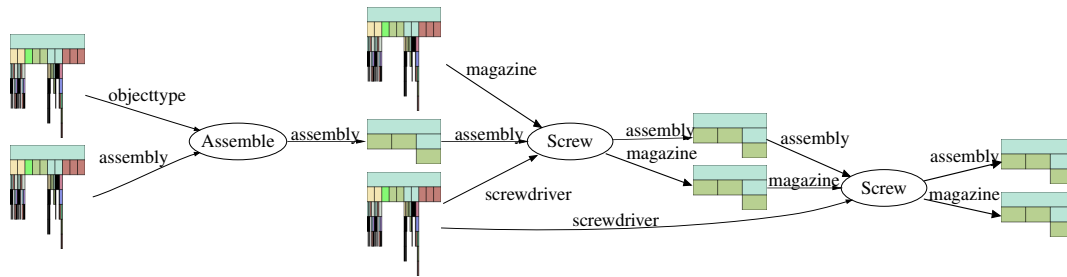


Figure 3.6.: Each piece of information of an instance is depicted by a fragment with a color specific for its type in the rectangles. The more fractured instances hold more detail, which is reduced in this abstract domain as soon as the oval operators act on them.

which is to ensure that we do not stop planning on a simulated level but execute it in the real world. We refine each operator on the detail dimension orthogonal to the target specification with the most detailed operations available in the domain.

3.2.2. Planning Algorithm

In the previous section, we discussed factorization based on hierarchical abstractions and set-based models. To solve this approach's emerging sub-problems, we can apply existing and dedicated search or planning algorithms, which we select based on domain properties such as PDDL, HTN-like, or task and motion planners. These algorithms perform well in some domains but have limitations that prevent a general application. For example, the task and motion planning algorithm can handle continuous problems well but struggles with many discrete states fragmenting the state space. A motion planner cannot deal with discrete states, while PDDL planners are weaker on continuous dimensions and usually only accept time to be a continuous variable. A further difficulty with cascaded planning algorithms, such as our hierarchical factorization approach, is the detection of dead ends and early terminations for the integrated planner. Therefore, we have developed an algorithm that takes advantage of our set-based model to cover general domains and acts as a fallback when other algorithms cannot leverage their strengths. It is a forward-state space planning approach, which can be configured for breadth-first or depth-first exploration. A detailed discussion of this planning approach, the integration with hierarchical factorization, and the data structures used can be found in [2].

Planning Process

In our planning approach, we iterate over all open states that have not been identified as dead ends. We apply all available operators on those states, with all combinations of available instances that meet the minimum input requirements specified by the modeled concepts on the operators' inputs. At this point, the model is also used to determine the instances relevant to the special inputs. These inputs include, as briefly mentioned in chapter 2, all instances available of a given type used for non-constructive operations such as path planning or negative inputs that mark instances that are prohibited for active use because the more abstract layers have not considered them, but must be

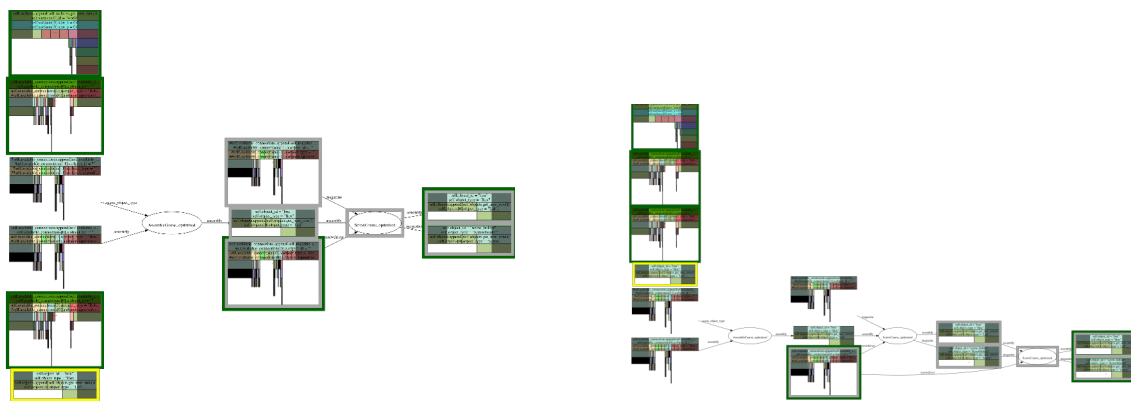


Figure 3.7.: The right planning state is a successor of the left planning state. The instances with green frames are available instances; the yellow framed instance is the goal, and the nodes marked in gray are the operator and instances touched in the last step that calculated the current state. We can see that the newly applied operator consumed instances that are no longer available to calculate the new instances.

at least considered passively in the refined layer. The resulting state is calculated for each of the successfully applied operators. In this state, the available facts are updated with respect to the modeled properties such as "consumed", and the operator's output is merged with the previously available facts as depicted in Figure 3.7. With this basic algorithm, the planning state graph, which contains the progress of this state space planning, would look like a tree, similar to other state space planning algorithms like RRT or EST. However, with a treelike structure, we cannot tell whether the new state has already been examined so that we could end up in loops. Motion planning algorithms have special strategies that are not applicable to our generalized models to explore unseen space and prevent those loops. They retain this tree structure but calculate distance measures to either take samples from the unexplored space and connect them to the nearest already visited nodes (RRT) or to take starting nodes that live in poorly explored regions and start the forward step from there (EST). For us, none of these approaches solves the problem of identifying and preventing dead ends since the calculation of distance measures is made difficult or even impossible by our heterogeneous state, which is composed of the accumulation of all existing instances at some point in time as shown in Figure 3.7.

Compact Planning Domains

We, therefore, use our set-based model to detect similar states as described in remark 9. If we encounter a new state already in our planning state, we will not explore it again. With this loop detection, we can also identify dead ends, at least for domains with a limited number of possible states. We can also achieve this by discretizing the value ranges for continuous properties. This approach is similar to the buckets used by probabilistic planners to ensure good search space coverage. The instances in a bucket are assumed to behave similarly for the next steps. In our framework, we would model these buckets as concepts. These concepts describe the similarity of their instances. When we plan on an abstract domain, in which no operator uses the level of

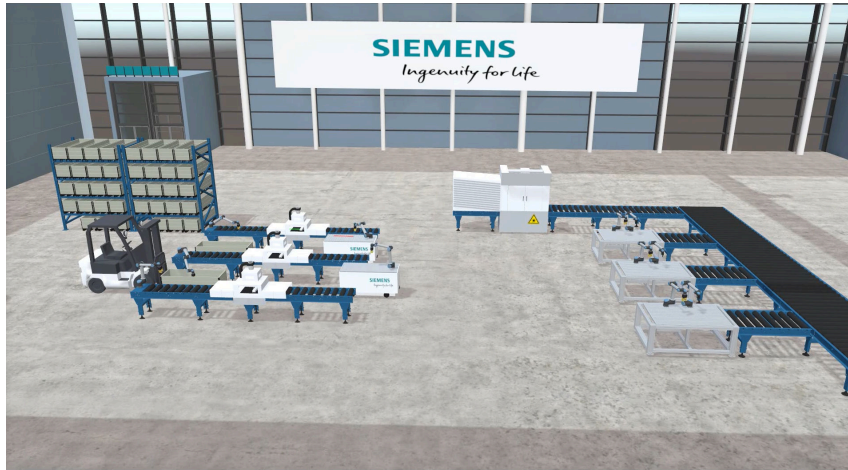


Figure 3.8.: Factories with multiple logistics and assembly robots require coordination of discrete and continuous actions, which are partly independent and thus factorizable. Therefore, this is a good field of application for our hierarchical planning approach.

detail of the instances provided, which is commonly the case for the coarse levels of planning, we automatically abstract the instances to the specific level of the current planning task. For our example 5, when we store boxes in the truck, all information about their contents and identity will be ignored, and they are only distinguished by their external measures.

An interesting problem that arises with the compactness of our planning state, as stated in remark 9, are instances that are identical on the current level but will be separate instances on a refined level. Take, for example, screws that have a position and are therefore identifiable on the refined level but group together on the more abstract level to form indistinguishable instances. This grouping happens as soon as we only consider properties that the operators of the abstract level take into account, such as size and material, and abstract all other properties away, which means that these instances are viewed through the glasses of a more abstract concept, making them identical to the tolerances defined in it. As mentioned above, our primary goal is to reduce the branching factor. If we have several similar instances, we expect operators to behave the same for each of them during the planning process. Otherwise, they would already contain additional information at this level, which the operators then explicitly use to adapt their behavior. Therefore, when we consider them individually, we expect redundant paths in our planning graph for each instance. So, for n similar instances and m uses in a successful plan, we get $\frac{n!}{m!}$ redundant paths. Because of our modeling approach, we can avoid this unnecessary combinatorial complexity by combining them into a single instance with a counter, which significantly reduces the associated effects at the abstract planning levels, especially for logistics and assembly scenarios with many similar instances as shown in Figure 3.8, such as screws, AGVs, boxes, raw materials, or modules. With this automatic abstraction, we extend the power of our hierarchy to yet another dimension, in addition to abstracted and cheaper operators and fewer steps to the goal.

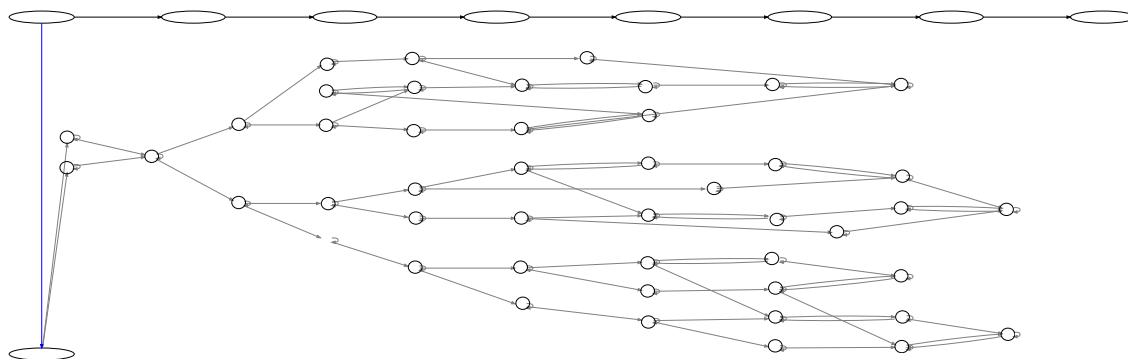


Figure 3.9.: This excerpt of the hierarchical planning state shows the abstract plan in the top row. Our planner is currently redefining the first step of this plan. However, the refined layer has no solution to this planning task. Therefore, we can observe the characteristic structure with multiple diamonds, each with its dead end.

Remark 9:

We modeled the planning process in a meta-domain. There, the planning state is a concept on which meta-operators act, which apply operators of the given planning domain. Therefore, the planning states during planning are instances of our meta-concept, for which we can calculate similarity as described in section 2.1. We assume that the succeeding planning states are the same in the given domain for similar concepts. Therefore, we can define a *compact planning domain* in which a single node represents similar states in the planning state graph. This reduces the number of planning states, improves the computational complexity of the planning process, and eases debugging and introspection. To calculate the similarity of two states, we consider the domain and the similarity, number, and creation times of the instances in this state. The available instances are automatically abstracted to the most detailed level the current operators work on, allowing for effective state compression in a coarse domain. That results in more similar instances, resulting in more similar states and reducing combinatorial complexity on abstract levels.

In compact planning domains, the planning state is no longer a tree but a graph, with possibly several parallel paths leading to a single state. Characteristic diamond-like structures, as depicted in Figure 3.9, emerge if dead ends arise, which can be investigated and described with category theory [17, 39]. Each diamond is generated by an operator that has consuming inputs. The succeeding part is formed by states that emerged from operators without consuming inputs. The state at the diamond's tip has executed all operator input combinations, representing the maximum number of possible facts available for a node that is an ancestor to the state generating this diamond. The difference between the possibly nested diamonds is the missing instances determined by the operators with consuming inputs.

Termination of Planning

We abort the search depending on the properties of the current domain. For "sufficient planning", we abort at the first state that meets all goals. We can have several goals that all have to be fulfilled as the operator defining the current task can have several outputs, or it has been specified that, for example, several packages must be delivered or the assembly must be completed, which is described by multiple goal instances. For each of these outputs, we need a separate instance that fulfills it and is therefore similar to this goal instance as described in section 2.1. Another reason for termination is the lack of open planning nodes, which can be further explored, meaning we are at a dead end and must backtrack and investigate another coarse plan. This cannot be detected with ordinary forward planners such as EST [40] since they would only sample the given space in more detail.

If we are working in a domain that requires an optimal solution, we continue planning until there are no more open planning nodes for expansion. In this case, we use the cost of an already found solution as an upper bound to reduce the unnecessarily explored space.

The cost of a state can be calculated in two ways, which in turn is defined by the properties of the current domain as discussed in remark 10.

Remark 10:

The cost of a planning state is equivalent to the **expected** execution cost (mostly time) on the most detailed layer. Therefore, each operator holds the expected execution cost of the most detailed planning task defined in his KPIs (key performance indicators). For a domain with "serial execution cost", we determine the cost of planning with the sum of all operators applied in the current state.

For domains with "parallel execution cost", they do not add up for parallel strings of operators. For those domains, we switch from an operator-centric approach to a calculation that tracks the expected cost incurred for creating each available instance. The most expensive instance in the current state then determines the total cost of that state.

The fingerprint of a planning state depends not only on the values of the instances but also on their expected creation times. Therefore, planning states with the same instances are still not fused to a single node in our compact planning domain because of different timestamps as described in remark 9. With this advanced calculation scheme, depicted in Figure 4.11, we can optimize our planning for a later parallelized execution.

Plan State Planning

Another view on our planning approach, which would allow for future extensions, is that of plan-space planning instead of state-space planning. In this case, we do not revise decisions on an abstract level and backtrack if a dead end for a subproblem is identified. Instead, for each planning step, we decide when to generate a new subproblem, for which step we generate this new subproblem, in which state we continue working, and which operator-input combination we simulate, which allows a heuristically controlled and probably domain-dependent search across all hierar-

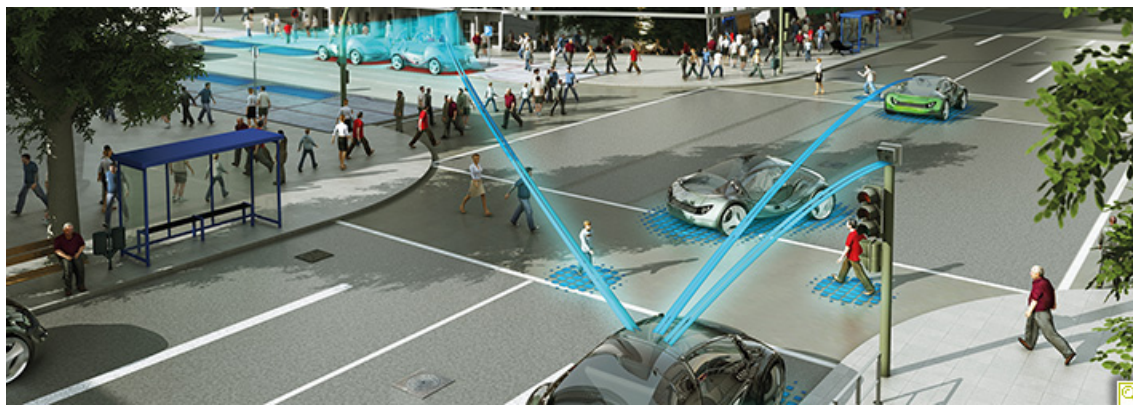


Figure 3.10.: Sada is an EU-funded project for which we applied the ideas of the set-based approach in the field of perception [3].

chical levels. The heuristics are then based on data from previous runs. We already generated distance functions for our concepts, which can be used to select the state that comes closest to the given goal. With the approach described in the previous sections, motion planning in labyrinths is problematic because it is tricky to detect dead-ends. With the heuristic approach, dead-end detection can be done before we run out of open planning states. However, the data structures, algorithms, and troubleshooting become more complex and challenging for a plan-space planning approach. From our point of view, the previously described hierarchical planning is a particular form of plan-space planning, where defined rules, which operations are performed on the current planning state, allow for the explanation and debugging functions discussed in the next section, which are powerful tools for setting up a new domain.

3.3. Domains, Explainability and Introspection

Today, building an autonomous system is a huge engineering task, which is frequently more time-consuming than designing multiple, more specialized, and less advanced systems that solve the same task. This dissertation's modeling and planning framework aims to reduce and eliminate the engineering time required to set up new autonomous systems. With reusable models and advanced planning algorithms, we exploit the potential of automation and push the limits of applicability.

Planning Perception Pipelines

The classical architecture of autonomous systems is based on perceive->think->act loops. We targeted each point of this structure and algorithmically solved engineering problems in each step. We started by automating environmental awareness, which is fundamental for every reactive system. Especially for multimodal systems, integrating various detection algorithms and data fusion is a cumbersome engineering process. Several publications that emerged from this work target this issue and automate the design of perception pipelines and sensor-data fusion with the planning algorithms proposed in this work [1, 41].

The use-case for this approach was initially set by the EU-funded project called *SADA* and later

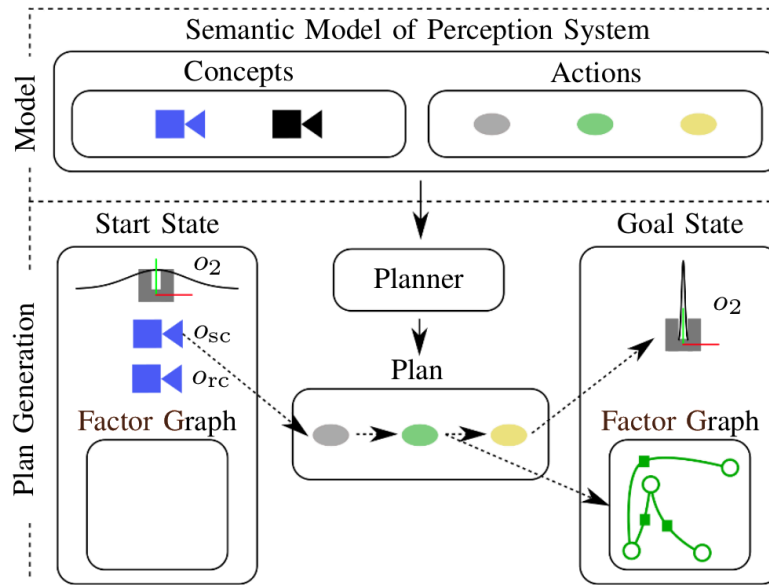


Figure 3.11.: This architecture of [1] allows our planner to handle domains with uncertainty to solve cell and perception configuration problems. Our model holds concepts about sensors and operators that implement perception and fusion algorithms. The goal state is represented by a maximal allowed variance. Our planner finds a sequence of data acquisition, extraction, and fusion algorithms to reach this goal.

on by the industrial assembly scenarios. SADA aims to create an intelligent infrastructure enabling autonomous vehicles to navigate urban areas. The system provides aggregated and relevant information from third-party, multimodal sensors that reduce the blind spots of cars as they are not necessarily mounted on the vehicle Figure 3.10. Examples include roadside radar or lidar units, sensors of other vehicles, or pedestrians' smartphones. Decisions about the information that should be incorporated into the sensor fusion must be made based on the relevance of the data and limiting factors such as bandwidth and computing power. With our modeling approach and a simplified, non-hierarchical version of the planner described in chapter 3, we could identify the relevant data and solve this problem.

With the advanced approach proposed in chapter 11, which uses a factor graph to take uncertainty into account, we can handle a similar multimodal domain for robot cell and perception configuration pipelines in industrial environments. The implementation is based on the model described in chapter 2, which holds the information for the state estimation and its probabilistic geometric uncertainty. Our current belief state is defined by all currently available information at a given state, which is then compiled into a factor graph that provides the probabilistic representation of the geometric uncertainties as depicted in Figure 3.11. The actions the robot must conduct during manipulation and assembly define the minimum requirements the uncertainty must fulfill. Our planner searches for a perception result, represented by the variance of the geometric state in our factor graph, that meets preconditions encoded in concepts. The operators are modeled based on the scheme presented in chapter 2 so that they can be used by a single-level planner similar to the one described in subsection 3.2.2. They implement processes for capturing new sensor inputs

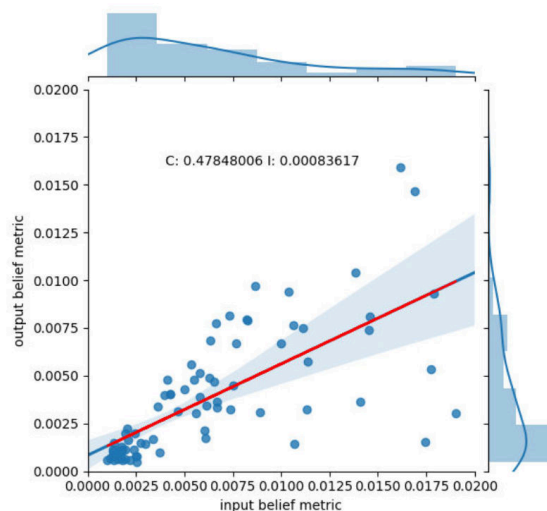


Figure 3.12.: This calibration, presented in [4], calibrates abstraction layers for a given task at hand such that our planner can rely on efficient and meaningful factorizations.

using active perception measures, statically mounted cameras, data processing that combines and converts between different measurements, and inference operators.

Processing, planning, and optimizing visual data is complex and requires a lot of computing power. A hierarchization of those domains would reduce this problem. However, as stated in [42], finding meaningful abstractions in perception is complex and challenging. Therefore, our first hierarchical approach of chapter 12 uses a specialized refinement algorithm to circumvent problems with hierarchical planning approaches that rely on declarative knowledge to identify the intermediate goals. The procedural and declarative knowledge is still based on the model presented in chapter 2. However, the search space is reduced by defining a template that abstractly describes all possible solutions of the planning phase. As stated in subsection 3.2.1, our approach makes using specialized planners for specific domains easy. In this case, we use this to apply an approach like plan-space-planning [43, 44]. Since the perception operators rely heavily on parameters, the refinement phase includes an optimization based on the structure found by the planning algorithm, which allows us to configure and optimize perception pipelines automatically.

The artificial restriction of the structure to the manually defined templates is relaxed by our approach presented in chapter 13. Again, this algorithm is based on the set-based models but uses the hierarchical planner described in section 3.2, which allows the solutions to react more flexibly to the given tasks and to cover a larger search space with reasonable planning times. For our hierarchical planning approach, however, we need suitable declarative abstractions. After circumventing this problem in the previous paper using templates, we now present an automatic calibration of the abstract layers to the refinements as depicted in Figure 3.12. By doing this, our planner can operate on the abstractions efficiently, and less manual work is required to model them. The solutions found with this approach showed a promising performance, even compared to an exhaustive search in the entire configuration space.

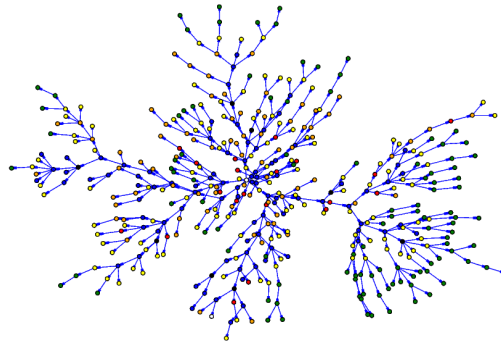


Figure 3.13.: Even for a task with only two discrete steps, state-of-the-art planners tend to visit thousands of nodes, which prevents debugging and results in poor scalability.

The industrial assembly domain

For the *think*-part of perceive->think->act loop, we applied the hierarchical planning and set-based modeling approach of this thesis to industrial assembly problems. Processes in this domain are highly hybrid, with numerous discrete properties modified by successive actions and complex continuous problems such as collision avoidance with arbitrary geometries or motion planning with different kinematics and reachabilities. Solutions in this area must master the combinatorics of discrete properties, even for several successive assembly and handling steps, while integrating geometric and continuous properties and their computationally intensive simulation. State-of-the-art task and motion planners fail even with a few involved objects or steps necessary to fulfill the task. In contrast, discrete planners are only loosely coupled with sophisticated simulators and real-world execution. Recent approaches such as [45, 46, 47] try to combine both worlds but restrict themselves to exactly two layers or planning phases. Others implement problem-specific solutions with more but still a fixed number of abstractions, such as [48, 49]. This domain is the perfect example for discussing our hierarchical planning and modeling approach, as we can learn from the way humans approach this task. We can even recognize the solution's hierarchical structuring in different user roles on the shop floor, highlighting the need for more than one level of abstraction. This industrial assembly domain is the primary example in our fundamental publications, cf. chapter 6 and chapter 7, as well as in papers that further extend this system, cf. chapter 8 and chapter 9.

Explainability

A valuable feature of the hierarchical approach is the easy traceability and isolation of problems. Unlike other types of planners, which easily have thousands of states that are structurally only loosely coupled, our primary goal is to limit the branching factor, the plan lengths, and thus the number of states as depicted in Figure 3.13. For this reason, debugging is feasible even for complex domains as we can visualize what the algorithm currently *thinks*. As discussed above, our hierarchical approach can be seen as a dynamic, modeled heuristic. With this perspective, we can

further direct attention to situations in which we expect to succeed and fix the model or implementation accordingly. This works for both the planning algorithms and when a person is investigating errors, bringing in intuition, or interacting for cooperation as depicted in Figure 3.14.

Human Robot Interaction

In [7], we discuss how the human workforce can be incorporated into our planning and execution process using expressive communication similar to tailored solutions such as [50, 51]. The only requirement is to formalize actions necessary for all human-robot cooperation [52]. Both means of execution have their respective strength and cost, reflected in their KPIs. In [53], a scheme for task assignment to a collaborative cell is discussed, which could be integrated into our planning algorithm using the KPIs of the respective operators. As discussed in [54], communication between the partners of a collaborative team is vital. Due to our expressive and intuitive representation based on the various levels of abstraction, we can facilitate this communication between human and machine as depicted in fig. 3.14. In this mode, the system not only executes parts of the plan by the robotic system, with actuators and tools that bring in different modes of interaction, such as screwing, grasping, or suction gripping, but also the human, which is directed to fulfill the overall planning task. In this sense, we have a multimodal system regarding execution abilities.

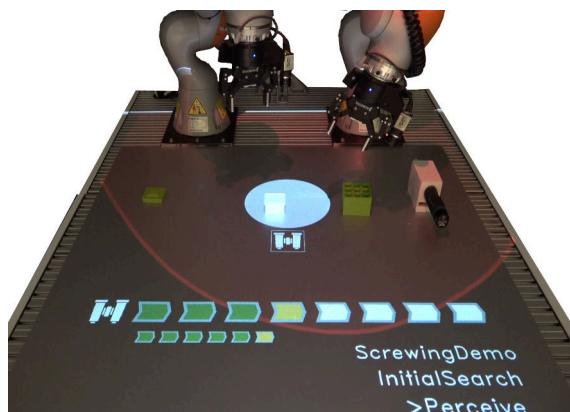


Figure 3.14.: With abstracted information, a more compact representation for interaction between humans and machines is available, which not only helps during debugging but also for cooperative or collaborative tasks. We can present the current progress, task, and relevant objects to a user, who is then prepared for the next actions of the autonomous system.

Root-Cause Analysis

Since the expected behavior is modeled, any backtracking is an anomaly for which we can store the planning status for retrospective analysis, debugging, and model improvements. In addition, we can detail which sub-task failed and thus narrow down the error region to a few planning states and applied or failed operators as depicted in Figure 3.15. In some cases, such as manipulation



Figure 3.15.: With our hierarchical approach, we can localize potential errors easily. For each required backtracking, we dump the planning state. We can then analyze the sub-task that was not refinable (top picture). In this sub-task, only the leaf nodes of the diamonds are of interest (bottom left picture). We can check which operations have already been applied, check the desired goals, and dump available instances to call operators that we expect to work but failed (bottom right picture). This process is supported by a graphical tool, which eases navigation, introspection, and debugging.

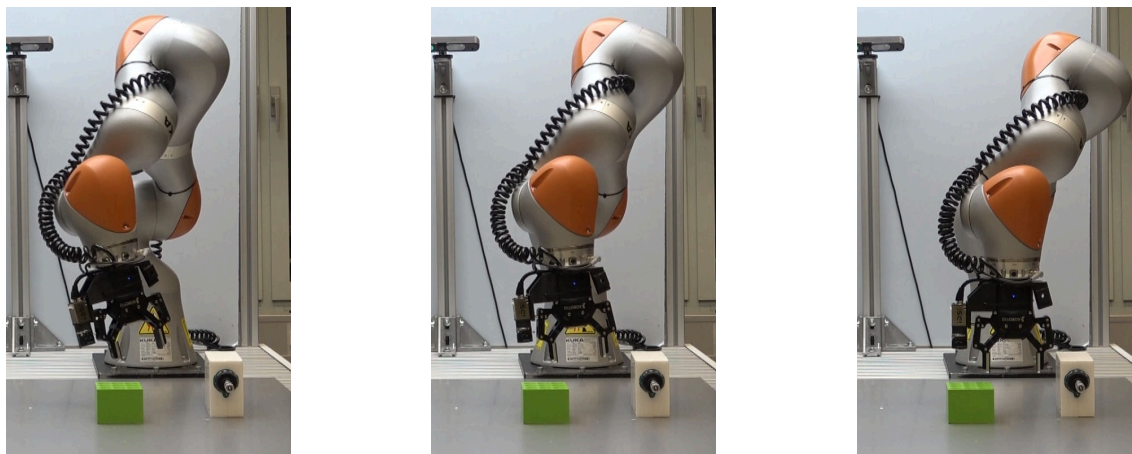


Figure 3.16.: Our planning algorithm provides introspection capabilities to localize errors. To further investigate problems, we can replace hard-to-debug solvers, such as sampling-based planners, with algorithms that provide feedback even in case of failures. In this sequence of pictures, we can see an error case for a constraint-based motion generation. The robot tries to grasp the screwdriver but fails due to a poorly chosen start configuration. With this feedback, an engineer can adapt the planning task so that solutions exist.

planning, where the chosen grasp transformation can affect the success of the subsequent assembly, several sub-tasks may lie between the failed action and the decision that led to this failure. However, even in these cases, we can quickly identify the root cause by stepping through the different planning states while backtracking and analyzing the failed actions. Once we identified the sub-planning task that did not find a solution, we need to find the operator that was expected to generate this solution but failed. Some operators are more difficult to debug because they contain a complex algorithm that only provides information for successful executions. Examples are motion generators based on probabilistic sampling, such as [55, 56, 57]. In case of an error, it is desirable to replace them with alternative approaches, which perform manipulations that are as goal-oriented as possible and from which the source of the error can then be derived, even if they fail. By that, we can provide a better approximation of the *think* part of the algorithm to the human. An alternative for sampling-based algorithms are, for example, constraint-based optimizers like [58, 59], as they generate a movement even if they cannot completely fulfill the specified goal. From this motion, its direction and premature end, the cause of failure, for example, an overlooked collision, the end of the working area, or axis limits, can be derived because the end configuration of the robot will be close to the reason for failure. Both motion-generating operators are interchangeable for some domains, although they have slightly different properties. Our proposed approach to improve the investigation in case of failure and to enable an explainable AI is to replace non-inspectable algorithms with their debuggable counterparts in case of failure so that the root cause can be further narrowed down quickly as shown in Figure 3.16.

We can further improve the explicitly modeled heuristics with statistical data for known domains. This way, we would know whether the failure in a sub-task is exceptional or relatively frequent. With this knowledge, which can also be stored in our model, we can control our plan-

ner's search algorithm and guide the engineer during troubleshooting. Since we have already implemented distance functions for declarative knowledge, states could be ranked by distance to the specified goal. Therefore, we could also rank planning states based on their most distant state during backtracking.

Further Tested Domains

We applied our algorithms to an extended version of the manipulation domain, which also considers material transport. Only because of its scalability can the planner deal with the problems of this large domain, with its dozen machines and several goal products, which was already discussed in Figure 3.8. The resulting plans consist of several hundred steps, and the planning times are still in the two-digit second range due to the significant acceleration through our hierarchical factorization.

Another domain we applied our algorithm to is the process industry, as it is vital for some use cases to introspect the algorithm and analyze what it *thinks*. Due to their static piping, these factories are not flexible and cannot benefit from the smaller batch sizes that our planner would make possible. However, we can help when a fault occurs, such as impurities detected during the final inspection, and identify the possible defect. We can formulate this root cause analysis problem similarly to our production tasks with small lot sizes. In this case, the goal is to create a contaminated product. The domain consists of nominal operators, which process the ingredients without unintended modifications, and fault-injecting operators, which cause impurities. Once our planner finds a solution that leads to the faulty state we defined as our goal, we can analyze the plan and identify the instances and operators that caused the problem.

For each of the real-world assembly use cases, we also have to consider the *act* of the perceive->think->act loop. Our system natively considers deviations between the real-world execution and the simulated planning and can react accordingly if necessary. Due to the set-based model, we can detect interdependencies of operations during *acting* and, therefore, reduce the impact of any parallel execution strands. In chapter 9, we proposed an asynchronous parallel execution on two independent robotic arms, for which the planner optimizes for maximal parallelization. During the *act*-phase, the set-based model separates the task-relevant resources from environmental information, which poses no constraints on the temporal ordering.

These different domains, from perception tasks over discrete manufacturing, process industry, and root-cause analyses, as well as the explainability and introspection capabilities, demonstrate our approach's versatility, flexibility, and expressiveness, which eases engineering and thus reduces the effort needed to implement new tasks. In the next chapter, we leverage our formal model's possibilities to improve models' reusability by automatically adapting them to new domains. By that, even less manual coding effort is required to solve new tasks.

4

Domain Optimizations

This chapter was published in a slightly modified version in the journal of chapter 10. Only rare and expensive specialists can set up and maintain autonomous systems. We need possibilities to facilitate and accelerate this process for broad applicability. This chapter discusses approaches supporting the engineer, automating some tasks, and enabling advanced modularization, especially during the application phase. We hope that in the future, more workers will be able to set up and operate autonomous machines and that the rare specialists will be able to do their work more efficiently and quickly, thus having a more significant positive impact on productivity. Together with the explainability discussed in the previous chapter, this can reduce or even eliminate the barriers to the widespread application of autonomous systems.

4.1. Motivation

Our goal is to provide autonomous systems for the masses. Autonomously manipulating household robots is a long way to go due to safety issues and hardware costs. However, current solutions are too expensive even for the industrial sector, although there is scope for more significant investment, operator training, and more controlled conditions. Flexible and autonomous systems are usually complex, especially regarding software and parametrization, and therefore, they are complicated and expensive to set up and use. There are few areas outside the scientific community for which the applicability of these advanced systems with a high degree of autonomy is feasible. Although those systems cover a wide range of tasks and can handle changes in the environment, potentially increasing their reliability and allowing automation of formerly unknown tasks, the cost of their installation and adaption often still does not pay off.

To overcome this issue, we can improve the reusability of solutions for similar problems, reducing the cost per installation. Since resilience to changes is one of the key features of flexible, autonomous systems, only minor modifications are required to apply them to new domains if a similar task was already targeted. We can further increase the number of reusable models by mod-

ularizing our domains and separating specific from generic code. Solutions for new tasks can then be merged from different sources without major additional implementations, which allows trained people who are not necessarily software engineers and experts in writing models to use our system. With current approaches, optimizing the domain for a specific task requires a lot of effort, time, and expertise, even if existing models are reused. Optimization prior to the planning steps is vital, especially for a domain composed of different origins.

We want to automate both the automation process and the domain's validation and optimization. Existing methods, such as skill-based architectures [60, 61, 62] or model-based descriptions of systems [63, 18] focus on the semantic description of either declarative or procedural knowledge. If the model neglects simulated abstractions of actions, it is impossible to validate the planned behavior of the system regarding the given task. Static analysis of non-functional properties, such as latencies, would still be possible [64, 65], but becomes irrelevant if the task itself fails due to erroneous parametrization. On the other hand, we could define resource allocation, for example, as a part of the task itself, such that non-functional properties are checked and fulfilled during our planning.

Consider the example 6 in which an AGV must navigate reliably without collision. As soon as we can tell if the system is safe and works based on its model, we can automatically plan valid compositions and optimize for secondary targets such as price tags. With this information and the provided model, we can calculate a maximum latency that our system must ensure, as it will fail to react adequately otherwise. We can check for this latency and validate the system's non-functional prerequisites. However, this will not tell us if the AGV will react in time to anything in its way. We do not know if the sensor can detect an assumed obstacle, which can depend on its material, nor can we tell if the signal is in time for a proper reaction, which depends on the range of the sensor.

Example 6:

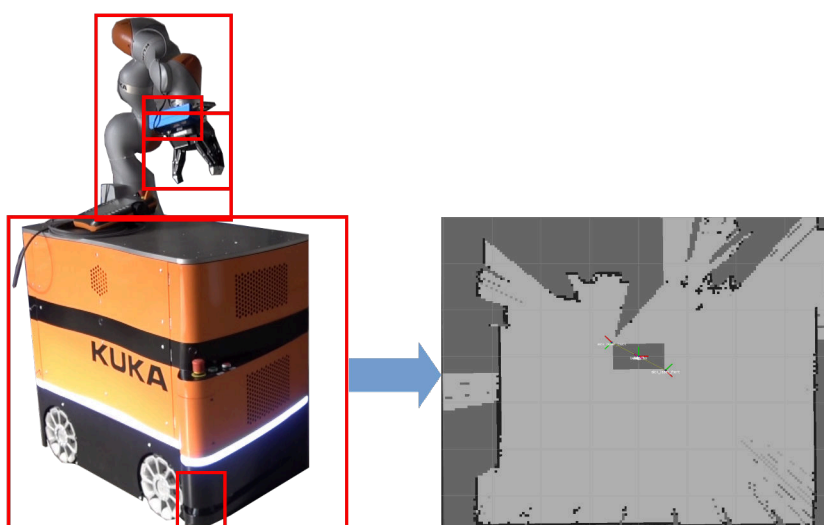


Figure 4.1.: For mobile robots, several components are integrated into an overall system, such as the mobile platform, lasers, manipulators, cameras, and grippers. The same applies to the software for localization and mapping or manipulation.

A mobile robot comprises different software and hardware modules described by independent models. For example, the robot of Figure 4.1 has a robotic manipulator with a wrist-mounted depth camera and gripper, which itself sits on a mobile platform equipped with laser scanners. Our model describes those components, the information they can provide, and algorithms to compute a representation of the environment based on the sensor input.

We look for a system that effectively prevents collisions and can localize and navigate a given environment to manipulate objects. The goal of the domain tells us the known speed and acceleration limits of the task. With this information and the provided model, we can calculate the maximum latency our system must ensure, as it will undoubtedly fail to react adequately otherwise.

Additionally, the models for the sensors, such as laser scanners, tof-cameras, 2d-cameras, and ultrasonic sensors, tell us which materials they can detect at which distances and accuracies. Together with the environment model and the properties of internal sensors such as odometry or the IMU, we can then tell if and how accurately the AGV can localize itself in the provided environment and choose the hardware accordingly.

Therefore, the non-functional properties, such as timing, allow us to check whether necessary conditions of the infrastructure are met in a composition. However, we cannot tell if the selected components will work as intended until the functional implementation has been validated successfully. To answer this question, we need a simulation of the domain's functional aspects, which runs the components in interaction with each other and, as a by-product, also checks for non-functional properties. This method, which is focused on the functional aspects, is also our approach for the hierarchical planner. Non-functional properties are not considered on the coarse layers but are integrated as soon as they become relevant for the success of the operations in the refined levels. When we compose models from different sources, we can, therefore, check for both functional and relevant non-functional properties in one go.

There is also no option to rely only on procedural knowledge and ignore the declarative types. We want to plan or manually define the concatenation of goal-oriented actions. To reduce the number of possibilities, we must pre-select the candidates for inputs and outputs based on our model, which encodes the domain-specific knowledge, before we start brute-forcing possible combinations. Furthermore, goal specification is not possible without a model of declarative knowledge, which completely prevents the use of planning algorithms and, therefore, hinders the implementation of intelligent and autonomous systems. Our hierarchical decomposition is based on declaratively defined intermediate goals, which require particular attention during fusion of domains to ensure good performance.

Our model of chapter 2 brings together declarative and procedural knowledge and expressiveness for functional and non-functional properties and is thus the ideal basis for the automatic composition from different sources into a meaningful domain. In this chapter, we discuss algorithms that facilitate the engineering process through automatic checks, optimizations, and even allow post-processing of successful plans to improve execution times further.

4.2. Alignment of the Domain to the Task

For most industrial applications, integration and software development are the decisive cost factor [66, 67]. If we bring autonomous systems into widespread use, we can reduce the cost per system by leveraging their scalability and distributing the development expenses to many installations. This only works if the software becomes reusable, which is not the case for current automation systems. As described in example 7, different parties would then contribute their expertise about a specific domain or task and develop components for the overall composed system.

Example 7:

In an ecosystem with semantically described modules, a camera supplier not only sells its sensors but also provides a library of software components with corresponding models. These models can then calibrate intrinsic and extrinsic parameters or determine the optimal exposure for a particular task. Other parties implement algorithms for data-driven object recognition or classification, robot manufacturers model different operating modes with various precision, speed, and dexterity levels, and integrators sell high-level models for task specification. All semantically described components can then be composed online, for example, with our hierarchical planner, to form a meaningful sequence of actions and solve a given task.

For example, we want to perform the assembly process of a control cabinet. For this, we must localize modules in a box. The perception algorithm must, therefore, handle a cluttered background and multiple instances of the desired object in a single picture. Possibly, no algorithm can provide this resilience to noise in conjunction with the required precision for the later assembly step. We must, therefore, combine a less precise localization with components that actively reduce the uncertainty later. In our example, the clutter-resistant algorithm operates on point clouds and is, therefore, inaccurate. We can then combine it either with a 2d based visual-servoing algorithm during insertion such as [68, 69, 70], an additional perception step with an edge-based algorithm [71, 72, 73], or a less rigid mode of the robotic arm, in combination with a trajectory, which considers the higher uncertainty [74, 75].

Experts in their fields typically provide different options to reduce uncertainty. For example, perception specialists develop visual-servoing and 2d localization algorithms while the control community proposes compliant motions with appropriate path generation to maximize success rates. Our approach can integrate these algorithms to solve each problem with the appropriate algorithm.

Splitting the task into manageable sub-tasks allows for the components for composition to be implemented by the experts of a single domain with their specific approaches and ways of thinking. Our system will then automate the costly manual integration step. During the planning process, we compose the modules using the hierarchical model and the respective algorithms, eliminating most of the manual integration effort. The operator only needs to define the task and list the sub-domains that might be relevant such that a quick and cost-effective setup ensures maximal flexibility. Nevertheless, there is still potential for optimization during this integration phase.

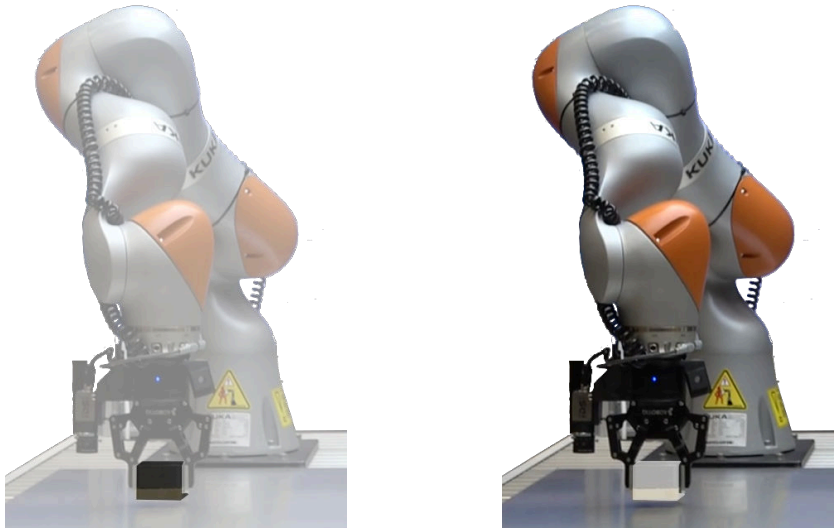


Figure 4.2.: Different experts describe the same situation either in an object-centric manner or with a plant-centric view. In the left picture, the box is the focus of attention, and the gripper and robot are the only side characters. This perspective is suitable to describe abstract tasks. On the right side, the robot is in the center of attention and manipulates an object, which happens to be our box [5]. This view is appropriate for low-level task and motion planning and component development.

The proposed planning algorithm of chapter 3 has its strength in solving complex and, therefore, practically relevant tasks. However, the results may be sub-optimal because of the intermediate goals calculated by the abstract levels if the abstractions do not fit each other as described in remark 11. The model, which describes the problem, strongly influences performance even if the same planning algorithms are used, which is highlighted by the task and motion planning domain, for which many different approaches exist, which sometimes only differ in detail, like a goal reference relative to an object instead of absolute coordinates, which allows for higher flexibility and robustness of the applicable motion generating algorithms [76, 77, 78]

We, therefore, propose domain optimization algorithms in this chapter so that different views and, thus, implementations of the same problem result in an optimized formulation.

Remark 11:

NP-hard problems prevent algorithms from determining the optimum solution within a limited timeframe. Since our primary goal is to offer sufficient, but not perfect, solutions within the limited computing time, our planner is designed to seek a promising but potentially sub-optimal solution within a time limit. The chosen abstraction, which determines the overall task's factorization, significantly impacts the scale of the additional cost added compared to the optimal solution. A good model with coordinated abstractions reduces the price we must pay for sub-optimality.

Hierarchical planning is a strong and flexible heuristic. Nevertheless, it can even lead to automatically composed domains that seemingly cannot be solved. This problem becomes even more relevant for industrial applications where the domains are designed by several independent teams



Figure 4.3.: The solutions for the original and optimized domain for the assembly of the box and fixture with four screws start with the same initial steps (first row). After the box is localized and picked, and the lid is refined, we conduct the assembly, re-localize, and pick up the assembled objects. The difference starts after the first screw is picked up and inserted into the box-lid assembly. The original domain has to place the screwdriver to hand over the assembly, as the screws would be facing down otherwise, and place it on the table to reach the intermediate goal. In the optimized solution, we can pick up the next screws right away. Therefore, we can skip the additional handover and pick-up of the screwdriver. Additionally to these described steps, prior to any pick, screw, and assembly, the parts must be localized. The robots must move above the object and take a picture with the wrist-mounted camera [5].

rather than a single person or a small group that knows the bigger picture of the overall structure. In a scientific setup, even if the problem formulation is sub-optimal, the expert who sets it up can correct it immediately. Moreover, in an academic environment, only a few problem classes with low reusability are targeted. Scenarios are often designed to primarily demonstrate the advantages of the specific planning approach, as opposed to the problem-oriented specification in the industrial environment. To eliminate the driving cost factor, no expert can re-adjust the models for each system. The only solution is to solve inconsistencies algorithmically. The algorithms proposed in chapter 8 address this problem and provide automatic domain optimization to shorten planning times and improve the solutions found by the planner. The main problem of composition within a hierarchical framework is the different views on the described process, which misaligns the levels of abstraction. As a result, intermediate goals cannot be achieved at all or only by an unnecessarily complex and long plan.

Example 8:

The sub-optimality of an unaligned hierarchical factorization becomes evident for an assembly of multiple parts modeled by a system integrator and a component supplier. We want to assemble a lid on a box and fixate it with four screws. On the abstract level, the system integrator describes the actions in an object-oriented manner. It explains how parts are manipulated, which is the intuitive way to talk about high-level tasks, such as picking the box, drilling four holes in the box, attaching the lid to the box, and fixing it with four screws. However, at the more refined level, component suppliers have a machine-centered view and formulate the actions only as changes to the machine state. In our example, these would be absolute or relative movements of the robot arm, actuation of the gripper, or drill activation. After a grasping step, the object is thus located in the gripper of the robot arm and thus no longer exists separately in the refined domain but only as part of the machine. While the abstract domain describes

a box and no robot arm, the operators on the refined level only speak of robots with attached objects, such as boxes, as depicted in Figure 4.2. Suppose intermediate targets are defined on the abstract level by the box’s properties. In that case, the finer level cannot reach them directly, but only if the box is separated from the robot again, which results in additional steps that must be planned and executed, which increases both the complexity and computation time and the hardware’s execution time. We address this problem by optimizing an isomorphic transformation grounded in our set-theoretic model. This reformulation, explained in Figure 4.4, allows us to reach the intermediate goals wrapped in another instance. In Figure 4.3, the comparison between the solutions of our hierarchical planner for the original and optimized domains is highlighted.

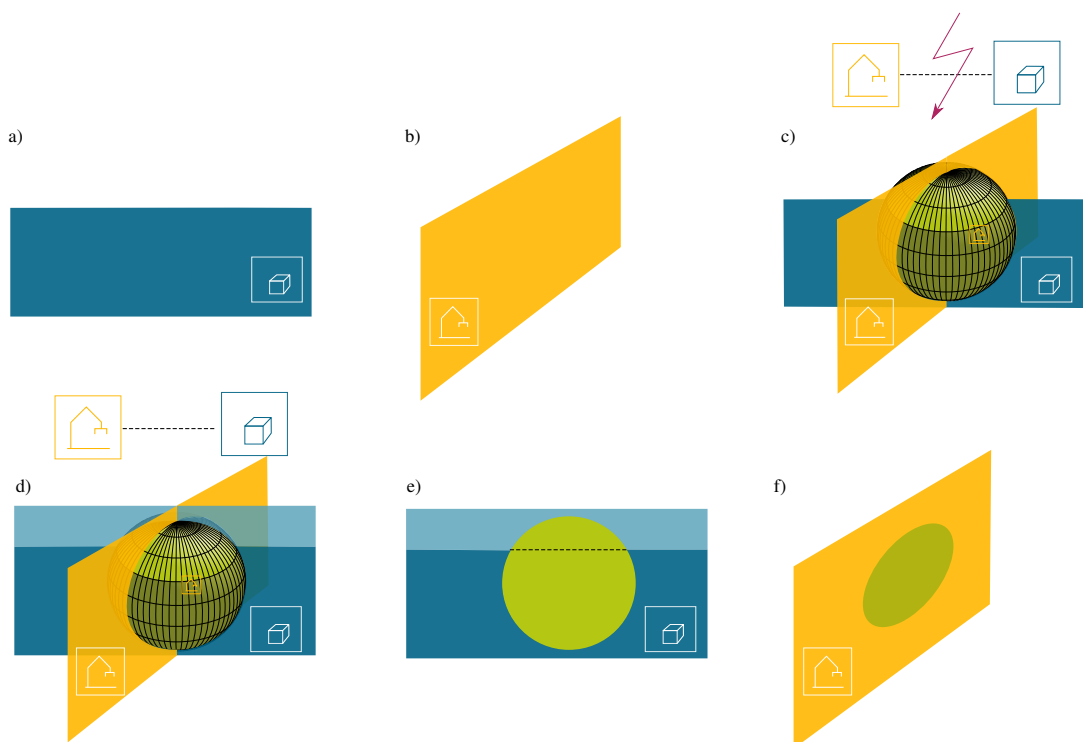


Figure 4.4.: The object-centric a) and plant-focused representation b) describe the same physical phenomenon (relation between box and robot), which is represented by the green sphere in a higher dimension. All instances but some corner cases can be represented in both variants. These exceptions are empty lists, such as a robot with an empty gripper, for which all grasped objects are represented with a list, as depicted in c). If we project this empty robot onto the object plane, we need a *no object* instance in which the robot can be additional detail. Therefore, we extend the less expressive formulation as in d) and can then project the physical world to each of those representations and thus use them interchangeably to represent our intermediate planning goals [5].

In example 8, we want to eliminate the misalignment, as it makes hierarchical planning more difficult, can prevent achieving intermediate goals, and leads to sub-optimal solutions. Our algorithm to accomplish this relies on the formal model of declarative and procedural knowledge and the hierarchical structuring of our domain. The fundamental problem of the misalignment is that

instances of concepts with different concept bases cannot be similar. Therefore, the instance at the refined level cannot directly fulfill the goal imposed by the abstract level, although they describe the same physical phenomenon. To avoid additional processing or conversion, we search for isomorphisms in the concepts of our domain and apply model-to-model conversion so that automatic hierarchization can place concepts of different conceptual bases into a common hierarchical order based on these isomorphisms as depicted in Figure 4.4.

The formal algorithm is discussed in detail with an in-depth analysis of the resulting planning times and plan lengths in chapter 8. The primary step of this alignment starts from the goal set on the abstract planning level. This goal defines the concept base against which all isomorphic concepts are aligned. Let us call this concept base B_{Γ} . Then, we search the domain and identify all concepts that use any concept with the concept base B_{Γ} . In our example 8, the goal on the abstract level was a box with a lid and four screws. The concept base of the respective concept is *box*. We can now search the entire domain for any concept that uses a box to impose restrictions on the concept base. For these concepts, like a robot using boxes in the list of attached objects, the isomorphic mapping is recursively computed and stored in the domain model so that any concept using a box anywhere in the graph is aligned with that concept base and thus, with the goal specified by the user. The set-based formal models allow this conversion without violating the correctness of the domain, thus enabling the automatic merging of models from different sources.

4.3. Reasoning for Data Fusion

The previous chapter discussed the challenges of composing domains from models created with different views on the same task due to varying user roles. We proposed an operator-centric algorithm for domain optimization using declarative knowledge of the concepts. This subsection discusses a similar problem for composed domains from different sources. We start with algorithms that solve minor inconsistencies within the declarative knowledge, such as different units for physical parameters. We then continue with approaches that perform a general data fusion and generate new operators. For this purpose, we analyze procedural and declarative knowledge in a combined manner.

Unit Conversion

We can observe the mixed success of standardization attempts, which, despite significant efforts, have not yet led to a unification in many domains. Even if we reinvent engineering and spend time, brains, and money formally defining every process and data element that needs to be considered during the engineering, automation, and task specification process, we will probably still be unable to establish common standards everyone adheres to. Let us take the system of units as an example. Even though SI units have become widely accepted, the implementation could use *mm* instead of *m* for lengths. On a larger scale, we must apply our algorithms to more complex data structures in misaligned domains and "brownfield" environments with differing implementations of various problems. Standards such as OPC-UA [79], SYSML [80], Automation ML [81], or the compos-

able models of RobMoSys [82, 83] have common goals, with overlapping fields of application but differing implementations. This is particularly relevant to our planning approach because of the vast possible field of applications, which consequently touch various fields of standardization. For this purpose, we want to reuse a new plant's existing tools and algorithms, utilizing the already standardized knowledge of other applications while keeping the integration effort as low as possible. Even though the sets of instances described by values of different units in our models are isomorphic, the planner cannot directly execute operators with concepts that do not use the specified unit. Therefore, we must convert to another representation before executing the given operator. Classically, engineers program these interfaces manually for each integration, which is a time-consuming and costly process. We would like to automate these conversions based on the models of the domains that are involved.

Example 9:

Consider the example of an operator that needs a box with its specified mass in g and an instance that comes from another abstraction layer and uses kg as depicted in Figure 4.5.

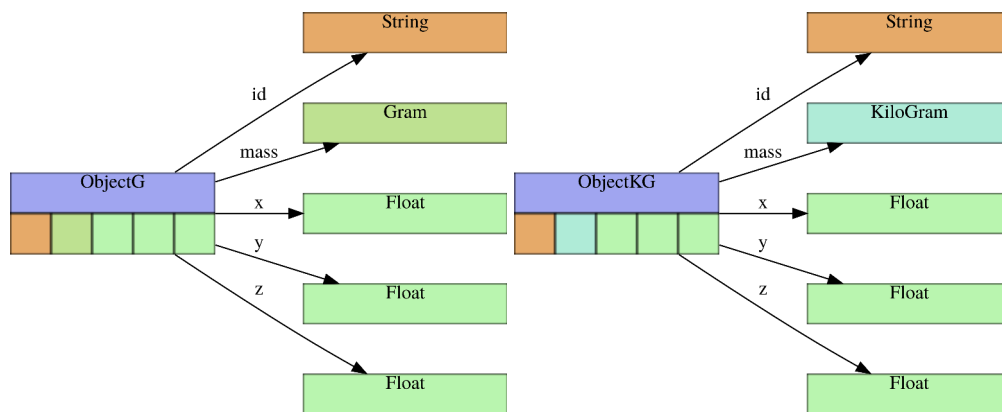


Figure 4.5.: Concepts of an Object with different units

Additionally, we have the conversion operator of Figure 4.6 that calculates g from kg for arbitrary masses.

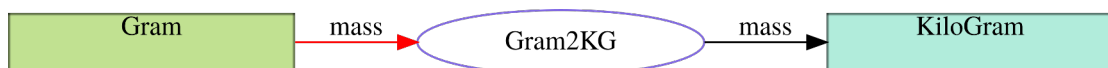


Figure 4.6.: Operator to convert between g and kg

However, when we have an instance of a concept that *has a* mass defined by one of the two units, we cannot directly apply this generic conversion operator since only a concept that *is a* mass can be converted. The concept base is wrong for the conversion, preventing the operator's application. Therefore, we need the special operator depicted in Figure 4.7 that converts the ObjectG to and ObjectKG.



Figure 4.7.: Operator to convert between an object with g and kg

Implementing this operator would apply the conversion operator on the sub-role and keep all other information untouched.

A naive approach for the automatic unit conversion of example 9 would be to use the same isomorphic transition we described in section 4.2. This way, we can change the basis of the concept and thus apply the conversion operator to the mass with a *box*. However, this will cause all information of this instance, except for the mass, to be lost. In contrast to the alignment between different levels of hierarchy, which only had to apply the similarity operation between two instances, in the case of conversion, we will not cross any boundary between abstractions. Therefore, we must not lose any information. Otherwise, we might get stuck in our planning process.

For manually modeled and implemented operations, the engineer ensures that every detail of the output instance is filled. We could define an operator that performs the conversion process manually, taking boxes with the mass of one unit and returning boxes with the mass of the other unit, preserving all details. In our modeling scheme, defining only the required data as inputs and outputs to the operator is good practice, which allows for broader applicability, use within the planner, and precise description, which is the ultimate goal of the model. In the case of the conversion operator, however, detailed inputs and outputs are required in the model, as we make the statement that even if the rest of the information about the box, apart from the mass, remains untouched, we have ensured consistency at this level of detail. In this way, we also declared that redundant information in our model, such as mass, volume, and density, is consistent. With the specialized approach, we need a conversion operator for each concept in the domain's concept hierarchy. We cannot take advantage of inheritance but must resort to manual coding. Considering domain-specific redundancies that depend on declarative and procedural knowledge, this tedious integration task can be automated using our set-based models.

In example 9, we discuss the conversion between different units for the mass of an object. Once we implemented a conversion operator between the different units, we have all the information needed to generate conversions between instances that have a mass with an arbitrary unit, such as boxes with g and kg with the algorithm described in remark 12.

Remark 12:

Our algorithmic approach to implementing the automatic conversion operator between concepts that use different units consists of three steps, which are highlighted based on example 9:

- 1) The information within the instance is decomposed so that individual instances are available for each role as depicted in Figure 4.8

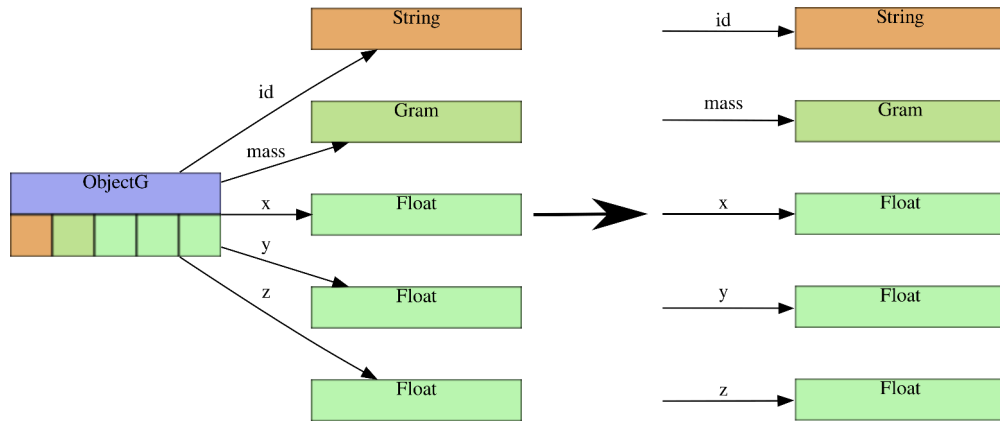


Figure 4.8.: We can decompose the concept so that the concepts on their roles float freely.

2) Figure 4.9 shows that the conversion operator can be applied to a subset of the instances once the required information has been extracted.

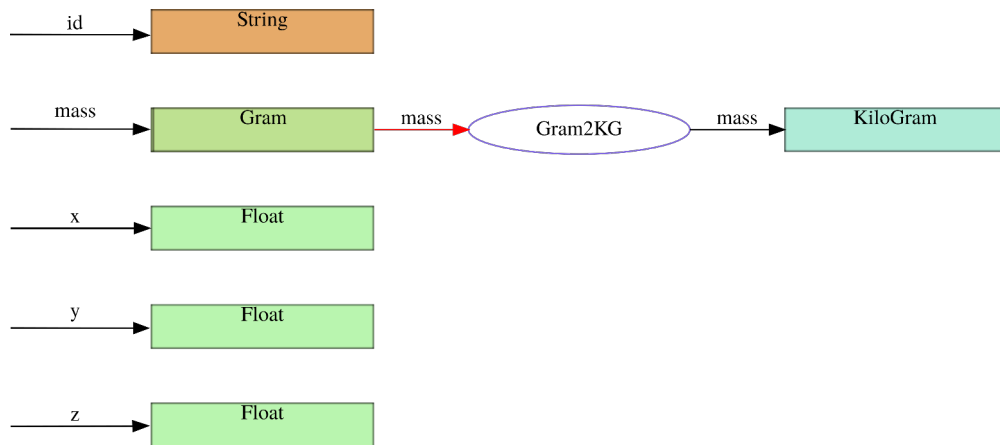


Figure 4.9.: The conversion operator can be applied to the extracted information of the concept.

3) The separate pieces of information are merged back into a single instance as depicted in Figure 4.10.

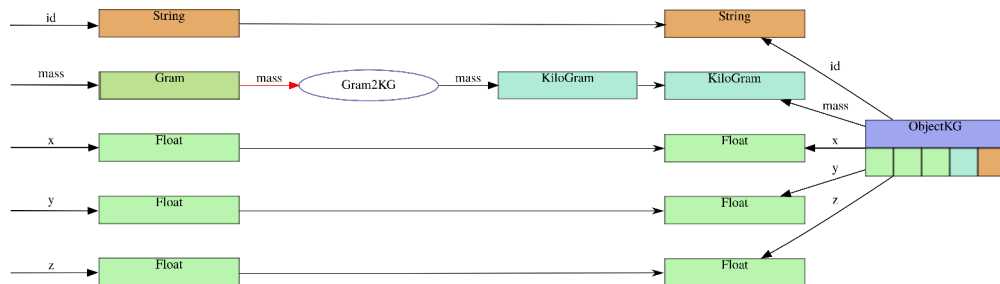


Figure 4.10.: With knowledge about the original roles, we can compose all available information to the new object with the converted units.

This way, we create a formally usable instance for our original operator. Challenges arise in the second and third steps as we must consider inconsistencies and assignment problems. We can address this with heuristics based on the roles and concept types. As a fallback, an engineer can choose from alternatives interactively.

For the fusion of step 3 in remark 12, we must decide which information should be placed in which role of the merged instance. To answer this question, we can set up a semi-automatic process where the assignments are pre-filled but can be corrected manually. We also know that the roles will probably remain the same for the unused information. So, we can concentrate on the newly computed information from our conversion operator. At this point, we can define another heuristic that analyzes an operator's inputs and outputs and identifies information that was probably only processed but not newly generated. For each consumed input, we, therefore, look for an output of the same concept and, if there is none, an output with the same concept base. These pairs will likely describe the same information; thus, we assign the former role to the newly computed instance. For the other outputs that do not yet have a candidate for their role, we can compare the structure of the newly created concept that neglects pieces of sub-information without assigned roles and compare that to more detailed concepts of the current domain. We can then find candidates with a structure that matches this prototype concept and use the same roles for additional information that complies with the concept or the concept base of unmatched roles.

In our example of the box with mass, which is also used in remark 12, the conversion operator would consume a mass in *kg* and return it in *g* with concepts that have the same concept base. We merge all available information extracted from the initial box concept to a new instance. In that case, we cannot assign the recalculated mass in *kg* and are missing the information for the mass in *g*. Thus, we search for other concepts of boxes in the current domain, which allows us to consume all available information with the role hint we gathered during the decomposition process. This concept must have a matching structure to our box without the mass but additionally use *mass in g* as a sub-concept. Once we have found a candidate concept, we can fill an instance of this concept with the pieces of information extracted and calculated from our box with mass instance based on their original roles. This solution is not without doubt error-free, but with this set of rules, we can find a reasonable estimate for plausible fusion operators, which only need to be suggested to the user for validation. This domain-specific conversion process, represented as a plan template, can then be stored as a new operator in our composed domain and is thus available to the planner. By that, further links between otherwise separate operators are possible during planning without requiring manual coding to bring together previously incompatible parts of a domain.

Arbitrary Operation on Sub-Information of an Instance

The conversion problem between different units is only a simplified special case of a more general meta-problem. We want to apply operators on a subset of the information within a concept. Multiple operators may exist; they do not necessarily consume all inputs, and we still must validate integrity when fusing the new pieces of information. As highlighted in example 10, the core challenge is redundant information within our composed concept, for which consistency must be ensured.

Example 10:

The challenges of domain-specific redundancies that prevent an automatic data fusion become evident in the intuitive example of a box with mass, volume, and density. We can calculate the mass even if we only have volume and density. Let us assume that the color is defined as an additional piece of information for our box in the example. According to our algorithm of remark 12, we decompose the box concept into three concepts with concept bases for volume, density, and color. In those concepts, we can apply the operator to calculate the mass, which consumes nothing. As we have no other operator in this domain to calculate the mass given color, density, and volume, the validity check succeeds, and we can proceed with the fusion. We can analyze the existing concepts, determine that the *mass* role is the correct one for the additional value we are calculating, and create a template to model our additional operator. In our second domain, we have an additional operator that classifies objects based on their color and calculates a mass based on that class. If we apply our algorithm to the same problem in this domain, we find that there is a redundant path to calculate the mass with the given information. Therefore, we cannot generate the fusion algorithm in this domain because we would generate redundant data and do not know whether the mass affects the color or vice versa.

This data fusion is commonly discussed in the field of sensor data fusion, where several sensor readings are combined to obtain a better estimate of the current state. These bits of information may be redundant because the same scene is recorded but with different sensors, modalities, and at different times. In this case, we need probabilistic models that extract the redundant part of the information from the noise of different measurements and reconcile possibly conflicting readings. This process is not possible without detailed probabilistic sensor and process models. For our data fusion problem, we cannot expect these probabilistic models to be generally available because they must be normalized for each domain. However, our operators are generally defined in different modules, and the domain can be automatically composed. Therefore, it is unlikely that we can rely on robust probabilistic models of the relationships between the operators. Nevertheless, from the operators in the domain, we can deduce which information within a concept is entirely independent of the other specified data. To check whether all outputs of an operator we want to apply to a subset of the information within a concept are independent of the other pieces of information, we specify a planning task in the meta-domain as highlighted in remark 13. In case there is a redundancy for the newly calculated outputs, a probabilistic sensor fusion with appropriate probabilistic information in our models would be necessary, which is targeted based on our models in chapter 13 but not the focus of this chapter.

Remark 13:

As mentioned in chapter 2, all operators and concepts we define in a domain can be declaratively described in a so-called meta-domain. In this domain, the concepts of an ordinary domain represent instances. The similarity property of these instances reflects the hierarchy of concepts within the ordinary domain. Additionally, we can implement operators in the meta-domain and define and solve planning problems. These operators can have meta-models of concepts as well

as meta-models of operators as inputs. Using a meta-operator that simulates the operators of our usual domain solely based on its input and outputs, we can then specify an initial state and goals that consist of instances that describe concepts. This setup allows us to check for independent information within our concept, making data fusion possible. For this purpose, we specify a planning task for each output of our conversion operator, with the goal of computing this output from all non-consumed inputs that are available as initial facts. In addition to these instances, the meta-concepts of all operators of this domain apart from the conversion operator are available as start instances. Since our planning algorithm can detect loops and thus aborts planning if no plan exists, we can use it to detect non-redundant data within the concept in a non-constructive manner. If we can find a plan that solves the task, there is an alternative way to compute the same information as our conversion operator has calculated, and thus, our concept contains redundant information. If this is not the case and we make the closed-world assumption for our domain, we can conclude that the checked information is not redundant and can, therefore, be merged without losing consistency.

Identification of Meaningful Networks to Process Sub-Information

We proposed an approach to solve the assignment problem in many cases and to switch to the semi-automatic process in case of ambiguous roles. Additionally, we discussed an algorithm to validate the consistency of the merged concept. However, we always considered a single, given operator, which we now check for applicability to sub-concepts. The even more exciting task is to identify meaningful applications of operators that can be applied to sub-concepts in our domain. This can be done offline, followed by the verification process described above, and ends in the generation of plan templates. These operators are then available during the planning process and envelop the manually implemented operators but work on other types of input and output concepts. The *operator* may be a sequence or network of operators that transform the initial facts provided by the sub-information of a concept into the set of information that is then merged into the new concept. This process can be formulated as a planning task in the meta-domain again. We can decompose any concept in the domain and specify any second concept within our domain as a goal. Each plan defines a feasible network that can be stored as a plan template. Only existing concepts make sense as goals since newly generated concepts are not used as input and are, therefore, useless to the planner. All conversion operators are only valid for the specific domain because additional operators can make sub-information within a concept redundant as example 10 shows.

4.4. Model Harvesting from Unstructured Sources

We can apply our modeling and hierarchical planning scheme to various domains with existing partial solutions and components. However, the implementation of these algorithms is rarely described with a formal, machine-readable model, so a direct application within our planning scheme is not possible. However, developers of modules provide an API with accompanying documentation for developers. With the help of natural language processing (NLP) such as [84] combined

with heuristics, we can automatically analyze and harvest the information and extract the formalized models we need for our planner from the API documentation. In [85], we applied this approach to various perception libraries. Valuable information for our models is the set of available functions for which we can generate operators. We must extract the inputs and outputs with their respective types for a minimal model of these operators.

For each of these types, we generate a new concept. The inheritance between these concepts is mapped to the concept hierarchy in our domain. Libraries that focus on object-oriented schemata are reformulated in a functional way so that they can be used in our planner. Since libraries are usually implemented in non-functional programming languages, we must separate input from output and treat them separately for the function call. Strong typing is also beneficial because it allows us to extract more information from the library, reducing the branching factor during planning. Despite implementing example libraries in the typed languages C and C++, each analyzed library implements generic types that cover various information types. For example, the generic class *image* has internal sub-types such as gray-scale, RGB, and edge images, which are not explicitly represented by different classes but are checked dynamically at runtime when the function is called. Invoking functions with the wrong type will always lead to failure, which would be valuable prior knowledge in our model. Engineers consult the documentation written in natural language and derive its correct usage. There are descriptions of the required types and parameterizations, as well as the preparation of the inputs and outputs. Our harvesting approach includes a Natural language processing (NLP) module that allows us to parse the function call and the associated documentation. This gives us an estimate for the classification between inputs and outputs, which can be stored in the model and verified with further planning steps.

4.5. Automatic Refinement and Abstraction of Operators

Even in an autonomous system, some processes are not planned or reasoned automatically but are defined and programmed manually. For example, we want to validate the feasible value ranges for some actions before we execute them on the actual hardware, or we want to define the exact interface data to be sent to the machine rather than mindlessly testing and hoping for valuable feedback. These processes are usually neglected in the discussion of autonomous systems but must be implemented for any system. Checks are necessary to avoid the destruction of real hardware by non-permitted commands, especially for probabilistic and data-driven approaches, which tend to find errors and shortcuts even in simulation. With our set-based, model-driven approach, we can also formally describe these validation processes so that we use them not only during simulation but also for the execution. There are two ways to represent this procedural knowledge within our framework. Either we program it in an arbitrary language and integrate it into our model as a black box, or we model the data flow, for example, in a behavior tree. The latter form is advantageous because it allows both advanced introspection and automatic refinement. We call them *plan templates* because they have the same graph structure as plans. Contrary to plans, concepts are used for declarative knowledge instead of instances, making them templates for other

instantiations during simulation or execution. We can either define these operators manually or extract and *learn* them as described in section 4.7.

The manual modeling process of a domain can either start from the most detailed level so that more and more abstract levels are defined during modeling or from top to bottom. In both cases, we can automatically generate proposals for plan templates. We only need to abstract or refine the plan template; the desired input and output concepts define the abstraction level. Other operators with defined inputs and outputs are used within the plan template's graph, which defines its behavior. To adjust the level of abstraction for our plan template, we select an operator for which all inputs are either manually defined or outputs of already checked operators. We then verify that for all inputs to that operator, the provided input concepts are more detailed than the modeled inputs and that there is no more abstract operator to which this property also applies. Otherwise, we replace it with the more abstract or refined operator and continue the process until all operators in the plan template are replaced or verified.

This way, we have automatically generated new abstractions or refinements for existing operators based on our set-based model. With this algorithm, we can only abstract or refine operators used when a one-to-one mapping exists with an operator at another level. This approach will fail if the refinement consists of several chained operators or the abstraction accumulates multiple operators in our plan template. However, we can extend the algorithm and apply meta-level planning to find valid sequences for refining an operator. The goals and facts in this meta-domain are defined by the inputs and outputs of the operator that needs to be replaced, and the operator instances are defined by the operator hierarchy described in section 2.2. For abstractions, we can search for operators that use the current operator as a refinement and try to accumulate neighboring operators that are in the same domain. These extensions are ambiguous and can only be suggestions for refinements or abstractions to an engineer.

4.6. Detection of Model Inconsistencies

The more complex the domains become, the more bugs and problems they can contain. Therefore, we need algorithms that support the engineer and verify the implemented models. This is not only relevant when assembling models from different sources but also during the implementation of sub-domains. For broad applicability, non-experts who are not familiar with modeling to the same extent as scientists will also have to contribute to the system. Especially for them, tools that detect inconsistencies in the model are beneficial. Examples of such verification algorithms can check the alignment of the formal model with the implemented black box code, consistency in the operator hierarchy, and inconsistencies between operators.

To check the consistency between the implementation and the formal model, we start by recording the operators used during planning. For the output instances, we can check if they match the output concepts of the model and if we have seen at least one instance for each output modeled. We can play back a modified version of the recorded input instances to verify the inputs and check if the outputs change. Suppose this is not the case for a configurable portion of the samples. In

that case, these input instances or sub-details of the input instances are likely irrelevant to the operator. Thus they should be eliminated to meet the requirement for minimally defined inputs to an operator.

We can verify the feasibility of the hierarchy of operators by setting up a meta-planning problem in which we try to compute the output instances of the more abstract operator by using the more detailed operators and inputs to the operator to define the domain. If we cannot find a solution without the additional constraints resulting from executing the code, there can be no successful refinement during planning. If individual operators are not used during this meta-planning, they are also irrelevant to the actual planning process. Additionally, an input consumed on the course operator should be consumed at the refined level. All those findings are valuable for the engineer, who can then guide his attention and examine the specific implementations and models.

The last example of a possible verification is similar to the approach presented in the previous subsection. We can analyze each output instance for feasibility by looking for redundant information using the same meta-planning problem definition we used in the previous section. This time, however, we can check whether the resulting newly calculated sub-information is similar to the information available in the recorded output instance, according to our definition of an instance. If this is not the case, we must assume that at least two of the operators we use in this domain calculate contradictory results and must be checked accordingly.

4.7. Data Driven Optimizations

Robotics is the art of integrating different strands of science and technology into a powerful system that manipulates the environment. In chapter 2, we have discussed how our approach can handle arbitrary code with only a few preconditions and thus contribute to the automatic integration of algorithms. We also discussed the application of different planning algorithms within our framework in chapter 3, which can be considered operators in the meta-domain. So far, we have mainly focused on model-based algorithms and manually specified declarative knowledge. However, our system's set-based foundations also allow us to seamlessly integrate data-driven approaches for various challenges within a robot system. A simple use case for optimizations based on recorded data is the KPIs of operators discussed in section 2.2. With their estimate of the cost, in our case, the execution time of the most refined level, they guide our optimization during planning to find the cheapest or fastest solution. Even with manually calculated KPIs, we consider the inputs to the operator for which we want to estimate the cost, which could be utilized by data-driven algorithms.

A significant problem with data-driven approaches is to collect enough data to start learning. The modeled classical solution can provide the running system needed to record data for a good starting point, which already covers a large portion of the possible scenarios. From this point on, subsequent optimization and exploration can be conducted in a data-driven manner. Indeed, this initialization can also manifest modeled local minima. However, data-driven approaches might not be possible without them, especially for domains with poor simulations and, therefore, tedious data acquisition such as example 11.

Example 11:

In a logistics application case, the abstract transport process has a reference map or a distance measure to calculate the expected time needed for this job. Generic classical algorithms efficiently find routes such as [86, 87, 88] without needing any recorded data. However, these algorithms will only consider modeled circumstances and not learn from the deviations between the plan and actual execution. With an installed and running system, we eventually conduct enough experiments to gather data not only for simple statistical analysis of the resulting execution times but also to train a deep neural network. These data-driven estimators can potentially avoid unmodeled bottlenecks and detect complex relations between the provided start and goal locations, time of day, weather, and the expected cost.

Similar heuristics as the KPIs of operators can be used to control the hierarchical search, where the most promising sub-planning tasks, planning states, and input/operator combinations must be selected. This is similar to the work of [89], however, extended to multiple layers of abstraction and with the additional benefit of our set-based definition of the declarative knowledge, which allows further specification of the domain. With our model-based approach, we can mark the recorded scenarios and train domain-specific heuristics. The generated distance functions are parameterized by the expected value ranges for each domain and, therefore, need to be calibrated with recorded data. They can serve as an additional source of information for these heuristics. With stronger heuristics, we can seamlessly move from a model-based planning scheme to a data-driven reinforcement learning approach. Since this setting is defined per domain, we can construct a hybrid system with data-driven approaches for problems for which we already have enough data. Of course, we have less experience with very detailed domains where every run differs. However, on the more abstract levels, we can start training earlier since the density of the recorded samples is higher the less detail the subdomain contains. In this way, our hierarchical approach helps us to apply data-driven algorithms based on modeled abstractions that automatically aggregate information from a given domain.

Another exciting field is the extraction of semantic knowledge from the recorded data. With data analytical approaches that detect clusters within the instances of a concept, we can define more detailed concepts that correspond to our set-based definition of declarative knowledge. For meaningful new concepts, clustering, which can be facilitated with our distance measures, must be based on the behavior of an operator. We can then combine all inputs for an operator into a single concept and determine whether the function call will likely fail or return a particular result. Since we are likely to omit some information there but gain computing time, this clustering with the following result estimate represents a new abstraction layer on this operator.

Semantic knowledge extraction from deep networks is another interesting strand of literature we could integrate into our system. We learned about the advantages of modeled operations in section 3.3, while data-driven approaches are generally black boxes. With algorithms like rule extractions from neural networks [90, 91, 92], we can extract an approximation to the functions described by the data in a model-based way and then regain the explainability during planning. In

contrast to skill extractions such as [93, 94, 95], we additionally have the foundation for a multi-level abstraction and typing, which allows reasoning about the aggregation of primitive skills and thus allows us to scale to larger problems.

In some domains, it would be very beneficial to plan backward. The idea is to start from the goals and apply the inverse of the available operators to these instances until all leaf nodes are within the initially provided instances. Since there are generally fewer goals than initial facts, the branching factor at the beginning of the backward search is smaller than that at the beginning of the forward search. Applying a few backward steps depletes this advantage, and we must alternate forward and backward searches based on the volume of the currently available facts. However, an operator's inverse is generally unavailable, which would be a good use case for data-driven algorithms. We modeled all input and output instances and can quickly calculate value ranges based on recorded data and the model. Based on this information, we can generate arbitrary data sets for purely functional operators and thus train their inversion.

4.8. Postprocessing for Parallel Execution

Our hierarchical planning approach can handle even large-scale problems, such as the complete assembly process, including the geometric assembly and the logistics problems that handle transporting goods within the factory. Especially in such a large domain, the actions can be geometrically or logically independent. This allows us to carry them out in parallel, reducing waiting times and improving the factory's performance. This capability may even be a prerequisite for some applications in an industrial context. In general, specialized schedulers only consider parallel tasks in discrete domains. Often, they need a feasible initial solution, which they optimize for reduced execution times, leading to a higher degree of parallelization. However, they only consider properties on a single discrete level. As discussed above, this is insufficient for robotic applications, as autonomous systems require more detailed simulations or even feedback from actual execution. Our hierarchical and model-based planning approach combines the search for a successful plan with the optimization for expected execution times and recursive validation of all steps. We can use the explicit information in our model about the resources of each task and can explicitly define that a particular domain can be parallelized. This is in contrast with solutions using deep learning or optimization such as [96], which neglect this resource information and fail to optimize the overall sequence prior to the parallelization during the post-processing or only parallelize a single motion of the overall task and motion problem [97, 98, 99, 100]. For us, parallelization directly impacts the cost estimate for the planning states, which influences our planner's behavior. This optimizes the effectiveness of parallelization, which is only applied during execution. This behavior differs from the approach of [101] by its optimization on all (three) levels, which could support parallelization in an offline process.

We can further improve the performance of our solution in the case of operators, which spend a significant amount of time in a preparation phase during which shared resources are not yet manipulated. For those scenarios similar to [102], we can allow our algorithm to automatically find

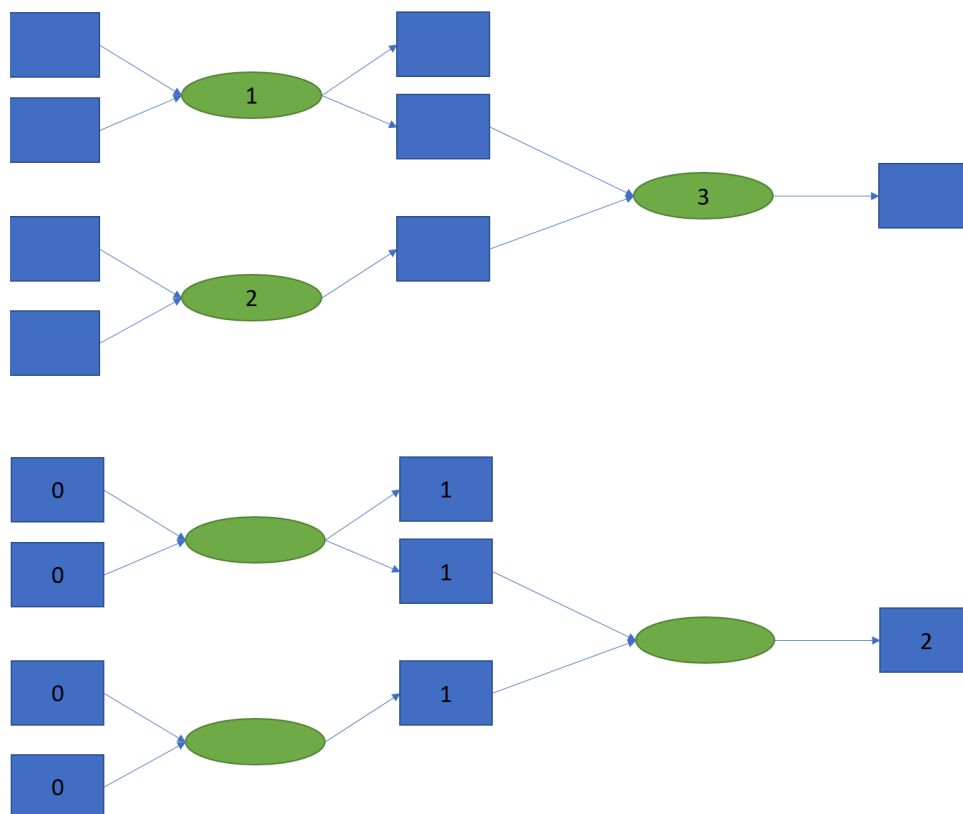


Figure 4.11.: For both scenarios, each operator is assumed to take one time unit execution time. The baseline calculation (top) for the cost of a planning node ignores the later parallelization during execution and just sums up the execution times of all applied operators. Our heterogeneous state allows us to implement a dedicated per-instance cost calculation scheme (bottom), which reflects a later parallelization chapter 9. There, we calculate the earliest time we can generate a specific instance. The cost of a planning state is then determined by the latest generated instance instead of the sum of all applied operators.

parallelized solutions by splitting up those operators into several smaller actions that occupy their acquired resources most of the time such that algorithmic access to this information is reflected in the model. An additional application of this parallelization scheme is a logistics task in which a dozen machines are handled in parallel. On the more abstract levels, we apply the actions in simulation in a strictly sequential order. However, we calculate the cost of the planning state not by the sum of the expected costs of all applied operators leading to this state but by the maximum of a *per-instance* cost.

This per-instance cost is calculated for the outputs of an operator by the sum of the most expensive input instance and the expected cost of the operator as depicted in Figure 4.11. Therefore, if the cost in an example domain is time, we add the expected execution time to the time when all input facts are available.

In this way, the costs we considered in planning reflect a parallel execution scheme that optimizes the sequential plan for subsequent parallelization as mentioned in subsection 3.2.2. Thus, even for parallel execution, we can consider continuous characteristics such as collisions or devi-

ations during execution since we use the same structures as for the sequential plan. An algorithm handles the parallel execution of the most refined level. It processes the sequential plan, extracts the preconditions for each step, and periodically checks whether new sub-tasks can be started. Two features of our model-based approach make this parallelization possible. First, our operators only use and manipulate a subset of the information of any state. State-of-the-art task and motion planning algorithms are opaque regarding the objects that are considered and manipulated for a given action that modifies the state. Therefore, dependencies cannot be recognized and considered during planning, similar to a resource-based parallelization [103]. We extend this scheme with our model that distinguishes between active and passive inputs. Active inputs only model the current environment during execution and are used for non-constructive operators such as collision avoidance. Instances for active inputs define the relative order between the operators and, therefore, represent a lock during parallelization. However, our parallelization algorithm composes passive inputs based on the current state defined by all available instances at a given time. The approach is discussed in more detail in chapter 9. There, we handled a robot cell with two independent arms and solved an assembly task with parallel strands of execution, for example. The robots receive tasks asynchronously, while a controller similar to low-level motion generation of [104, 105] ensures collision-free movements.

5

Conclusions

Previously, we proposed a formalism to describe declarative and arbitrary procedural knowledge within a single set-based model. Combined with the hierarchical planner, the means to introspect the planning results, and the pre- and post-processing algorithms this thesis describes a possibly game-changing framework for industrial automation that eases the application of multimodal systems to different domains.

For an autonomous, multimodal robotic system, experts from all engineering disciplines must come together to integrate their hardware and algorithms to create a meaningful system. This integration, in combination with a robust handling of disturbances during execution, is the core of robotics.

We observed that the engineering effort to compose the components for new use-cases is a bottleneck in the industry. Especially for systems with a higher degree of autonomy and thus greater resilience to external disturbances, programmed rule-based designs interweave modules with each other, reducing reusability and increasing complexity.

This results in labor-intensive integration for systems, which should eliminate time-consuming manual tasks. Hence, automation frequently shifts efforts from physical to cognitive work without any increase in efficiency.

We can now automate the work of engineers who previously automated the physical process, pushing the boundaries of automation and opening new fields of application with composable autonomous systems. We solve two problems simultaneously: Firstly, the system gets more flexible and robust compared to rule-based approaches through online planning, which can react to external disturbances without explicitly handling every corner case. When the system state deviates from the expected, planned situation, we can simply re-plan from the then valid state. Due to the hierarchical structure of the planned solution, only partial recalculations are necessary for minor deviations.

Secondly, modularization allows for a high degree of reusability, such that the experts of each domain can concentrate on their focus area rather than the later integration process, even for mul-

timodal systems. By that, we address both cornerstones of robotics, the robustness of execution and the integration from various sources.

The core enabler for this kind of system is the hierarchical planner. By factorizing the planning problem, even large-scale domains can be solved efficiently. Additionally, the hierarchization allows for easier debugging and interaction with a user.

The critical requirement for a universal planner is an expressive model that holds all required information, ranging from the most abstract semantic levels to continuous real-world execution. Without such a model, no generalized planner is possible. We presented such a model based on formal set theory. These models are flexible enough to describe every party's modules that contribute to a robotic system. We even incorporate different simulations in our models to validate the results of our operators. That way, we can handle completely different domains with the same algorithmic implementation of the solver and the same description language.

The formalization not only allows for online composition of the modules but also for an offline optimization to align the dialects of experts in different fields. This by-product of the formal models, which are inputs to the hierarchical planner, pushes the automation of the integration process to the next level. We cannot only adapt for different units but also adjust the structure of how information is stored to incorporate information from various sensor modalities.

A further benefit of the formal model is the possibility of asynchronous parallelization. For each component, we can differentiate between task-relevant and environmental information. By doing so, we do not add unnecessary temporal constraints during execution. Still, we can parallelize more tasks on close resources, such as collaborative robots, which can optimize their waiting times by solving tasks independently whenever possible.

With these results, we can answer the research questions stated in the introduction:

Research Question 1: Can we provide an automatic and flexible online decomposition of planning tasks to solve large planning problems?

Answer: Yes, our hierarchical planner can factorize arbitrary domains described by our models. We observed linear scaling in planning times for large assembly problems, for which the solution consisted of hundreds of steps.

Research Question 2: Can we integrate an MPC-like behavior into our planning algorithm so that deviations during planning or execution are detected by different sensors and handled automatically, such that hybrid problems are solved for real environments?

Answer: Yes, our planner natively incorporates an MPC-like behavior, which we applied to handle deviations of the real-world for a dual robot arm. We extended this concept and handled the two interfering robotic arms independently, so deviations occurred even without external uncertainty. The system could still operate reliably based on sensor information from force sensors, rotary encoders, and camera feedback.

Research Question 3: Can we define a modeling language that covers arbitrary discrete and continuous properties as well as actions, simulations, or executions from the discrete level to real-world execution for our generic planning algorithm?

Answer: Yes, our set-based modeling language can handle arbitrary black-box actions. We can automatically compute partial orderings between the pieces of declarative knowledge for our planning algorithm and annotate the required information on operations.

Research Question 4: Can we design a system that enables automatic optimization of the models and explainability for the developers?

Answer: Yes, we can reformulate the declarative and procedural knowledge based on our models. The factorization eases debugging and allows for a high degree of introspection and explainability.

A key advantage of our approach compared to upcoming, data-driven algorithms is computational frugality. Model-free approaches rely on pure luck to reach a goal at the beginning of the training. If there is no hint, such as a distance measure to the goal, but only a binary result about whether the goal is reached, the initial search phase might be too computationally expensive and prevent the application of such algorithms. We could integrate such algorithms and provide a *hot-start* in such cases, which, on the one hand, biases our solution. On the other hand, it could allow us to apply the system to a broader set of tasks. Compared to purely hierarchical planning, we could benefit from faster and better solutions due to the learning and optimization from previous runs.

II

Cumulative Part

6

Bridging the Gap Between Semantics and Control for Industry 4.0 and Autonomous Production

Title	Bridging the Gap Between Semantics and Control for Industry 4.0 and Autonomous Production
Authors	Bernd Kast , Sebastian Albrecht, Wendelin Feiten, and Jianwei Zhang
ISBN/ISSN	978-1-7281-0356-3/2161-8089
DOI	10.1109/COASE.2019.8843174
Status	published
Publisher	IEEE
Contribution of Bernd Kast	I developed the theory based on and refined by the discussions with the co-authors. I implemented the concepts described in this paper and carried out the experiments. The paper benefited from the discourse with the co-authors.

Summary	<p>Challenges in Industry 4.0, and for autonomous systems in general, are hybrid problems for which discrete and continuous properties come together. The first step towards automated planning systems, which enable autonomy and flexibility, is the modeling language to describe the problem at hand. Classical planning, with their respective PDDL-related modeling languages, describes discrete problems and allows us to solve them with various algorithms. The motion planning community traditionally targets continuous problems with probabilistic planning algorithms independently of the discrete properties. While progress towards a merger of both worlds has been made in recent years in task and motion planning, these approaches lack formal models for a generic problem description. We provide a formal model based on set-theory that can represent both discrete and continuous properties and embed arbitrary code so that collision checking is possible, for example. We motivate the modeling of both, declarative as well as procedural knowledge, the isomorphism between planning and execution, and the hierarchization and factorization of the planning problem for scalable planning. An example of an autonomous assembly task that uses a hierarchical planner, proposed in a second paper, concludes the paper.</p>
---------	---

Bridging the Gap Between Semantics and Control for Industry 4.0 and Autonomous Production

Bernd Kast¹, Sebastian Albrecht¹, Wendelin Feiten¹ and Jianwei Zhang²

Abstract— Small lot size production requires decoupled specification of the product and the production system to allow for flexible manufacturing, which leads to autonomous systems. Their key are algorithms, that are able to sequence and parametrize skills in a meaningful way such that a given task is fulfilled. This planning process requires suitable models of all relevant aspects of the production which includes the system's state and estimates of its behavior.

In this paper we propose formal models for declarative and procedural knowledge, which unify the symbolic and sub-symbolic representations. Currently, the curse of dimensionality prevents state of the art techniques to solve non-trivial real-world tasks that pose mixed discrete and continuous states. Our models are designed hierarchically from the very beginning and thus allow an automatic decomposition of planning problems. This allows us to compute solutions online for complex tasks that are only defined by the goal configuration of the product.

We validate our approach with an assembly use-case that illustrates end-to-end autonomous robotic production in a real-world setting. The separate definition of product and hardware, as well as the number of steps required to fulfill this task demands a multilayered planning and execution for online control.

I. INTRODUCTION

Today's production lines are specifically designed for a limited and known set of tasks. The design processes of the parts to be produced and the production systems interleave to ensure feasibility and ideally robustness as well as performance [9]. Engineering efforts are enormous and often even small changes in the product or the production hardware require re-engineering of the overall process. For economic reasons, this prohibits production systems for small lot sizes, which come with a high variance in tasks, product properties or hardware setups [7], [14], [24]. A possible solution to that are autonomous systems, that increase flexibility by an automatic calculation of an effective action sequence. This would require only little or even no engineering effort for new tasks due to the increased autonomy.

However, in order to keep engineering efforts manageable, those algorithms require suitable models for their computations, that describe each task and component of the production system independently. These pieces of information should then be brought together by planners, that calculate a meaningful sequence of actions online. Key to such a successful planning system are expressive models that describe

¹Siemens AG, Corporate Technology, Otto-Hahn-Ring 6, 81739 Munich, Germany. bernd.kast@siemens.com

²University of Hamburg, Faculty of Mathematics, Informatics and Natural Sciences, Vogt-Kölln-Str. 30, 22527 Hamburg, Germany.

The presented research is financed by the TransFit project which is funded by the German Federal Ministry of Economics and Technology (BMWi), grant no. 50RA1701, 50RA1702, and 50RA1703.

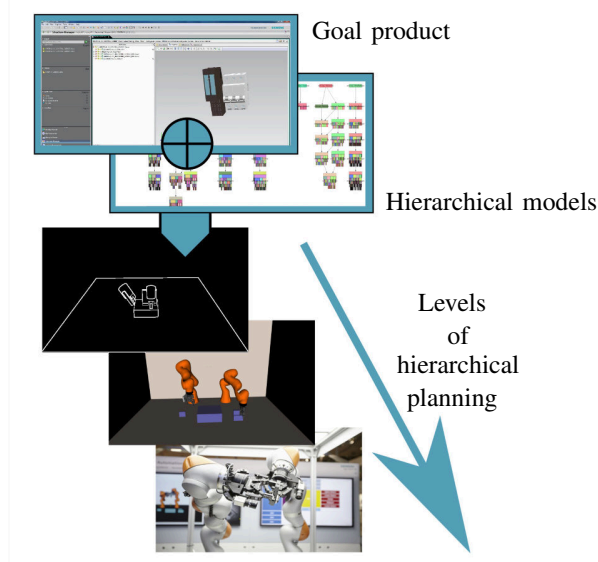


Fig. 1: The goal product and hierarchical models are the basis hierarchical planning. Symbolic plans can be refined to sub-symbolic properties and executed on real hardware.

the available pieces of information (concepts) and possible actions (operators). A deep digitalization of the production process and its components is the corner stone of Industry 4.0 and indispensable for autonomy.

The models must cover the overall system with all its components even those that interface with the real-world. This includes simulations, the actual hardware, as well as algorithms and databases. As the modeling and planning algorithms are part of that system, the models must provide the expressiveness to describe themselves and algorithms that work on them.

Throughout this paper we consider an assembly task, that uses a robot to click a module on a top-hat rail, as an example. On the symbolic level a plan might be the sequence of the actions: *grasp*, *move* and *click*. This determines the objects that are manipulated and the operations on them. However, this plan is not executable on the real hardware if there is no collision-free grasp for the object, that also allows for a collision-free assembling on the rail. Collisions can hardly be represented on this symbolic level, though. This example illustrates, that, in general, collisions and motion trajectories for the robot cannot be determined on that level of abstraction. We need models that can handle sub-symbolic

information and suitable planning algorithms that make use of them.

Planning tasks that involve symbolic and sub-symbolic properties pose a huge complexity. This prevents the direct calculation of the problem's solution within the strict time requirements of online control. A common approach is to simplify the task by a factorization of the problem. Our contribution are models that allow an autonomous reasoning about this factorization based on the solutions of abstracted levels. The specific number of abstractions, their content and the respective transitions are calculated automatically, as pre-defining these levels would generate undesirable interdependencies between models. This would prevent an automatic composition to domains and therefore limit the flexibility and application scenarios. The models have to provide all information to allow for this reasoning.

However, this can only be achieved if the same principles are used to describe information and actions, as they are strongly interconnected and define the levels of abstraction only when considered together.

In this paper we propose new formal and intrinsically hierarchical models that unify the declarative and procedural knowledge for both, the symbolic and sub-symbolic world. They are expressive enough to describe arbitrary calculations like simulations, algorithms, that make use of them or directly control real hardware.

We leverage the strengths of different modeling and planning techniques rather than to replace them. Our models allow to interface these well-established methods and enable the needed information exchanges. Despite the undeniable linkage between models and planning algorithms, we only focus on the models in this paper. To demonstrate their benefits, we use them in an assembly scenario. and breadth first forward searches.

II. RELATED WORK

Due to its unifying character, this paper touches numerous strands within the scientific community of robotics and autonomous systems. In the following, we present a small selection of those approaches. In subsection II-A we describe concepts, that introduce manually designed abstractions of arbitrary components of production plants. They ease adaptations to changing tasks but require a manual sequencing as they neither allow for planning nor reasoning of hierarchies. However, they lift the continuous properties to a symbolic level, that enables an enrichment with models discussed in subsection II-B such that they can be used with planners presented in subsection II-C. The first flaw of this setup is the manually defined hierarchy, with the fixed number of layers and the gap between the symbolic and sub-symbolic level. This is discussed in subsection II-D along with the second problem which is the coupling of the models to a specific problem class. In subsection II-E approaches are presented that tackle the end-to-end problem while pertaining the separation of task and plant descriptions. Those approaches, however, don't offer a hierarchical decomposition of the problem and therefore can't scale to larger problems.

A. Industry 4.0, Model-predictive Control and Skills

Industry 4.0, the next step of automation, provides the so-called digital twin. It offers high-fidelity models that describe properties, states, processes as well as the system's dynamics [19], [36], [46].

Advanced robotics is not viable without such reliable and desirably probabilistic models as algorithms, that account for deviations in measurement and actuation, rely them [44]. Those algorithms are required to tackle the inevitable difference between simulation and real-world, as they react appropriately on small deviations during execution [10], [41].

Given those robust algorithms, hand-crafted hierarchy levels for the capabilities of machines can be designed, that lead to the term *skill* [6], [22]. Such skills encapsulate certain aspects of the hardware and therefore enable engineers to easily combine and compose components for new tasks [25]. Concepts that make use of such skills range from behavior trees [30] to advanced hierarchically compositions [8].

However, they still rely on manual work, as they lack the information, like a simulated outcome, needed for planning.

B. Models for Reasoning and Planning

The plan generation of autonomous systems is based on suitable models that, describe a calculated estimate of all task-relevant aspects. Therefore, a formalization of the required knowledge [21], [37] is necessary, which combines the declarative and procedural information [1], [3], [18]. A special focus has to be put on procedural models of different abstractions, which can provide calculated estimates of real-world processes with various accuracy. The respective declarative representations of their in- and output can range from pure symbolic to (mixed) sub-symbolic information. On the most detailed level, skills, discussed in subsection II-A, can be used.

A lot of research has been done regarding ontologies and taxonomies (such as RDF and OWL). However, only few papers are proposed for a common description of declarative and procedural knowledge [35], [38]. Autonomous production poses further requirements. This includes flexible expandability as well as evaluation of black-box code in order to make use of high-fidelity simulations. Additionally, hierarchies for both, declarative and procedural knowledge, are required to overcome the curse of dimensionality. To the best of our knowledge there exists no approach in the context of robotics, that covers all those aspects.

C. (Hierarchical) Symbolic Planning

Many modeling languages formalize planning problems on a symbolic level, e.g. PDDL and its variants [16], [31]. They focus on procedural knowledge of a domain which is complementary to most ontologies that mainly describe the declarative structures. Note that composite objects with complex structures modeled in an ontology cannot easily be expressed in PDDL. Several planners for suitable subclasses of PDDL problems have been proposed, e.g. [12], [23], [34]. However, somewhat realistic setups require to handle a lot of sub-symbolic quantities with nonlinear effects.

The resulting problem complexity renders online application of these planners for robotic control impossible.

Hierarchies are a widely used technique to increase scalability. For instance, partial-order planning (POP) [48] handles partially specified action decompositions, in which the planner fills in the missing pieces. Another example is the huge class of hierarchical task networks (HTN) which refine each abstract method by a network of sub-methods, e.g. [13], [20], [33]. An overview of different HTN methods can be found in [5]. Both methods require some prior knowledge of the task to model the domain, and don't allow for an arbitrary number of black-box calculations or simulations which hugely limits the flexibility.

Closely related to that problem is, that many planners rely on the downward refinement property: it guarantees that an abstract solution can be refined to a primitive one [2]. An example of such an algorithm is presented in [29] where even abstract tasks possess preconditions and effects. Their underlying semantics ensures that an abstract plan can be refined to a primitive solution. Several variants of HTN planning and hybrid planning [4], [13], [32], [40] have been proposed. In general, the performance gain of HTN methods comes at the cost of flexibility. In most cases the later application has to be actively considered during engineering of HTN actions as the (purely symbolic) calculations on the abstract level have to provide a definite and correct answer to the outcome of the real-world.

D. Combined Task and Motion Planning in Robotics

In recent years the combination of task planning and motion planning has gained a lot of interest in the robotics community. By combining existing task planners with motion planners [11], [42] collision-free paths and some geometric constraints can be handled. Others lift the sub-symbolic world to the symbolic planning [17], [28] by extending the heuristics of the symbolic planners or considering unbound variables. Disadvantages of those approaches include scaling issues and a discretization that has to be specified in advance. On the other hand, the symbolic properties can also be added to the sub-symbolic path planning problem that determines the best intermediate and final states [45].

Most of the former approaches consider almost exclusively manipulation problems and neglect robot dynamics and/or temporal constraints. The ScottyActivity planner [15] considers problems, that include such properties. It uses methods of convex optimization in combination with relaxed plan graphs and suitable heuristics for searching. However, the applicability is limited by fundamental assumptions, like the absence of obstacles and linear dynamics for the robots. Both, nonlinear dynamics and collisions are considered by [39] proposing an asymptotically optimal manipulation planner. Their approach extends sampling-based roadmap planners to efficiently explore configuration spaces.

E. Autonomous Production Systems

The total problem of autonomous robotics in production is to find and execute a sequence of actions based only on a

description of the hardware and a CAD model of the desired product. All tasks, such as perception, arm movements or gripper actuations need to be planned for multiple pieces avoiding the curse of dimensionality.

The pioneering system [27] is one of the few who tried to solve this hard problem. Due to the technological limitations at that time, the user has to provide substantial input, for example recommended sub-assemblies or assembly paths. The approach just generates a nominal control code, not considering sensors and is tested only in simulation.

In [43] a more evolved version is presented that, among other models, uses CAD data and the desired goal configuration as an input. After computing assembly sequences, the optimal plan is selected and mapped to the robot's skills which are then executed on the hardware. However, as there is no reasoned hierarchy, this approach suffers from the curse of dimensionality.

III. HIERARCHICAL KNOWLEDGE MODELING

The focus of this paper are formal models that enable hierarchical planning. In this section, we introduce the formal structures for *concepts*, i.e., the declarative models, that describe pieces of information, and for *operators*, i.e., the procedural knowledge models which characterize actions. These models fulfill the two basic requirements of hierarchical planning and bridge the symbolic and sub-symbolic worlds:

First, our approach allows to use the same structures for all concepts, independently of whether they are fully symbolic, fully sub-symbolic or hybrid. This solves the hardest part, which is the transition between the different worlds.

Second, the operators can be defined as black boxes which are evaluated online during planning. This enables for example the integration of collision checks, advanced simulations and a seamless transition to the real-world. By that, our formal models standardize the interfaces for all aspects of autonomous systems.

A. Concepts

A concept captures the declarative knowledge about an object. It has inherently a specific level of abstraction when compared to another concept. For example, consider the working surface of an assembly unit. On an abstract level, it suffices to have a unique identifier for each surface in the workspace to answer whether or not a placement is possible. On more detailed levels, other properties like dimension, poses or even friction coefficients become relevant. Each combination of those properties specifies a specific concept C and the set of all concepts for working surfaces results in a concept class Γ .

These ideas can be grounded on set-theoretic definitions which are also depicted in Figure 2.

Definition 1:

- A concept base B_Γ is the set of instances, which is not necessary finite.

- A concept C is a subset of B_Γ , i.e. $C \subseteq B_\Gamma$.
- A concept class Γ is the set of concepts C_i that have a common concept base B_Γ , i.e. for all $C_i \in \Gamma, b \in C_i : b \in B_\Gamma$.
- A partial order \mathcal{M} , that describes *more detailed than*, can be defined on Γ : $(C_i, C_j) \in \mathcal{M}$ iff $C_i \subset C_j$.

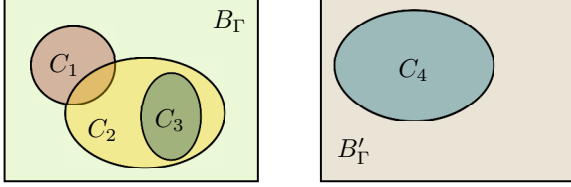


Fig. 2: Illustration of set-theoretic definitions of concepts. The three concepts C_1, C_2, C_3 are subsets of a common concept base B_Γ and thus form a concept class Γ . Concept C_4 has another concept base and concept class. Concept C_3 is more detailed than C_2 . For the other concept pairs, no such relation is true.

In this sense all concepts are sets. However, they sometimes have more structure than these non-composite concepts. Especially when humans construct and use them. In many cases it is reasonable to denote the meaning of each value of an instance. Therefore, we introduce roles r and composite concepts in the following:

Definition 2:

- A role $r \in \mathcal{R}$ is a unique identifier, e.g. a string or integer, where \mathcal{R} contains all possible roles.
- The set of all roles is assumed to be ordered, thus for each subset $\mathcal{R}_i \subseteq \mathcal{R}$ with $n_{\mathcal{R}_i} := |\mathcal{R}_i|$ a bijective mapping $\mathcal{I} \rightarrow \mathbb{N}_{n_{\mathcal{R}_i}} := \{1, \dots, n_{\mathcal{R}_i}\}$, i.e. both mappings $\mathcal{I}(r) \in \mathbb{N}_{n_{\mathcal{R}_i}} \forall r \in \mathcal{R}_i$ and $\mathcal{I}^{-1}(n) \in \mathcal{R}_i \forall n \in \mathbb{N}_{n_{\mathcal{R}_i}}$ exist.
- A composite concept $C = \prod_{r \in \mathcal{R}_C} C_r$ with \mathcal{R}_C is the specific set of roles for this composite concept.

Note that this composition structure of a concept defines a (directed) graph, see figure 3. Each node corresponds to a concept structure. Edges contain the role and point from the composite concept to a sub-concept. The composite concept structure from above can easily be extended to both, ordered, finite lists and unordered finite sets, by enhancing the roles by semantic information: $r[\]$ or $r\{\}$.

For the non-composite concepts, we already introduced a partial order, that describes which concept is more detailed. This notion can directly be generalized to composite concepts:

Corollary 2.1: Given two composite concepts of one concept class $C, \bar{C} \in \Gamma$. Then the partial order \mathcal{M} can be defined recursively by:

$$(C, \bar{C}) \in \mathcal{M} \text{ iff } (C_r, \bar{C}_r) \in \mathcal{M} \forall r \in \mathcal{R}_{\bar{C}}.$$

Thereby C is isomorph to a subset of \bar{C} .

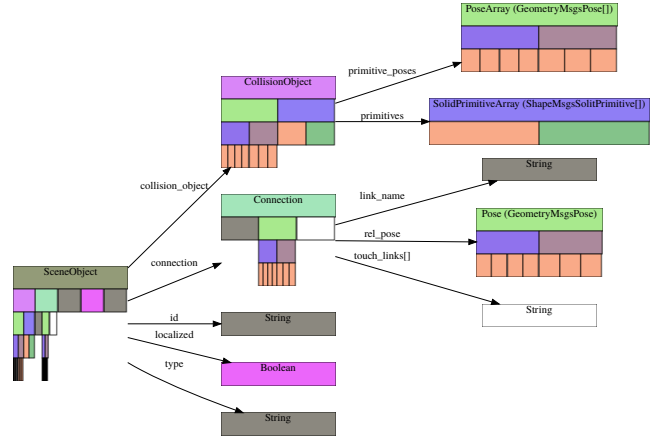


Fig. 3: Example of a composite concept with composite sub-concepts. An individual color is randomly picked for each concept. The images illustrate the tree-structures which are used for composite concepts.

Proof: The definition of the partial order requires that C has all the roles of \bar{C} . Rearranging the indices of the roles, an isomorph concept is obtained:

$$C \cong \prod_{r \in \mathcal{R}_C \cap \mathcal{R}_{\bar{C}}} C_r \times \prod_{r \in \mathcal{R}_C \setminus \mathcal{R}_{\bar{C}}} C_r := C'.$$

The recursive definition of \mathcal{M} then yields:

$$\prod_{r \in \mathcal{R}_C \cap \mathcal{R}_{\bar{C}}} C_r \subseteq \bar{C}.$$

Therefore:

$$C' \subseteq \bar{C} \times \prod_{r \in \mathcal{R}_C \setminus \mathcal{R}_{\bar{C}}} B_{\Gamma(r)} \cong \bar{C}. \quad \blacksquare$$

An interesting aspect is the similarity of instances, i.e. when do two instances of the same concept represent the same thing. This is especially important due to measurement noise, numeric accuracies and symmetries. For example, each sensor measurement varies slightly and each number in a computer comes with a rounding error.

We suppose that each instance represents a set, i.e. all $b_i \in B_\Gamma$ correspond to sets. Then a relation, that describes the similarity, can be based on the intersection $b_i \cap b_j$. If this intersection is non-empty, the instances are assumed to be similar.

Note that in this way similarity is symmetric, but not necessarily transitive. If transitivity is mandatory, the instances x_i have to represent equivalence classes, i.e. a partition of B_Γ is needed. Depending on the use case, transitivity is in some cases strictly required and in others not even possible. Therefore, for each concept C a relation, the so-called *compare function*, is introduced: $F_C : C \times C \rightarrow \mathbb{B}$.

For a composite concept the definition of a valid compare function is straight forward due to the product structure: Two composite concepts can be defined to be similar if all their corresponding leaves are similar.

B. Operators

Declarative knowledge is only useful if elements of the procedural knowledge, so-called *operators*, use them. An example is depicted in Figure 4. The following definition introduces models for operators and subsequently additional assumptions are stated:

Definition 3:

- An operator π describes changes in the declarative knowledge. Therefore, it is a mapping of given input concepts I_{r_i} to output concepts O_{r_j} with input roles $r_i \in \mathcal{R}_{\pi,I}$ and output roles $r_j \in \mathcal{R}_{\pi,O}$, i.e., $\pi : \prod_{r_i \in \mathcal{R}_{\pi,I}} I_{r_i} \rightarrow \prod_{r_j \in \mathcal{R}_{\pi,O}} O_{r_j}$.
- Certain operators describe modifications of instances, opposed to just generating new instances. This means that pieces of knowledge are invalidated as soon as such an operator is executed. Those inputs are called *consumed*. The set of roles corresponding to inputs that are consumed by the operator is denoted by $\mathcal{R}_{\pi}^c \subseteq \mathcal{R}_{\pi,I}$.

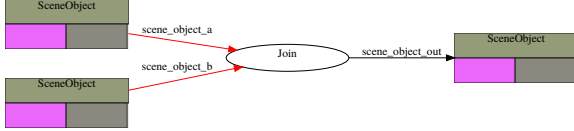


Fig. 4: Example of an operator with multiple inputs and outputs. Consumed inputs are depicted with red arrows. Inputs and outputs may be instances of the same set and are thus depicted by the same concept.

Assumption 3.1:

- There is no limitation on how output instances are computed, given the input instances. This can be specified explicitly by symbolically representable mappings (mathematical formulas) or implicitly as the result of some experiment in simulation or real-world. Therefore, even sampling-based planners or random processes are possible.
- Operators are fully functional: they do not have an internal state and they cannot return any input, only a copy of it.

Naturally, operators are elements of a functional space. Thus, they can also be modeled as instances of an operator-concept on a higher level of abstraction that is often called *meta level*. In addition to the information described above, the operator-concept can contain *meta information*, such as performance indicators (KPI) and entropy characteristics.

The available structures for the operator allow to deduce hierarchies of operators from the hierarchical structures of concepts. These hierarchies represent different abstractions of simulators. The most detailed operator is the execution in real-world. These abstractions allow for a usage in planning algorithms.

Corollary 3.1: An operator π_1 is more detailed than another π_2 if the following conditions hold true:

- all input and output roles of operator π_2 are elements of the role set of operator π_1 , i.e. $\mathcal{R}_{\pi_2,I} \subset \mathcal{R}_{\pi_1,I}$ and $\mathcal{R}_{\pi_2,O} \subset \mathcal{R}_{\pi_1,O}$,
- all (common) inputs $I_{r,1}$ and outputs $O_{r,1}$ of operator π_1 are more detailed than those of operator π_2 , i.e. $(I_{r,1}, I_{r,2}) \in \mathcal{M}$ for all input roles $r \in \mathcal{R}_{\pi_2,I}$ and $(O_{r,1}, O_{r,2}) \in \mathcal{M}$ for all output roles $r \in \mathcal{R}_{\pi_2,O}$,
- the key performance indicators and the entropy characteristics of operator π_1 are more detailed than those of operator π_2 .

Summing up, we now introduced formal models for declarative and procedural knowledge based on set theory. These models are designed to capture different abstraction levels, which allow easy algorithmic generation of hierarchies. In particular, these models are able to bridge the gap between the symbolic and sub-symbolic world. However, they are only useful for an autonomous production unit if a planner can use them to automatically generate valid action plans. Therefore, we will now sketch how we represent and find plans while we avoid combinatorial explosion in many cases.

IV. CONCEPTS FOR PLANNING

The model hierarchies for concepts and operators were introduced to enable hierarchical planning. The goal of the following sections is to show that these structures can actually be used to plan and control an autonomous assembly system. We only sketch the general hierarchical planning approach, as the in-depth discussion is beyond the scope of this paper and can be found in [26].

Planning is an evaluation of operators on the set of currently available instances in order to reach one or multiple goal instances. In our approach, execution is no different from any other calculation, but just a very good simulation of the consequence of actions. The operators can use or consume available instances and generate new instances. Consuming instances happens frequently when manipulating real-world objects: A box that is moved to another place can no longer be picked at the previous location. On the other hand, some information, like the size of the object, can be used multiple times and is therefore used but not consumed.

To discuss the relevant elements during planning, we first introduce the concepts of a *planning task*, a *plan family* and a *plan*, which are all concepts in the meta domain:

1) *Planning Task:* A planning task $PT(X_{init}, X_{goal}, X_{op})$ specifies the combination of three sets: a set of initial instances $X_{init} := \{b_i, i = 1, \dots, n_{init}\}$, a set of goal instances $X_{goal} = \{b_j, j = 1, \dots, n_{goal}\}$ and a set of available operators $X_{op} := \{\pi_k, k = 1, \dots, n_{op}\}$ with $n_{init} \in \mathbb{N}_0$ and $n_{goal}, n_{op} \in \mathbb{N}$.

2) *Plan Family:* We introduce the concept of a *plan family* \mathcal{G}_{PF} , that contains the initial facts and connects operators with their input and output instances. Consequently, the plan family \mathcal{G}_{PF} is an extended bi-partite graph (V, E, E^C) where the first node set V_B contains nodes representing instances only and the second set V_π consists solely of operators. The extension of the graph means that there exists a special subset

of edges: the set of edges $E^C \subseteq E$ which denotes edges corresponding to consumed operator inputs.

3) *Plan*: A plan is a special plan family in which each fact can only be consumed once. Additionally, a plan has to contain a (non-consumed) node representing an instance that is more detailed than an element of the goal set X_{goal} . This means $\exists v \in V_B, v_g \in X_{\text{goal}} : (v, v_g) \in \mathcal{M}$.

V. PLANNING FOR AUTONOMOUS SYSTEMS

Planning considers sets of instances X_i , $i \in \mathbb{N}$. Each set of instances X_i is obtained by the execution of one operator $\pi_k \in X_{\text{op}}$ on a selection of instances $(b_1, \dots, b_{n_{\pi_k}})$ with length $n_{\pi_k} := |\mathcal{R}_{\pi_k}|$. Those instances are elements of a previous set of instances X_j , $j \in \{0, \dots, j-1\}$ (with $X_0 := X_{\text{init}}$):

$$X_i = (X_j \cup \pi_k(b_1, \dots, b_{n_{\pi_k}})) \setminus \{b_{\hat{n}} \mid \mathcal{I}^{-1}(\hat{n}) \in \mathcal{R}_{\pi_k}^c\},$$

in which the instances must be an element of the correct concept $b_{\mathcal{I}(r_i)} \in C_{r_i} \cap X_j$.

The set of currently available instances X_i is the representation of our state. Since the information regarding input and output instances is stored in the plan family, the sets X_i correspond to a subset of nodes V . Considering a set of instances X_i that contains a all goal instances, i.e. $\forall b_g \in X_{\text{goal}} : \exists b \in X_i$ with $(b, b_g) \in \mathcal{M}$, the corresponding plan is the sub-graph of the plan family that has the following properties:

- all leaf nodes correspond to instances in X_i ,
- all root nodes correspond to instances in X_{init} or are a node from V_{π} ,
- each node can have at most one outgoing edge that is within the consumed set.

Hierarchical planning deals with the curse of dimensionality in a divide and conquer fashion. This means, a plan is determined on an abstract level of the hierarchy first and then this plan is refined step by step to the most detailed level, which is the execution on the real hardware (cf. figure 1). Thus, the hierarchical planning is a meta-planning approach, in which for each step individual algorithms are called. As mentioned above, two of those planning types are classical planning and task and motion planning. While they cannot handle the total problem on their own, they are still suitable to solve sub-tasks. Opposed to previous approaches, we are not limited to exactly two layers and can handle multiple layers with hybrid states. Additionally, we allow backtracking and therefore don't rely on the downward refinement property. Nevertheless, our performance only benefits if it holds often.

In general, the reasoning of suitable hierarchy levels and sub-planning domains for a given planning task and the search of according refinement strategies are hard optimization problems on their own and therefore topics for future research. However, for specific application examples, the automatic computation of relevant hierarchies is straight forward and already implemented (see for example next section).

The basic idea of the hierarchical planning is, that each step of a course plan defines a new planning task, that recursively be solved until the real hardware is actuated. Due to our models, this refinement can be determined automatically and online. In more detail, the execution of a given operator $\pi : \prod_{r_i \in \mathcal{R}_{\pi, I}} I_{r_i} \rightarrow \prod_{r_j \in \mathcal{R}_{\pi, O}} O_{r_j}$, which is a single step in the plan, leads to the following mapping of instances:

$$\{b_r \mid r \in \mathcal{R}_{\pi, I}\} \mapsto \{\bar{b}_r \mid r \in \mathcal{R}_{\pi, O}\}.$$

The refinement of this planning step defines the goal set of new planning task $PT(X'_{\text{init}}, X'_{\text{goal}}, X'_{\text{op}})$, as the set of operator outputs $X'_{\text{goal}} := \{\bar{b}_r \mid r \in \mathcal{R}_{\pi, O}\}$. The operator set X'_{op} results from the automatically calculated hierarchy level and the initial facts X'_{init} are determined by a subset of the available facts of the most refined previous plan step. This means, all instances of the input concepts, but the ones actually used in the coarse step, are removed from that set to restrict the search space. The principle idea of this algorithm is depicted in Figure 5.

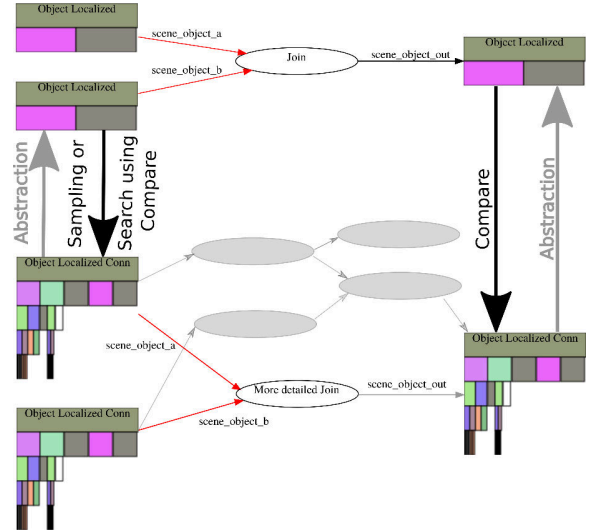


Fig. 5: Basic planning idea: An easier planning task can be obtained by abstracting the original instances and goals (arrows pointing upwards). Each step in the coarse plan defines a new task in the refined level. Either a refined operator exists (white ellipse) or a plan has to be found (gray ellipses). Facts and goals are determined using the compare function as their abstractions need to comply with the instances of the course plan.

Multiple realizations of this general hierarchical planning idea are possible. We use a planner that supports backtracking and uses breadth first search. Therefore, we don't rely on the downward refinement property, while it helps to fight the curse of dimensionality. As the downward refinement property holds often, we observe a linear scaling of the planning times in the number of modules for this example.

Note that this hierarchical planning method finally closes the former gap between the symbolic planning and the sub-symbolic planning.

VI. APPLICATION EXAMPLE: AN AUTONOMOUS MANUFACTURING SYSTEM

An assembly task is chosen as an example to demonstrate the capabilities of the proposed hierarchical modeling and planning framework: four modules (one type of such a module is shown in figure 6) must be assembled onto a top-hat rail which is a common part in control cabinets. The product specification uses fully-symbolic concepts of these modules which are completely hardware-independent. The precise poses of all components are unknown in the beginning and have to be determined by suitable perception steps.

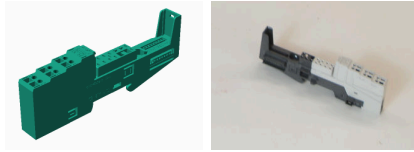


Fig. 6: CAD model and image of one of the modules to be assembled on a top-hat rail.

The autonomous system is a two-arm-robot setup shown in the bottom picture of figure 1. Both arms are equipped with a parallel gripper and a wrist-mounted camera to manipulate and localize parts.

Depending on the actual pose of a component, the part can either be picked and mounted by a single arm or a handover to the other manipulator is necessary. This may happen when all grasping positions would result in mounting motions which are not collision-free. Confer [39] for details on low-level task and motion planning, which selects grasps and calculates collision free robot arm motions in order to fulfill the overall task. The actual joining movement of the component onto the top-hat rail is a difficult process because of the presence of measurement uncertainties. It is solved by actively considering these uncertainties and using suitable planning techniques [47].

The provided operators on the most detailed level include:

- perceive: given a coarse initial guess, a precise 6D pose of the object is computed.
- mount: given the precise 6D pose of a component and the rail, the autonomous system grasps this object and clicks it on the hat-rail.
- push: the precise 6D pose of the rail with more than one mounted component provided, the parts are pushed together to connect them.

The hierarchical structure of the domain is directly visible in the concept hierarchy of objects. Figure 7 shows 5 concepts that specify an object on different abstraction levels. All of them include an object id. The more detailed concepts that are still purely discrete additionally describe the manipulation and localization state. The most detailed concepts include geometry and position information. The overall task input is modeled with instances that correspond to the lower left concept. The task goal is an element of the concept in the top row. In our domain, however, there

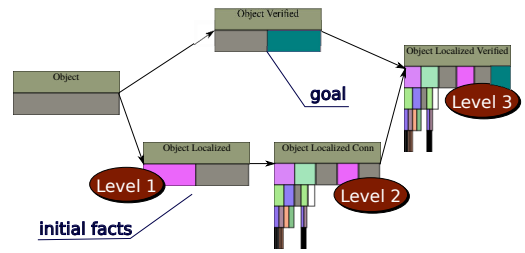


Fig. 7: The concept hierarchy of manipulation objects. Together with concepts of the initial and goal instances it defines the planning hierarchy in this example.

exists no operator or sequence of operators such that an instance of the goal concept is obtained. Since an instance of a more-detailed concept is in particular an instance of the more-abstract one, the goal of hierarchical planning framework is to obtain an instance of the more-detailed concept that fulfills all properties of the initially-specified goal. The domain properties allow to algorithmically select one additional planning level between the initial input level and the goal level: a level corresponding to the lower right concept. The operators for each level are determined using backward chaining.

In our example, the different levels of hierarchical planning correspond to the following abstractions: The planner on the first planning level checks only if sampled start and goal poses exist, such that no collisions occur. On the second planning level, coarse trajectories that are throughout collision-free are computed. The final level combines detailed planning with the actual execution: the trajectories are further optimized, and the hardware is controlled actuated. In Figure 1 these different levels of the planning process are depicted. Actions, which result from a combination of the cell and the task, like handover procedures, are automatically computed online. Confer [47], [39] for more details. Note that those layers are automatically determined by the operators available for this task and are not defined in advance.

Note that the formal knowledge models and the hierarchical planning approach allowed to specify the goal on an abstract symbolic level independent of the actual production hardware. Additionally, the operators of the production system and their sequences are not tailored to the application - opposed to current automation strategies. Furthermore, we are able to calculate solutions online with linear time dependency in the number of assembled modules, without any task specific modeling. Thus, the autonomous production system combined with hierarchical planning allowed for an end-to-end control with decoupled descriptions for task and hardware.

VII. CONCLUSION

The proposed formal models for concepts and operators allow a seamless transition between the symbolic and sub-symbolic levels. This enables new applications for relevant tasks in Industry 4.0, like the proposed assembly of top-

hat rails with multiple components. Building up on the hierarchical structures of the formal models, we sketched a hierarchical planning strategy that avoids the curse of dimensionality for reasonably modeled domains. At the same time, the separation of plant and model description avoids manual work for changing tasks and ensures flexibility.

All proposed elements are already structured such that a direct extension to other research fields like self-X and co-simulation is possible. Analyzing the interplay of these approaches is planned for future research.

REFERENCES

- [1] K.R. Apt, H.A. Blair, and A. Walker. Towards a theory of declarative knowledge. In *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Elsevier, 1988.
- [2] F. Bacchus and Q. Yang. Downward refinement and the efficiency of hierarchical problem solving. *Artificial Intelligence*, 71(1):43–100, 1994.
- [3] C. Baral. *Knowledge representation, reasoning and declarative problem solving*. Cambridge University Press, 2003.
- [4] P. Bechon, M. Barbier, G. Infantes, C. Lesire, and V. Vidal. Hipop: Hierarchical partial-order planning. In *STAIRS*, pages 51–60, 2014.
- [5] P. Bercher, D. Höller, G. Behnke, and S. Biundo. More than a name? on implications of preconditions and effects of compound htn planning tasks. In *ECAI*, pages 225–233, 2016.
- [6] A. Billard, S. Calinon, R. Dillmann, and S. Schaal. Robot programming by demonstration. In *Springer Handbook of Robotics*, pages 1371–1394. Springer, 2008.
- [7] E.K. Bish, A. Muriel, and S. Biller. Managing flexible capacity in a make-to-order environment. *Management Science*, 51(2):167–180, 2005.
- [8] S.G. Brunner, F. Steinmetz, R. Belder, and A. Dömel. Rafcon: A graphical tool for engineering complex, robotic tasks. In *IROS*, pages 3283–3290. IEEE, 2016.
- [9] A.M. Bryan, J. Ko, S.J. Hu, and Y. Koren. Co-evolution of product families and assembly systems. *CIRP Annals*, 56(1):41–44, 2007.
- [10] E.F. Camacho and C.B. Alba. *Model predictive control*. Springer, 2013.
- [11] S. Cambon, R. Alami, and F. Gravot. A hybrid approach to intricate motion, manipulation and task planning. *Robotics Research*, 28(1):104–126, 2009.
- [12] M. Cashmore, M. Fox, D. Long, and D. Magazzeni. A compilation of the full pddl+ language into smt. In *AAAI Workshop: Planning for Hybrid Systems*, 2016.
- [13] L.A. Castillo, J. Fernández-Olivares, O. Garcia-Perez, and F. Palao. Efficiently handling temporal knowledge in an htn planner. In *ICAPS*, pages 63–72, 2006.
- [14] N.F. Edmondson and A.H. Redford. Generic flexible assembly system design. *Assembly Automation*, 22(2):139–152, 2002.
- [15] E. Fernandez-Gonzalez, B. Williams, and E. Karpas. Scottyactivity: Mixed discrete-continuous planning with convex optimization. *Artificial Intelligence Research*, 62:579–664, 2018.
- [16] M. Fox and D. Long. Pddl+: Modeling continuous time dependent effects. In *Int. NASA Workshop on Planning and Scheduling for Space*, volume 4, page 34, 2002.
- [17] C.R. Garrett, T. Lozano-Pérez, and L.P. Kaelbling. Ffrob: An efficient heuristic for task and motion planning. In *Algorithmic Foundations of Robotics XI*, pages 179–195. Springer, 2015.
- [18] M.P. Georgeff and A.L. Lansky. Procedural knowledge. *Proc. of the IEEE, Special Issue on Knowledge Representation*, 74(10):1383–1398, 1986.
- [19] E. Glaessgen and D. Stargel. The digital twin paradigm for future nasa and us air force vehicles. In *AIAA Structures*, page 1818, 2012.
- [20] R.P. Goldman. Durative planning in htns. In *ICAPS*, pages 382–385, 2006.
- [21] N. Guarino. Formal ontology, conceptual analysis and knowledge representation. *Human-Computer Studies*, 43(5-6):625–640, 1995.
- [22] T. Hasegawa, T. Suehiro, and K. Takase. A model-based manipulation system with skill-based execution. *Transactions on Robotics and Automation*, 8(5):535–544, 1992.
- [23] M. Helmert. The fast downward planning system. *Artificial Intelligence Research*, 26:191–246, 2006.
- [24] K. Hitomi. *Manufacturing Systems Engineering: A Unified Approach to Manufacturing Technology, Production Management and Industrial Economics*. Routledge, 2017.
- [25] S.J. Hu, J. Ko, L. Weyand, H.A. ElMaraghy, T.K. Lien, Y. Koren, H. Bley, G. Chryssolouris, N. Nasr, and M. Shpitalni. Assembly system design and operations for product variety. *CIRP Annals*, 60(2):715–733, 2011.
- [26] B. Kast, V. Dietrich, S. Albrecht, W. Feiten, and J. Zhang. A hierarchical planner based on set-theoretic models: Towards automating the automation for autonomous systems. In *ICINCO 2019*. SCITEPRESS Digital Library.
- [27] S.G. Kaufman, R.H. Wilson, R.E. Jones, T.L. Calton, and A.L. Ames. The archimedes 2 mechanical assembly planning system. In *ICRA*, volume 4, pages 3361–3368. IEEE, 1996.
- [28] T. Lozano-Pérez and L.P. Kaelbling. A constraint-based method for solving sequential manipulation planning problems. In *IROS*, pages 3684–3691. IEEE, 2014.
- [29] B. Marthi, S.J. Russell, and J.A. Wolfe. Angelic hierarchical planning: Optimal and online algorithms. In *ICAPS*, pages 222–231, 2008.
- [30] A. Marzinotto, M. Colledanchise, C. Smith, and P. Ögren. Towards a unified behavior trees framework for robot control. In *ICRA*, pages 5420–5427. IEEE, 2014.
- [31] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. Pddl - the planning domain definition language. *The AIPS-98 Planning Competition Comitee*, 1998.
- [32] M. Molineaux, M. Klenk, and D.W. Aha. Planning in dynamic environments: extending htns with nonlinear continuous effects. In *Conf. on Artificial Intelligence*, pages 1115–1120. AAAI Press, 2010.
- [33] D.S. Nau, T.-C. Au, O. Ilghami, U. Kuter, J.W. Murdock, D. Wu, and F. Yaman. Shop2: An htn planning system. *Artificial Intelligence Research*, 20:379–404, 2003.
- [34] W. Piotrowski, M. Fox, D. Long, D. Magazzeni, and F. Mercurio. Heuristic planning for hybrid systems. In *Conf. on Artificial Intelligence*, pages 4254–4255. AAAI Press, 2016.
- [35] E. Prestes, J.L. Carbonera, S.R. Fiorini, V.A.M. Jorge, M. Abel, R. Madhavan, A. Locoro, P. Goncalves, M.E. Barreto, and M. Habib. Towards a core ontology for robotics and automation. *Robotics and Autonomous Systems*, 61(11):1193–1204, 2013.
- [36] R. Rosen, G. v. Wichert, G. Lo, and K.D. Bettenhausen. About the importance of autonomy and digital twins for the future of manufacturing. *IFAC-PapersOnLine*, 48(3):567–572, 2015.
- [37] S.J. Rosenschein. Formal theories of knowledge in ai and robotics. *New generation computing*, 3(4):345–357, 1985.
- [38] C. Schlenoff, E. Prestes, R. Madhavan, P. Goncalves, H. Li, S. Balakirsky, T. Kramer, and E. Miguelanez. An iee standard ontology for robotics and automation. In *IROS*, pages 1337–1342. IEEE, 2012.
- [39] P.S. Schmitt, W. Neubauer, W. Feiten, K.M. Wurm, G. v. Wichert, and W. Burgard. Optimal, sampling-based manipulation planning. In *ICRA*, pages 3426–3432. IEEE, 2017.
- [40] V. Shivashankar, R. Alford, U. Kuter, and D.S. Nau. The godel planning system: A more perfect union of domain-independent and hierarchical planning. In *IJCAI*, pages 2380–2386, 2013.
- [41] B. Siciliano and O. Khatib. *Springer handbook of robotics*. Springer, 2016.
- [42] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. In *ICRA*, pages 639–646. IEEE, 2014.
- [43] U. Thomas and F.M. Wahl. Assembly planning and task planningtwo prerequisites for automated robot programming. In *Robotic Systems for Handling and Assembly*, pages 333–354. Springer, 2010.
- [44] S. Thrun. Robotic mapping: A survey. *Exploring artificial intelligence in the new millennium*, 1(1-35):1, 2002.
- [45] M. Toussaint. Logic-geometric programming: An optimization-based approach to combined task and motion planning. In *IJCAI*, pages 1930–1936, 2015.
- [46] T.H.-J. Uhlemann, C. Lehmann, and R. Steinilper. The digital twin: Realizing the cyber-physical production system for industry 4.0. *Procedia Cirp*, 61:335–340, 2017.
- [47] F. Wirschofer, P.S. Schmitt, W. Feiten, G. v. Wichert, and W. Burgard. Robust, compliant assembly via optimal belief space planning. In *ICRA*. IEEE, 2018.
- [48] R. M. Young, M.E. Pollack, and J.D. Moore. Decomposition and causality in partial-order planning. In *AIPS*, pages 188–194, 1994.




7

**A hierarchical planner based on set-theoretic models:
Towards automating the automation for autonomous systems**

Title	A hierarchical planner based on set-theoretic models: Towards automating the automation for autonomous systems
Authors	Bernd Kast , Vincent Dietrich, Sebastian Albrecht, Wendelin Feiten, and Jianwei Zhang
ISBN/ISSN	978-989-758-380-3/2184-2809
DOI	10.5220/0007840702490260
Status	published
Publisher	Scitepress
Contribution of Bernd Kast	The hierarchical planning scheme, with the construction of new sub-planning tasks, was developed and implemented by me. The details of the planning scheme, as well as the manuscript benefited from the discussions with the co-authors. Vincent Dietrich helped me to conduct the experiments on the real hardware.

Summary	<p>Real-world planning problems comprise discrete as well as continuous properties in general. Established planning algorithms can either not cope with such a hybrid domain or suffer from the curse of dimensionality and are, therefore, limited to small examples relevant to research only. In this paper, we propose a hierarchical planning scheme that factorizes the domain using the formal models from our previous paper. The generic model, which incorporates declarative and procedural knowledge, allows us to automatically calculate a partial ordering that defines the refinements of each step in a course plan. By a recursive definition of new sub-planning problems, we can limit the branching factor and reduce the effective plan lengths to the next goal. Therefore, we can, in theory, scale nearly linearly with the length of the overall successful plan at the cost of non-optimality. We discuss our set-based forward state-space planner, which works on a graph rather than a tree. By that, we can detect the closure of a domain with its characteristic structure in the planning states. We apply our algorithm to two different assembly scenarios and discuss how execution is integrated into the planning process, the internal data structures, and the properties of our approach.</p>
---------	---

A Hierarchical Planner Based on Set-Theoretic Models: Towards Automating the Automation for Autonomous Systems

Bernd Kast¹ ^a, Vincent Dietrich¹ ^b, Sebastian Albrecht¹ ^c, Wendelin Feiten¹ ^d,
and Jianwei Zhang²

¹Siemens AG, Corporate Technology, Otto-Hahn-Ring 6, 81739 Munich, Germany

²University of Hamburg, Faculty of Mathematics, Informatics and Natural Sciences, Vogt-Kölln-Str. 30, 22527 Hamburg, Germany
bernd.kast@siemens.com

Keywords: Hierarchy, Planning, Autonomy

Abstract: The complexity of today's autonomous systems renders the manual engineering of control strategies or behaviors for all possible system states infeasible. Therefore, planning algorithms are required that match the capabilities of the system to the tasks at hand. Solutions to typical problems with robotic systems combine aspects of symbolic action planning with sub-symbolic motion planning and control. The problem complexity of this combination currently prohibits online planning without task specific, manually defined heuristics. To counter that we use a set-theoretic approach to model declarative and procedural knowledge which allows for flexible hierarchies of planning tasks. The coordination of the planning tasks on different levels, the classification of information and various views on data are the core functions of hierarchical planning. We propose suitable graph structures to capture all relevant information and discuss the elements of our hierarchical planning algorithm in this paper. Furthermore, we present two use-cases of an autonomous manufacturing system to highlight the capabilities of our system.

1 INTRODUCTION

Setting up autonomous systems still requires huge integration and engineering efforts. In order to make them widely applicable, we need a holistic approach that even considers aspects of component integration. This is especially important in production, where a higher degree of automation, notably for small lot sizes, can account for changing customer demands.

A limiting factor of today's automation strategies is the interwoven product and production design which requires costly manual effort even for small changes of some component (Hitomi, 2017), (Bryan et al., 2007). To reduce this effort, the design process of the product and the production system have to be decoupled (Hu et al., 2011) and then matched again by an autonomous system. Thus, the autonomous system has to solve on its own certain engineering tasks

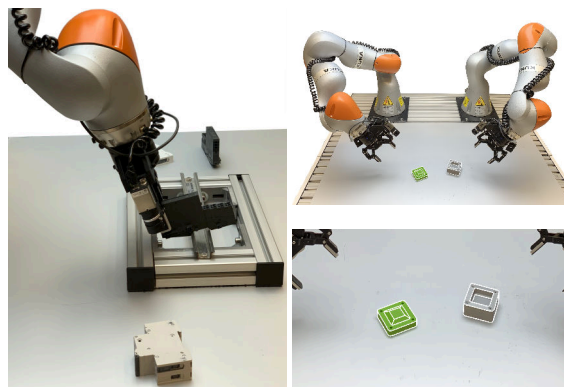





Figure 1: Images of the robotic system (upper right) and the hat-rail with components (left) and the box next to its lid (lower right).


that these days are addressed manually when automating a production system, i.e. the automation of automation (Schmitz et al., 2009) is necessary for flexible autonomous systems.

This requires general models which characterize objects by their properties (declarative knowledge) and describe actions modifying them (proce-

^a  <https://orcid.org/0000-0001-7838-3142>

^b  <https://orcid.org/0000-0003-0568-9727>

^c  <https://orcid.org/0000-0002-3647-4043>

^d  <https://orcid.org/0000-0002-7593-6298>

dural knowledge). These models define domains for suitable planners. Real-world manufacturing domains suffer from a huge computational complexity due to their mixed continuous and discrete nature. Discrete symbolic action planning and scheduling bring in the problem of combinatorial explosion while sub-symbolic tasks such as motion planning or control introduce time, accuracy and stability constraints. A common approach to address this curse of dimensionality is to use abstractions or hierarchies to decompose the problem. This allows to extract smaller sub-problems that are easier to handle. It is especially important to deduce those hierarchies for both, procedural and declarative knowledge, automatically from the models in order to keep the task and plant description separate. Additionally, assumptions that limit the flexibility of the system have to be avoided. One example is the *downward refinement property* which demands that each coarse plan can be refined on all more detailed levels. This is in general not viable for real-world problems with decoupled models for the problem and the plant. This becomes obvious for task specific sub-symbolic properties, such as possible grasp positions, which can for the general case not be modeled on an abstract, purely symbolic level.

Throughout this paper, we illustrate our approach with an industrial assembly use-case, cf. Figure 1, which is modeled for maximal flexibility. This means in particular, that not only the actions of the robot, but even the interplay between software components is planned, cf. section 8. We present a planner that benefits from models that integrate declarative and procedural knowledge and allow for an automatic calculation of abstractions. This planner uses the graph structures of our models on different abstraction levels to factorize the problem, integrates special single-level planners for individual subproblems but does not depend on the downward refinement property.

The paper is structured as follows: In section 2 we present related work regarding hierarchical symbolic planning, combined task and motion planning in robotics and autonomous production systems. After that, we discuss a set-theoretic approach for declarative and procedural models that natively enables the deduction of hierarchical structures in section 3. The data structures for the single-level planning are described in section 4 and the respective single-level planner is discussed in section 5. In section 6 we extend these concepts for the hierarchical planning context. Finally, we present the hierarchical planning method in section 7 and demonstrate its features on two examples in section 8.

2 RELATED WORK

Planning algorithms are often tailored to modeling languages and rely on their expressiveness. In this section we discuss both symbolic and sub-symbolic planning, as well as the application specific aspects related to for autonomous production systems.

2.1 (Hierarchical) Action Planning

The key for flexible systems is a modeling language that formalizes the description of the domain. On the symbolic level, there are many languages such as PDDL (McDermott et al., 1998) and its dialects that focus on the procedural knowledge of a domain. A long series of planning competitions have resulted in a large set of fast planners for PDDL domains such as (Helmert, 2006). Extensions to PDDL add support for certain sub-symbolic properties such as time. An example is PDDL+ (Fox and Long, 2002) that introduces events and processes to model exogenous change and support domains with mixed discrete and continuous dynamics. Corresponding solvers, such as (Cashmore et al., 2016) or (Piotrowski et al., 2016), make use of this representation and handle domains with nonlinear continuous change. They approximate the dynamics of the system and handle the resulting discretized model with uniform time steps and step functions. Other PDDL dialects, e.g. (Dornhege et al., 2009), extend the formalism by using semantic attachments that allow for an evaluation of externally specified functions. Though, this requires modifying standard PDDL planners and to create domain-specific PDDL action choosing the specific external functions.

Another approach is partial-order planning (POP) (Young et al., 1994) which involves partially specified action decompositions. In this approach, the planner is only allowed to fill in missing pieces of a fixed plan template which hugely reduces the search space. However, it is difficult to ensure the separation of hardware and task description with this approach, which reduces the flexibility.

A large community addresses hierarchical task networks (HTN) that refine each abstract skill by a network of sub-methods, e.g. (Castillo et al., 2006), (Goldman, 2006), (Nau et al., 2003). In their basic form they were state-oriented and could not deal with time constraints or concurrent actions. Nevertheless, the formalism is more expressive than that of first principle planners (Erol et al., 1994), HTN methods can improve planning times (Nau et al., 2003) and even support plan reparation (Gateau et al., 2013). In (Bercher et al., 2016) an overview of HTN meth-

ods is provided that discusses the expressiveness of hierarchical planning formalisms as well as implications of preconditions and effects of abstract methods. A mandatory and limiting condition for HTNs is the downward refinement property that enforces refinements to all abstract solutions (Bacchus and Yang, 1994). Yet, the generation of suitable abstract models for a domain is a challenging or even impossible task and often results in a coupling between task and hardware description.

This becomes obvious in (Marthi et al., 2008). They propose expressive models, which allow for the definition of domains with the downward refinement property. In this approach, an underlying semantic that defines preconditions and effects even for abstract tasks, ensures that abstract plans can always be refined to a primitive solution. However, this demands that all relevant effects and preconditions of the primitives have to be considered even on the most abstract level. This is only possible if all properties can be described symbolically and hugely limits the benefits of the hierarchical abstractions.

To overcome the limitations induced by the downward refinement property, several variants of HTN planning and hybrid planning (Kambhampati et al., 1998), (Schattenberg, 2009) have been proposed. The hierarchical partial-order planner, introduced in (Bechon et al., 2014), uses additional knowledge that describes sets of abstract actions with optional methods to increase flexibility during refinement. Another approach to improve versatility is the combination of HTN and POP in a domain-specific planner with a strict separation between several hierarchical levels (Castillo et al., 2003). In both approaches the effects of the downward refinement property are mitigated at the cost of models which are dependent on the hardware and the task at the same time.

Temporal reasoning within an HTN planner is discussed in (Castillo et al., 2006) and an HTN extension for (nonlinear) continuous temporal environments can be found in (Molineaux et al., 2010), in which the SHOP2 planner (Nau et al., 2003) is matched to features from PDDL+ (Fox and Long, 2002). Those approaches offer a suitable performance even in domains with sub-symbolic properties but don't support general geometric constraints such as collisions. An overview of HTN planning can be found in (Georgievski and Aiello, 2015).

Each of those approaches extends the application area. Summarizing we state that manually drafted hierarchies, the strict requirement of the downward refinement property or a limited support of continuous properties prevents an application in our general manufacturing domain.

2.2 Combined Task and Motion Planning in Robotics

A challenging and widely discussed subproblem for autonomous systems in production is task and motion planning. In (Srivastava et al., 2014) existing task planners and motion planners are combined by the introduction of new symbolic abstractions. Motion planning is used to refine the plans from symbolic planning to a sub-symbolic level. The general idea is to add further abstract poses to the symbolic planning problem every time the refinement failed because no collision-free path could be found. This allows to consider continuous properties on an abstract level without manual prior modeling. However, the computational complexity is shifted to the abstract level. The advantages of the hierarchy are additionally reduced by the limited number of abstractions that are possible with this approach.

Instead of combining of two separate planners (Garrett et al., 2015) extends the symbolic planners and their heuristics to motion planning and thus lifts the sub-symbolic world to the symbolic planning. The heuristic is based on domain-dependent literals that represent reachability for example, which can be evaluated lazily on demand. The planner maintains a corresponding reachability graph of sampled configurations. Geometric constraint-satisfaction problems (CSP) determine plan templates with unbound variables like robot and object poses (Lozano-Pérez and Kaelbling, 2014). The central disadvantages of these approaches are that a discretization has to be specified in advance and that they run into scalability issues if a fine discretization is needed.

Another possibility is to add the symbolic properties to the sub-symbolic path planning problem. If a simple symbolic planner is used to generate action sequences, large optimization problems can be formulated that determine optimal intermediate and final states (Toussaint, 2015). They demonstrated the strength of their approach for a stacking problem in which towers of cylinders and plates have to be assembled. However, it is computationally too demanding for online planning and scales exponentially with the number of involved objects.

The ScottyActivity planner (Fernandez-Gonzalez et al., 2018) is one of the few approaches that consider dynamics and/or temporal constraints along the manipulation problem. It combines strong heuristics with convex optimization and relaxed plan graphs. However, the absence of obstacles and the limitation to linear dynamics for the robots prohibits the applicability for real-world problems.

In (Schmitt et al., 2017) an asymptotically optimal

manipulation planner is proposed that considers both, nonlinear dynamics and collisions. Without any hierarchical decomposition or heuristics, their approach, which extends sampling-based roadmap planners to explore configuration spaces, scales exponentially.

2.3 Autonomous Production Systems

There are three main challenges for autonomous robotics in production: planning to generate a coarse, symbolic plan, mapping the plan to the actual hardware and controlling it accordingly. The pioneering system (Kaufman et al., 1996) is one of the few who proposed an integrating approach to target all three challenges at a time. However, this approach relied on substantial task specific manual modeling of sub-assemblies, which limits flexibility. Additionally, the approach is tested in simulation only and therefore it neglects sensor input and generates a nominal control code only. In (Thomas and Wahl, 2010) an improved version is presented that uses CAD data and other models to compute assembly sequences that comply with the desired goal configuration. The optimal plan is chosen, mapped to the robot's skills and executed on the hardware. Due to the lack of a hierarchical decomposition, this approach, however, suffers from the curse of dimensionality.

3 FORMAL MODELS

The planning process requires models for declarative and procedural knowledge. For our hierarchical planning approach, it is especially important that both types are naturally hierarchically structured and that these hierarchies match each other. We accomplish that with the following set-theoretic definitions, which have been introduced in more detail in (Kast et al., 2019).

3.1 Concepts

In this work we use the term *concepts* for elements of the declarative knowledge. An example of such a concept is a simple box. Each specific box is an instance of the abstract concept "box", but the properties of boxes differ in detail. For example, the shape might be different, as one box is cubic and the other cylindrical. Thus, we have an abstract concept of a box and two specializations capturing subsets. This notion of concepts can be formally grounded by following set-theoretic definitions:

- A concept base B_Γ is the set of instances, not necessarily finite.

- A concept C is a subset of B_Γ , i.e., $C \subseteq B_\Gamma$.
- A concept class Γ is the set of concepts C_i that have a common concept base B_Γ , i.e., $\forall C_i \in \Gamma, b \in C_i : b \in B_\Gamma$.
- A partial order \mathcal{M} , describing *more detailed than*, can be defined on Γ : $(C_i, C_j) \in \mathcal{M}$ iff $C_i \subset C_j$.

Extending the previous example: In most cases you do not only want to know the shape of the box but also the characteristics of the dimensions. Further properties, like the weight or the current location, might also be interesting. This directly leads to special types of concepts, which we call *composite concepts*.

The following definitions introduce composite concepts formally:

- Given an ordered set \mathcal{R} of identifiers, e.g. strings, its elements $r \in \mathcal{R}$ are called roles. Thus, for each subset $\mathcal{R}_i \subseteq \mathcal{R}$ with $n_{\mathcal{R}_i} := |\mathcal{R}_i|$ there exists a bijective mapping \mathcal{J} to $\mathbb{N}_{n_{\mathcal{R}_i}} := \{1, \dots, n_{\mathcal{R}_i}\}$, i.e., $\mathcal{J}(r) \in \mathbb{N}_{n_{\mathcal{R}_i}} \forall r \in \mathcal{R}_i$ and $\mathcal{J}^{-1}(n) \in \mathcal{R}_i \forall n \in \mathbb{N}_{n_{\mathcal{R}_i}}$.
- A composite, recursively defined concept $C = \prod_{r \in \mathcal{R}_C} C_r$ with \mathcal{R}_C being the specific set of roles for this composite concept

This definition of a composite concept corresponds to a (directed) graph, cf. Figure 2: the nodes correspond to (sub-)concepts and the edges point from the composites to their sub-concepts.

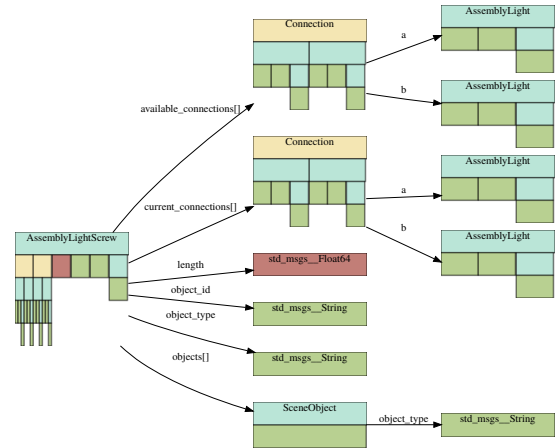


Figure 2: Example of a composite concept that describes objects in the later examples. Here, only the composite structure of the first level of sub-concepts is additionally depicted. Note that the small images for each node visualize the composite structures. A common color is used for each concept class. The edges are attributed with roles.

Having two composite concepts of one concept class $C, \bar{C} \in \Gamma$, the partial order \mathcal{M} can be deduced recursively:

$$(C, \bar{C}) \in \mathcal{M} \text{ iff } (C_r, \bar{C}_r) \in \mathcal{M} \forall r \in \mathcal{R}_{\bar{C}},$$

which requires that C has all the roles of \bar{C} . This means that (see Figure 3):

$$C \cong \prod_{r \in \mathcal{R}_C \cap \mathcal{R}_{\bar{C}}} C_r \times \prod_{r \in \mathcal{R}_C \setminus \mathcal{R}_{\bar{C}}} C_r \subseteq \bar{C} \times \prod_{r \in \mathcal{R}_C \setminus \mathcal{R}_{\bar{C}}} B_{\Gamma}(r).$$

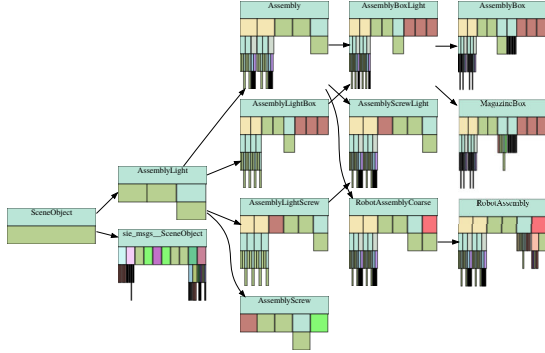


Figure 3: Example hierarchy of the concept class for objects in the later examples. The graph is directed and acyclic, but not necessarily a tree.

By definition the leaves of a concept are considered to be atomic at the modeling detail of the provided concept. Each element of a concept is called an *instance*. For composite concepts this results in the necessity to specify values for each leaf concept of a composite concept.

In order to answer the question whether two instances of the same concept represent the same thing, we assume that for each concept C a relation, the so-called *compare relation* F_C , is defined. This means two instances b_i, b_j are similar with respect to a concept C if $(b_i, b_j) \in F_C$. The product representation allows to derive similarity of two composite concepts when all their corresponding leaves are similar. Note, that similarity does not mean that the instances are identical, but rather that the information provided by the second instance does not add any additional information on a given level of abstraction of a domain. This compare relation will help to introduce *compact domains*, in which different results are mapped to the same graph node if the corresponding instances are similar.

3.2 Operators

Declarative knowledge is hardly useful if elements of the procedural knowledge, so-called *operators*, do not use them. We define operators as follows (see Figure 4):

- An operator $\pi \in P$ is a mapping of given input concepts I_{r_i} to output concepts O_{r_j} with given input roles $r_i \in \mathcal{R}_{\pi, I}$ and output roles $r_j \in \mathcal{R}_{\pi, O}$, i.e., $\pi : \prod_{r_i \in \mathcal{R}_{\pi, I}} I_{r_i} \rightarrow \prod_{r_j \in \mathcal{R}_{\pi, O}} O_{r_j}$.

- The output instances can be explicitly specified by symbolically-representable mappings (mathematical formulas) or implicitly as the result of some calculation or experiment in simulation or the real world, operating on input instances

- Certain operators describe modifications of instances, i.e., elements of knowledge are invalidated when such an operator is executed; such inputs are called *consumed*. The set of roles corresponding to inputs that are consumed by the operator is denoted by $\mathcal{R}_{\pi}^c \subseteq \mathcal{R}_{\pi, I}$.

- Operators are fully functional, i.e., they do not have an internal state.

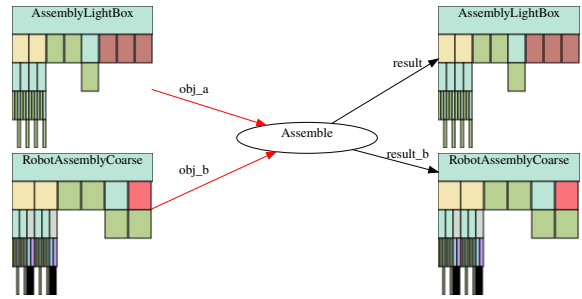


Figure 4: Example of an operator with multiple concepts as in- and outputs. A consumed input is depicted by a red edge.

Since operators are elements of a functional space, they can also be modeled as instances of an operator concept on a higher level of abstraction that is often called *meta level*. In addition to the input and output structures and a reference to the executable code, an instance of this operator-concept can contain *meta information* like key performance indicators (kpi).

A hierarchy of operators is obtained from the hierarchical structures of concepts. An operator π_1 is more detailed than another π_2 in case the following conditions hold true (see Figure 7):

- all input and output roles of operator π_2 are elements of the role set of operator π_1 : $\mathcal{R}_{\pi_2, I} \subset \mathcal{R}_{\pi_1, I}$ and $\mathcal{R}_{\pi_2, O} \subset \mathcal{R}_{\pi_1, O}$,
- all (common) input concepts $I_{r,1}$ and outputs $O_{r,1}$ of operator π_1 are more detailed than those of operator π_2 : $(I_{r,1}, I_{r,2}) \in \mathcal{M} \forall r \in \mathcal{R}_{\pi_2, I}$ and $(O_{r,1}, O_{r,2}) \in \mathcal{M} \forall r \in \mathcal{R}_{\pi_2, O}$,
- the meta information, e.g. key performance indicators, of operator π_1 are more detailed than those of operator π_2 .

4 CONCEPTS AND GRAPHS FOR PLANNING

In our context planning is based on the evaluation of operators, working on the set of currently available instances, to reach one or multiple goal instances. The operators can use or consume available instances and generate new ones. Many application domains pose problems combining symbolic and sub-symbolic instances. However, standard approaches run into the curse of dimensionality. The approach of hierarchical planning is based on the hierarchies of concepts and operators. It tries to solve the planning task on an abstract level and to refine the plan step by step and abstraction level by abstraction level until a plan on the most detailed level is obtained.

To formalize the structures obtained during planning we first introduce formal models for a planning task, a planner and a plan.

4.1 Planning Task

A planning task $PT(X_{\text{init}}, X_{\text{goal}}, X_{\text{op}})$ specifies the combination of three sets: the set of initial instances $X_{\text{init}} := \{b_i, i = 1, \dots, n_{\text{init}}\}$, the set of goal instances $X_{\text{goal}} = \{b_j, j = 1, \dots, n_{\text{goal}}\}$ and the set of available operators $X_{\text{op}} := \{\pi_k, k = 1, \dots, n_{\text{op}}\}$ with $n_{\text{init}} \in \mathbb{N}_0$, and $n_{\text{goal}}, n_{\text{op}} \in \mathbb{N}$. Consequently, a planning task is a concept in the meta domain.

4.2 Plan Family

We introduce the concept (in the meta domain) of a *plan family* that contains the following information: the initial instances, the executed operators, the input instances of executed operators with their roles, and the output instances of executed operators.

To represent this plan family, we use an annotated bi-partite graph $\mathcal{G}_{\text{PF}}(V, E, E^{\text{con}})$, in which all annotated edges in $E^{\text{con}} \subseteq E$ correspond to consumed operator inputs. The one node set V_B only contains nodes representing instances and the other V_π consists of operators only. The bi-partite property ensures that $V = V_B \cup V_\pi$, $V_\pi \cap V_B = \emptyset$ and $\forall e \in E: e = (v_1, v_2)$ with $\{v_1, v_2\} \not\subseteq V_\pi$ and $\{v_1, v_2\} \not\subseteq V_B$. The representation mapping $f_v: V \rightarrow B \cup P$ of nodes in \mathcal{G}_{PF} to instances and operators is assumed to be bijective.

Adding an executed operator to a plan family For every operator that is executed during planning, all new nodes are added to the graph as long as the graph does not already contain the information. The result is the bi-partite graph $(\bar{V}, \bar{E}, \bar{E}^{\text{con}})$ with $V \subseteq \bar{V}$ and $E \subseteq \bar{E}$. In more detail, this means that, with the given

operator $\pi: \prod_{r_i \in \mathcal{R}_{\pi, I}} I_{r_i} \rightarrow \prod_{r_j \in \mathcal{R}_{\pi, O}} O_{r_j}$, the execution results in the following mapping between instances:

$$\{b_r \mid r \in \mathcal{R}_{\pi, I}\} \mapsto \{\bar{b}_r \mid r \in \mathcal{R}_{\pi, O}\}.$$

The graph is only modified if the entropy (number of different results for a given input set) of the operator, is not yet exhausted by former executions. Thus, a node v_π is added to V_π and all inputs are connected via a directed edge to this new node. Edges corresponding to consumed inputs are marked by adding them to E^{con} . Additionally, nodes for all output instances, i.e., $\{v_{f_v^{-1}(\bar{b}_r)} \mid r \in \mathcal{R}_{\pi, O}\}$, are added to V_B and v_π is connected to all these outputs via directed edges.

We introduce a so-called *compact domain*, in which new nodes are only added for instances that carry new information, i.e., node $v \in V$ is added if $\forall v_i \in V_B$ with $v_i \neq v: (f_v(v_i), f_v(v)) \notin \mathcal{M}$ or $(f_v(v), f_v(v_i)) \notin \mathcal{M}$. If the plan family already contains a node related to an instance with the same information as the output instance, the mapping f_v points to that node instead. Thus, compact domains have a considerably smaller number of nodes enabling better scaling. In particular, the number of considered operators during planning can exceed the number of nodes in the corresponding plan family (see Table 1 for the graph sizes in the robotic example). Note that in consequence a plan family in a compact domain can have cycles, since the operator outputs can be mapped to ancestor nodes of the operator node v_π .

Thus, the following equations hold for the sets of the plan family:

$$\begin{aligned} \bar{V}_\pi &= V_\pi \cup \{v_\pi\}, \\ \bar{V}_B &= V_B \cup \left\{ v_{f_v^{-1}(\bar{b}_r)} \mid r \in \mathcal{R}_{\pi, O} \right\}, \\ \bar{E} &= E \cup \left\{ \left(v_{f_v^{-1}(b_r)}, v_\pi \right) \mid r \in \mathcal{R}_{\pi, I} \right\} \\ &\quad \cup \left\{ \left(v_\pi, v_{f_v^{-1}(\bar{b}_r)} \right) \mid r \in \mathcal{R}_{\pi, O} \right\}, \\ \bar{E}^{\text{con}} &= E^{\text{con}} \cup \left\{ \left(v_{f_v^{-1}(b_r)}, v_\pi \right) \mid r \in \mathcal{R}_{\pi, I} \right\}. \end{aligned}$$

4.3 Planner State Graph

Planning considers sets of instances $X_i, i \in \mathbb{N}$. The initial set is X_{init} . A plan is available if, by applying operators, a set X_m is generated that fulfills the goal set X_{goal} , i.e.,

$$\forall b_i \in X_{\text{goal}} \exists b_j \in X_m: (b_j, b_i) \in F_C,$$

in which $b_i \in C_i$. Each set of instances X_i is obtained by executing one operator $\pi_k \in X_{\text{op}}, k \in \mathbb{N}$, on one selection of instances $(b_1, \dots, b_{n_{\pi_k}})$ with $n_{\pi_k} := |\mathcal{R}_{\pi_k}|$

from a previous set of instances X_j , $j \in \{0, \dots, j-1\}$ (with $X_0 := X_{\text{init}}$):

$$X_i = (X_j \cup \pi_k(b_1, \dots, b_{n_{\pi_k}})) \setminus \{b_{\hat{n}} \mid \mathcal{J}_{\pi_k}^{-1}(\hat{n}) \in \mathcal{R}_{\pi_k}^c\},$$

in which the instances have to be an element of the correct concept type $b_{g(r_l)} \in C_{r_l} \cap X_j$.

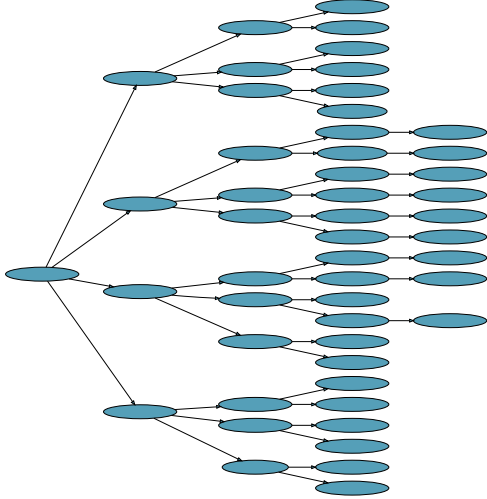


Figure 5: Part of the planner state graph for the later example of a hat-rail assembly.

Therefore, we introduce a directed graph $\mathcal{G}_{\text{ps}}(V, E)$ to describe the planner state as shown in Figure 5. Each node $v_i \in V$ corresponds to a set of instances X_i (and a subgraph of the plan family). There exists a directed edge $e = (v_i, v_j) \in E$ iff X_j was obtained from X_i by applying an operator. We assume a compact graph structure similar to a compact domain, i.e., $\forall v_i, v_j \in V, v_i \neq v_j : X_i \setminus X_j \cup X_j \setminus X_i \neq \emptyset$. This property reduces the size of the planning space considerably, because plan duplication is avoided for similar paths leading to the same intermediate planner state X_i . Additionally, this structure allows to detect dead ends, which is later used by the backtracking techniques for hierarchical planning.

5 SINGLE-LEVEL PLANNING

The classical planning problem is named *single-level planning* (SLP) as it doesn't consider hierarchies of concepts or operators. A planning task $PT(X_{\text{init}}, X_{\text{goal}}, X_{\text{op}})$ is solved in a constructive manner by applying operators from X_{op} to available instances.

A planning step is comprised by three actions:

1. choose the planning node v_i with the corresponding instances X_i to proceed from,
2. select an operator $\pi_j \in X_{\text{op}}$ with its input concepts I_r for $r \in \mathcal{R}_{\pi_j, I}$,

3. single out instances from X_i and map them to the inputs of π_j such that $b_r \in I_r \forall r \in \mathcal{R}_{\pi_j, I}$.

This general structure is valid for all kinds of planners ranging from PDDL to motion planning. Even classical graph algorithms like depth-first and breadth-first search can be used to realize such a single-level planner. In our implementation we let the search algorithm determine the next node v_i and iterate over all operators and all possible combinations of corresponding input instances for that state. More evolved planners have either better heuristics and sampling strategies or use suitable problem reformulations for specific sub-problems. Nevertheless, the core elements of forward planners correspond to the three steps described above.

Since each node v_i corresponds both to a set of instances X_i and a subgraph of the plan family, this subgraph captures the respective plan if the goal instances X_{goal} are met by the instances of X_i .

6 GRAPHS FOR HIERARCHICAL PLANNING

The computational complexity of the planning problem, which results from the mixture of symbolic and sub-symbolic properties, can be handled if hierarchies of suitable concepts and operators are considered.

The core idea is to generate a plan on an abstract level where only a few instances and operators exist. Then each step of this plan has to be recursively refined until the maximal level of detail is reached for all operators. In case a plan step cannot be refined on a more-detailed level, a back-tracking procedure is required that is described in subsection 7.2. This hierarchical divide and conquer approach ideally scales logarithmically with the number of required plan steps. Nevertheless, it is only beneficial with the right number of layers, if abstract plans sort out steps that are less promising on a more detailed level and if the downward refinement property holds often, thus most steps can be refined later on. In turn this requires the models of the concepts and operators to be suitably structured. Determining such models is a hard task on its own. However, the set-theoretic approach of the introduced formal models for concepts and operators helps to manually or automatically define them.

6.1 Parent Child Mapping

A central aspect of hierarchical planning is to refine an operator by posing a new planning task on a more-

detailed level. This mainly corresponds to a selection of suitable operators. Thus, we introduce the set-valued parent child mapping $\mathcal{F}_{PC} : P \rightrightarrows P$ such that $\mathcal{F}_{PC}(\pi_i) := \mathcal{F}_{PC}(\{\pi_i\}) = X_{op}$.

Currently, we assume that such a mapping is provided and engineered during modeling of operators. For some domains an algorithmic extraction of that mapping just from the sets of concepts and operators was successfully tested. For general domains, however, this is a problem to be addressed in future research.

In the following we introduce two graphs for a formal description of dependencies between plans on different levels of abstraction: the *layer graph* \mathcal{G}_{LG} addresses the hierarchical dependencies between plan steps and planning tasks, and the *extended planner state graph* \mathcal{G}_{EPS} represents temporal dependencies. These two combined with the previously described plan family, which captures the causal dependencies between instances and operators, could be represented in a single multi-layer graph. In this paper we stick to a presentation with separate graphs though as it is easier to understand.

6.2 Layer Graph

Since our planning approach does not assume a fixed number of hierarchy levels, each planning task PT defines some *layer*. Note that consequently the number of layers can vary during the planning process.

We introduce the (directed) bi-partite layer graph $\mathcal{G}_{LG}(V, E)$ to capture dependencies between planning states and layers, cf. Figure 6. Each node within the first set V_l corresponds to a planning task. As described in subsection 4.3, several planner states result during (single-level) planning for a given planning task. Each of these states is represented in the layer graph by a node in the set V_p . Note that the two sets V_l and V_p are disjoint.

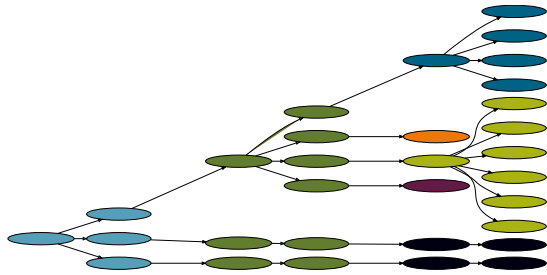


Figure 6: Part of the layer graph for the box-lid-example (subsection 8.2). The node colors match the temporal order during planning, compare Figure 8.

The set of vertices E represents dependencies of two kinds, which are discriminated by the direction

relative to layer nodes. The first type connects planner state nodes obtained during planning with their respective layer nodes which represent planning tasks, i.e., $\forall e = (v_1, v_2) \in E, v_1 \in V_l : v_2 \in V_p$. The second type connects a planner state to exactly one layer node if the layer node specifies the planning task to refine the operator that led to the planner state (cf. subsection 4.3), i.e., $\forall e = (v_1, v_2) \in E, v_1 \in V_p : v_2 \in V_l$ and $\forall v_1 \in V_p : |\{e \mid e = (v_1, v_2) \in E\}| \leq 1$.

Consequently, the layer graph \mathcal{G}_{LG} is tree-structured and captures the hierarchical refinements of plans. For later references, the mapping of each planning state to its corresponding layer is given by: $\mathcal{F}_{PS,L} : V_p \rightarrow V_l$ with $\mathcal{F}_{PS,L}(p) := l$ such that $(l, p) \in E$.

6.3 Blacklist

The planning task that refines a plan step, has a reduced set of instances for two reasons. First, the computational complexity is reduced, as less combinations are possible with fewer instances. Second, this assures that the plan is in accordance with the higher-level plan. This is particularly important for domains that are persistent and thus can't be set to an arbitrary state (e.g. real-world execution).

For example, assume that one out of two boxes has to be moved to a different working surface. In the coarse plan the smaller box is picked and then transported. If this pick is refined, only the smaller box is available. Otherwise, the more detailed planner could choose the bigger box for this task and an inconsistency between the plans on the different levels of abstraction would result.

The idea is to set all instances on a black list X_{Black} that belong to a concept corresponding to an operator input on the more abstract level. Let the coarse planning task $PT(X_{init}^c, X_{goal}^c, X_{op}^c)$ and a corresponding plan be given as a plan family (V, E, E^{con}) :

$$\forall v_1 \in V_B, v_2 \in V_\pi, (v_1, v_2) \in E : v_1 \in X_{Black}$$

and

$$\forall v_1 \in V_B, v_2 \in X_{Black}, v_1, v_2 \in C : v_1 \in X_{Black}.$$

Let $v_2 \in V_\pi$ be one operator in the plan that has no preceding operators, i.e., $\forall v_1 \in V_B$ with $(v_1, v_2) \in E$ holds $v_1 \in X_{init}^c$, then the corresponding planning task for refinement is:

$$PT_{v_2} \left(\begin{array}{l} X_{init}^c \setminus X_{Black} \cup \{v_1 \mid (v_1, v_2) \in E\}, \\ \{v_3 \mid (v_2, v_3) \in E\}, \\ \mathcal{F}_{PC}(\pi_{v_2}) \end{array} \right).$$

If the operator has predecessors, the set of initial instances has to be replaced by resulting instances of the refinement of that preceding operator. In order

to formally capture these dependencies, we have to extend the planner state graph \mathcal{G}_{PS} of the single-level planning.

6.4 Extended Planner State Graph

Hierarchical planning requires encoding of temporal relations between instances of different layers. Therefore, we extend the planner state graph \mathcal{G}_{PS} and discuss the related blacklists.

A single-level plan that resulted from a refinement step, is defined by the sequence of planning states in the \mathcal{G}_{PS} and a blacklist X_{Black}^c . The planning task corresponds to the first planning state p_0 in that sequence as it maps to the layer node in the layer graph. Solving this task results again in a separate planner state graph. However, the goal is to combine all planner state graphs in one large graph, the extended planner state graph \mathcal{G}_{EPS} , for unified data handling during hierarchical planning. Thus, we extend the node information of the (one-layer) planner state graphs by their layer information given by $\mathcal{F}_{PS,L}$ to avoid mixing of layers.

Assuming that a refining plan for the planner state p_i is found, the definition of the planning task corresponding to the layer $l_{i+1} = \mathcal{F}_{PS,L}(p_{i+1})$ has to use the resulting instances of that plan. Therefore, we add one additional node to $\mathcal{G}_{EPS}(V, E)$ for each planner state that reached the goal set X_{goal} . Let p_j be such a planning state and v be the corresponding additional node, then a matching edge is added $(p_j, v) \in E$. The set of instances corresponding to the node v contains the union of instances from the preceding node $X(p_j)$ and the blacklisted instances in the corresponding layer $X_{\text{Black}}(l)$: $X(v) := X(p_j) \cup X_{\text{Black}}(l)$.

In order to refine the next planner state of the coarse plan, one of the plans refining the preceding node has to be selected, leading to the node with blacklisted instances v . Let the planner state p_{i+1} correspond to the operator π in the plan family \mathcal{G}_{PF} . Now, the set of blacklisted instances for the next layer can be specified:

$$X_{\text{Black}}(l_{i+1}) := (X(v) \cap X_{\text{Black}}^c) \setminus \{v_k \mid (v_l, \pi) \in E_{\mathcal{G}_{PF}}, (v_k, v_l) \in F\}.$$

Then the set of initial instances for the next layer l_{i+1} results: $X_{\text{init}}(l_{i+1}) := X(v) \setminus X_{\text{Black}}(l_{i+1})$.

This means that for each plan refining the previous planner state p_i a new set of initial instances results and a corresponding new layer is generated. To also keep track of these dependencies, we add further edges to the extended planner state \mathcal{G}_{EPS} linking the node v with the added blacklist elements to the planner state p corresponding to the initial instances $X_{\text{init}}(l_{i+1})$.

7 HIERARCHICAL PLANNING

The goal of hierarchical planning is to find a sequence of planner states in \mathcal{G}_{EPS} such that no corresponding operator has a further refinement. For this task, the graphs of the plan family \mathcal{G}_{PF} , the extended planner state \mathcal{G}_{EPS} and the layers \mathcal{G}_{LG} are used.

The basic idea to reduce the fanout and thus counter the curse of dimensionality, is to use prior knowledge that is encoded by abstractions, like visualized in Figure 7. Once a plan was found on an abstract level, each step in this plan defines a new planning task that can either be refined by a single, more detailed operator or poses a new planning task on its own.

However, there is freedom in choosing the order of refining planning states on different levels of abstraction. The performance of such a refinement strategy depends highly on the domain at hand. If, for instance, the downward refinement property holds, one can always refine all operators to the most detailed level before proceeding to the next step. On the other hand, if some refinements repeatedly fail even on a rather coarse level, it is suboptimal to refine the first steps to maximal detail before checking the refinements of later steps on coarser levels.

If a refinement fails previous planning steps need to be adapted accordingly. Such backtracking methods are the key to real-world scenarios in which the coarser level cannot consider all details.

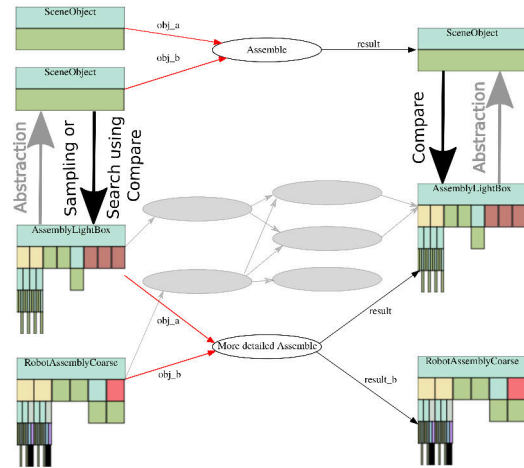


Figure 7: Visualization of the basic planning idea to reduce the fanout. An abstracted planning task is obtained based on the original instances and goals (arrows pointing upwards). New tasks in the refined level are posed by each step in the coarse plan. Either a primitive, single operator plan exists (white ellipse) or a planner has to be used to find a suitable plan (gray ellipses). The compare function is used to ensure that the abstractions of available instances and goals comply with the instances of the coarse plan.

7.1 Hierarchical Strategies

The two most basic strategies follow the spirit of depth- and breadth-first search and define the extrema of possible approaches. Naturally, intermediate strategies, possibly including heuristics, could increase planning performance for specific domains, however this is a topic for further research.

The breadth-first strategy refines all operators of one level once.

7.2 Backtracking

Only the downward refinement property could assure that all plan steps can be refined until the finest level is reached. However, this property cannot be guaranteed for all real-world problems. Therefore, the hierarchical strategies have to be complemented by backtracking procedures.

Such a backtracking method triggers re-planning on former layers. Since the overall problem is assumed to be solvable, there exists a set of layers and some selection of corresponding plans such that all operators are fully refined.

In most cases it is reasonable to choose the backtracking procedure that inverts the hierarchical (forward) search strategy. In the case of a breadth-first search, tasks that are predecessors within the extended planner state \mathcal{G}_{EPS} of the failing step are considered first. Only, if there exist no alternative plans in these layers that lead to a succeeding refinement, the more abstract layer, i.e., an ancestor in \mathcal{G}_{LG} of the node given by $\mathcal{F}_{PS,L}$, is addressed.

Re-planning means that the original goals are blacklisted and the (single-level) planner continues its search then.

8 EXAMPLE: AUTONOMOUS ASSEMBLY CELL

Our example setup consists of two robot arms with seven degrees of freedom each, cf. Figure 1. The workspaces of the arms have considerable overlap to enable cooperation and wrist-mounted cameras to perceive the environment. To demonstrate the capabilities of our approach, two different use-cases are analyzed. The first one considers a setup for the assembly of control cabinets: circuit breakers and other components have to be automatically assembled on a hat-rail. The focus of this example is the scalability with an increasing number of necessary steps. The second use-case is simpler, thus we can discuss the backtracking algorithm on the real graph structures.

The difficulty for both scenarios is the combination of high dimensional continuous properties (14 degrees of freedom for the robotic arms and up to five times three dimensions for the objects) and a even higher number of discrete options. Those discrete options not only result from the actions directly visible on the robot, such as perception, grasping, clicking, pushing or releasing, but also from different sequences of varying operators that calculate parameterizations such as grasp configurations, view poses or initialize the hardware. The operators in our models are kept rather atomic, to ensure flexibility, easy extension and composability as demanded for autonomous manufacturing systems. Additionally, these requirements result in subsection 4.3 in the definition of planner states being the combinations of available instances. In consequence, inhomogeneous planner states have to be considered during planning in opposition to most motion planner that assume a homogeneous state for performance reasons.

8.1 Hat-rail Assembly

For the hat-rail example four components and the hat-rail are initially placed in the working area of the robot arm. The precise configuration of the parts is not known in the beginning, thus a perception step is required. The goal of the task is specified on an abstract level, which only states the final position of the components on the rail.

To actually control the robotic system the sequence of the four assembly operations of the coarsest plan have to be refined three times. The first refinement adds further symbolic properties to the first domain. Then, the second refinement actually simulates all necessary robot motions assuring collision-free paths and reachability. Finally, the actual control of the robotic hardware is the most-detailed layer.

In this experimental setup we use the breadth-first strategy to hierarchically refine the coarse plan. Our numerical result (cf. Table 1) shows that the hierarchical approach scales almost linearly in the number of pieces to be assembled, which is the best we can expect without (automatically generated) additional abstraction levels for longer tasks. Note, that those planning times include all execution times of called operations. This includes computation and simulation of controllers for both robot arms considering the full nonlinear robot dynamics (~ 0.3 [s] per motion task). A considerable offset for the planning times is caused by the loading and construction of the geometric scene (~ 1.2 [s] once). For comparison we used a simple breadth first search on the same problem. For a single piece that is assembled, this algorithm is one

order of magnitude slower than our hierarchical approach. For two pieces it took at least three orders of magnitude longer than the hierarchical algorithm and did not find any solution before we ran out of memory. Arguably a faster algorithm can be found to solve this problem, which will be faster at least for the single piece assembly. However, the interesting point is how the solution will scale with increasing numbers of pieces. One can expect, that non hierarchical approaches run into the curse of dimensionality.

Table 1: Planning times t , and number of nodes in the plan family and planner state over the number of pieces to assemble. The first block states the results of the hierarchical planner and the second block a single-level approach which highlights the complexity of the task.

	1 pc.	2 pcs.	3 pcs.	4 pcs.
t [s]	3.7	5.5	7.9	10.5
$ \mathcal{G}_{\text{EPS}} $	159	330	502	688
$ \mathcal{G}_{\text{PF}} $	89	200	314	439
t [s]	79.2	$> 7 \cdot 10^3$	*	*
$ \mathcal{G}_{\text{EPS}} $	32,203	$> 2 \cdot 10^6$	*	*
$ \mathcal{G}_{\text{PF}} $	981	$> 4 \cdot 10^3$	*	*
Plan	16	36	56	76

These results demonstrate that the hierarchy successfully handles the curse of dimensionality. Note that the models for this example fulfill the downward refinement property, thus no backtracking is needed, which explains the almost perfect results.

8.2 Box Lid Assembly

In order to show the backtracking features of our planning approach, we consider a simple assembly process, in which a lid has to be placed on top of a box.

The coarse level has to choose which robot arm should pick the lid and assemble it on the box. However, the box is only reachable by one of the arms, as it is placed outside of the workspace overlap of the two robots. If the coarse level now chooses the wrong arm to do this task, there exists no valid refinement on the sub-symbolic level. Therefore, backtracking is needed. Note that only the second step of the coarse plan cannot be refined, since both arms can pick the lid, but only one can put it on the box.

Consequently, the backtracking procedure tries to find an alternative plan in the temporal preceding step first and then chooses a new plan on the coarser layer. See Figure 8 for a visualization of the planner state of this example.

The orange area shows the part where no plan that refines the second step in the coarse plan is found. The backtracking approach then tries to find an alternative plan in the preceding layer (red area). How-

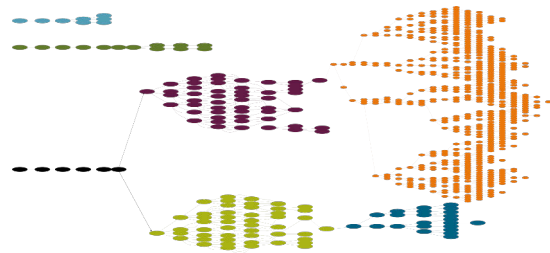


Figure 8: The planner state \mathcal{G}_{PS} with backtracking. No refined plan is found (orange area), therefore backtracking considers the preceding layer (red area) first. A second plan on the coarser level (dark green area) can then be successfully refined (green and petrol area).

ever, there exists no plan to pick the lid in a way that allows assembly, with the left arm, as the box is out of reach. Thus, the backtracking goes to the more abstract level (dark green area) and selects an alternative plan that uses the right arm instead of the left one. Here, the backtracking stops, and the breadth-first search is restarted. In this second coarse plan both actions, i.e., picking and assembling, can be refined successfully (green and petrol area, respectively).

9 CONCLUSIONS

The approach to ground the declarative and procedural knowledge on set theory enables to flexibly reason about hierarchies. Based on these structures we introduce a hierarchical planner that allows to seamlessly traverse different abstraction levels and obliterate the differences between symbolic and sub-symbolic domains. This approach additionally allows for modeling robot and task description independently. This leads to highly flexible and composable systems. We discussed examples of autonomous production systems with real hardware that is controlled with this hierarchical approach. Our examples show that hierarchy can help to avoid the curse of dimensionality and backtracking allows to handle cases that do not have the downward refinement property.

REFERENCES

- Bacchus, F. and Yang, Q. (1994). Downward refinement and the efficiency of hierarchical problem solving. *Artificial Intelligence*, 71(1):43–100.

The presented research is financed by the TransFit project which is funded by the German Federal Ministry of Economics and Technology (BMWi), grant no. 50RA1701, 50RA1702, and 50RA1703.

- Bechon, P., Barbier, M., Infantes, G., Lesire, C., and Vidal, V. (2014). Hipop: Hierarchical partial-order planning. In *STAIRS*, pages 51–60.
- Bercher, P., Höller, D., Behnke, G., and Biundo, S. (2016). More than a name? on implications of preconditions and effects of compound htn planning tasks. In *ECAI*, pages 225–233.
- Bryan, A., Ko, J., Hu, S., and Koren, Y. (2007). Co-evolution of product families and assembly systems. *CIRP Annals*, 56(1):41–44.
- Cashmore, M., Fox, M., Long, D., and Magazzeni, D. (2016). A compilation of the full PDDL+ language into SMT. In *AAAI Workshop: Planning for Hybrid Systems*.
- Castillo, L., Fernández-Olivares, J., Garcia-Perez, O., and Palao, F. (2006). Efficiently handling temporal knowledge in an htn planner. In *ICAPS*, pages 63–72.
- Castillo, L., Fernández-Olivares, J., and Gonzalez, A. (2003). Integrating hierarchical and conditional planning techniques into a software design process for automated manufacturing. In *ICAPS*.
- Dornhege, C., Eyerich, P., Keller, T., Trüg, S., Brenner, M., and Nebel, B. (2009). Semantic attachments for domain-independent planning systems. In *Conf. on Automated Planning and Scheduling*.
- Erol, K., Hendler, J., and Nau, D. (1994). Htn planning: Complexity and expressivity. In *AAAI*, volume 94, pages 1123–1128.
- Fernandez-Gonzalez, E., Williams, B., and Karpas, E. (2018). Scottyactivity: Mixed discrete-continuous planning with convex optimization. *Artificial Intelligence Research*, 62:579–664.
- Fox, M. and Long, D. (2002). Pddl+: Modeling continuous time dependent effects. In *Int. NASA Workshop on Planning and Scheduling for Space*, volume 4.
- Garrett, C., Lozano-Pérez, T., and Kaelbling, L. (2015). Pfrob: An efficient heuristic for task and motion planning. In *Algorithmic Foundations of Robotics XI*, pages 179–195. Springer.
- Gateau, T., Lesire, C., and Barbier, M. (2013). Hidden: Cooperative plan execution and repair for heterogeneous robots in dynamic environments. In *IROS*, pages 4790–4795. IEEE.
- Georgievski, I. and Aiello, M. (2015). Htn planning: Overview, comparison, and beyond. *Artificial Intelligence*, 222:124–156.
- Goldman, R. (2006). Durative planning in htms. In *ICAPS*, pages 382–385.
- Helmert, M. (2006). The fast downward planning system. *Artificial Intelligence Research*, 26:191–246.
- Hitomi, K. (2017). *Manufacturing Systems Engineering: A Unified Approach to Manufacturing Technology, Production Management and Industrial Economics*. Routledge.
- Hu, S., Ko, J., Weyand, L., ElMaraghy, H., Lien, T., Koren, Y., Bley, H., Chryssolouris, G., Nasr, N., and Shpitalni, M. (2011). Assembly system design and operations for product variety. *CIRP Annals*, 60(2):715–733.
- Kambhampati, S., Mali, A., and Srivastava, B. (1998). Hybrid planning for partially hierarchical domains. In *AAAI/IAAI*, pages 882–888.
- Kast, B., Albrecht, S., Feiten, W., and Zhang, J. (2019). Bridging the gap between semantics and control for industry 4.0 and autonomous production. In *CASE*. IEEE.
- Kaufman, S., Wilson, R., Jones, R., Calton, T., and Ames, A. (1996). The archimedes 2 mechanical assembly planning system. In *ICRA*, volume 4, pages 3361–3368. IEEE.
- Lozano-Pérez, T. and Kaelbling, L. (2014). A constraint-based method for solving sequential manipulation planning problems. In *IROS*, pages 3684–3691. IEEE.
- Marthi, B., Russell, S., and Wolfe, J. (2008). Angelic hierarchical planning: Optimal and online algorithms. In *ICAPS*, pages 222–231.
- McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., and Wilkins, D. (1998). Pddl - the planning domain definition language. *The AIPS-98 Planning Competition Committee*.
- Molineaux, M., Klenk, M., and Aha, D. (2010). Planning in dynamic environments: extending htms with nonlinear continuous effects. In *Conf. on Artificial Intelligence*, pages 1115–1120. AAAI Press.
- Nau, D., Au, T.-C., Ilghami, O., Kuter, U., Murdock, J., Wu, D., and Yaman, F. (2003). Shop2: An htn planning system. *Artificial Intelligence Research*, 20:379–404.
- Piotrowski, W., Fox, M., Long, D., Magazzeni, D., and Mercorio, F. (2016). Heuristic planning for hybrid systems. In *Conf. on Artificial Intelligence*, pages 4254–4255. AAAI Press.
- Schattenberg, B. (2009). *Hybrid planning and scheduling*. PhD thesis, University of Ulm, Germany.
- Schmitt, P., Neubauer, W., Feiten, W., Wurm, K., v. Wichert, G., and Burgard, W. (2017). Optimal, sampling-based manipulation planning. In *ICRA*, pages 3426–3432. IEEE.
- Schmitz, S., Schluetter, M., and Epple, U. (2009). Automation of automation definition, components and challenges. In *Conf. on Emerging Technologies & Factory Automation*. IEEE.
- Srivastava, S., Fang, E., Riano, L., Chitnis, R., Russell, S., and Abbeel, P. (2014). Combined task and motion planning through an extensible planner-independent interface layer. In *ICRA*, pages 639–646. IEEE.
- Thomas, U. and Wahl, F. (2010). Assembly planning and task planning two prerequisites for automated robot programming. In *Robotic Systems for Handling and Assembly*, pages 333–354. Springer.
- Toussaint, M. (2015). Logic-geometric programming: An optimization-based approach to combined task and motion planning. In *IJCAI*, pages 1930–1936.
- Young, R. M., Pollack, M., and Moore, J. (1994). Decomposition and causality in partial-order planning. In *AIPS*, pages 188–194.





8

Domain Optimization for Hierarchical Planning Based on Set-Theory

Title	Domain Optimization for Hierarchical Planning Based on Set-Theory
Authors	Bernd Kast , Vincent Dietrich, Sebastian Albrecht, Wendelin Feiten, and Jianwei Zhang
ISBN/ISSN	978-989-758-442-8/2184-2809
DOI	10.5220/0009823007590766
Status	published
Publisher	SCITEPRESS
Contribution of Bernd Kast	The idea, development, and implementation of the approach is my contribution to this paper. The co-authors proofread the manuscript. They also implemented example domains to detect common misalignments and verify the relevance of the proposed approach. Vincent Dietrich helped me with the experiment and figures.

Summary	<p>A key factor for successfully disseminating autonomous machines is reusable models. Creating the models that describe the domain, actions, and all available information that specify the current task takes a lot of time. Without intensive reuse and easy composability of the models in our autonomous system, the effort is only shifted from programming to modeling compared to a classical system. While trained persons can specify tasks easily, composing new domains requires more knowledge, experience, and time. This paper discusses an algorithm to ease that process and allow for a greater reuse of models. The basis of the algorithm is the set-based theory. Based on that, we identify possible optimizations of the composed domain. We then reformulate the declarative and procedural knowledge to find an isomorphic representation that is more favorable for our hierarchical planning. The intermediate goals are aligned across the different layers and thus require fewer additional actions than the optimal solution. We evaluate our algorithm on tasks in an optimized and non-optimized domain with up to 1000 steps. With this algorithm, we can handle ten times the number of parts compared to the original domain. Additionally, we can observe near-linear scaling with the number of parts in our experiment, which was also conducted on a real dual-arm cell.</p>
---------	--

Domain Optimization for Hierarchical Planning Based on Set-Theory

Bernd Kast¹ ^a, Vincent Dietrich¹ ^b, Sebastian Albrecht¹ ^c, Wendelin Feiten¹ ^d,
and Jianwei Zhang²

¹Siemens AG, Corporate Technology, Otto-Hahn-Ring 6, 81739 Munich, Germany

²University of Hamburg, Faculty of Mathematics, Informatics and Natural Sciences, Vogt-Kölln-Str. 30, 22527 Hamburg, Germany
bernd.kast@siemens.com

Keywords: Hierarchy, Planning, Autonomy, Robotic Assembly


Abstract: The design of planning domains for autonomous systems is a hard task, especially when different parties are involved. We present a domain optimization algorithm for hierarchical planners that uses a set-based formulation. Due to an automatic alignment we can compose models from different sources to a larger domain for efficient planning. The combination of domain optimization and hierarchical planning can handle large scale domains very efficiently. Our algorithm reduces the effects of the non-optimality that comes with the hierarchical approach. We demonstrate the scalability with a task and motion planning problem. In the scenario of a robotic assembly with up to 62 parts and plan lengths of over 1000 steps the planning times are kept within 15 minutes. We show the execution of our plans on a real-world dual-robot setup.


1 INTRODUCTION


Robotic systems bring together hardware and software components from different sources to solve a specific task. Only for recurring tasks it is viable for an engineer to compose the components and write or parametrize algorithms to manage the different pieces in a meaningful way.


However, even in an industrial environment, the cost of setting up the system can easily exceed the savings by the automated process. This is especially true for smaller lot sizes or even lot size one production. In order to address the automation of such a highly flexible production, we need algorithms for the composition of such systems and decision making during their runtime.

The algorithms for decision making have to handle a mixed problem, which is depicted in Figure 1 with continuous geometric and discrete properties, like attachment status or grasps. These planning problems become large for non-trivial tasks which results in unreasonable computation times due to the curse of dimensionality. This can be handled with a hierarchical approach, as presented in (Kast et al., 2019b).

^a  <https://orcid.org/0000-0001-7838-3142>

^b  <https://orcid.org/0000-0003-0568-9727>

^c  <https://orcid.org/0000-0002-3647-4043>

^d  <https://orcid.org/0000-0002-7593-6298>

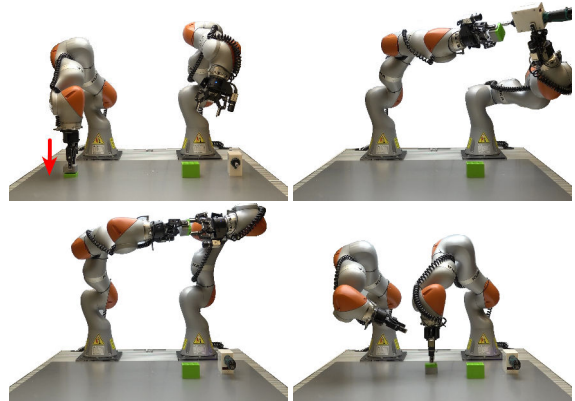


Figure 1: The core operations of the process: assembly, screw, handover, and place. Each action is planned and executed with the hierarchical planner.

In an industrial environment several different parties, such as integrators or component suppliers, with varying user roles are involved to define modules, objectives, and available resources. It must be possible to provide the planner with these different pieces of information in a modular way while each module might implement its own modeling scheme. These different styles of expressing the declarative and procedural knowledge should not affect the performance of the online planning.

In this paper we propose a domain optimization al-

gorithm that aligns the models for efficient planning. We can either apply this algorithm on-demand, prior to the planning, or as an offline step. We rely on our set-theoretic foundation to reformulate the declarative and procedural knowledge of the domain without infringing their validity. We analyze the performance of the optimization in a simulated setup with 800 test runs and tasks of varying lengths. The final experiment is conducted on a real two-arm robotic system.

2 RELATED WORK

There are two strands in the literature that target domain optimization problems.

A recently very active branch are data-driven algorithms, notably reinforcement learning approaches, which optimize heuristics for a specific domain. These methods show very good performances for some easy to simulate problems, like board games (Silver et al., 2016), (Silver et al., 2017) or computer games (Mnih et al., 2013), (Vinyals et al., 2019). However, application on real-world scenarios are still difficult due to the limited data available. Attempts to overcome this include large scale pick and place setups with hundreds of robots (Kalashnikov et al., 2018) and, due to the difficult nature of physics, hard but rather short tasks (Xie et al., 2019).

In (Schmitt et al., 2019) reinforcement learning is used in combination with an abstraction layer. This layer allows eased simulation, ensures viability, like collision-free movements, and provides an interface for real-world execution that handles small deviations. Still, training requires large datasets and processing power. Additionally, the trained model is a black box and thus hard to debug or transfer. In our approach we rely on models and rules rather than single data points that define the behavior of the resulting system. The explicit representation enables introspection, which is a key feature during development and for industrial environments. Additionally, less computational power is needed for our approach. However, it lacks theoretical optimality and still requires experts to program. In (Nägele et al., 2018) another approach, which relies on domain specific heuristics is proposed. In this case, however, the heuristics are computed online by analyzing the desired goal. Geometric interdependencies are broken up and the resulting plan is executed in simulation.

Another strand of related work covers optimization of modeled domains. In (Kang and Nnaji, 1993) a scheme for aligned and manually designed domains is elaborated. This, however, covers a single domain only and brings in its benefits only when strictly fol-

lowed. In a real-world scenario different parties bring in their modules, which are used for completely different problems as well, to compose the overall domain. Therefore, it is very hard to align everyone to a common scheme. In our approach each party fulfills their user-role with the representation they prefer. Just before planning, we harmonize the representations automatically.

For this type of optimization many algorithms that operate on discrete, mainly PDDL domains have been proposed. Two strands can be identified for algorithms which either transform the problem to suite the planner, or pick a planner, which can cope with the characteristics of the original problem.

Portfolio planners, such as (Seipp et al., 2012), (Katz et al., 2018) apply different planners with different heuristics on the domain and try to switch to the most appropriate combination for the current problem formulation.

In (Haslum et al., 2007), (Vallati et al., 2015) automated optimization algorithms for PDDL-domains are proposed, which allow even non-experts to apply generic planning algorithms on general PDDL-domains. In (Areces et al., 2014) a formalization for an optimization scheme was found, that was formerly applied manually on the domains. This even allows an engineer to inspect the optimized formulation to validate and further optimize it easily. However, it comes with the difficulties and limitations of PDDL to handle continuous domains.

3 APPROACH

Before we discuss our new domain optimization algorithm, we highlight the properties of its set-theoretic foundation, which was introduced in (Kast et al., 2019a).

Our new algorithm is effective in combination with a hierarchical planner only. We point out the relevant properties of an example hierarchical planning algorithm, which was proposed in (Kast et al., 2019b).

3.1 Declarative Knowledge

Following the line of (Kast et al., 2019a), we call elements of the declarative knowledge *concepts*. For us, a concept C is a subset of some concept base B_Γ which is a set of instances that is not necessarily finite. Concepts with the same concept base B_Γ belong to the same concept class Γ . The partial order *more-detailed than* \mathcal{M}_Γ between two concepts holds true if each instance described by the first concept is also an

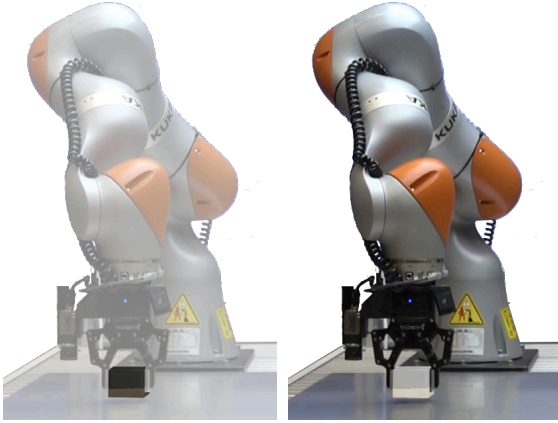


Figure 2: Either the Box is the main actor and the robot only the supporting actor, thus every property is described product-centric, or it is vice versa and the robot has properties that describe the objects in its grippers.

element of the second one. Properties that are common to a set of instances define *composite concepts*. These properties in turn can be expressed by value ranges or sets, which are concepts themselves:

$$C \cong \prod_{r \in R_C} C_r,$$

where C_r with $r \in R_C$ are the sub-concepts and the role set R_C defines the composite structure.

Examples to this set-theoretic view, which we will reconsider in subsection 3.4, are the concept bases of *Objects* and *Manipulators* with exemplary concepts of a box or a robotic arm.

The box with no property but an identifier can be detailed by concepts that define dimensions or its position, like *on the table*, *grasped by the robot* with even more-detailed concepts that specify continuous positions relative to some coordinate system.

The concept of the robot is detailed by concepts that specify its current position and state. Part of the state is the content of the gripper, which can be an object, like an instance of a box, or it is empty as nothing is currently grasped.

This example shows that some phenomena can equivalently be expressed by instances of concepts with different basic types, e.g. a robot holding a box or a box in a robot's gripper as visualized in Figure 2.

3.2 Procedural Knowledge

Planning is all about actions, which are represented by the procedural knowledge. Only the procedures make the declarative knowledge useful. We call these building blocks of planning and execution *operators* π . They map between given instances of input concepts to instances of output concepts:

$$\pi : \prod_{r_i \in R_{\pi, \text{in}}} C_{r_i} \rightarrow \prod_{r_j \in R_{\pi, \text{out}}} C_{r_j},$$

where the sets $R_{\pi, \text{in}}$ and $R_{\pi, \text{out}}$ describe the roles of the respective input and output concepts. The mapping can either be specified explicitly by a formula or implicitly by a black box of code, some library, or even a simulated or real-world experiment.

A set of operators is more-detailed than another operator $\tilde{\pi}$, when a sequence or network $\hat{\pi}$ of these operators exists with matching, more detailed outputs compared to the original operator and inputs, which are either a subset of the original inputs or orthogonal to all of them, i.e. have a different concept bases and are therefore independent to each other:

$$\forall r_i \in R_{\tilde{\pi}, \text{in}} \exists r_j \in R_{\tilde{\pi}, \text{in}} : (\hat{C}_{r_i}, \tilde{C}_{r_j}) \in \mathcal{M}_{\Gamma(\tilde{C}_{r_j})},$$

$$\forall r_i \in R_{\tilde{\pi}, \text{out}} \exists r_j \in R_{\tilde{\pi}, \text{out}} : (\hat{C}_{r_i}, \tilde{C}_{r_j}) \in \mathcal{M}_{\Gamma(\tilde{C}_{r_j})},$$

and for all $r_i \in R_{\tilde{\pi}, \text{in}}$ holds:

$$\begin{aligned} & |\{r_j \in R_{\tilde{\pi}, \text{in}} \mid \Gamma(\tilde{C}_{r_j}) \subseteq \Gamma(\hat{C}_{r_i})\}| \\ &= |\{r_j \in R_{\tilde{\pi}, \text{in}} \mid \Gamma(\hat{C}_{r_i}) \subseteq \Gamma(\tilde{C}_{r_j})\}|. \end{aligned}$$

The more-detailed operators can consider additional information and are in general more costly to be applied as they implement a more comprehensive simulation or even real-world execution to tell the outcome of the action.

This partial ordering is later used by the hierarchical planner to define new sub-planning problems.

3.3 Hierarchical planner

The hierarchical planner, which we proposed in (Kast et al., 2019b), is a forward state space planner that can handle domains with both, discrete and continuous properties efficiently. The basic idea of the planner is to divide the overall planning problem into small subproblems repeatedly, such that the curse of dimensionality can be alleviated. We do this by planning in an abstract domain first. In this domain the goal can be reached with a relatively small number of steps, as the abstracted operators cover huge changes of the state. Additionally, the branching factor is small due to the smaller number of possible operators that can be applied in that domain.

Once we found a solution on the coarse level, each operator that was applied in this plan by itself defines a new sub-planning problem with its inputs as starting values and outputs as goals. The operators that can be applied in this new, refined domain are the more-detailed operators as described in subsection 3.2. We apply this process recursively to each

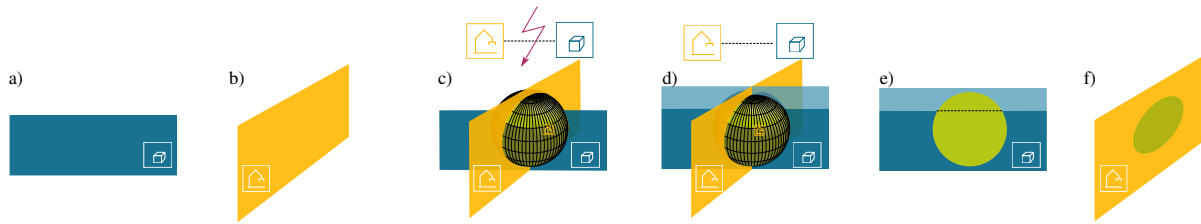


Figure 3: Geometric representation of the connection between two sets. The dark blue plane segment depicts the concept base for all objects (a) while the yellow plane represents the manipulators concept base all robots belong to (b). The intersection is a sphere (green). The projection of the sphere to the box-segment would miss the light green volume (c). This is due to a possibly empty list of grasped objects for the robot, which has no representation in the original box-set. Therefore, our algorithm extends the box-plane with the light blue segment depicting the "no object" set (d). After this extension, the volume can either be projected on the combination of the two blue plane segments (e) or the robot plane (f). This enables a reformulation of the planning domain.

of the newly generated planning problems until there is no further refinement for the operators. As the abstracted domains and their planned solutions can only be approximations of the real behavior, there can be errors and unsolvable subtasks on any level. Our solution to avoid dependency on the downward refinement property is backtracking, which means that plans on the abstract level are dismissed and new sequences to the goal are recalculated if a refinement fails. In our system, execution is the final refinement and error handling a case of backtracking. Therefore the planning approach represents a model predictive control scheme. Both execution and error handling are first class citizens with our planning approach. Under optimal conditions, when the coarse level is a good approximation to the behavior of the real-world, our planning approach can scale linearly with the length of the task. This, however, holds only true if the downward refinement property is always guaranteed. For a bad approximation the strategy still grows exponentially as the full problem is np-hard.

Additionally, our solutions are not necessarily optimal. It depends on the modeling of the coarse domains to propose good intermediate goals for factorization to have overall solutions close to the optimum. This, however, can be a burden to the engineering, especially when different people model the levels of the domain according to their respective user roles.

3.4 Optimization Algorithm

The key idea of our domain optimization algorithm is to align the different levels of abstraction. This is especially important, when engineers with differing viewpoints on the problem, for example due to different user roles, model different parts of the domain. As described in subsection 3.3 the key to an efficient factorization with our hierarchical planner are the intermediate goals that arise from the abstract level plan.

Remember that we need a sequence of operators on the refined level, which produces more detailed instances than the coarser level. According to our definition described in subsection 3.1, a concept, and therefore an instance of that concept as well, cannot be more-detailed than another, if their concept bases differ.

Consider the example of the box and the robot in subsection 3.1. Both can describe the same phenomenon, but one from a product-centric view and the other from an actor or tool-based angle. As they use different concept bases, the robot with the box cannot be a solution to the box in the gripper of the robot nor of any of the abstractions of the box with the robot. However, it is quite likely that the engineer modeling the coarse processing steps has a product-centric view on the plant, while the person that designs the cell with the robot or some other machining equipment has a tool-based angle.

During planning the coarse level would define intermediate goals, which are only reachable for the refined domain, if box and robot are separated. This causes additional steps to be planned and executed, which results in solutions that are farther off the optimal plan. To overcome this issue, we can identify concepts $C \cong \prod_{r \in R_C} C_r$ having a sub-concept $C_j, j \in R_C$ with a concept base matching our overall goal, i.e. $B_\Gamma(C_j) \subseteq B_\Gamma(C_{\text{goal}})$. Then we reformulate these concepts such that they have the same concept base as the goal concept while the set that they represent is identical: Assume that $C_j \cong \prod_{r \in R_{C_j}} C_{r_j}$ then define the reformulated concept by

$$C' := \prod_{r_j \in R_{C_j}} C_{r_j} \times \prod_{r \in R_C \setminus \{j\}} C_r.$$

This concept is isomorph to a subset of C_j and thus has the concept base $B_\Gamma(C_j)$. This is the formal description of the isomorphism described in subsection 3.1. With this transformation, which is depicted in Figure 3, we can directly derive intermediate goals

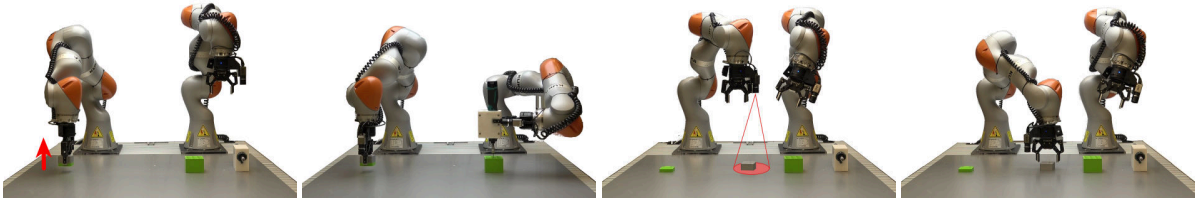


Figure 4: Extension to Figure 1 of important actions. Picking of the assembled box, localization of objects, screw out from the magazine and initial pick of the box.

and can successfully compare newly created instances with our existing subset, as the concept bases match.

However, this is only possible if all elements of the concept can be expressed in the other concept base, which is not always true. Consider the example where one concept is a composite concept having an array of another concept type as a sub-concept. For our robot-box-example, the composite concept with an array is the robot and elements in this array are of type box. Then not all composite concepts with the array can be expressed equivalently by the sub-concept, because the array might be empty, i.e. the robot’s gripper is empty.

Therefore, we cannot express some instances directly in the form of the first concept base. To overcome this, we expand the first concept base with a set which contains the empty set, i.e.

$$B_{\Gamma}^* := B_{\Gamma}(C_j) \cup \{\{\emptyset\}\}.$$

Afterwards corresponding sub-concepts of all elements of that extended concept base can have additional attributes:

$$C' := \{\emptyset\} \times \prod_{r \in R_C \setminus \{j\}} C_r.$$

For example, a *no-box-object* can have the attribute *robot*. This allows to cover empty arrays as well and therefore include all relevant instances.

We then generated wrapping operators which replace the original ones that have optimizable input or output concepts. They internally call the original operators but change the type of the inputs and outputs, such that the aligned and newly generated concepts are returned. For a domain, which is optimized this way, the intermediate goals defined by the abstract layers for the refinement layers are relaxed, while the specified overall goals remain firm. Therefore, the overall result of the plan is the same for the original and the optimized domain, while possibly less steps are required for the optimized domain.

4 EXPERIMENT

In the real-world we conduct our experiment on a dual arm robot with a total of 14 degrees of freedom. Each of the arms has a two-finger gripper and a RGB-camera that is used to refine the object poses. The workspace is observed with a stationary 3D camera, which is used to estimate the objects starting locations. We use a cordless screwdriver which is remotely controlled.

The assembly scenario we use to analyze the performance of our algorithm includes the mounting of a lid to a box and the insertion of multiple screws to join them. The relevant actions that need to take place are depicted in Figure 1 and Figure 4. It is necessary to refine the object positions with one of the wrist-mounted cameras, before any interaction takes place, which means prior to grasping or picking up screws. Fastening the screws can only happen with the box fixated in one gripper and the screwdriver actuated with the other arm. The objects (screwdriver and box with lid) can be placed on the table which can create loops in the state space. The box can additionally be handed over between the two arms to change the relative orientation in the gripper.

We generate collision-free robot trajectories with the constraint-based solver described in (Schmitt et al., 2019). This allows for good reachability with relatively few intermediate positions for motion planning.

The configuration of the perception system is grounded on (Dietrich et al., 2018).

We can vary the number of screws that are inserted to modify the difficulty of the task. The real box has only four screw holes, which results in maximum of six parts that can be assembled. In simulation we added virtual screw holes such that we can examine scalability of our approach for up to 62 pieces.

We analyze the performance of the optimization algorithm on a domain that was first designed on an abstract, discrete layer only and later extended with the continuous layers. There are a total of 4 modeled abstraction layers for the task at hand. They include two completely abstract levels, with and without lo-

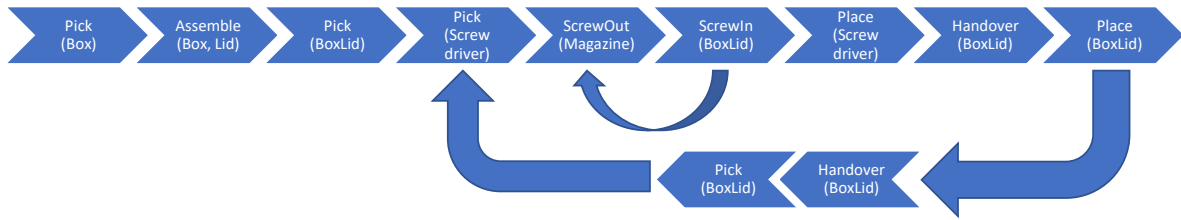


Figure 5: Nominal sequence of actions for the assembly process. Localizations, which must be executed prior to each pick, screw, or assemble are omitted in this graph, but must be added to the plan as those actions fail otherwise. Additional screws require the repetition of the last actions actions of this sequence. For the optimized domain only the small loop with two actions (and additional localizations) must be repeated. The original domain requires the execution of actions in the larger loop, as the intermediate goal is more restrictive. This includes the placement of the screwdriver such that the handovers of the box can be performed.

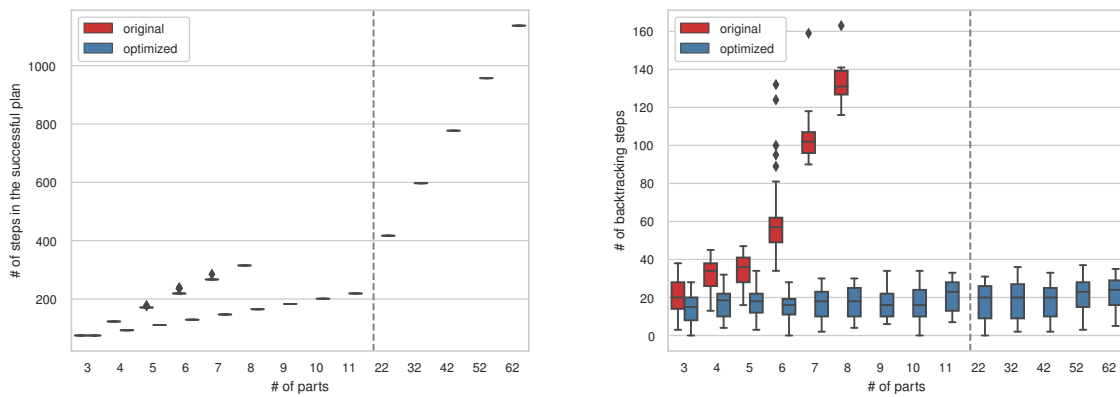


Figure 6: The number of steps in the successful solution for the task increases linearly. For the original domain more steps are necessary per additional part. For the optimized domain the intermediate product is not placed and nevertheless recognized as the interim step due to the reformulation. This is also reflected in the number of necessary backtracking steps shown on the right figure. Both numbers go hand in hand, as shorter plans have less operations, that may fail during refinement. Please notice the change in step size right of the dashed line.

calization of parts, simulated and real-world execution. Only the first layer is workpiece centric, while the others are robotic centric. The robot can localize each object in its workspace, pick the box, place it on the lid, localize the merged pieces again to pick both up together. The other arm can then pick the screwdriver, pick up a screw from the magazine, and insert it in the air to the box with lid as depicted in Figure 4. In order to reach the intermediate goals in the optimized domain, the arm with screwdriver can localize the magazine right away and continue to pick up screws from the magazine and fasten the lid with them. For the original domain on the other hand, the box must be separated from the robot after each intermediate step to fulfill the intermediate goals posed by the coarse level. This means that the box must be placed on the table. However, the box is grasped up-

side down to allow access for the screwdriver. Therefore, no collision-free way to place the box upright with a single arm exists. That means an additional handover is needed, and the planner must find out that the screwdriver must be placed to free the second arm for that as highlighted in Figure 5. Afterwards the box and the screwdriver must be localized again before the process can continue similar to the optimized domain. In Figure 6 the resulting lengths of the successful plans is depicted on the left hand side. We can see that the plan lengths grow significantly faster for the original domain due to the described process of additional placements. We tested each domain 100 times with increasing number of screws and a noise of 5 cm added to the initial object positions. The variance in plan lengths is a result of additional free-space movements to reach the goal positions. On the right

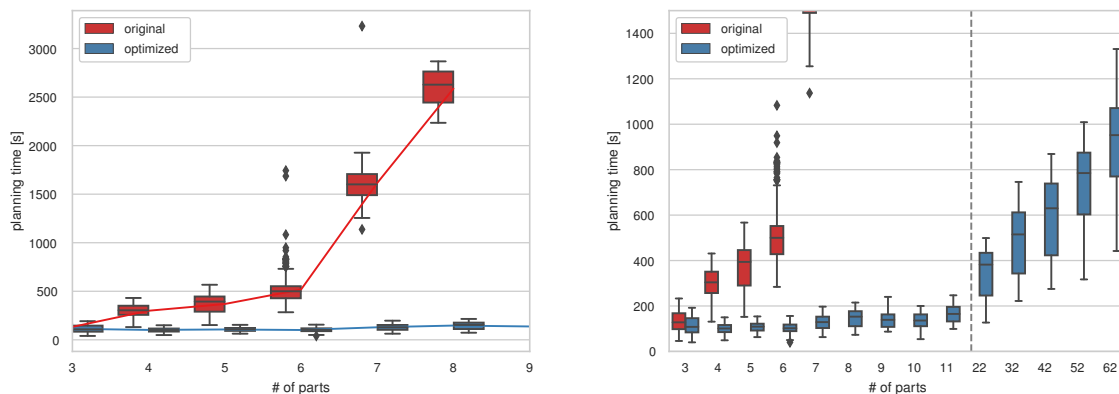


Figure 7: Planning times for the original and optimized domain for a task with three to 8 parts that need to be assembled. We can observe a near linear relationship between planning times and length of the task. However, the optimized domain scales significantly better than the original domain. Please notice the change in step size right of the dashed line.

hand side the number of necessary backtracking steps is depicted. This number grows for the original domain faster as well, as the additional steps that need to take place compared to the optimized domain add further points of possible failure during refinement. Therefore, not only the final plan length is shorter for the optimized domain, but also the planning process examines less dead ends during the search. This is also reflected in the planning times. Figure 7 shows them for each of the domains for different number of parts. We can observe that the optimized domain scales almost linearly even for a huge number of parts. The original domain performs good as well, however it has a significantly faster increase in planning times compared to the optimized domain. Note that even the original domain scales near linear, despite the very long plans of up to 300 steps. Non-hierarchical planners would typically scale exponentially with this plan length. The reason for this is that the complexity is handled on the abstract levels of our domain which results in very few calls to the very expensive trajectory generation. The linear scaling probably ends when the abstract planning problems gains more weight on the overall planning times than the problems on the refined levels which individually stay constant in size.

5 CONCLUSIONS

In this paper we presented a novel optimization algorithm for planning domains, that are represented in a set-based formulation. We use a hierarchical planner that makes use of this formulation and provides near linear scalability for our problem at the cost of non-optimal solutions. Poorly modeled domains naturally result in costly and computationally expensive solutions.

Our optimization approach reduces these effects and therefore allows for an easy composition of different models to an overall planning domain. This is an indispensable prerequisite for scalable industrial autonomy and flexible manufacturing.

An alternative to this would be perfectly aligned descriptions of each part of the overall domain. This, however, can only be guaranteed in small demo scenarios, which are designed by a single person or a small team. As soon as several groups or companies bring in modules which enable easy integration and alignment of the models must be enabled by algorithms. An important advantage of the explicit model of the domain compared to learned heuristics is the ease to debug and ability to explain the behavior of the algorithm.

The presented research is financed by the TransFit project which is funded by the German Federal Ministry of Economics and Technology (BMWi), grant no. 50RA1701, 50RA1702, and 50RA1703.

REFERENCES

- Areces, C. E., Bustos, F., Dominguez, M., and Hoffmann, J. (2014). Optimizing planning domains by automatic action schema splitting. In *Twenty-Fourth International Conference on Automated Planning and Scheduling*.
- Dietrich, V., Kast, B., Schmitt, P., Albrecht, S., Fiegert, M., Feiten, W., and Beetz, M. (2018). Configuration of perception systems via planning over factor graphs. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–7. IEEE.
- Haslum, P., Botea, A., Helmert, M., Bonet, B., Koenig, S., et al. (2007). Domain-independent construction of pattern database heuristics for cost-optimal planning. In *AAAI*, volume 7, pages 1007–1012.
- Kalashnikov, D., Irpan, A., Pastor, P., Ibarz, J., Herzog, A., Jang, E., Quillen, D., Holly, E., Kalakrishnan, M., Vanhoucke, V., et al. (2018). Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293*.
- Kang, T.-S. and Nnaji, B. O. (1993). Feature representation and classification for automatic process planning systems. *Journal of manufacturing systems*, 12(2):133–145.
- Kast, B., Albrecht, S., Feiten, W., and Zhang, J. (2019a). Bridging the gap between semantics and control for industry 4.0 and autonomous production. In *Int. Conf. on Automation Science and Engineering*. IEEE.
- Kast, B., Dietrich, V., Albrecht, S., Feiten, W., and Zhang, J. (2019b). A hierarchical planner based on set-theoretic models: Towards automating the automation for autonomous systems. In *Int. Conf. on Informatics in Control, Automation and Robotics*. SCITEPRESS Digital Library.
- Katz, M., Sohrabi, S., Samulowitz, H., and Sievers, S. (2018). Delfi: Online planner selection for cost-optimal planning. *IPC-9 planner abstracts*, pages 57–64.
- Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.
- Nägele, L., Schierl, A., Hoffmann, A., and Reif, W. (2018). Automatic planning of manufacturing processes using spatial construction plan analysis and extensible heuristic search. In *ICINCO (2)*, pages 586–593.
- Schmitt, P. S., Wirmshofer, F., Wurm, K. M., Wichert, G. V., and Burgard, W. (2019). Planning reactive manipulation in dynamic environments. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.
- Seipp, J., Braun, M., Garimort, J., and Helmert, M. (2012). Learning portfolios of automatically tuned planners. In *Twenty-Second International Conference on Automated Planning and Scheduling*.
- Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484.
- Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017). Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359.
- Vallati, M., Hutter, F., Chrapa, L., and McCluskey, T. L. (2015). On the effective configuration of planning domain models. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.
- Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., et al. (2019). Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354.
- Xie, A., Ebert, F., Levine, S., and Finn, C. (2019). Improvisation through physical understanding: Using novel objects as tools with visual foresight. *arXiv preprint arXiv:1904.05538*.

9

Hierarchical Planner with Composable Action Models for Asynchronous Parallelization of Tasks and Motions

Title	Hierarchical Planner with Composable Action Models for Asynchronous Parallelization of Tasks and Motions
Authors	Bernd Kast , Philipp Schmitt, Sebastian Albrecht, Wendelin Feiten, and Jianwei Zhang
ISBN/ISSN	978-1-7281-5237-0
DOI	IRC.2020.00029
Status	published
Publisher	IEEE
Contribution of Bernd Kast	Philipp Schmitt helped me to integrate his implementation of the constraint optimization algorithm, which is used in the operators for motion generation in the experiment. I implemented the rest of the model. I also developed and refined the theoretic foundations based on discussions with the co-authors.

Summary	<p>The separate approaches for discrete planning problems and continuous tasks are especially significant for domains where actions happen in parallel. While the scheduling community, which targets discrete optimization and assignment problems, provides algorithms optimized for various scenarios with parallel or asynchronous actions, no solutions exist for hybrid domains. A naive approach for those domains would be a task and motion planner, for which the state-space and, in extension, all actions that can operate in parallel are multiplied out. However, this results in a yet higher branching factor and computation times that prevent an application on relevant scenarios. Additionally, parallel actions are synchronized to the slowest operation in the current batch. With our approach, we can asynchronously parallelize our solutions and maintain the branching factor, which allows for an application even in plant-wide scenarios. The key to our algorithm is the heterogeneous state, for which our model tells us the relevant information for each applied action. By that, we can separate instances that define a partial sequence from information about the state at the time of application, which can be dynamically composed right before the call, and irrelevant parts of the current state that are not used or considered by our current operator. Combined with a scheduling scheme that considers this information for a later parallelization, we can apply the same hierarchical planning proposed in a former paper on these problems and only parallelize the sequential plan in a special refinement step for execution. Operators cannot use the same instances in parallel. However, the same instance can be considered by one and used by a second operator simultaneously. The operators must handle this dynamic modification internally. A constraint-based optimization algorithm ensured this integrity for our experiment, which involved two 7-dof real-world robots mounted next to each other.</p>
---------	---

Hierarchical Planner with Composable Action Models for Asynchronous Parallelization of Tasks and Motions

Bernd Kast¹, Philipp S. Schmitt¹, Sebastian Albrecht¹, Wendelin Feiten¹, and Jianwei Zhang²

Abstract—Task and motion planning is a relevant yet hard to solve problem in robotic manipulation. Large number of degrees of freedom with multiple manipulators and several objects require specialized algorithms, which can deal with the hybrid planning and optimization problem.

An additional challenge is the asynchronous parallelization of single robot actions on interacting manipulators.

In this paper we propose a system with a hierarchical planner, which solves the task and motion problem and optimizes for a subsequent parallelization. We use action models based on a constraint formulation; thus, the execution engine can parallelize the sequential plan without synchronization between different tasks.

In the experiment, we solve a task and motion problem with difficult geometric constraints and combinatorial complexity. The asynchronously parallel execution of that plan is demonstrated on a real world dual-arm robot.

I. INTRODUCTION

Many tasks in industrial domains require multiple collaborating robots. Traditionally, the synchronized movements and actions of the manipulators involved are programmed manually and triggered in a centralized manner. The setup of such a system is very time consuming, error prone, and therefore expensive. Additionally, these systems are inflexible and require reprogramming for minor changes in the product or the plant.

Our aim is to increase flexibility, such that small lot size production becomes viable. In order to achieve this, the system must compute all movements and actions autonomously.

Two major difficulties emerge from the requirement to plan each action and movement of the system. Firstly, a planner must be able to handle tasks with dozens of steps even for small problems. However, the computation time grows exponentially with the degrees of freedom involved and the number of discrete actions that are required to fulfill a given goal. To overcome this problem, strong heuristics that integrate expert knowledge as well as introspection capabilities for the planning algorithm are desirable for these kinds of tasks.

Our second requirement is parallelization on different agents. Most actions only require a single agent, such as a single robot arm picking an object. However, other operations like the dual arm screwing depicted in Figure 1 require multiple manipulators. We want a planner which computes motions for multiple agents. Furthermore, the system should

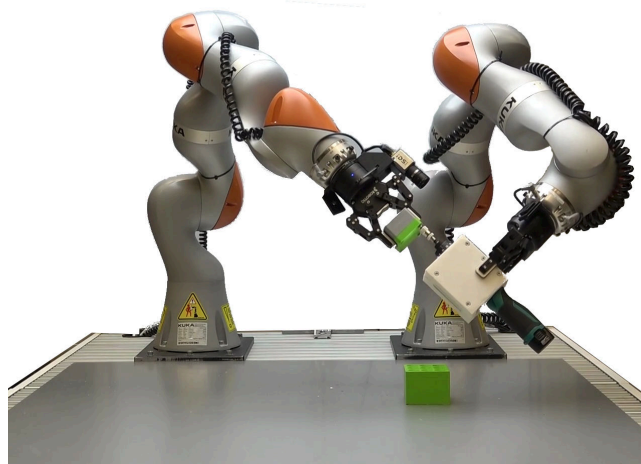


Fig. 1: Dual arm robot cooperating to fixate a lid on a box with a screw.

leverage parallelization opportunities, consider resource conflicts, and manage collaborative tasks.

In this work, we propose a system, which allows a parallelized execution of tasks with a constraint-based formulation on different, interacting robots. Our hierarchical planner orchestrates the actions even for long tasks, allows introspection and considers different resources such that the parallelization potential is considered during planning.

II. RELATED WORK

This work integrates and extends literature of two different strands. The first strand covers task and motion planning and discusses potential implementations and limitations for a parallel execution with existing algorithms.

The second strand is about the constraint-based motion formulation which eases asynchronous execution due to its controller structure, which results in reactivity and composability.

A. Task and Motion Planning

Task and motion planning is a hybrid problem with discrete and continuous properties, which can either be considered by a single or multiple composed planning algorithms.

Approaches that use specialized planners for the symbolic and geometric sub-domains include [1] or [2]. They either use a classic Strips planner [3] or HTN related planners

¹ Siemens Corporate Technology, Munich, Germany

² Faculty of Mathematics, Informatics and Natural Sciences, University of Hamburg, Germany The presented research is partially financed by the TransFit project which is funded by the German Federal Ministry of Economics and Technology.



Fig. 2: Comparison between sequential (first row) and parallel execution. The coloring denotes different resources used for some task (for example different robots which manipulate or localize). Note that the start of the tasks is not necessarily synchronized. The block, spanning two rows, is a task, which requires both instances of some resource, for example during handover both arms are blocked.

[4] on the task level. A very challenging requirement of these approaches is the discretization of most properties, like positions, which need to be considered on the task level. Real world applications fall victim for the combinatorial growth encountered in task planning problems. We use a hierarchical approach to handle the combinatorial complexity but instead of nested planners use a single planner, which can handle discrete and continuous properties in combination with local controllers for motion generation.

In [5] a kinodynamic task and motion planner is proposed, which also handles both domains with a single planner. The authors use a constraint-based local motion generation, which eases planning and allows for a reactive execution of the plan. Their actions operate on the complete task, which works well for problems with a small number of objects and a limited number of steps in the successful plan.

Asynchronous, parallel execution is difficult to handle with both systems. For the approaches with separate planners, both must consider time and parallelization. High level planners like Shop2 [6] can handle continuous time domains, however, the hierarchical decomposition prevents efficient parallelization.

The combined task and motion planner's actions operate on the complete state space. Thus, independent strands of actions cannot be identified, which prevents an asynchronous parallelization.

B. Constraint-Based Task Definition

Constraint-based task definitions offer a compact and composable form to specify robot motion [7]. Another distinct advantage is their handling of runtime disturbances, which were not or could not be considered during the planning phase. In combination with a planner, as presented in [5], even complex tasks can be solved very flexibly. However, parallel actions are not possible with this concept, as each task operates and possibly manipulates all resources of the current state. Therefore, task irrelevant and thus unused resources, like passive robots or objects, cannot be identified and operated on in parallel.

As the constraint-based controllers are resilient to disturbances, loosely coupled constraints can be composed and thus enable parallel execution of tasks. In [8] a scheduler is proposed, which identifies composable constraints and executes them in parallel. As this is a local optimization and

no planner is involved, it can only be part of a solution for a flexible asynchronously executing task and motion system.

The approaches presented in [9], [10] target time optimal manipulation. As a by-product the resulting solution is a parallelized task and motion plan. Their approach performs an offline trajectory optimization, which is not trivially extensible to account for sensory feedback and execution on real-world robots.

III. PROBLEM DESCRIPTION

In this paper, we target the generation of sequences of asynchronously executable actions which solve a task and motion problem. The three challenges therein are:

- task planning: handles resources, discrete properties and actions
- motion generation: generate collision-free movements in a high dimensional continuous space
- parallel execution on the real robots.

All parts are strongly interwoven and induce the requirement to strongly integrate the algorithmic components of the system.

For example, asynchronous parallel execution is only possible, if the motion generation can calculate paths or control the robots in parallel for different high level tasks without synchronizing start or end of the individual tasks as depicted in Figure 2. The high-level planning must consider geometric constraints, like reachability and graspability as well, which can only be thoroughly be achieved by an integration to the motion generation algorithm in simulation.

IV. HIERARCHICAL TASK PLANNER

For an asynchronously parallelizable system architecture, the task planning level must resolve resource conflicts in a fine-grained manner. We use a hierarchical planning approach for the orchestration and task and motion planning, which was proposed in [11]. It uses a heterogeneous state and strong typing, such that operations can act on a subset of the information available at the current state. As the actions define which subset of the current state they need and manipulate, resource conflicts can be identified and avoided during planning. This is a necessity to identify independent and parallelizable operations during execution.

The basic building blocks of our hierarchical planning approach are the description of the declarative knowledge

(concepts) and procedural knowledge (operators), which are then composed to a planning domain. We extract hierarchically ordered planning tasks from this model and solve and execute them recursively. The hierarchy not only allows scaling to long and complex tasks, but is essential for an efficient recovery and backtracking, when the execution failed due to non-foreseeable circumstances during execution.

A. Declarative Knowledge

We use a set-based model to describe the declarative knowledge (concepts) of the domain with a strong typing as described in [12]. From this formulation we can calculate a partial ordering *more general than* whenever a concept covers all instances of another concept. This partial ordering allows an automatic hierarchization of the domain which is used during planning to factorize the tasks.

A key element are the *compare functions* which are automatically generated based on the computed hierarchy. They calculate whether an instance is within the set described by a second instance. We use these functions to check for reached (intermediate) goals during planning.

B. Procedural Knowledge

All actions that can take place act on the previously defined concepts. They operate on a defined set of input instances and provide a set of output instances. The code to calculate applicability and the output is a black-box, such that arbitrary simulations and calculations can take place. Our operators not only cover movements and their abstractions, but any arbitrary operation like hardware initialization, localization, attachment of objects to the robot or execution on real hardware like gripper operations or arm movements.

The operators have a partial ordering like our concepts. In their case it represents a more detailed and therefore computationally more expensive simulation. A single operator can be refined by multiple more detailed operators. For example, the coarse operator "Pick" may be refined by "SampledMove", "Localize", "Approach" and "CloseGripper". In this example the refinement is not a simple replacement, but a planning task on its own, in which each operator may be required multiple times or not at all. We apply our hierarchical planning algorithm until each step is refined to the execution level.

The partial ordering of the operators can be compiled to a single *operator-graph*, in which the root nodes denote the most abstract operators and the leaf nodes represent the operators that are required during real world execution. In between are operators of every abstraction from symbolic to mixed discrete, continuous levels, with collision checking or even physical simulation.

C. Planning Algorithm

The planner must find a sequence of operators which transforms a given set of initial instances, which are pieces of information or resources that represent our state during planning, to instances that fulfill all the elements of the required set of goal instances.

There are two sub-algorithms involved to find this solution efficiently. The first defines the next sub-planning task based on the current planning status and the second solves it with a forward planner afterwards.

For the first sub-planning task, we allow only operators that are root nodes of the ordered operator-graph and are therefore the most abstract and computationally cheapest operators of our domain. This high level of abstraction ensures that we only need few computationally cheap steps to reach the goal thus we can find this coarse solution very fast.

After the planner returns a solution, we define a new planning task for each operator in the successful plan, which has children in the operator-graph. The goals of the new planning tasks are the outputs of the operator in the coarse plan and the operators that can be applied are the successors of this operator in the operator-graph. Important for the later parallelization is the selection of the initial facts for the sub-planning problem. We divide the pieces of information and resources of the current planning state in two groups, based on the inputs of the operator in the coarse plan. *Available facts* can be used, consumed, and manipulated by the operators of the subsequent planning task. *Blacklisted facts*, however, may only influence the outcome while operators can neither consume nor actively manipulate them. During planning, instances from the blacklisted facts are fed to the operator only in specially marked inputs. As an example, the refinement of a "Pick"-operation, which was applied in the coarse plan on a specific box, results in a sub-planning task with this specific box as an available fact and all other objects only as blacklisted facts. This way decisions made on the coarse level are enforced in the refined level, which limits the branching factor with the downside of sub-optimality. The objects represented by blacklisted instances can be considered by the operator which checks for collisions and therefore it can verify feasibility in the complete state. However, they are passive only and cannot be manipulated. This comes into play again during parallel execution, when the passive instances of one task can be manipulated by a simultaneously active task. Details about the calculation of blacklisted and available facts can be found in [11].

The second sub-algorithm involved in planning is the forward chaining. In this phase we identify inputs from the current planning state and apply operators on them which generates new instances and therefore a new state. Important for the parallelization and new compared to our former algorithm is the optimization for the expected execution duration that we introduce as a cost. Each operator holds an approximation of the expected real-world execution duration. For every resource and piece of information we note the expected time it becomes available. When we apply a new operation on instances the output's timestamps are the sum of the expected execution duration and the last time stamp of all inputs (blacklisted facts are ignored as they are passive). The virtual timestamp for each state is the maximum timestamp of all instances in this state. With that, we can pick the cheapest goal state later.

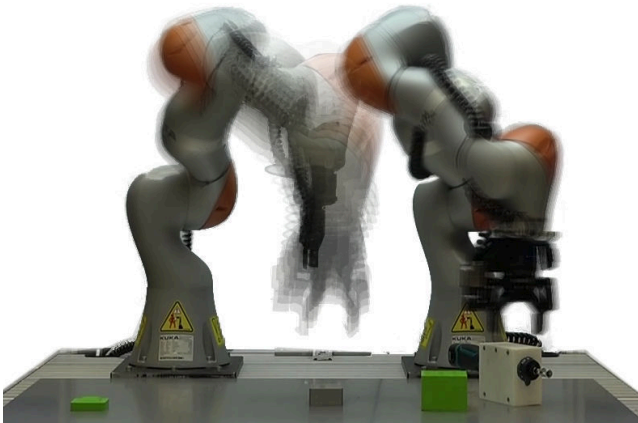


Fig. 3: The active left robot picks the box. The right arm must move to avoid collisions, even though it is passive at that moment.

V. ACTION MODELS FOR PLANNING AND PARALLEL, ASYNCHRONOUS EXECUTION

A core problem of planning and then executing parallel manipulation tasks for a robot lies in the complexity of modeling parallel motions. Let us review the requirements for such a model:

- **Complex Motions:** Motions for manipulation are inherently complex to model. Consider the exemplary screw driver task of Figure 1. This task involves two robots with a total of 14 axes but only five degrees of freedom are required to position the screwdriver. Yet, the two robots form a closed kinematic chain while inserting the screw and must not cause self-collisions or violate their dynamic limits.
- **Reactivity:** If the system behaves deterministically all motions could be pre-computed and then simply executed as a trajectory-following task. However, some operations, for example the operation of the screwdriver, have a non-deterministic runtime. For the asynchronous execution of multiple parallel tasks this means that it is not viable to pre-compute trajectories. Instead, **we need a reactive action model**, which can evade the other manipulator as depicted in Figure 3.
- **Composability:** It must be possible to compose multiple motion tasks sequentially, in parallel, and with partial overlap in time.

To address these requirements, we utilize a constraint-based task specification as an action model within our planner and during execution. This model is introduced in the next subsection followed by an instantaneous control-scheme that is used to translate the action model to acceleration commands for the robot.

A. Constraint-Based Task Specification

For our task specification, we call the current configuration of the system q , which is composed of object poses and positions of the robot axes. We describe positioning motions using a constraint function $f(q)$. This vector-valued function

is used as a non-linear inequality constraint on the configuration q

$$f(q) \leq 0. \quad (1)$$

Examples for these constraints include position-limits for the robot axes or collision avoidance formulated as a minimal distance between primitive shapes.

Naturally, two of such constraints with different signs can be combined to formulate equality constraints. Examples for equality constraints include a target position and orientation of the screwdriver relative to another object or the state of attachment of an object (for example grasped or placed).

Constraints can also be defined based on velocities via the constraint function $v(q)$.

$$\frac{d}{dt}v(q) \leq 0 \quad (2)$$

A typical example for this is an axis velocity limit.

Finally, we limit the acceleration of our system with

$$-\ddot{q}_{\max} \leq \ddot{q} \leq \ddot{q}_{\max}. \quad (3)$$

We can now describe two types of motion tasks as pairs of constraint functions: (f, v) . The first type of task is called default task (f_d, v_d) and comprises constraints that ensure safety and physical plausibility of motions. This includes collision avoidance, axis limits, and the fact that objects only move when attached to a gripper.

Additional tasks are the $k \in \mathbb{N}$ positioning tasks synthesized by operators in our planner. The i -th task, with $i \in \{1 \dots k\}$ has the constraint functions (f_i, v_i) . Note that each task may have a different number of constraints, which means a different dimensionality of f_i and v_i .

When multiple positioning tasks are executed in parallel a motion controller tries to reach a state (q, \dot{q}) , where all constraints of the positioning tasks are fulfilled. As soon as all constraints of one task are fulfilled, this task was successfully executed, and the execution engine may disable and replace them by constraints of another task. At no time may the constraints of the default task be violated.

B. Asynchronous Execution

We extended the control-scheme, which is based on [7], to operate on accelerations as proposed in [5]. This allows switching of constraints during motions while preserving continuity of the velocities.

We would like to guarantee the constraint dynamics

$$\ddot{f} \leq -K_p f - K_d \dot{f},$$

with diagonal matrices K_p and K_d that ensure a stable, at least critically damped system. The constraint function f_p is composed of all constraints imposed by the currently active tasks $f_p^T = [f_1^T, \dots, f_k^T]$. This is not always possible as positioning tasks may interfere with each other or the default task. Therefore we introduce a slack variable which relaxes the optimization problem:

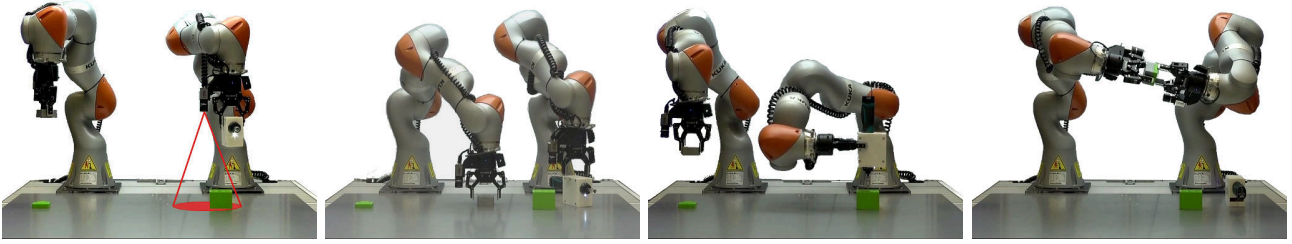


Fig. 4: Depiction of the single-robot actions locate, pick and screw out and the dual-robot action handover.

$$\begin{aligned} \ddot{f}_d(0) + \frac{\partial \ddot{f}_d}{\partial \ddot{q}} \ddot{q} &\leq -K_p^{f_d} f_d - K_d^{f_d} \dot{f}_d, \\ \ddot{f}_p(0) + \frac{\partial \ddot{f}_p}{\partial \ddot{q}} \ddot{q} &\leq -K_p^{f_p} f_p - K_d^{f_p} \dot{f}_p + \varepsilon. \end{aligned} \quad (4)$$

With these boundary conditions we solve the quadratic program:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & x^\top H x \\ \text{subject to} \quad & L_A \leq A x \leq U_A \\ & L \leq x \leq U, \end{aligned} \quad (5)$$

with the optimization variable $x = [\ddot{q}, \varepsilon]^\top$. It is composed of the slack variable ε and the control \ddot{q} , which we then use to move the robots in simulation for planning or the real-world during actual execution.

VI. PARALLELIZATION AND INTEGRATION OF EXECUTION AND PLANNING

The result of our hierarchical planner is a strict sequence of states, with operators that are applied on a subset of the available facts. In contrast to common state-space planners, it is transparent to our algorithm which instances an operator actively used. Therefore, we can relax the temporal ordering which was imposed by the planner and parallelize based on the data-flow.

Our execution engine algorithm iterates over the data-flow graph and starts every operator, for which all required inputs exist, in a new thread. Inputs can be available for three reasons:

- The instances were present right from the start as they are part of the start state.
- The preceding operator in the data-flow graph was successfully executed and returned the instance on the respective output to the data-flow graph.
- The input receives blacklisted instances and is fed with any unused instance of the defined type, which is unique to our design.

As described in subsection IV-C blacklisted instances are provided as an additional input to provide a complete representation of the current state. Therefore, our execution engine keeps track of the current state, which is composed of the available information and resources like manipulators or objects. It then separates the instances of this state

into available facts and blacklisted facts. We compare the structure of the data-flow graph of the current state with the graph that was calculated during planning. Outputs that end as "normal" inputs in the current operator are handled like available facts, all others in the current state belong to the blacklisted facts.

By that, actions that don't share any common resource are executed in parallel. Resources that influenced the result during planning, like additional objects or a second manipulator which was not directly involved in the current action are considered, but do not introduce a lock which prevents a parallel action. Combined with the optimization during planning, the execution engine algorithm allows parallel execution of independent strands effectively.

As the parallel execution changes the time different actions take place compared to the simulation when planning, some actions that interfere with each other may fail. We notice this deviation easily, as the result from the execution does not fulfill the intermediate goal of the coarser simulated level when we apply the compare function described in subsection IV-A. In this case, re-planning is necessary from the currently valid state, which is composed of the available instances after all running actions are finished, possibly returning an error state. We need to backtrack from this state then as described in [11].

In order to prevent repeated failures due to the ignored crosstalk of the tasks during planning, parallelization must be disabled for the next run. Our approach, with the hierarchical plan result allows to disable the parallelization for a small number of steps, until the next intermediate goal on the coarser level is reached. By that, parallelization can be used again even if it failed previously.

VII. IMPLEMENTATION AND EXPERIMENTS

We conduct our experiments on a dual-arm robot setup with 14 degrees of freedom. The goal of the task is to produce a box sitting on the table with a lid fastened on it by a screw. Each manipulator is equipped with a parallel gripper and a wrist mounted camera. The screwdriver used for this task is remotely controlled and can be picked by the manipulators.

To fulfill the task two 3D-printed parts, the box and the lid, must be localized and assembled. Then the screwdriver must be localized and picked up to fetch a screw from the magazine. After that, one arm needs to pick up the box-lid assembly and hold it in the air for the second manipulator to

insert and fasten the screw. The last step is to place the box on the table. As this is not possible with the screws facing down, a handover must be performed. For that the second arm must be freed by placing the screwdriver after which the handover and placement can be executed.

Box, lid, magazine and screwdriver are placed on the table without any fixation. Therefore, before every interaction, the current position must be updated with a localization procedure. For this refinement one arm is positioned above the estimated location of the object such that this position is in the view of the wrist mounted camera. After that an edge-based algorithm on the 2D-image updates the objects position.

The operators on the simulation and on the execution level, which are partially depicted in Figure 4 include:

- pick of an object with different grasp positions
- pick with both arms, which includes a handover and therefore results in another orientation of the object in the gripper compared to the normal pick
- assembly of the box on the lid
- placement of an object on the table
- placement with both arms, which performs a handover and puts the object on the table upside down
- arm movement, which is a necessary element of the successful plan, to prevent local minima of the controllers
- localization to update the current object position
- pick-up of a screw, which receives a screw from the magazine while rotating the screwdriver anticlockwise
- insertion of a screw into an object while rotating the screwdriver clockwise to fasten the screw

Additionally, operators to initialize the robots exist. The planner must calculate a successful plan with these operators without a predefined sequence. Physical feasibility of the actions, for examples if a grasp is possible without collisions, is ensured by testing convergence of the controllers in simulation.

From this list of operators, insert screw and pick-and-place with two arms require both arms. All other actions need only a single robot and can be executed in parallel, if they operate on different objects.

The execution engine runs the current set of constraints for 0.1 seconds, after which we check for convergence and achieved subtasks and update the set of active constraints accordingly. This high rate for the update of constraints ensures that new tasks can start any time the necessary inputs for the operator are available. The axes positions based on the accelerations of the constraint controller is commanded with 200 Hz.

In our example task, we can parallelize the first steps, until the screw is inserted. One arm can localize, pick and assemble the box on the lid, while the other robot localizes and picks the screwdriver to localize the magazine and pick a screw.

In our experiments, the possible parallelization is successfully exploited. An interesting observation is the crosstalk between the arms when tasks change. The manipulator with the longer running constraints is repelled by the second arm

with the new constraints to a degree, where it moves further away from its target, until it starts converging again which highlights the robustness of the controllers and our general approach.

VIII. CONCLUSION

In this paper we presented an architecture, for asynchronous task execution combined with a hierarchical planner for task and motion planning. Despite the sequential nature of the resulting plans, they are optimized with a subsequent parallelization in mind. The action model relies on a constraint-based formulation for the controllers which are robust to the deviations. Therefore, the synthesized controllers can be executed successfully even if the world state deviates slightly between execution and planning. This allows to parallelize the previously sequential plan and reorder independent strands of the plan. During execution, tasks are activated and finished independent of specific states of other tasks or the environment at a high rate of 10 Hz.

In future we would like to verify the success of the parallelized plan in simulation before executing it on the real hardware. This is not trivially possible, as timing must be simulated precisely even for non-manipulating actions like perception, opening/closing of the gripper, screwing, etc.

REFERENCES

- [1] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel, "Combined task and motion planning through an extensible planner-independent interface layer," in *IEEE Int. Conf. on Robotics and Automation*. IEEE, 2014, pp. 639–646.
- [2] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now," in *IEEE Int. Conf. on Robotics and Automation*. IEEE, 2011, pp. 1470–1477.
- [3] M. Helmert, "The fast downward planning system," *Artificial Intelligence Research*, vol. 26, pp. 191–246, 2006.
- [4] F. Bacchus and Q. Yang, "Downward refinement and the efficiency of hierarchical problem solving," *Artificial Intelligence*, vol. 71, no. 1, pp. 43–100, 1994.
- [5] P. S. Schmitt, F. Wirnshofer, K. M. Wurm, G. v. Wichert, and W. Burgard, "Modeling and planning manipulation in dynamic environments," in *Int. Conf. on Robotics and Automation*. IEEE, 2019.
- [6] D. Nau, T.-C. Au, O. Ilghami, U. Kuter, J. Murdock, D. Wu, and F. Yaman, "Shop2: An htn planning system," *Artificial Intelligence Research*, vol. 20, pp. 379–404, 2003.
- [7] E. Aertbeliën and J. De Schutter, "eTaSL/eTC: A constraint-based task specification language and robot controller using expression graphs," in *Int. Conf. on Intelligent Robots and Systems*. IEEE, 2014, pp. 1540–1546.
- [8] E. Scioni, G. Borghesan, H. Bruyninckx, and M. Bonfè, "Bridging the gap between discrete symbolic planning and optimization-based robot control," in *Int. Conf. on Robotics and Automation*. IEEE, 2015, pp. 5075–5081.
- [9] M. Toussaint, K. Allen, K. A. Smith, and J. B. Tenenbaum, "Differentiable physics and stable modes for tool-use and manipulation planning," in *Robotics: Science and Systems*, 2018.
- [10] M. Toussaint, "Logic-geometric programming: An optimization-based approach to combined task and motion planning," in *IJCAI*, 2015, pp. 1930–1936.
- [11] B. Kast, V. Dietrich, S. Albrecht, W. Feiten, and J. Zhang, "A hierarchical planner based on set-theoretic models: Towards automating the automation for autonomous systems," in *Int. Conf. on Informatics in Control, Automation and Robotics*. SCITEPRESS Digital Library.
- [12] B. Kast, S. Albrecht, W. Feiten, and J. Zhang, "Bridging the gap between semantics and control for industry 4.0 and autonomous production," in *Int. Conf. on Automation Science and Engineering*. IEEE, 2019.

10

Automatic Domain Extension and Optimization based on Set-Theory

Title	Automatic Domain Extension and Optimization based on Set-Theory
Authors	Bernd Kast , Vincent Dietrich, Sebastian Albrecht, Georg von Wichert, Wendelin Feiten, and Jianwei Zhang
ISBN/ISSN	978-3-030-92442-3
DOI	10.1007/978-3-030-92442-3
Status	published
Publisher	Springer
Copyright	Springer
Contribution of Bernd Kast	I conducted the algorithms, primary theoretic considerations, and experiments. During implementation, I had discussions with Sebastian Albrecht, Wendelin Feiten, and Vincent Dietrich. Additionally, Vincent helped with the implementation of the experiments. All co-authors brought in their experience for different use-cases and their specific challenges.

Summary	<p>Setting up models for new tasks and domains is a challenging engineering task, which has the potential to only shift effort from programming to modeling. In the context of scalable and flexible setups, we address the challenge of setting up models for new tasks and domains, which often leads to the common error of misaligned models. This error is particularly prevalent in domains composed of different sources, posing a significant obstacle for reusable systems. To mitigate this, we present an algorithm that algorithmically facilitates the modeling task. Based on our set-theoretical foundations, we transform the representations such that the planning task is eased and the solutions optimized without manual interference necessary. We present the algorithm, the discussion of its validity, and an experiment on a real-world robot. We analyze the improvement in planning time as well as execution time and discuss the reasons for those reductions and changed behaviors.</p>
---------	--

Automatic Domain Extension and Optimization based on Set-Theory^{*}

Bernd Kast¹[0000-0001-7838-3142], Vincent Dietrich¹[0000-0003-0568-9727],
Sebastian Albrecht¹[0000-0002-3647-4043], Georg von Wichert¹,
Wendelin Feiten¹[0000-0002-7593-6298], and Jianwei Zhang²

- ¹ Siemens AG, Corporate Technology, Otto-Hahn-Ring 6, 81739 Munich, Germany
bernd.kast@siemens.com
- ² University of Hamburg, Faculty of Mathematics, Informatics and Natural Sciences,
Vogt-Kölln-Str. 30, 22527 Hamburg, Germany

Abstract. Autonomous machines promise more flexibility and robustness changes in their environment compared to manually programmed solutions in industrial applications. However, the autonomous planning of actions involves discrete as well as continuous properties, which results in a np-hard planning problem. Especially for multiple machines and long planning horizons the design of domains requires a lot of fine tuning and thus manual effort. Modularization and reuse of existing domain knowledge with formalized models is one solution to this issue. However, the models of different projects tend to be misaligned, in particular when several parties contributed to the project, which deteriorates the performance.

In this paper, we present two domain optimization and extension algorithms, which adapt the models to facilitate planning. The first algorithm handles inconsistent units, or even misaligned pieces of sub-information. It automatically generates conversions and allows to call operations with a wider range of input types. The second algorithm aligns models from different sources with varying modeling views. After this reformulation, we can compose models more efficiently to a larger domain.

For both optimizations, we rely on the formal set-based models that we also use in our previously presented hierarchical planning algorithm. Our hierarchical approach allows an almost linear scalability with the length of the plan. However, it comes with non-optimality effects due to the imposed intermediate goals that depend on the quality of the model. The optimization algorithms of this paper allow to adapt and extend the model so that valid shortcuts reduce these suboptimalities. We conduct experiments on a task and motion assembly problem, demonstrating scalability for up to 62 parts and plans with over 1000 steps, which either result in discrete state or high-level position changes, with planning times of less than 15 minutes. Our experiments also include the successful plan execution on a real-world dual arm setup.

Keywords: Domain Optimization · Automatic Data Fusion · Hierarchical Planning · Robotic Assembly

^{*} The presented research is financed by the TransFit project which is funded by the German Federal Ministry of Economics and Technology (BMWi), grant no. 50RA1701, 50RA1702, and 50RA1703.

2 B. Kast et al.

1 Introduction

Robotics is the melting pot for various engineering disciplines. From the mechanical design over control theory, and perception to decision-making algorithms, components from different sources have to be composed to obtain a seamlessly working system.

At the moment engineers bring together the hardware and software manually. They must select, adapt, and configure the right components to solve the task at hand. This process is both time-consuming and expensive, so that automation is currently only considered for recurring tasks in industry. In addition, the systems have limited flexibility so that they can react to disturbances but not to changes in the setup, environment, or task. Then again, qualified and expensive engineers are required to reconfigure the system even if the hardware would still be able to solve the task.

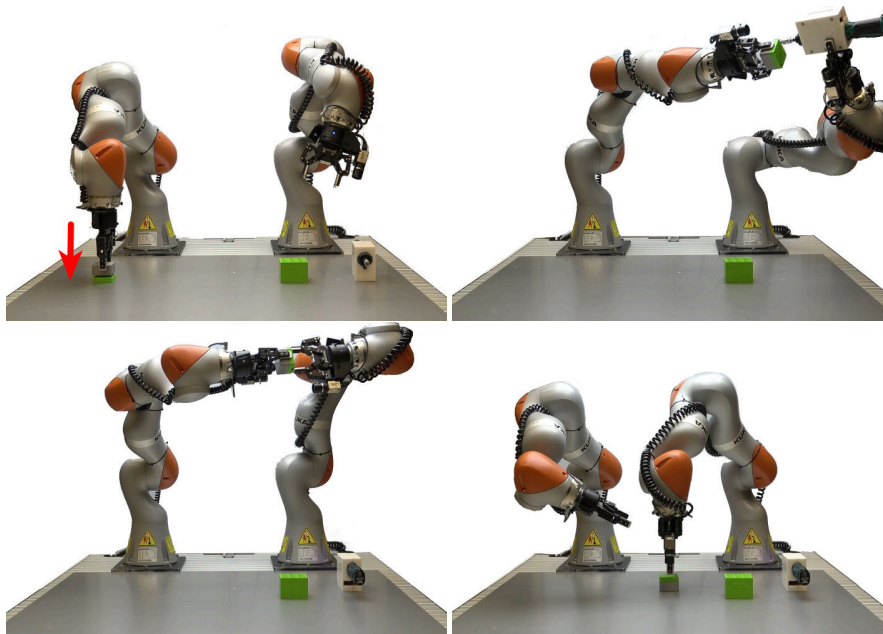


Fig. 1: In our experiment we conduct a task and motion planning problem on a dual arm robot, for which we conduct the assembly, screw, handover, and place operations. Originally published in [11].

These costs counteract the benefits of the automation and prevent a broad application of robots. The trend towards highly individualized products, shorter product life cycles, and expensive workforces increases the desire for automated small lot sizes, or lot size one production. Such a highly flexible production requires general purpose manipulators equipped with multi-modal sensors and algorithms for decision-making, perception of the environment, and reaction on errors.

The software components of such a system are becoming increasingly complex. To reduce the effort of setting up or adapting to new problems, we must reuse as many components as possible, develop multi-purpose algorithms that can replace specialized solutions, and allow modularization so that several parties can contribute reusable modules.

In this paper we build on algorithmic decision-making algorithms, which use models that standardize domain knowledge. These planning algorithms allow for a modularization of the domain and consider sensory input to control the hardware and perform the task at hand solely based on the provided models.

The algorithms for decision-making must handle hybrid problems like the manipulation scenario shown in Figure 1, which has continuous geometric and discrete properties, such as attachment states or grasping positions. These planning problems become large for non-trivial tasks, which leads to unreasonable computation times due to the curse of dimensionality. We counter that with the hierarchical approach presented in [10].

Theoretically, recursive hierarchization allows for a near linear scalability with the length of the task. Nevertheless, the performance of the planning algorithm depends strongly on the quality of the models. However, manual model optimization is exactly what we want to avoid. The advantage we have are the formal models, which can be automatically optimized with different algorithms presented in the following.

In this paper we focus on the automatic and facilitated integration of components from different parties. We discuss two algorithms that optimize a domain composed from mismatching modules. Both rely on our formal models and their set-theoretic foundation to modify declarative and procedural knowledge while preserving their integrity.

Especially in an industrial environment integrators, component suppliers, and manufacturers define objectives or contribute modules and resources with varying perspectives depending on their user role. Each of them implements a slightly different modeling scheme that represents the individual approach to the problem.

The first domain optimization algorithm we discuss in this paper is a generalized form of an automatic unit transformation. If two modules use different units, functions from one unit cannot be called with data from the other module. They are incompatible and, despite formalized information to reach a given goal might exist, there is no possible path that can be calculated by the planning algorithm.

This algorithm that was first proposed in [11] prevents performance degradation during online planning. Our algorithm automatically adapts the different declarative models by transforming them according to the task at hand. In this way, we avoid unnecessary intermediate steps, which are an artifact of the hierarchical approach that allowed us to handle these large domains.

In this paper, we also propose a second algorithm, which was not previously published and thus an extension to our previous work of [11]. It analyzes the domain and reformulates the existing declarative knowledge by decomposing it, applying general conversion operations to a subset of the pieces of information and recombining them again. We automatically generate conversion operators, which are checked for validity based on the formal models. In this way, we can merge subdomains with so far incom-

4 B. Kast et al.

patible formulations for example, due to different units used in the declarative knowledge. We algorithmically extend the domain such that conversion operators ensure the compatibility.

In the following we discuss the related work and theoretic foundation of our algorithms. Based on this we introduce our optimization algorithms, the extended conversion approach, and the automatic alignment. Afterwards we analyze the effectiveness of our algorithms in a dual-arm manipulation scenario. We simulate 800 assembly tasks with increasing complexity and conduct the final experiment on a real two-arm robotic system. Finally, we conclude our work and discuss possible future extensions.

2 Related Work

The three strands of literature related to the algorithms presented in this paper are data fusion, data-driven optimizations, and model-based domain optimizations.

The crucial step of our conversion algorithm is the recombination of the different pieces of information without infringing the validity of the model. This problem is like the (sensor) data-fusion task, which is a well-studied domain. In fact, Dasarathy et al. formulated the sensor-data-fusion as the inverse of data fission in [3]. Fusion and fission of data are two of the three important steps we conduct in one of our domain optimization algorithms. The difference to most sensor-data-fusion approaches is, however, that they mostly focus on the fusion of redundant information, according to the classification of algorithms presented in [5], to increase the confidence of the state estimate. This requires a good sensor model to weigh the different pieces of information, that possibly contradict each other. This model is not available in general, such that we must make sure, that we only fuse completely complementary pieces of information, which cannot conflict as they do not overlap. Another relevant field of research is data analysis for the internet of things [2]. The difference to their data-fusion scheme, which can be similarly model-driven than our approach, is the absence of procedural knowledge. With this additional information, we can make statements about the consistency of the fusion given a specific domain and provide operators that combine the fission-conversion-fusion-process.

An overall different approach to our model-driven domain optimization algorithms are data-driven approaches. Among this recently quite active domain, reinforcement learning, which provides an optimized heuristic for a given domain, is the branch with the most similar target. Their strength lies in domains with lots of training data or easy to simulate problems like board [17], [18] or computer games [13], [20]. In these scenarios they can outperform humans or even tailored algorithms.

A limiting factor is the training data, which prevents application on real-world scenarios or any domain, which is hard or impossible to simulate. In [7] the great effort and cost to gather enough training data in real-world becomes evident. Hundreds of robots performed pick and drop tasks for weeks in parallel and shared the experience, until the task was completed successfully for this specific setup. In our system, we aim to reuse the knowledge compiled in the model. With the abstractions and implementations made, we target a more flexible solution without training runs before operation. This, however, comes at the cost of possible suboptimality. Another impressive demonstra-

tion of the capabilities of data-driven approaches can be found in [21], which is a very hard but rather short task. With our hierarchical approach, we can factorize the problem, such that we can scale near linearly with the length of the solution and calculate plans with hundreds of steps, possibly integrating data-driven approaches for specific sub-problems.

As soon as we abstract the low-level machine control, the relative amount of data increases which eases data-driven approaches. In [15] this was exploited by combining reinforcement learning with an constraint-based motion-generation. With this layer, not only simulation and therefore data-generation is eased. It also reduces the dimensionality of the problem and avoids any collision. This expands the valid value range of the control parameters, which in turn accelerates the training. Another benefit of this abstraction is the identical interface for the training and the execution in the real-world, which additionally handles small deviations during the runtime.

A downside of this approach is still the high demand of processing power to generate the data and conduct the training. Especially for tasks with a high number of required steps, the time until a first successful path is found by chance, without any prior knowledge, can consume a lot of computation time. Opposed to our model-driven approach, the decision-making process is performed by a black box network, which is trained on specific data. It is therefore harder to debug and introspect. Additionally, a retraining is required as soon as changes in the setup or environment occur that were not included in the original training set.

In our approach, on the other hand, the hierarchization allows to solve large scale problems online without prior training. However, the introduced intermediate goals, which factorize the planning problem and enable this performance, possibly result in sub-optimal solutions. Opposed to this, reinforcement learning approaches are theoretically optimal given infinite computational power. With our explicit models, rules, and the factorization, the planning algorithm is introspectable and thus easy to debug, opposed to data driven approaches. This not only eases development but is a key requirement for many industrial applications.

The third strand of related work, apart from data-fusion and data-driven approaches, targets the design and optimization of modeled domains.

A best-practices scheme for manually constructed domains is presented in [8]. It targets the representation of the declarative knowledge and offers consistent models. It requires a strict adherence of the implementation to the defined rules, which is not always possible, due to different parties contributing to the overall system, or existing modules, that should be reused and incorporated. In those cases, manual effort to wrap, adapt, and integrate the non-conformant modules is required.

We accommodate this problem with the automatic alignment algorithm discussed in this paper. It allows each party to use the preferred representation and automatically integrates the modules based on the task at hand, without additional manual effort. By that, we facilitate collaboration as each party can fulfill their user-roles.

Several approaches target the problem of matching task and planning algorithm, primarily for discrete, PDDL domains. As planners differ in their search strategy and applied heuristics, their performance depends on the structure of the domain they are applied on and the parameters used. Therefore, portfolio planners, such as [16], [12],

6 B. Kast et al.

wrap around existing algorithms and apply different configurations on the given domain, analyze the behavior for a limited time and try to pick the best planner, without modification of the original domain. In competitions, portfolio planners have shown a very good performance. Our architecture is similar to the portfolio planners in the sense, that we have a planning algorithm that wraps, parametrizes and calls another planner. While different planners are possible in principle, we did not focus on the parametrization or the choice of the planning scheme at the moment.

Another approach to match planner and domain is to reformulate the domain to work efficiently with a given planner. The construction of efficient PDDL-domains requires experienced experts. With the optimization algorithms proposed in [6], [19] the hurdle is lowered, such that non-experts can automatically reformulate a domain for efficient planning with generic algorithms.

A benefit of offline optimization schemes, such as [1], which automatically applied common optimization schemes on a given domain, is the possibility to inspect the resulting domain. By that, validation, and further manual optimization, as well as a combination with portfolio planners, becomes viable.

The downside of the domain optimization schemes, and portfolio planners is related to the difficulties PDDL and PDDL planners have with hybrid domains, that also contain continuous properties, and the application of arbitrary code to control real-world hardware. In our approach, we handle the scalability issues of hybrid domains with the factorization and allow to call arbitrary operations.

3 Theoretical and Algorithmic Foundation

In this section, we discuss the foundations of our domain optimization algorithms, namely the definitions and representations we use for the declarative and procedural knowledge, as well as the hierarchical planning approach we optimize for.

3.1 Declarative Knowledge

We base the definitions of our declarative knowledge on the set-theory as proposed in [9] section III A. This is not only an intuitive approach, related to the human language and way of thinking, but also a way for machine-interpretable and thus algorithmically modifiable models.

Our definitions can describe physical objects as well as non-tangible pieces of information. We call these pieces of information *instances*. In our declarative knowledge, we aggregate instances to sets, which we call *concepts*. They are the building blocks of our declarative knowledge. All concepts have a specific concept base B_Γ which is a not necessarily finite set of instances that defines to which concept class Γ the concept belongs.

For concepts of the same concept class, we can define a partial order *more-detailed than* \mathcal{M}_Γ , which tells us, if the first concept is a subset of the second concept.

We can define sub-concepts in the set of a concept to aggregate instances with common properties. This recursive approach leads to so-called *composite concepts*, which

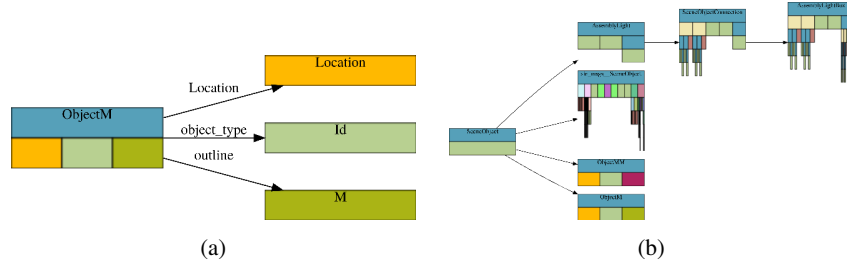


Fig. 2: The face of a concept is composed of the colors and recursive structure of its sub-concepts. In the hierarchical domain, the directed edges denote the more-detailed than property.

are a intersection between multiple concepts, that specify the value ranges or sets of a specific property or *role*:

$$C \cong \prod_{r \in R_C} C_r,$$

where R_C defines the composite structure and C_r with $r \in R_C$ are the intersecting sub-concepts and their role. The more roles the composite concept has the more detailed it is, as it represents a region of instances and is thus more precise. The same is true, if the intersecting sub-concepts are more detailed themselves. Based on that, we can calculate the partial ordering *more-detailed than* \mathcal{M}_T for composite concepts as well. Confer section III A of [9] for more information.

We present our concepts as directed graphs to the user. The root node holds the concept base and the descendants are the sub-concepts with the roles on the edges. The different sets are denoted by colors, which results in a recursively splitting *face* for composed concepts as depicted in Figure 2a.

When we bring together the partial ordering with the compact representation of our concepts, we can visualize the declarative knowledge in a hierarchical domain, which is a forest of directed subgraphs as depicted in Figure 2b. We use this ordering, in combination with automatically generated *compare functions* based on this model, to check for the fulfillment of intermediate goals during planning, and to compile the sub-planning tasks during factorization.

3.2 Procedural Knowledge

The declarative knowledge on its own would be completely useless, as no one could use it without the procedural knowledge. During planning, we apply actions to a subset of the information in the current state until we reach the desired goals. The aggregation of instances to different concepts is motivated by similar expected outcomes of the actions and the procedural knowledge is defined by its effect on the declarative knowledge. Thus, procedural and declarative knowledge go hand in hand and modeling only one in isolation would make no sense.

8 B. Kast et al.

We call the building blocks of our procedural knowledge operators. They describe the mapping between input instances of specific concepts and the possible instances of defined output concepts as described in [9] section III B and depicted in Figure 3. Thus, an operator π with $R_{\pi,\text{in}}$ describing the set of input roles and $R_{\pi,\text{out}}$ the set of output roles with the according concepts has the following structure:

$$\pi : \prod_{r_i \in R_{\pi,\text{in}}} C_{r_i} \rightarrow \prod_{r_j \in R_{\pi,\text{out}}} C_{r_j}.$$

We have little restrictions on the way the mapping between input and output instances is implemented.

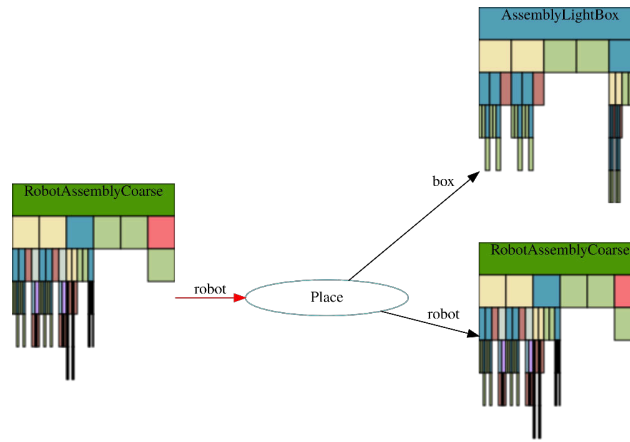


Fig. 3: The model of an operator defines one or multiple inputs and outputs, their types, as well as additional properties. For further reading consult section III B of [9].

We only impose that inputs cannot be modified, unless explicitly stated, no internal state is allowed, unless explicitly stated, and the types of all possible outcomes must be defined. This allows us to support explicit formulas, implicit black box implementations of arbitrary libraries, simulations, and real-world executions.

Similar to the partial ordering of concepts, we define the more-detailed-than relationship for operators. For the declarative knowledge, however, it is not necessarily a one to one, but a one to n relationship. We call a set of operators $\tilde{\pi}$ more detailed than an operator, as soon as a network $\hat{\pi}$, which can also be a sequence, exists, for which we can find a more detailed concept on the leaf nodes for each output of our operator. Additionally, the concepts on the root nodes of the network must either be more detailed than a input of the operator, or it must be orthogonal to all input concepts, i.e. it must have a concept base, which was not used in any of the inputs. According to section III B in [9] this requirement can be formalized in the following way:

$$\forall r_i \in R_{\pi,\text{in}} \exists r_j \in R_{\tilde{\pi},\text{in}} : (\hat{C}_{r_i}, \tilde{C}_{r_j}) \in \mathcal{M}_{\Gamma}(\tilde{C}_{r_j}),$$

$$\forall r_i \in R_{\tilde{\pi}, \text{out}} \exists r_j \in R_{\tilde{\pi}, \text{out}} : (\hat{C}_{r_i}, \tilde{C}_{r_j}) \in \mathcal{M}_{\Gamma(\tilde{C}_{r_j})},$$

and for all $r_i \in R_{\tilde{\pi}, \text{in}}$ holds:

$$\begin{aligned} & |\{r_j \in R_{\tilde{\pi}, \text{in}} \mid \Gamma(\tilde{C}_{r_j}) \subseteq \Gamma(\tilde{C}_{r_i})\}| \\ &= |\{r_j \in R_{\tilde{\pi}, \text{in}} \mid \Gamma(\hat{C}_{r_j}) \subseteq \Gamma(\tilde{C}_{r_i})\}|. \end{aligned}$$

The goal of the partial ordering of operators is the representation of a stepwise increase in simulation detail, complexity, and hence cost, which can be computational time or usage of the real hardware without the need to introspect the operators. We can additionally specify a partial ordering between operators, which does not interfere with this condition, manually.

We use this partial ordering for the factorization of the planning problem in the next section.

3.3 Hierarchical Planning Algorithm

The basic strategy of our hierarchical planning scheme, first proposed in [10], is a recursive divide-and-conquer approach to alleviate the curse of dimensionality. It consists of two distinct algorithms. The first is a single-level state-space forward planner, which takes a set of operators and initial instances to reach defined goals. We could replace this strategy with existing PDDL-planners, reinforcement learning algorithms, or task and motion planners.

The second algorithm configures those planning algorithms and conducts the factorization of the planning problem. There are three factors, which determine the computational requirements for a planning task: The number of possibilities per step, the number of steps required to reach the goal, and the cost per step. While the cost per step contributes only linearly to this calculation, the length of the plan and the branching factor are exponential facets. Hence, our primary goal is the minimization of the latter two aspects with the help of our factorization.

We start with the most abstract domain for a given problem, as it should be relatively simple to solve. It has only a small set of available operators, which have a small number of inputs and therefore few input combinations. Additionally, they cover huge steps towards the goal. This results in a relatively small branching factor and possibly small successful plan lengths.

We then pick one of the solutions and refine each of the steps in that plan recursively. We construct the new planning task with the more-detailed operators of the applied action in that step, reduce the set of usable instances based on the input instances, and define the former outputs as the new goals. We continue this recursion, until no more-detailed operators exist. In our model, the real-world execution is a more-detailed operation of the most detailed simulation. Therefore, we seamlessly integrate real-world execution in our hierarchical refinement scheme.

With our recursive factorization, we reduce the branching factor and the effective plan length, which allows us to calculate solutions of an idealized domain in linear time. This, however, only holds true, if the intermediate goals imposed by the abstract layers are on the direct path even in the most detailed level. As we stick to the decisions

10 B. Kast et al.

made on the more abstract level during our refinements, and the abstractions might not cover all possible outcomes or problems, the plan in the refined level might be sub-optimal or not even exist. Opposed to HTN planners such as [14], we do not rely on the downward refinement property, which would require the consideration of all aspects of the most detailed level on all abstractions to avoid non-refinable steps. However, this breaks nearly all benefits of the hierarchization in realistic setups. To avoid this problem, we implement a backtracking strategy, such that another abstract plan is picked, if the former choice was not refinable. For more information confer section 7.2 of [10]. During execution, this results in a naturally emerging model-predictive-control scheme as soon as deviations between execution and planning occur. However, we cannot avoid performance degradation for misaligned abstract planning levels, which provide poor intermediate goals and therefore misleading hints for the more detailed levels. Since the problem without the downward-refinement property is np-hard, we can observe exponential scaling in some misaligned domains.

To accommodate this issue and ease the engineering, we discuss the automatic alignment algorithm in subsection 4.1, which improves the models coming from different sources and thereby minimizes the sub-optimality effects of the intermediate goals and simultaneously improves the planning times.

4 Algorithmic Domain Optimization For Hierarchical Planning

In this section we discuss our domain optimization algorithms, which facilitate the engineering of planning domains for hierarchical planning. We focus on the alignment of models that were implemented with different views on the same physical aspects. This is a common issue when we reuse models or compose a domain from different sources.

The first algorithm aligns a given domain to the task at hand. It reformulates the declarative knowledge and wraps the operators, such that the outputs and therefore intermediate goals have the same concept-base as the final goal. This prevents unnecessary steps, facilitates planning, and improves the quality of the results. We proposed this algorithm first in [11].

The second algorithm addresses differing units or sub-concepts within two concepts, which prevent the application of operators, despite conversions between this pieces of sub-information exist. After this optimization, the planning algorithm can apply a larger set of actions, and thus solve additional problems, or find shorter paths to the goal.

4.1 Automatic Alignment of a Domain

The basic concept of our alignment algorithm is strongly interlinked with the principles of our hierarchic planning algorithm. The outputs of the operators on the more abstract level are the new goals for the refined level. During planning, we constantly check, whether the available instances fulfill this specified goals with the automatically generated *compare function* of the goals. This compare function checks, if one instance has a non-empty intersection with the goal instance, which can only be detected, if both instances have the same concept base.



Fig. 4: For the relation between robot and box, the same situation can either be expressed hardware, or object centric. E.g. either the box has a robot attached, or the robot holds the box. Originally published in [11].

The issue in misaligned domains is, however, that for the same physical phenomenon, different modeling solutions might exist. Especially when we describe the relation between two objects, one of them is the primary, which defines the concept base, and one object is the secondary, which adds detail as a sub-concept. We need this distinction and cannot work with triples, for example, for performance reasons, as the hierarchization would be impossible to calculate with that representation. We can, however, automatically convert one representation to the other, for example if we reuse modules or receive models from a third party, which had a different view on the same problem due to the differing user-roles.

Let's consider the example of a box, which is currently fixed to a robot's gripper. When we model this situation, we define a *box*-concept, possibly with an id and additional properties. As soon as it is grasped, we add the robot to this concept, such that we know, which robot it is attached to. Another approach would be to start from the robot, which initially has an empty gripper. As soon as it grasps the box, the list of attached objects would hold the box.

Both representations are perfectly valid. The first is possibly preferred by an engineer, who constructs the high-level task and ignores the specific hardware involved. The later will be the chosen representation for lower-level motion planning and control algorithms, which are commonly hardware centered. This duality of expressing the same instance with different base types is depicted in Figure 4.

During our hierarchical planning, we now have the abstract layer, which specifies the sequence in which different parts are assembled with instances of the base type *box*. The operators, that implement the task and motion control, however, take and return boxes only when the objects are picked, placed, or refined. But not when they are manipulated in the grippers. Therefore, the robot must place the box every step to reach the intermediate goal.

We could prevent this unnecessary step by reformulating the robot with attached box to a box with robot. The formal process to achieve this algorithmically and show the validity is based on the set-theoretic foundation of our model.

The algorithmic solution for the misalignment of different levels reformulates all concepts in a given domain, which *have* a sub-concept with the same concept base

12 B. Kast et al.

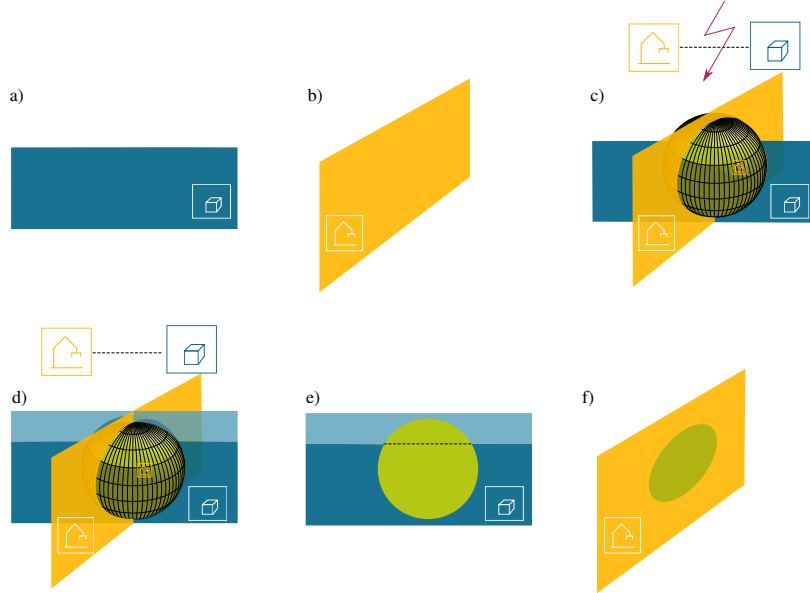


Fig. 5: Three-dimensional representation of our optimization approach. Originally published in [11]. Each depicted plane stands for a concept base (dark blue: all objects (a), yellow: all robots (b)). Instances with a connection between those planes are within the green sphere (c). As we cannot represent this volume with our concepts, we need to project it to either plane, which is possible for the yellow, but not for the original blue plane in (c). Therefore, we extend the blue plane with the empty set, according to our algorithm (d) and can now represent instances of the sphere with either concept base (e), (f).

as the goal concept, to be a concept with this concept base. Thus, we search for concepts $C \cong \prod_{r \in R_C} C_r$ that have a sub-concept C_j , $j \in R_C$ with the same concept base as our overall goal, i.e. $B_\Gamma(C_j) \subseteq B_\Gamma(C_{\text{goal}})$. In the next step, we transform each of those concepts, preserving the set they describe, to be of the concept base of the specified goal:

With $C_j \cong \prod_{r_j \in R_{C_j}} C_{r_j}$ the transformed concept is defined by:

$$C' := \prod_{r_j \in R_{C_j}} C_{r_j} \times \prod_{r \in R_C \setminus \{j\}} C_r.$$

With this definition we constructed a concept C' that is isomorph to a subset of C_j , which adheres to the definition of a concept with concept base $B_\Gamma(C_j)$ according to subsection 3.1.

As depicted in Figure 5, this transformation is only true if all elements of the original concept are covered by the new concept base. However, this is not trivially true in all cases. Consider our example of the box and robot again. The composite robot-concept has an array of attached objects as a sub-concept, which can be empty. In this

empty case, we cannot find a *no-box*-instance in the box-concept set, as it does not exist naturally unless we extend the original concept base of boxes with the empty set:

$$B_{\Gamma}^* := B_{\Gamma}(C_j) \cup \{\{\emptyset\}\}.$$

With this extended concept base, we can now construct the robot with empty gripper as an element of the concept base box, starting from the empty set element:

$$C' := \{\emptyset\} \times \prod_{r \in R_C \setminus \{j\}} C_r.$$

This corresponds to an *no-box-object* with all attributes of the original *robot* as a composed sub-concept.

The graphical representation of this process is depicted in Figure 5. With this formal transformation, we can now align all concepts in our domain algorithmically such that we can reach intermediate goals more directly, which in turn reduces planning times and increases the quality of the planning results.

As the optimization is dependent on the specified goal, we perform it online prior to the planning. We algorithmically analyze the domain, identify the concepts for the optimization and search for operators, that use these concepts as inputs or outputs. We then automatically generate conversions between either form of representation and wrap the original operators to accept and return the aligned representations of the concepts. We extend the original domain with the new operators, such that the planner naturally uses them to construct the new solutions. By that, the intermediate goals have a new type, as they are defined by the outputs of the aligned operators. These goals are easier to reach and thus relaxed, which results in different plans, while the overall goal keeps the original type such that the overall result is the same as without our optimization.

4.2 Extended Unit Conversion and Reasoning for Data Fusion

In this subsection, we start with the discussion of algorithms that solve small inconsistencies within the declarative knowledge, such as different units for physical parameters. We then continue with approaches that perform a general data fusion as well as the generation of new operators and for this purpose analyze the procedural knowledge in combination with the declarative knowledge.

Unit Conversion Even in the event that we reinvent engineering, spend time, brains, and money to define every process and data element that needs to be considered during the engineering, automation, and task specification process, we will probably still not be able to establish a standard that everyone adheres to. The more realistic scenario is that we must apply our algorithms in a "brown field" environment with existing implementations to various domains, from the private sector to health care and industrial applications. For this purpose, we want to reuse the existing tools and algorithms of a new plant with the already standardized knowledge of other use-cases while keeping the integration effort as low as possible. Naturally, inconsistencies will occur during this integration such as different units for the same physical quantity. Despite the fact that in

14 B. Kast et al.

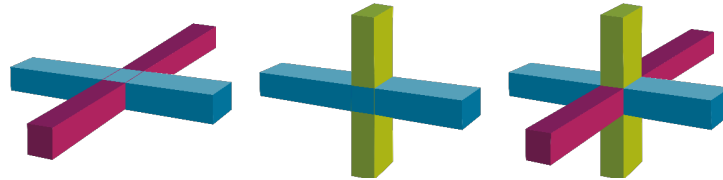


Fig. 6: Our concepts represent sets, which we can visualize by volumes in the 3d-space. In this depiction, the blue cuboid represents the abstract concept *box*. When we define additional properties with roles and respective sub-concepts, we intersect this volume with the volumes of the sub-concepts. This results in a smaller set of instances. In this example, we intersect the box concept (blue) with either the dimensions in mm (purple) or in m (green). We can observe, that the resulting volume is the same, as we can find a 1:1 mapping between either representation, i.e. a transformation, which transfers one volume to the other (in this case a rotation of around the axis parallel to the blue cuboid).

our models the sets of instances described by values of different units are isomorphic, the planning algorithm cannot directly execute operators with concepts that do not use the specified unit as depicted in Figure 6.

Therefore, we need a conversion to the other representation before we can execute the given operator. Classically, engineers program these interfaces manually for each integration, which is a time consuming and costly process. We propose an algorithmic solution to these conversions, based on the models of the domains that are involved during the integration process.

Consider the example of a pick operator that needs a box with its outlines specified in *mm* and an instance that comes from another level of abstraction and uses *m*.

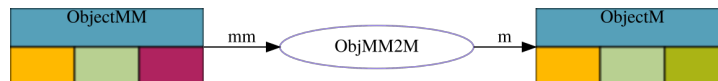


Additionally, we have a conversion operator that calculates *mm* from *m* for arbitrary distance measures, which we need to model and implement only once.



However, when we have an instance of a concept that *has a dimension* defined by one of the two units, we cannot directly apply this generic conversion operator, since

only a concept that *is a* dimension can be converted. For the conversion, the concept base is wrong, which prevents the application of the operator. Therefore, we need a specialized operator that converts the ObjectMM to an ObjectM.



The implementation of this operator would apply the conversion operator on the sub-role and keep all other information untouched.

A naive approach for this automatic unit conversion would be the isomorphic transition of subsection 4.1. This way we can change the concept bases and thus apply the conversion operator to the dimension, that has a *box*. However, this will cause all information of this instance, except for the dimensions, to be lost, as the operator only handles the information for the dimension. It can take more detailed inputs but will ignore additional information for the calculation of the output. In contrast to the alignment between different levels of hierarchy, which must only apply the similarity operation between two instances after the conversion, we will not cross any boundary between abstractions and therefore we must not lose any information. Otherwise, we might get stuck in our planning process.

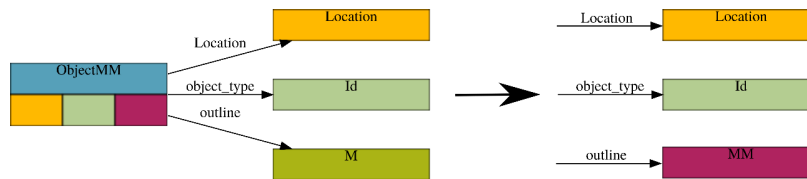
For manually modeled and implemented operations, the engineer makes sure that every detail of the output instance is filled in. We could define an operator that performs the conversion process manually, taking boxes with the dimension of one unit and returning boxes with the dimension of the other unit, preserving all details. In our modeling scheme, it is good practice to define only the required data as inputs and outputs to the operator. This allows for broader applicability, use within the planning algorithm, and precise description of the code, which are the primarily goals of the model. In the case of the conversion operator, however, the detailed inputs and outputs are required in the model, as we make the statement that even if the rest of the information about the box, apart from the dimensions, remains untouched, we have ensured consistency at this level of detail. In this way, we also declared that redundant information in our model, such as the dimension, volume, and density, are consistent. With the specialized approach, we need a conversion operator for each concept in the domain's concept hierarchy and cannot even take advantage of inheritance but have to resort to manual coding. This tedious integration task, considering redundancies that are domain-specific and depend on both declarative and procedural knowledge, can be automated using our set-based models.

In our unit conversion example we have all the information needed to generate conversions between instances that have a dimension with an arbitrary unit, such as boxes with *mm* and *m* once we implemented a universal conversion operator between the different primitive units.

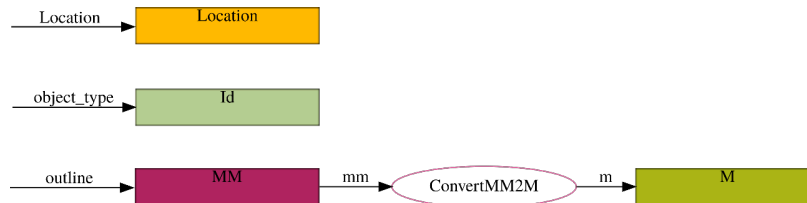
16 B. Kast et al.

Our algorithmic approach to implement the automatic conversion operator between concepts that use different units consists of three steps:

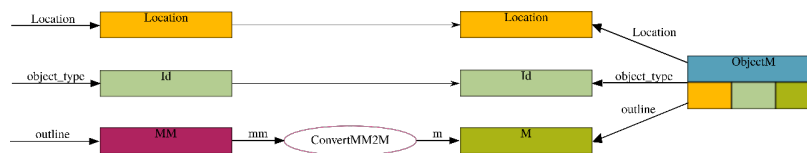
1. The information within the instance is decomposed so that individual instances are available for each role.



2. Once the required information has been extracted, an operator is applied to a subset of these instances.



3. The separate pieces of information are merged back into a single instance.



In this way, we create a formally usable instance for our original operator. For the fusion of step 3, we must decide which information is to be placed in which role of the merged instance. To answer this question, we can set up a semi-automatic process where the assignments are pre-filled but can be corrected manually. We also know that for the unused information the roles will probably remain the same. So, we can concentrate on the newly computed information from our conversion operator. At this point we can define another heuristic that analyzes the inputs and outputs of an operator and identifies information that was probably only processed but not newly generated. For each consumed input we therefore look for an output of the same concept and, if there

is none, an output with the same concept base. These pairs are likely to describe the same information, thus we assign the former role to the newly computed instance. For the other outputs that do not yet have a candidate for their role, we can compare the structure of the newly created concept that neglects pieces of sub-information without assigned roles and compare that to more detailed concepts of the current domain. We can then find candidates with a matching structure to this prototype concept and use the same roles for the additional pieces of information that comply with the concept or the concept base of unmatched roles.

In our example of the box with dimension the conversion operator would consume a dimension in mm and return it in m with concepts that have the same concept base. If we now merge all available information extracted from the initial box concept and cannot assign the recalculated dimension in m , then we look for other boxes in the current domain with a matching structure to our box without the dimension, but additionally use *dimension in m* as a sub-concept. We can then fill an instance of this concept with the pieces of information extracted and calculated from our box with dimension. With this set of rules, we can find a good estimate for plausible fusion operators, which then only need to be suggested to the user for validation. This domain-specific conversion process can then be stored as a new operator in our composed domain and is thus available to the planning algorithm. By that, further links between otherwise separate operators are possible during planning without the need for manual coding to bring together previously incompatible parts of a domain.

Arbitrary Operation on Sub-Information of an Instance However, this conversion problem is only a simplified special case of a more general problem. We want to apply operators on a subset of the information within a concept. There are possibly multiple operators in our domain, which do not necessarily consume all inputs, and we still must validate integrity when fusing the new pieces of information together. The core challenge is redundant information within our composed concept, for which consistency must be ensured. They prevent an automatic data fusion, which becomes evident at the intuitive example of an box with mass, volume and density. We can calculate the mass even if we only have volume and density. Let us assume that for our box in the example the color is defined as an additional piece of information. According to our automatic conversion algorithm, we decompose the box concept and get three concepts with concept bases for volume, density, and color. On those concepts we can apply the operator to calculate the mass, which consumes nothing. As we have no other operator in this domain to calculate the mass given color, density and volume, the validity check succeeds, and we can proceed with the fusion. We can perform an analysis of the existing concepts, determine that the *mass* role is the correct one for the additional value we are calculating, and create a template to model our additional operator. In our second example domain, we have an additional operator that classifies objects based on their color and calculates a mass based on that class. If we apply our algorithm to the same problem in this domain, we find that there is a redundant path to calculate the mass with the given information. Therefore, we cannot generate the fusion algorithm in this domain because we would generate redundant data ad hoc and do not know whether the color is affected by the mass or vice versa as visualized in Figure 7.

18 B. Kast et al.

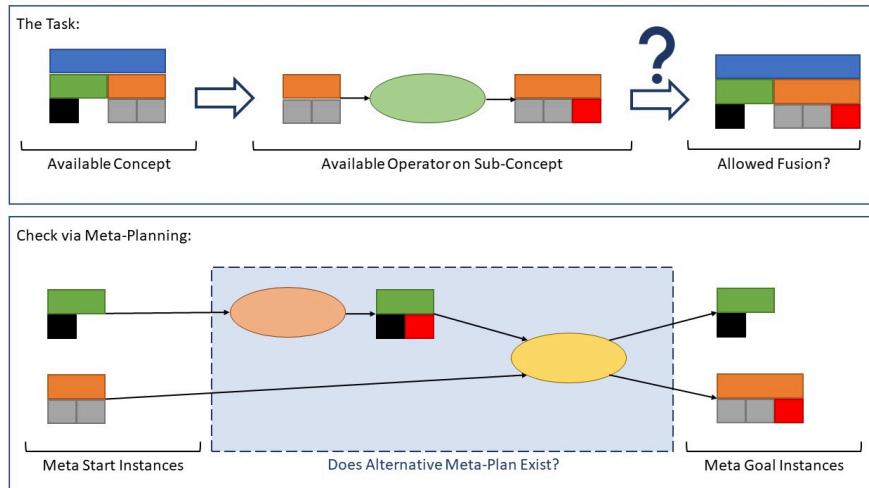


Fig. 7: We validate the given fusion task, which emerges from the decomposition of an available concept, the application of operators, and the final fusion with a meta-planning task. If an alternative path to the same pieces of sub-information exists, the fusion is not allowed, as inconsistencies might exist. Otherwise we can automatically conduct this conversion.

In the following we want to formalize this example introducing the so-called meta-domain, but first we want to discuss the differences between our task and the problems in the field of sensor data fusion, where several sensor readings are combined to obtain a better estimate of the current state. These bits of information may be redundant because the same scene is recorded, however with different sensors, different modalities, and at different times. In this case, we need probabilistic models that extract the redundant part of the information from the noise of different measurements and reconcile possibly conflicting readings. This process is not possible without detailed probabilistic sensor and process models. For our data fusion task, we cannot expect these probabilistic models to be generally available, because they have to be normalized for each domain. However, our operators are generally defined in different modules, and the domain can be automatically composed. Therefore, it is unlikely that we can rely on robust probabilistic models of the relationships between the operators. Nevertheless, from the operators in the domain we can deduce which information within a concept is completely independent from the other specified data.

For introduction purposes we start the formalization of our approach now by addressing the consistency of a single operator on a subset of an instance and later on discuss the extended case, where the sub-concept relation between each two concepts can be analyzed. To check whether all outputs of an operator that we want to apply to a subset of the information within a concept are independent from the other pieces of information, we specify a planning task and check if there is a solution, which indicates

that a dependent plan exists. In detail, we can construct a so-called meta-domain, in which the concepts of an ordinary domain represent instances with our domain. The similarity property of these instances reflects the hierarchy of concepts within the ordinary domain. Additionally, we can also implement operators in the meta-domain as well as define and solve planning problems. These operators can have meta-models of concepts as well as meta-models of operators as inputs. Using a meta-operator that simulates the operators of our usual domain solely based on a mapping of input concepts to output concepts (i.e. not mapping instances as the usual operator would do, but just providing the input and output structures), we can then specify an initial state and goals that consist of instances that describe concepts. Such an instance representation of a concept follows the lines of subsection 3.1, where concepts are introduced via basic sets and compositions with roles. Thus the graph structure of a concept (confer for example Figure 2) describes the instance in the meta-domain. With this setup, we can now check for independent information within our concept so that a data fusion is possible. For this purpose, we specify for each output of our conversion operator a planning task, that has the goal to compute this output from all non-consumed inputs that are available as initial facts. In addition to these instances, the meta-concepts of all operators of this domain apart from the conversion operator are available as start instances. Since our planning algorithm can detect loops and thus aborts planning if no plan exists, we can use it to detect non-redundant data within the concept in a non-constructive manner. If we can find a plan that solves the task, there is an alternative way to compute the same information as our conversion operator has calculated, and thus there is redundant information within our concept. If this is not the case and we make the closed world assumption for our domain, we can conclude that the checked information is not redundant and can therefore be merged without loss of consistency.

In case there is a redundancy for the newly calculated outputs, a probabilistic sensor fusion with appropriate probabilistic information in our models would be necessary, which is targeted based on our models in [4], but not the focus of this paper.

Identification of Meaningful Networks to Process Sub-Information We proposed an approach to solve the assignment problem in many cases and to switch to the semi-automatic process in case of ambiguous roles. Additionally, we discussed an algorithm to validate the consistency of the merged concept. However, we consider so far only a single, given operator, which we check for applicability to sub-concepts. The even more interesting task is to identify meaningful applications of operators that can be applied to sub-concepts in our domain, which is depicted by the block for selecting another concept pair at the top of Figure 8. This can be done in an offline step, followed by the verification process described above and end in the generation of new operators. These operators are then available during the planning process and envelop the manually implemented operators but work on other types of input and output concepts.

The *operator* may possibly be a sequence or network of operators that transform the initial facts provided by the sub-information of a concept into the set of information that is then merged into the new concept. This process can be formulated as a planning task in the meta-domain again. Obviously, this process is just a variation of the planning task in the meta-domain from above. This means that we can decompose any concept

20 B. Kast et al.

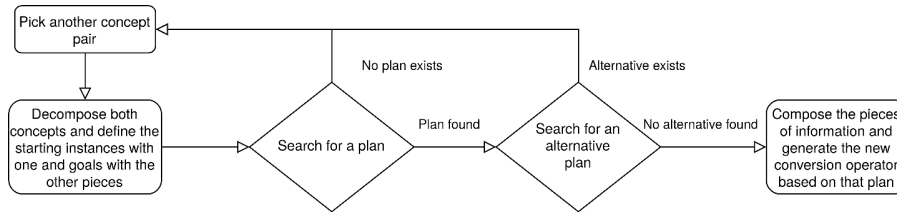


Fig. 8: Flow diagram of the overall extended conversion algorithm. We pick the two concepts of our domain between which we want to find a conversion. We construct a planning task, in which the goals are the decomposed sub-concepts of one concept while the starting instances are the sub-concepts of the other concept. The applicable operators are the shells of the operators matching input concepts to output concepts (i.e. no instance computation based on the implementation, preconditions, etc.). If exactly one plan exists (i.e. no alternative path between starting instances and goal instances exists), a valid conversion operator is found. It is the combination of the operators in that plan.

in the domain and specify any second concept within our domain as a goal. Each plan we find defines a feasible network that can be stored as a new operator. Only existing concepts make sense as goals, since newly generated concepts are not used as an input and are therefore useless to the planning algorithm. All conversion operators are only valid for the specific domain, because additional operators can make sub-information within a concept redundant as shown above.

5 Experiment

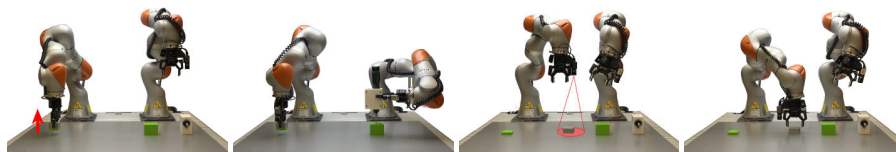


Fig. 9: The hierarchical planning algorithm picks, configures and executes each of the possible actions. Additional important steps of the successful plan to Figure 1 are the grasping of the assembled box, refinement of objects, pick up of the screw, and the initial pick of the box. Originally published in [11].

We conduct the experiments in two separate parts. Firstly, we analyze the performance of our optimization algorithms in simulation, as we can perform a large number of test runs on longer lasting problems. Secondly, we validate our setup on a dual arm robot cell to check for real-world applicability and robustness to the effects of imperfect simulation.

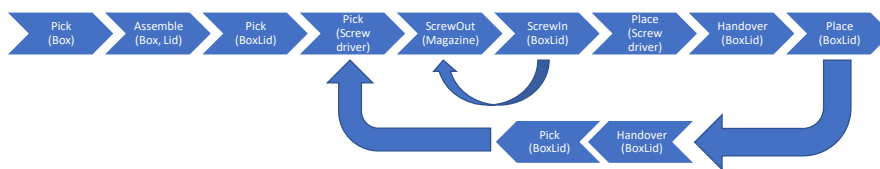


Fig. 10: In this nominal plan localizations are omitted for brevity. As stated in subsection 3.3 the MPC-like behavior will result in completely different sequences if external disturbances occur. The basic difference between the optimized and original domain is the length of the loop to insert additional screws. Without optimization, the box and screwdriver are placed on the table for each step, which results in handovers, pick, and place actions with respective refinements for each screw, which are unnecessary in the optimized domain. Originally published in [11].

Our real-world setup, which we also modeled for the simulated test runs, is a dual arm robot with 14 degrees of freedom and a parallel gripper on each end-effector. We conduct the localization with an RGB-camera, which is attached to each arm, such that the refinement requires the robots to position in a certain area. We additionally monitor the workspace with a 3D camera, which gives us initial estimates of the object's locations, while the precision is not high enough to directly manipulate the items. The screwdriver is graspable by the robots and remotely controlled to pick up and fixate the screws.

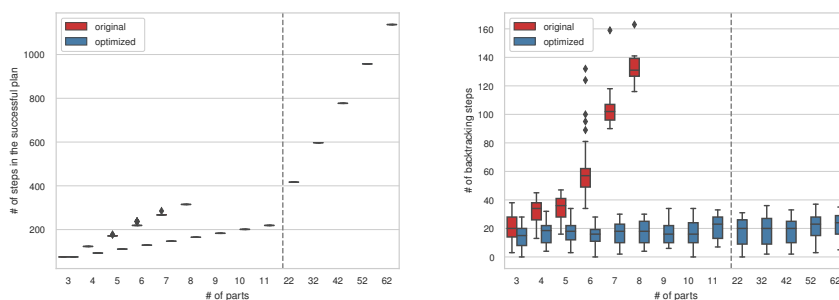


Fig. 11: For both domains, the successful plan lengths grow linearly (change of step size right of dashed line) with the number of parts in the goal product, while the optimized domains require less steps due to no unnecessary placements. This also results in fewer required backtrackings (right diagram). Originally published in [11].

22 B. Kast et al.

The task involves the assembly of two parts, which are then fixated by a configurable number of screws as depicted in Figure 1 and Figure 9. We modeled the actions, which manipulate objects, such as pick, assemble, or screw, to reduce the precision of unfixed objects but require a high precision in the beginning. Therefore, the planning algorithm must find a solution, which incorporate the refinements prior to each of those operations. Screwing is only possible in the air, with the object fixated by one robot and the screwdriver in the second robot's gripper. As we allow the placement of all picked objects with a certain orientation, such that loops in state space are possible. Screwing is only possible with a certain orientation as well, such that the screws face outwards, which can be achieved with a handover. For the motion synthesis we use the constraint-based approach of [15], which eases planning due to the little intermediate positions required for a large coverage of the state-space.

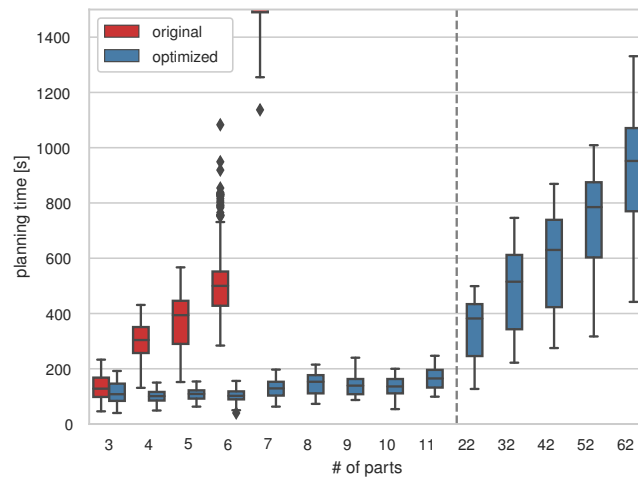


Fig. 12: The planning times for both domains grow near linearly, while the slope is about a factor of 5 steeper for the original domain. This correlates with the fewer steps and less backtracking procedures required in the optimized domain. Note the change in step size right of the dashed line. Originally published in [11].

In this paper we propose an offline optimization algorithm, which extends the given domain with automated unit transformation operators and wrappers, and an online optimization that aligns the domain to the specified goal. For the experiment, we can apply the planning algorithm on the original, fully optimized, and partially optimized domains.

In the original domain, the most abstract level is workpiece centric, while the other levels of abstraction are hardware centric. This misalignment continues in the usage of

different units for the dimensions of the objects, which is checked to validate graspability by the robots. In total, there are 4 distinct layers of abstraction, which emerge from the hierarchical ordering of the operators. In the second layer, the refinement operations and accuracy of the locations come in, the third layer checks for collisions in simulation, while the most detailed layer executes the actions on the real hardware.

For the original and the domain without offline optimization, no successful plan was found. This is due to the unit mismatch between the first and second level of abstraction, which results in no available action to pick objects successfully in the second level. Only after the unit conversion and wrapping with the offline optimization algorithm solutions can be found. For the further analysis of the alignment optimization, we thus compare the performance between the fully optimized domain and the offline optimized original domain, which we refer to as original domain for brevity in the following.

We can fixate the lid only with up to 4 screws for the real-world test, due to the limited number of holes in the objects. These experiments succeed with both domains, while the optimized is significantly faster.

To test the further scalability of our approach and measure execution times with a larger number of runs and thus reduce the variance of our results, we switch to simulated tests. The difficulty of the task is configured by the number screws we require for fixation. While the real box has only four screw holes, we tested up to 60 screws in simulation, which resulted in 62 involved parts for assembly. We conduct each experiment 100 times with randomized starting positions of the objects.

For the original domain, we had to abort the experiments at 7 involved parts, as the planning times exceeded 10 minutes. The optimized domain, however, could assemble up to 62 parts within that time as depicted in Figure 12. For both domains, we could observe a near linear increase in planning times with the number of involved parts. This shows the effectiveness of our hierarchical planning approach. Nevertheless, the steepness of the original domain was about a factor of 5 higher than for the aligned domain and it had a larger variance.

The reason for this can be found in the lengths of the successful plans, depicted in Figure 11. For the same number of parts involved in the goal product, the number of steps in the successful plan is about double the number in the aligned domain. As the number of abstractions between both domains is equal, the plan lengths of the sub-planning tasks increase as well. This results in larger contributions in the exponential component, which results in the long planning times. Additionally, these longer plans can fail more often during refinement. Therefore, we can observe a more frequent need for backtracking, which further increases planning times.

As we analyze the reasons for the longer plans in the original domain, we can observe, that the intermediate goal of a box with a specific number of screws can be achieved directly with the box fastened in the robots gripper for the optimized domain. Thus, the next screw can be directly inserted, opposed to the original domain for which the box must be separated from the robot and thus placed on the table. This placement involves several additional steps, as a direct drop is not possible due to the wrong orientation of the box with the screws facing downwards. Thus, a handover is required, which can only happen after the screwdriver is placed. Afterwards all actions must be reverted

24 B. Kast et al.

to fasten the next screw, which results in the significantly longer and less efficient plans of the original domain as depicted in page 21.

6 Conclusions

In this paper we discussed two domain optimizations algorithms, which build upon our set-based formal models to optimize and extend a given domain.

They address different aspects to facilitate the integration of existing models to new domains. The domain optimization and alignment algorithm, proposed in [11], minimizes the suboptimalities that come with the factorization conducted by our hierarchical planning algorithm. We show that it allows our planning algorithm to bring its strengths of good scalability to larger problems, even for poorly modeled domains. As the modeling of domains requires expensive experts, the widespread application of planning approaches relies on such optimizations, which ease the engineering and allow more people to configure and contribute to autonomous systems.

The second algorithm, which was first published in this paper, is an extension to this idea of facilitated composition of domains. We further automate tasks of the engineer and can algorithmically detect possible unit conversions, and general applications of operators on sub-information of instances. By that, we enlarge the pool of possible actions that can be used by our planning algorithm which allows to solve additional tasks as shown in our experiment. As our algorithms generate human-readable models for both, the declarative as well as the procedural extensions. They are easy to interpret, such that engineers can be extended and build upon those additional pieces of knowledge. This can be a powerful tool for an automated composition of modules from different sources and facilitate the setup of autonomous systems. Especially in industrial manufacturing, additional fields of applications, for which current systems are too cumbersome to setup might now be feasible.

An interesting field for future research is the combination of our hierarchical and model-based planning approach with data-driven algorithms. The strengths of our approach lie in the scalability due to the factorization of large problems. Data-driven approaches, on the other hand, promise better results, once they gathered enough data and finished training. The combination of both approaches could either generate the initial training data by using our planning algorithm, which could provide a hot start to the training with the risk of local minima. Another option is the training of heuristics for both, the hierarchical factorization and the forward-planner used in our approach. As a third option, we could use trained networks as operators. All three options do have their pros and cons, which would be worthwhile to examine.

References

1. Areces, C.E., Bustos, F., Dominguez, M., Hoffmann, J.: Optimizing planning domains by automatic action schema splitting. In: Twenty-Fourth International Conference on Automated Planning and Scheduling (2014)
 2. Bleiholder, J., Naumann, F.: Data fusion. *ACM computing surveys (CSUR)* **41**(1), 1–41 (2009)
-

3. Dasarathy, B.V.: Sensor fusion potential exploitation-innovative architectures and illustrative applications. *Proceedings of the IEEE* **85**(1), 24–38 (1997)
4. Dietrich, V., Kast, B., Albrecht, S., Beetz, M.: Data-driven synthesis of perception pipelines via hierarchical planning. In: *International Conference on Robotics in Alpe-Adria Danube Region*. Springer (2020)
5. Durrant-Whyte, H.F.: Sensor models and multisensor integration. In: *Autonomous robot vehicles*, pp. 73–89. Springer (1990)
6. Haslum, P., Botea, A., Helmert, M., Bonet, B., Koenig, S., et al.: Domain-independent construction of pattern database heuristics for cost-optimal planning. In: *AAAI*. vol. 7, pp. 1007–1012 (2007)
7. Kalashnikov, D., Irpan, A., Pastor, P., Ibarz, J., Herzog, A., Jang, E., Quillen, D., Holly, E., Kalakrishnan, M., Vanhoucke, V., et al.: Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293* (2018)
8. Kang, T.S., Nnaji, B.O.: Feature representation and classification for automatic process planning systems. *Journal of manufacturing systems* **12**(2), 133–145 (1993)
9. Kast, B., Albrecht, S., Feiten, W., Zhang, J.: Bridging the gap between semantics and control for industry 4.0 and autonomous production. In: *Int. Conf. on Automation Science and Engineering*. IEEE (2019)
10. Kast, B., Dietrich, V., Albrecht, S., Feiten, W., Zhang, J.: A hierarchical planner based on set-theoretic models: Towards automating the automation for autonomous systems. In: *Int. Conf. on Informatics in Control, Automation and Robotics*. SCITEPRESS Digital Library (2019)
11. Kast, B., Dietrich, V., Albrecht, S., Feiten, W., Zhang, J.: Domain optimization for hierarchical planning based on set-theory. In: *ICINCO 2020*. SCITEPRESS Digital Library (2020)
12. Katz, M., Sohrabi, S., Samulowitz, H., Sievers, S.: Delfi: Online planner selection for cost-optimal planning. *IPC-9 planner abstracts* pp. 57–64 (2018)
13. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013)
14. Nau, D., Au, T.C., Ilghami, O., Kuter, U., Murdock, J., Wu, D., Yaman, F.: Shop2: An htn planning system. *Artificial Intelligence Research* **20**, 379–404 (2003)
15. Schmitt, P.S., Wirnshofer, F., Wurm, K.M., Wichert, G.V., Burgard, W.: Planning reactive manipulation in dynamic environments. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2019)
16. Seipp, J., Braun, M., Garimort, J., Helmert, M.: Learning portfolios of automatically tuned planners. In: *Twenty-Second International Conference on Automated Planning and Scheduling* (2012)
17. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al.: Mastering the game of go with deep neural networks and tree search. *nature* **529**(7587), 484 (2016)
18. Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al.: Mastering the game of go without human knowledge. *Nature* **550**(7676), 354–359 (2017)
19. Vallati, M., Hutter, F., Chrapa, L., McCluskey, T.L.: On the effective configuration of planning domain models. In: *Twenty-Fourth International Joint Conference on Artificial Intelligence* (2015)
20. Vinyals, O., Babuschkin, I., Czarnecki, W.M., Mathieu, M., Dudzik, A., Chung, J., Choi, D.H., Powell, R., Ewalds, T., Georgiev, P., et al.: Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature* **575**(7782), 350–354 (2019)
21. Xie, A., Ebert, F., Levine, S., Finn, C.: Improvisation through physical understanding: Using novel objects as tools with visual foresight. *arXiv preprint arXiv:1904.05538* (2019)

11

Configuration of Perception Systems via Planning Over Factor Graphs

Title	Configuration of Perception Systems via Planning Over Factor Graphs
Authors	Vincent Dietrich, Bernd Kast , Philipp Schmitt, Sebastian Albrecht, Michael Fiegert, Wendelin Feiten, and Michael Beetz
ISBN/ISSN	978-1-5386-3081-5/2577-087X
DOI	ICRA.2018.8460955
Status	published
Publisher	IEEE
Contribution of Bernd Kast	I implemented the underlying modeling and planning for this paper. Vincent Dietrich developed the probabilistic models, which benefited from discussions about the theoretic foundations of our modeling scheme with me. The other co-authors provided the general scientific conditions and use-cases for this paper.

Summary	<p>A key factor for autonomous machines is the perception subsystem, which is generally composed of different types of sensors at varying locations. Evaluation algorithms accompany this subsystem, extracting compiled information from the raw data to feed it to a sensor-data fusion algorithm. The configuration problem for a new autonomous system is structurally like other planning algorithms in our framework. Hence, we can reuse our existing modeling and planning algorithms and implement only models for this new domain. We define the goals by a required maximal uncertainty level, which can be achieved by fusing different measurements from independent locations. The planner can select adequate sensors at reasonable positions and appropriate feature extraction algorithms, which are then fused in a factor graph. This flexible autonomous configuration approach is validated on an industrial assembly cell.</p>
---------	--

Configuration of Perception Systems via Planning over Factor Graphs

Vincent Dietrich¹Bernd Kast¹Philipp Schmitt¹Sebastian Albrecht¹Michael Fiegert¹Wendelin Feiten¹Michael Beetz²

Abstract—Sensor guided, automated systems require the composition of various sensors and data processing algorithms to obtain relevant information for performing their task. Many applications have additional requirements such as a certain accuracy, which has to be achieved despite sensor noise and calibration errors. In this paper we model the configuration of perception systems as a planning problem over probabilistic graphical models. We work on a subset of the full configuration space of perceptions systems, specifically the used sensors, data processing algorithms and view poses. Based on a semantic description of the goal, available sensors and data processing algorithms, our system plans perception steps and sensor data fusion autonomously. The planner operates by constructing a factor graph until the accuracy requirements of tasks are fulfilled or unobtainable with the available action set. We validate our approach in an industrial assembly scenario.

I. INTRODUCTION

Automated control systems in industrial applications, such as logistics or manufacturing, require reliable and accurate data about the state of the system and its surroundings. This data must be obtained despite low signal to noise ratios, ambiguous sensor readings and indirect measurements of the quantities of interest. The state of the art approach to fulfill these perception tasks in industry is to engineer problem specific combinations of sensors and algorithms for perception and data fusion. Often, sensor based solutions are avoided entirely by designing expensive hardware, such as fixtures or precise part feeding, for the task at hand.

For applications with short product life cycles or large product variety the approach of manually engineering perception systems is not viable. In the extreme case a special perception task might occur only once. For these scenarios there is a pressing need to autonomously configure perception systems. Solutions to this problem must address the following challenges:

- Large variety of sensors and algorithms: Sensors as different as RGB cameras, LIDAR and sonar range finders are required to capture the full spectrum of applications. In the same way as there is not one single type of sensor for all applications, a wide range of perception algorithms is needed. An automated approach to the configuration of perception systems must cope with this variety.

¹Siemens Corporate Technology, Otto-Hahn-Ring 6, 81739 Munich, Germany

²Institute for Artificial Intelligence, University Bremen, Am Fallturm 1, 28359 Bremen, Germany

*This work was supported by BMWi IKT III SADA Project <http://www.projekt-sada.de/>.

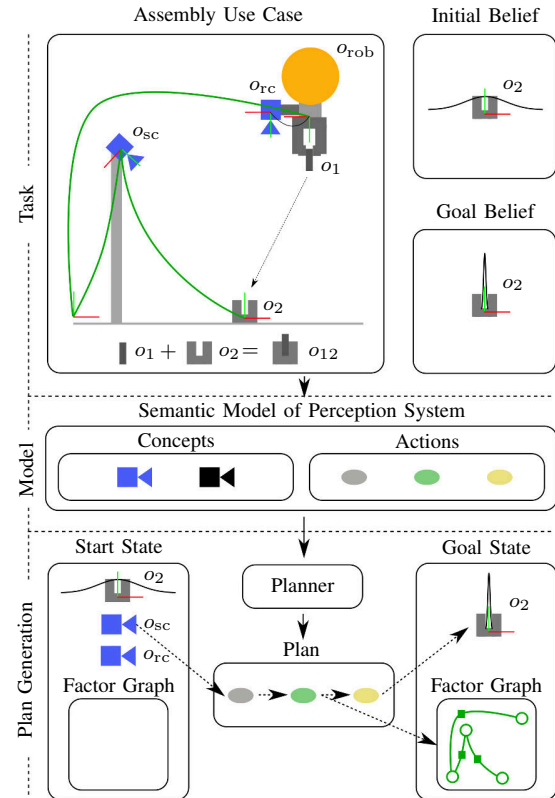


Fig. 1: Overview of the configuration approach for an exemplary assembly use case, where a robot o_{rob} should assemble the part o_1 into o_2 . Therefore, part o_2 needs to be localized precisely. The task and perception system description is grounded in a semantic model, which comprises concepts such as a RGB camera and actions such as an image based object detection. Using the semantic description of the problem, the planner determines a sequence of actions to satisfy the goal. The plan includes the automatic generation of a factor graph which is used to estimate the object pose.

- Inherent perceptual uncertainty: Perception is subject to uncertainties ranging from noisy raw data over false measurements to calibration errors. Currently, the perception engineer has the often implicit knowledge of the involved uncertainties and chooses an appropriate perception system. In order to enable automatic configuration, an explicit representation and automatic handling of these uncertainties is required.
- Task dependent requirements for perception: The information that is required to successfully perform a task has large qualitative differences from one application to

another. For a sorting application a basic classification may be sufficient, while for an assembly task highly accurate relative poses of objects are required. A single measure, such as entropy of a belief state, does not cater well to the variety of tasks.

The contribution of this paper is an approach for the configuration of perception systems that addresses these challenges. The approach is exemplarily depicted in Fig. 1. The variety of sensors and perception algorithms is captured with semantic models of equipment and algorithms that are used to plan both perception and data fusion steps. Our planner operates over belief distributions that are represented as factor graphs, which efficiently handle high dimensional geometric probability distributions. It proceeds with building factor graphs using simulated perception algorithms until the information required by the task is obtained. We show the effectiveness of the approach in a simulated, industrial assembly scenario.

II. RELATED WORK

Several approaches have been proposed in the past for the problem of automatic configuration of perception systems. An approach based on unstructured information management is implemented in the *RoboSherlock* framework by Beetz *et al.* [1]. In combination with the knowledge bases *KnowRob* [2] and *OpenEASE* [3] and the *Semantic Robot Description Language SRDL* [4], successful perception pipelines can be determined according to situation requirements. The authors focus on semantic reasoning grounded in *OWL* [5] ontologies, but also show how probabilistic reasoning can be leveraged for perception, specifically for object classification [6]. Geometric uncertainties induced by perception actions and calibration errors are not in the main focus of the framework.

Another approach for modeling and situation aware adaptation of perception actions can be found in the work of Hochgeschwender *et al.* [7]. The authors introduced and use the *Robot Perception Specification Language RPSL* [8] to describe the perception task and employ a reasoning mechanism to find a suitable perception plan.

The research area of world modeling for autonomous systems does also provide solutions for adaptive perception. Elfring *et al.* [9] present an approach to keep a consistent probabilistic world model based on probabilistic multiple hypothesis anchoring. The probabilistic world model is further updated with strategies that maximize information gain and allow for a basic task dependency [10].

Another relevant area of research is the field of active perception [11]. Research ranges from view selection [12] over adaptive parameter tuning [13] to concepts of a framework which enables autonomous configuration of perception and sensor fusion [14]. The latter work focuses on the underlying software framework and anticipates the autonomous configuration as future work.

Handling geometric uncertainty, for instance in assembly applications does have a long history. In early work from Su *et al.* [15] geometric relations including uncertainty

are modeled within a directed graph of transformations with covariance matrices. Using uncertainty propagation and sensor fusion with Kalman filter, the covariance and pose between coordinate systems is determined. Furthermore, the authors provide an approach based on backward propagation to determine the admissible set of actions as well as required perception actions. The approach for handling geometric uncertainty is still commonly used, for example in the work of Blumenthal *et al.* [16]. For applications such as simultaneous localization and mapping, handling of uncertainty with factor graphs has become the dominant approach [17].

The most common way to express planning tasks is the *Planning Domain Definition Language PDDL* [18] subsuming, for example, the problems addressable with the *Stanford Research Institute Problem Solver STRIPS* language [19] and the *Action Description Language ADL* [20]. Two standard approaches to solve planning tasks are classical planning [21], [22] and *Hierarchical Task Network (HTN)* planning [23], [24]. Planning represents an important step in the automatic configuration of perception systems, but the performance depends on the model and the specific task.

III. APPROACH AND NOTATION

We consider the problem of planning a sequence of data gathering, data linkage and inference steps to obtain a required set of information about the state of the robot and its surroundings. The information about the unknown state x of our system is represented as a probability distribution or belief $\text{bel}(x)$. Probabilistic graphical models have shown success in modeling and estimating probabilistic relations in a large variety of applications [25]. The focus in this paper is on geometric uncertainty as it plays an important role in many domains, such as industrial assembly. Specialized probabilistic graphical models known as factor graphs are a well-suited representation for distributions involving geometric uncertainty and allow efficient inference using optimization techniques [26]. Therefore, we encode the belief as a factor graph F_G denoted by the tuple (V, E) , where V encodes the vertices of the factor graph and E the edges or factors.

A state within our planning problem is specified by the tuple $s = (F, F_G)$. In this tuple, F denotes a set of facts that hold in the current state. These facts encode currently available variables, such as encoder positions, as well as previous measurements, such as outcomes of a object detection procedure or the robot's encoder values when this measurement was taken. The factor graph F_G that encodes the belief $\text{bel}(x)$ is constructed using these facts. At all times the system can choose from three types of actions:

- *Data Gathering*: By performing (simulated) perception actions, the planner gathers new data, which corresponds to adding new facts to F . An example is an object localization routine that produces a fact about the localization outcome, e.g. a relative pose between camera and object.
- *Data Linkage*: This action type sets facts into relation with each other. In the presented scenario, it is used to

construct the factor graph F_G . Following the example of an object localization routine, the set of edges E of F_G is extended by the uncertain measurement. Depending on the already existing vertices in V , new vertices are added, e.g. for the camera pose or the object.

- *Inference*: Actions of this type target the generation of new facts by inference. In the localization example, an inference action may extract for instance a new object pose estimate by optimizing the variables within the factor graph F_G .

Formally, an action $a \in \mathcal{A}$, with \mathcal{A} being the set of all available actions, may only be applied if the current state s fulfills its preconditions: $s \in \text{Pre}(a)$. After action a is applied we obtain a new state $s' = \text{Post}(s, a)$. With this we can define valid plans (of length $k \in \mathbb{N}$) as sequences of states $\{s_i\}_{i \leq k}$ and actions $\{a_i\}_{i \leq k-1}$:

Definition 1 (Valid Plan) A plan $\{s_i\}_{i \leq k}$, $\{a_i\}_{i \leq k-1}$ is valid iff

- 1) $s_i \in \text{Pre}(a_i)$ for $i \in 1 \dots k-1$ and
- 2) $s_{i+1} = \text{Post}(s_i, a_i)$ for $i \in 1 \dots k-1$.

Starting from an initial state s_{start} the aim of the planner is now to reach a desired state of information within the set \mathcal{S}_{goal} . With this we can define feasible plans:

Definition 2 (Feasible Plan) A plan $\{s_i\}_{i \leq k}$, $\{a_i\}_{i \leq k-1}$ is feasible iff it is valid and

- 1) $s_1 = s_{start}$ and
- 2) $s_k \in \mathcal{S}_{goal}$.

An important assumption that we make during planning is that our system only produces maximum likelihood measurements as proposed in [27]. This has three key advantages:

- *Deterministic planning domain*: As our system's actions do not increase uncertainty, the belief state evolves as follows:

$$\text{bel}_{t+1}(x) = \gamma p(z | x) \text{bel}_t(x), \quad (1)$$

where z is a measurement and γ a normalization variable. As the measurement z is random, the evolution of the belief is random as well. Assuming maximum likelihood measurements yields a deterministic evolution of the belief and a deterministic planning domain.

- *Decoupling of perception planning and construction of the factor graph*: As measurements always correspond to the mode of the measurement model, the generation of new facts during planning does not require the evaluation of the belief represented by the factor graph. This decouples planning of perception steps and sensor data fusion.
- *No repeated fusion of the same measurement*: Our measurements are assumed to be deterministic during planning. Therefore, perceiving objects with the same algorithm from the same robot configuration results in the same measurement. Thus the resulting fact is not

added repeatedly to the set F . This prevents adding the same or similar data repeatedly into the factor graph.

For the experimental evaluation, we use a basic breadth-first search that performs all valid actions $a \in \mathcal{A}$ on s , where $s \in \text{Pre}(a)$ holds.

IV. MODELING

We use a generic graph-based modeling approach (being a subset of *OWL* [5]) to describe the perception domain. Every fact $f \in F$ is an instance of a so-called concept, which models its properties. For instance, the RGB camera concept has an intrinsic concept, which itself models the parameters of a pinhole camera model. Actions are modeled via the input concepts that they require and output concepts that they produce. The model does not contain an explicit formulation of pre- and post conditions. The actions are implemented such that they fail upon execution if the input facts do not fulfill the pre-conditions. An example is the *move-close* action as introduced in Sec. VI-B.2, which requires to know the object position with certain accuracy. Furthermore, actions may only be executed if actual facts of all modeled input concepts are available in F .

V. EXEMPLARY PLANNING SEQUENCE

For the sake of comprehensibility, we use this section to visualize and describe the mode of operation of our approach in a basic example with the help of Fig. 2. We assume a setting where an object o_p is visible to a camera o_c and has to be located with respect to a reference coordinate system c_{ref} . The goal is given by $x_{ref,p,-}$, specifying a maximum positional uncertainty allowed for the task at hand. The goal $x_{ref,p,-}$ does not pose requirements on the time stamp of the result. A prior pose estimate between c_{ref} and the coordinate system c_p of o_p is given in form of a factor in the factor graph $F_{G,0}$. Additionally, the extrinsic calibration $m_{ref,c}$ of o_c is given in form of a fact. This initial state is contained within s_0 . Note that all facts and goals are grounded in a semantic model as indicated in the top part of Fig. 2. The visualized model contains only the required concepts and actions, but is in no way limited to these.

In the following, one feasible plan is presented in form of a possible sequence of actions to reach the goal. In the first step, 3 independent actions specified in \mathcal{A}_{s_0,s_1} are executed: *2D-meas*, *add-meas* and *query-F_G*. The action *2D-meas* simulates a perception algorithm that may provide an estimate of an object position in a 2D image, e.g. the deep learning approach called *YOLO* [28]. As input, *2D-meas* requires a RGB camera and, if successful, produces a 2D measurement. In the given example *2D-meas* acts on o_p and o_c and produces $m_{c,p}$. It belongs to the introduced subset of actions that gathers new facts from existing facts. The second action *add-meas* belongs to the subset of actions that link information in a graph representation. In the example it inserts the measurement $m_{ref,c}$ into the factor graph $F_{G,1}$ of s_1 . Finally, the third action *query-F_G* performs an optimization of the factor graph with respect

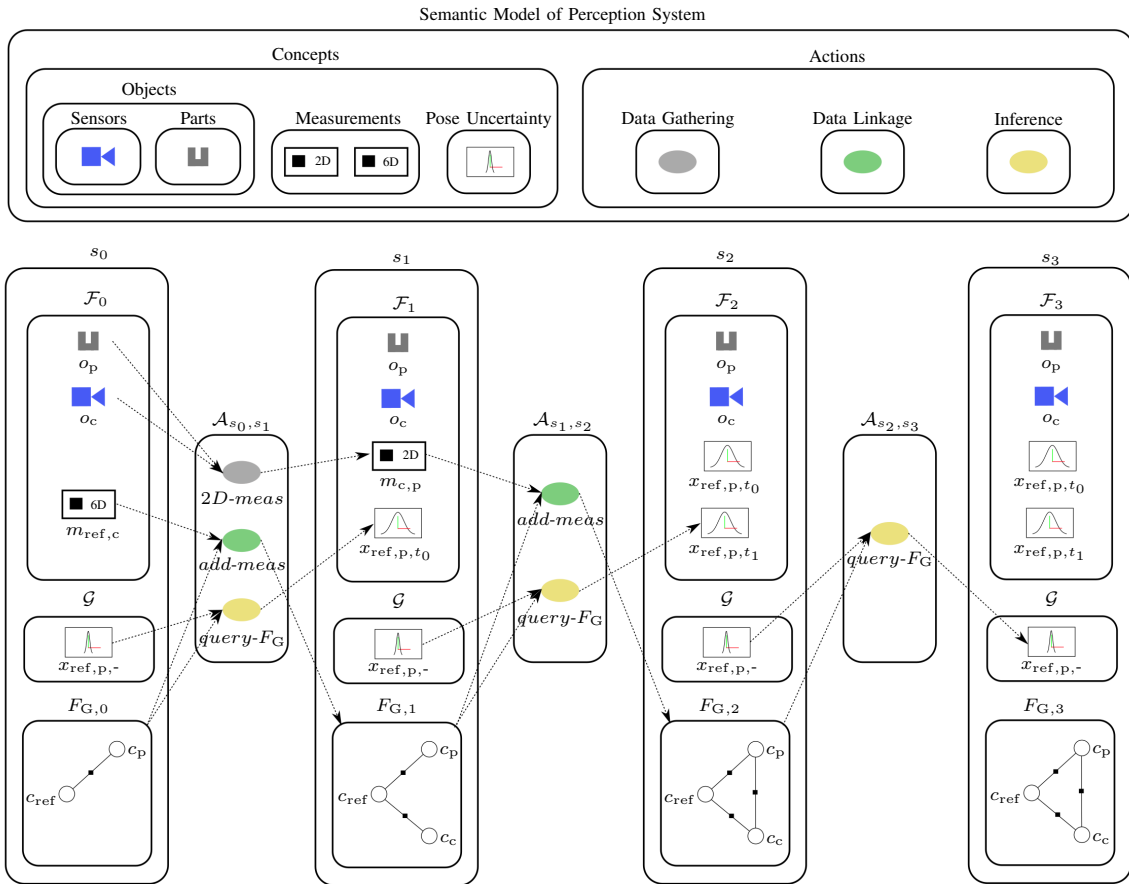


Fig. 2: Exemplary visualization of the planning procedure for a basic setting with one object and one camera. The goal is to reduce the belief uncertainty over the object pose under a value specified within the goal set. In the upper part an exemplary model of the perception domain is outlined. In the lower part, the different system states from the start set s_0 to the final state set s_3 , where the goal could be satisfied, are shown. A more detailed description of this figure can be found in Sec. V.

to the reference frame of the goal input. As output, a pose with uncertainty, specifically x_{ref,p,t_0} is produced. This pose, representing the pose prior over o_p , does not fulfill the goal requirements. Therefore the action sequence continues.

The next step, the transition between the states s_1 and s_2 is achieved by executing the action set A_{s_1, s_2} . The action set contains two actions *add-meas* and *query-FG*. Again, a measurement is added to the factor graph, in this case the previously produced 2D projection $m_{c,p}$. Note here that using factor graphs, we are able to handle quite different pose constraints like a noisy 6D pose or a noisy 2D image projection. The general procedure and effect of the action *query-FG* has been introduced in the previous paragraph and is similar for this state transition. As can be seen in the factor graph, the previously added extrinsic calibration $m_{\text{ref},c}$ of the camera has no effect on the pose estimate between the reference frame c_{ref} and the target object o_p coordinate frame c_p . Therefore the output of *query-FG* is an equally uncertain pose estimate x_{ref,p,t_1} as x_{ref,p,t_0} .

Finally, in the last step of the exemplary plan another factor graph optimization is performed. This last action *query-FG* of A_{s_2, s_3} acts on the factor graph $F_{G,2}$ that

contains the measurement between target object and camera. The resulting pose uncertainty is significantly lower and satisfies in this example the requirements encoded in the goal set G .

VI. EVALUATION

We chose an assembly use case, where geometric uncertainties have significant effect on the process success. After an introduction of the use case, we demonstrate the capability of our approach to find plans that achieve the accuracy required by the assembly process in minimal execution time.

A. Implementation

The modeling and planning environment is a self-developed system that we intend to use in larger extent for machine knowledge management and robot autonomy. More detailed publications about this system will follow. For the factor graph representation and optimization we build upon the open source library GTSAM [26].

B. Experiment

1) *Setup*: Industrial assembly is a domain with high requirements on geometric uncertainty quantification as many

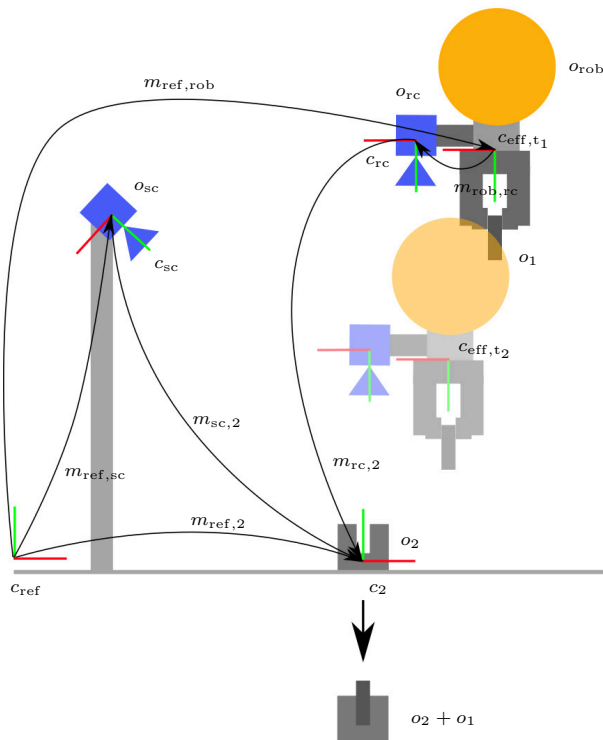


Fig. 3: Overview of the assembly use case. The symbol o denotes objects, c denotes coordinate systems and m denotes pose constraints due to measurement or calibration. The graph of black arrows visualizes the factor graph in its full extent.

assembly processes require tight positioning tolerances. In our specific use case an object has to be grasped for a follow-up assembly process as depicted in Fig. 3.

The perception goal consists of localizing the target object o_2 with respect to the robot end effector coordinate system c_{eff} . There is a static camera o_{sc} as well as an end effector camera o_{rc} mounted on the robot o_{rob} . Moreover, there exists an arbitrary chosen reference coordinate system c_{ref} . The overall application is to assemble o_1 into o_2 , where the former already resides within the robot gripper. For the sake of brevity, we assume that o_2 has been localized and grasped with perfect accuracy. Furthermore, to show that our approach can cope with different viewpoints, an important parameter in perception tasks, we let the system decide to take a close view of the object. The action *move-close* that performs this operation requires the object to be known with a certain accuracy that is not given by the prior belief over the object pose. If the relative uncertainty specified by the goal is met, the assembly process can be successfully executed. The uncertainty introduced by the relative movement of the robot for the assembly process is not considered in this example.

2) *Action Description*: In this use case we allow the following actions:

- Data Gathering
 - *2D-meas*: Simulates an object detection algorithm working on RGB images that outputs the object

center as a 2D point on the image screen. This action is similar to the 2D projection action described in Sec. V.

- *6D-meas*: Simulates a 6D pose estimation algorithm based on RGB images. For instance, it could represent the approach to estimate the 8 object bounding box corners using convolutional neural networks (*BB8*) as described in [29].
- *move-close*: Moves the robot o_{rob} such that the end-effector camera o_{rc} has a closer and centered view on the object. The pre-condition for this action is that the standard deviation of the positional uncertainty σ_1 of the target object o_2 is lower than a threshold σ_{thres} . This is motivated by the fact that a robot can not take a close look on an object whose position is not known.
- Data Linkage
 - *add-meas*: This action inserts a measurement to the factor graph. See also Sec. V.
- Inference
 - *query-F_G*: The graph query has a pose as input that specifies reference frame and target frame. The action optimizes the factor graph with respect to the reference frame and returns the pose and uncertainty between reference frame and target frame. See also Sec. V.

C. Results

In the following, results of the experiment will be discussed from two different viewpoints. First we analyze the reduction of uncertainty in the belief. Subsequently, an analysis of the temporal characteristics of the generated plans is given.

1) *Uncertainty Reduction*: We only consider geometric uncertainty, more specifically positional and rotational uncertainty. For the assembly use case we are interested in the maximum standard deviation. Therefore, we define the standard deviations σ in the following as the maximum of the principal components of the separate 3x3 covariance matrices for position and rotation. The joint distribution of position and rotation is currently not considered.

In Fig. 4 we visualize the positional and rotational uncertainty of all pose estimates generated by the planning system. Note here again, that the pose estimate between the robot end-effector coordinate system c_{eff} and the target object coordinate system c_2 is considered. This has important implications for the following analysis. Hereafter, we will discuss specific poses denoted by p and marked in the figure:

- p_0 : The pose p_0 marks the initial belief over the target object o_2 . Positional and rotational uncertainty are both relatively large.
- p_1 : The pose p_1 denotes the belief after a 2D measurement between the static camera and the target object $m_{\text{sc},2}$. Due to the nature of the 2D measurement, the rotational uncertainty is not reduced with respect to p_0 .

- p_2 : The pose p_2 denotes the belief after a 2D measurement between the end-effector camera and the target object $m_{rc,2}$. The positional uncertainty is clearly lower than p_1 . This is due to the application setting. The end-effector camera is calibrated relative to the robot end-effector, with the calibration denoted by $m_{rob,rc}$. The uncertainty of p_2 is most heavily influenced by the measurement chain of camera calibration $m_{rob,rc}$ and measurement $m_{rc,2}$, which is a subset of the full factor graph as depicted in Fig. 3. Contrarily, for the uncertainty of p_1 the measurement chain of robot positioning uncertainty $m_{ref,rob}$, static camera calibration $m_{ref,sc}$ and static camera observation $m_{sc,2}$ is decisive. Obviously, this longer measurement chain including the robot positioning uncertainty induces larger uncertainty. The influence of the prior $m_{ref,2}$ on p_1 and p_2 is identical for both cases. As we model the belief as a factor graph, all measurement chains are jointly considered and the correct determination of the uncertainties is automatically handled by our approach.
- p_3 : The pose p_3 represents the belief after a 6D measurement with the static camera. The uncertainty is much lower than solely using 2D measurements.
- p_4 : This pose marks the lowest achievable belief uncertainty for this exemplary planning setting. It is based on the fusion of all 2D and 6D measurements for all cameras and robot configurations, namely the static camera o_{sc} , the end-effector camera c_{rc} in the initial pose of the robot and the end-effector camera c_{rc} in the close view pose of the robot.

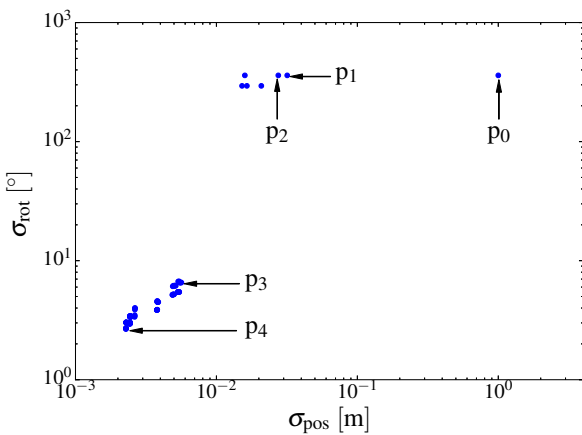


Fig. 4: Visualization of resulting pose uncertainties generated via the proposed configuration system for the assembly use case. Note that due to the axis configuration this figure reads from right to left and top to bottom as opposed to Fig. 5. A detailed description can be found in Sec. VI-C.1. The corresponding experimental setup is visualized in Fig. 3.

2) *Timing*: In this section we analyze the temporal extent of different plans. We therefore assume that an image is directly available whenever a perception action is applied. In Fig. 5 all belief poses are visualized by time stamp and positional uncertainty. Contrary to the previous section we

will additionally address sets of poses denoted by P . Furthermore, state transitions allowed in this planning problem are visualized via gray arrows.

The different actions have different timing characteristics. For the simulated experiment we assume the following values motivated by a real use case:

- *2D-meas*: 30 ms
- *6D-meas*: 100 ms
- *move-close*: 500 ms
- *add-meas*: 1 ms
- *query- F_G* : 10 ms

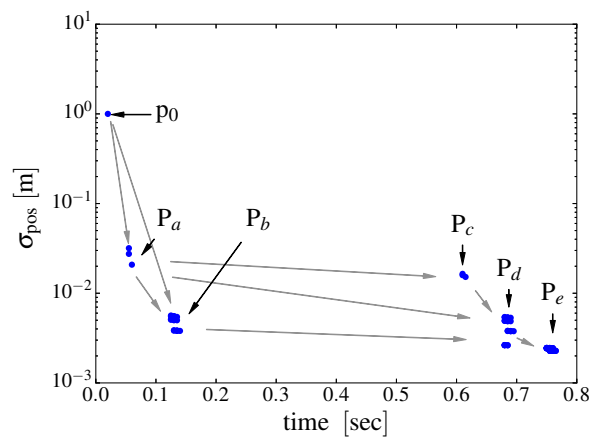


Fig. 5: Visualization of resulting pose uncertainties and the required execution time in a real system to achieve the belief state quality. A detailed description can be found in Sec. VI-C.2. The corresponding experimental setup is visualized in Fig. 3.

The characteristic of the different pose sets is discussed in the following.

- p_0 : The initial belief.
- P_a : Pose set P_a contains all poses that are estimated using fast 2D measurements without moving the robot.
- P_b : Pose set P_b contains all poses that are estimated using at least one 6D measurement without moving the robot.
- P_c : To reach this set, the robot view pose is changed and further 2D measurements are considered from the new robot pose. The motion of the robot takes a large amount of time, and the uncertainty is not decreased with respect to the pose set P_b . From application view it does not make sense to reach this belief.
- P_d : Pose set P_d is similarly reached via robot motion. Further it requires at least one 6D measurement from the new view pose.
- P_e : Pose set P_e is the set with the lowest reachable belief uncertainty in the given use case. It is reached by changing the view pose and fusing up to all possible measurements.

Concluding, it can be seen that time can easily be incorporated into the approach. Depending on the application requirements, the planning system generates different action sets. For instance, in an application where a fast position

estimate of an object is required, the actions leading to P_a are sufficient. In a less demanding assembly task, where the full pose needs to be known, the action sequence leading to P_b is chosen. Finally, in an application which requires very high accuracy, moving the robot to a better view pose is required and an action sequence leading to P_e is used. The key is that the configuration system, including the planner, can autonomously decide the right steps to take depending on the task.

The presented system represents an initial version of the general approach. To achieve autonomous configuration for a broad range of applications, the action set needs to be further extended and more advanced planning techniques evaluated to efficiently cope with the complexity of the task.

VII. CONCLUSION

In this paper we proposed a novel method to model and plan the configuration of perception systems. Our approach addresses the selection of sensor input and perception algorithms, perception planning and sensor fusion in an integrated yet modular fashion. Based on a semantic and probabilistic description of sensors and perception algorithms, our system plans elementary data gathering, data linkage and inference steps to achieve the goal. Uncertainties are handled in a generic manner by automatically building and inferring from a factor graph that represents the probabilistic relations between the involved entities. In this instance of the approach the probabilistic relations are of geometric nature, covering e.g. robot positioning tolerance, extrinsic camera calibration uncertainty and noisy observations. The entire process up to elementary operations is planned, which is thus enabling flexible adaptation to different settings and requirements.

We validated our approach in an industrial assembly scenario, where our planner successfully employs different sensors, data processing steps and view poses to localize the target part with sufficient accuracy in minimal time. An avenue for future work is the incorporation of further probabilistic properties, such as object classification probability, and evaluating the system on larger problems as well as a more extensive set of problem categories.

REFERENCES

- [1] M. Beetz, F. Bálint-Benczédi, N. Blodow, D. Nyga, T. Wiedemeyer, and Z.-C. Márton, "RoboSherlock: Unstructured information processing for robot perception," in *IEEE Int. Conf. on Robotics and Automation*. IEEE, 2015, pp. 1549–1556.
- [2] M. Tenorth and M. Beetz, "Knowrob: A knowledge processing infrastructure for cognition-enabled robots," *The Int. J. of Robotics Research*, vol. 32, no. 5, pp. 566–590, 2013.
- [3] M. Beetz, M. Tenorth, and J. Winkler, "Open-EASE," in *IEEE Int. Conf. on Robotics and Automation*. IEEE, 2015, pp. 1983–1990.
- [4] L. Kunze, T. Roehm, and M. Beetz, "Towards semantic robot description languages," in *IEEE Int. Conf. on Robotics and Automation*. IEEE, 2011, pp. 5589–5595.
- [5] 2004. [Online]. Available: <http://www.w3.org/Submission/OWL-S/>
- [6] D. Nyga, F. Balint-Benczedi, and M. Beetz, "PR2 looking at things - Ensemble learning for unstructured information processing with markov logic networks," in *IEEE Int. Conf. on Robotics and Automation*. IEEE, 2014, pp. 3916–3923.
- [7] N. Hochgeschwender, M. A. Olivares-Mendez, H. Voos, and G. K. Kraetzschmar, "Context-based selection and execution of robot perception graphs," in *IEEE Conf. on Emerging Technologies & Factory Automation*. IEEE, 2015, pp. 1–4.
- [8] N. Hochgeschwender, S. Schneider, H. Voos, and G. K. Kraetzschmar, "Towards a robot perception specification language," *4th Int. Workshop on Domain-Specific Languages and Models for Robotic systems*, 2013.
- [9] J. Elfring, S. van den Dries, M. Van De Molengraft, and M. Steinbuch, "Semantic world modeling using probabilistic multiple hypothesis anchoring," *Robotics and Autonomous Systems*, vol. 61, no. 2, pp. 95–105, 2013.
- [10] J. Elfring, R. van de Molengraft, and M. Steinbuch, "Semi-task-dependent and uncertainty-driven world model maintenance," *Autonomous Robots*, vol. 38, no. 1, pp. 1–15, 2015.
- [11] R. Bajcsy, "Active perception," *Proc. of the IEEE*, vol. 76, no. 8, pp. 966–1005, 1988.
- [12] R. Eidenberger and J. Scharinger, "Active perception and scene modeling by planning with probabilistic 6d object poses," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*. IEEE, 2010, pp. 1036–1043.
- [13] H. Hu and G. Kantor, "Efficient automatic perception system parameter tuning on site without expert supervision," in *Conf. on Robot Learning*, 2017, pp. 57–66.
- [14] S. Govindaraj, J. Gancet, M. Post, R. Dominguez, and F. Souvannavong, *InFuse: A Comprehensive Framework for Data Fusion in Space Robotics*. Infinite Study, 2017.
- [15] S.-F. Su and C. G. Lee, "Manipulation and propagation of uncertainty and verification of applicability of actions in assembly tasks," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 22, no. 6, pp. 1376–1389, 1992.
- [16] S. Blumenthal, H. Bruyninckx, W. Nowak, and E. Prassler, "A scene graph based shared 3d world model for robotic applications," in *IEEE Int. Conf. on Robotics and Automation*. IEEE, 2013, pp. 453–460.
- [17] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Trans. on Robotics*, vol. 32, no. 6, pp. 1309–1332, 2016.
- [18] M. Fox and D. Long, "PDDL 2.1: An extension to PDDL for expressing temporal planning domains," *J. of Artificial Intelligence Research*, 2003.
- [19] R. E. Fikes and N. J. Nilsson, "STRIPS: A new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, vol. 2, no. 3-4, pp. 189–208, 1971.
- [20] E. P. Pednault, "ADL and the state-transition model of action," *J. of Logic and Computation*, vol. 4, no. 5, pp. 467–512, 1994.
- [21] M. Helmert, "The fast downward planning system," *J. of Artificial Intelligence Research*, vol. 26, pp. 191–246, 2006.
- [22] J. Hoffmann and B. Nebel, "The FF planning system: Fast plan generation through heuristic search," *J. of Artificial Intelligence Research*, vol. 14, pp. 253–302, 2001.
- [23] K. Erol, J. Hendler, and D. S. Nau, "HTN planning: Complexity and expressivity," in *AAAI*, vol. 94, 1994, pp. 1123–1128.
- [24] D. S. Nau, T.-C. Au, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu, and F. Yaman, "SHOP2: An HTN planning system," *J. of Artificial Intelligence Research*, vol. 20, pp. 379–404, 2003.
- [25] D. Koller and N. Friedman, *Probabilistic graphical models: principles and techniques*. MIT press, 2009.
- [26] F. Dellaert, "Factor graphs and GTSAM: A hands-on introduction," Georgia Institute of Technology, Tech. Rep., 2012.
- [27] R. Platt Jr, R. Tedrake, L. Kaelbling, and T. Lozano-Perez, "Belief space planning assuming maximum likelihood observations," in *Robotics: Science and Systems*, 2010.
- [28] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *IEEE Conf. on Computer Vision and Pattern Recognition*, 2016, pp. 779–788.
- [29] M. Rad and V. Lepetit, "Bb8: A scalable, accurate, robust to partial occlusion method for predicting the 3d poses of challenging objects without using depth," in *Int. Conf. on Computer Vision*, 2017.

12

Automatic Configuration of Perception Pipelines

Title	Automatic Configuration of the Structure and Parameterization of Perception Pipelines
Authors	Vincent Dietrich, Bernd Kast , Michael Fiegert, Sebastian Albrecht, and Michael Beetz
ISBN/ISSN	78-1-7281-2467-4
DOI	ICAR46387.2019.8981611
Status	published
Publisher	IEEE
Contribution of Bernd Kast	I brought in the experience of hierarchization and templating. Vincent Dietrich implemented and conducted the experiments. He also wrote the manuscript, which benefited from the co-authors' scientific remarks.
Summary	The setup of a perception system is not only about the right choice of components, such as sensors and algorithms and their wiring, but also about the parameterization of each component for the specific domain. This paper compares and discusses different approaches to incorporating parameter optimization and component sequencing. The basic structure for all examined algorithms is hierarchically refinable, which acts as a plan template. Due to our formal model, we can compile engineering knowledge into templates, which reduces the plan space and ensures reasonable planning times.

Automatic Configuration of the Structure and Parameterization of Perception Pipelines

Vincent Dietrich¹Bernd Kast¹Michael Fiebert¹Sebastian Albrecht¹Michael Beetz²

Abstract—The configuration of perception pipelines is a complex procedure that requires substantial amounts of engineering effort and knowledge. A pipeline consists of interconnected individual perception operators and their parameters, which leads to a large configuration space of pipeline structures and parameterizations. This configuration space has to be explored efficiently in order to find a solution that fulfills the specific requirements of the target application. In this paper, we present an approach to perform automatic configuration based on structure templates and sequential model-based optimization. The structure templates allow to reduce the search space and encode prior engineering knowledge. We introduce a structure template based on hypothesis generation, hypothesis refinement, and hypothesis testing to demonstrate the effectiveness of the approach. Experimental evaluation with state-of-the-art operators is performed on data from the T-LESS dataset as well as in a real-world robotic assembly task.

I. INTRODUCTION

Object detection and 6D pose estimation is crucial in many application domains ranging from augmented reality, over autonomous driving to manufacturing and logistics. Especially in the domains which involve physical interaction with objects such as material handling and assembly, the targeted objects, environments, and accuracy requirements vary significantly across applications. Therefore, the task specific engineering effort for the configuration of perception pipelines is high.

The pipeline building blocks are individual perception operators, which are typically taken from available software projects and libraries. Given a specific sensor setup, the perception engineer has to build and parameterize a pipeline that fulfills the application requirements. The process to get to a working pipeline usually requires engineering knowledge and intuition in choosing the right operators and tuning the parameters. The goal of the presented approach is to provide a practical method to automate this process.

Automatic configuration of perception pipelines is a difficult problem due to the following aspects:

- **Complexity:** Both the number of possible operators and pipeline structures as well as the parameter space are large, which leads to a combinatorial explosion. Not all variants and parameter settings can be evaluated in a brute force fashion. Therefore, approaches are required to efficiently structure and reduce the search space.
- **Uncertainty:** Perception operators are generally subject to uncertainty and errors in their predictions. Therefore, single data points are not enough to assess the system

¹Siemens Corporate Technology, Munich, Germany

²Institute for Artificial Intelligence, University Bremen, Germany

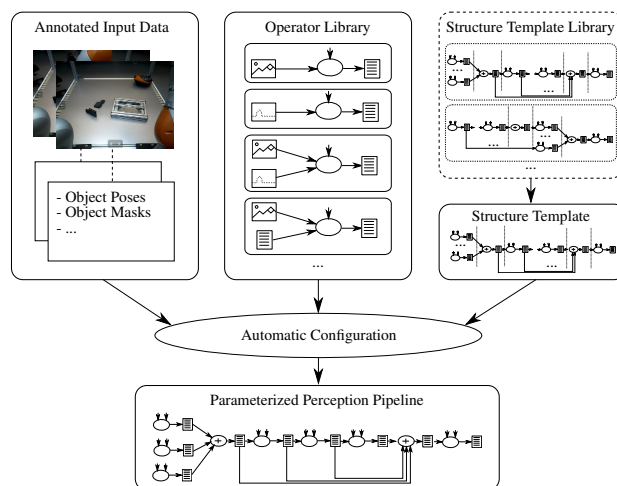


Fig. 1: Overview of the automatic configuration approach. The application specific task is specified via an annotated input data set. A pipeline structure template is used to reduce the search space. The template can be chosen from a structure template library. Operators from an operator library are automatically selected, placed and parameterized within the template during the configuration procedure.

performance. Rather large datasets have to be used, which are computationally demanding to evaluate.

- **Sensitivity:** Despite current advances, perception pipelines are still sensitive to the application setting. Varying lighting conditions, sensor noise, ranges, and texture can all contribute to decreasing performance in different application settings. This is especially true for model-based (classical) operators, whereas data-driven operators such as neural networks highly depend on the provided training data. The dependency of pipeline performance on object properties can be clearly seen in the BOP benchmark [1].

In this paper, we present an approach for the automatic configuration of perception pipelines. An overview of the approach is depicted in Fig. 1. The main contributions are:

- a template based structuring approach for perception pipelines to reduce the configuration space and encode prior engineering knowledge,
- a pipeline structure template based on hypothesis generation, hypothesis refinement, and hypothesis testing,
- an approach to automatically configure the perception pipeline structure and parameterization for combined model-driven and data-driven setups, and
- a demonstration of the applicability of the approach

on an open source dataset as well as in a real-world assembly scenario.

After an overview over related work, we first present a problem formalization in Sec. III and then introduce our approach in Sec. IV. Finally, in the experimental section we demonstrate the applicability and discuss characteristics of our approach.

II. RELATED WORK

A. AutoML

AutoML addresses the tuning of hyper-parameters, algorithm choice, and architecture search for machine learning. For the former, approaches like Bayesian Optimization [2], Genetic Programming [3], and Sequential Model-Based Optimization [4] have been proposed. Tools such as AutoWEKA [5], Auto-sklearn [6], AutoKeras [7], and TPOT [3] are designed for specific machine learning libraries, while others such as SMAC [4] and Hyperopt [8] are designed for general use. The authors of AutoWEKA [5] introduce the problem of *Combined Algorithm Selection and Hyper-parameter Optimization* (CASH) for machine learning operators. We address a related problem for a mixed set of perception operators. Neural Architecture Search (NAS) addresses the adaptation of network architectures and is a promising field. For a review we refer to [9]. In this publication we target mixed pipelines of classic algorithms and neural networks, which is not in the scope of NAS.

AutoML techniques have been applied to, among other data pre-processing pipelines [10], parameter tuning, and algorithm selection for classification [6] and SAT solver parameter tuning [4]. In this work AutoML techniques and tools are applied in the concrete application domain of perception pipeline parameter and structure configuration for a diverse set of operators.

B. Perception System Engineering

Another relevant field of research is the automated design of perception and sensor fusion software systems, which is typically solved by some sort of design space exploration. One line of research is the use of semantic models in order to describe the task and generate appropriate perception pipelines, such as [11], [12], [13]. Beetz *et al.* [14] use a query-answering approach and semantic models in order to generate perception pipelines at run-time. Hochgeschwender *et al.* [15] use the Robot Perception Specification Language [16] in order to describe and select perception pipelines and choose the pipeline parameterization from a pre-configured set depending on the current environment state. In our previous work [17], we use a semantic model in order to plan a sequence of perception and sensor fusion actions in order to achieve the required pose estimation accuracy for a given task.

Other approaches directly target the online and offline adaptation of perception pipeline parameters. For instance, Sakar *et al.* [18] model the parameters within a Bayesian parameter dependence network in order to cope with the search space complexity and dependencies. The authors

of [19] propose an approach to automatically tune the system parameterization online without expert supervision. Durner *et al.* [20] use logged execution data in order to optimize the parameterization of different perception pipelines.

We present an approach for the joint configuration of perception pipeline structure and parameterization in contrast to previous work where both are regarded as separate problems.

III. PROBLEM FORMALIZATION

In order to formalize the targeted problem, we start with the representation of the available knowledge about the system, as specialization of the prior work [21] and [22]. Subsequently, the notations of dataset and error metrics are introduced and finally, the pipeline configuration problem formalized.

A. Concept, Operator and Pipeline

First, we denote the different data types and representations, the declarative knowledge, as *concepts*. Exemplary concepts are image, bounding box, object mask, and pose. An actual given image is an instance of the *image* concept. The set of all instances is denoted as \mathbb{M} and a concept C is a subset thereof: $C \subseteq \mathbb{M}$.

An *operator* $O \in \mathbb{O}$ denotes a function that maps a set of input instances to a set of output instances and optionally takes a parameter vector λ :

$$O : \begin{cases} C_{in}^1 \times \dots \times C_{in}^{n_{in}} \times \Lambda \rightarrow C_{out}^1 \times \dots \times C_{out}^{n_{out}} \\ (i_{in}^1, \dots, i_{in}^{n_{in}}, \lambda) \mapsto (i_{out}^1, \dots, i_{out}^{n_{out}}) \end{cases}, \quad n_{in}, n_{out} \in \mathbb{N}, \quad (1)$$

with $C_{in}^j, j = 1, \dots, n_{in}$, and $C_{out}^k, k = 1, \dots, n_{out}$, being input and output concepts of the operator, i_{in}^j and i_{out}^k the respective instances of aforementioned concepts, Λ the set of parameterizations, and \mathbb{O} the set of operators. A set of instances is denoted as I , such as the set of input instances I_{in} and output instances I_{out} . An example with the Single Shot Pose operator [23] is depicted in Fig. 2. An operator that is parameterized with the parameter vector λ is denoted as O_λ .

A *pipeline* $P \in \mathbb{P}$ is composed of several operators whose inputs and outputs are connected in a graph structure. The set \mathbb{P} denotes the set of all possible pipeline structures, which depends on the available instances and operators and may be restricted by additional constraints. An example is depicted in Fig. 2, where the Single Shot Pose operator is succeeded by an iterative closest point matching (ICP). Both require different inputs, the former a RGB image and the latter a point cloud and initial poses in the form of a hypothesis list. A pipeline can be represented as an individual operator as shown in Fig. 2. Both operators have the individual parameterizations λ_S and λ_I , which are concatenated to a single parameter vector λ_{SI} .

B. Dataset and Error Metrics

In this work, we characterize the targeted application domain of the perception pipeline via a given annotated

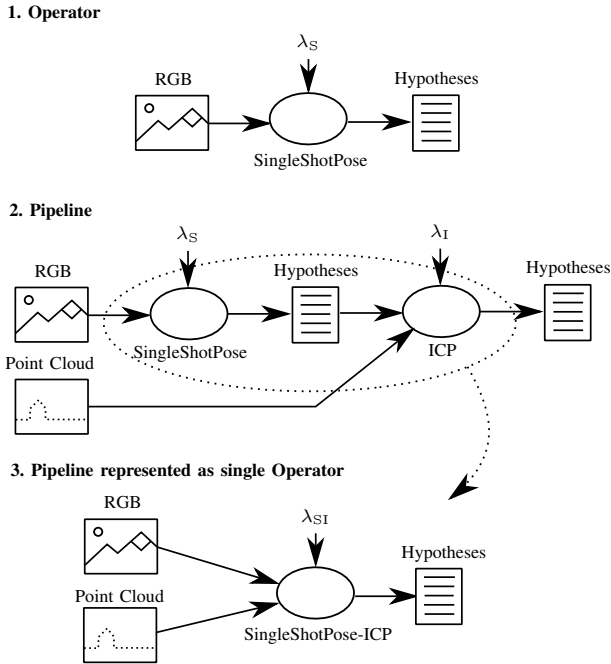


Fig. 2: Exemplary operator, pipeline, and reduction of a pipeline to an individual operator.

dataset \mathbb{D} . The dataset needs to reflect the application properties, e.g., the expected distributions of scenes, sensor noise, and environment conditions.

We model the dataset as a list of scene instance sets d_j , with $j = 1, \dots, m$:

$$\mathbb{D} = \{d_1, d_2, \dots, d_m\}, m \in \mathbb{N}, \quad (2)$$

where each scene instance set contains the data for a specific scene. The scene instance sets are composed of input instances I_{in} , ground truth instances I_{gt} , and expected pipeline output instances I_{exp} :

$$d = (I_{in}, I_{gt}, I_{exp}), \text{ with } I_{in}, I_{gt}, I_{exp} \subseteq \mathbb{M}. \quad (3)$$

Input instances include different sensor data inputs, such as images and point clouds and prior belief knowledge. Ground truth instances describe the scene state in different (intermediate) representations such as object poses, masks, and bounding boxes. Expected output instances represent the information that is required by the application, such as a list of objects and their poses. The expected output instances are typically a subset of the ground truth instances.

In order to formulate the actual configuration problem, a metric is required to quantify the pipeline performance on a dataset. Therefore, we define the metric $e_{\mathbb{D}}$ for the parameterized pipeline P_{λ} and the dataset \mathbb{D} as the average of the individual error for each scene instance set $e_d(P_{\lambda}, d_j)$:

$$e_{\mathbb{D}}(P_{\lambda}, \mathbb{D}) = \frac{1}{m} \sum_{j=1}^m e_d(P_{\lambda}, d_j), \quad (4)$$

with $d_j \in \mathbb{D}, m = |\mathbb{D}|$.

The scene instance set error $e_d(P_{\lambda}, d_j)$ requires the execution of the pipeline and computation of the instance error metric e_i between the pipeline output I_{out} and the expected instances I_{exp} :

$$\begin{aligned} e_d(P_{\lambda}, d_j) &= e_d(P_{\lambda}, (I_{in}, I_{gt}, I_{exp})) \\ &= e_i(P_{\lambda}(I_{in}), I_{exp}) \\ &= e_i(I_{out}, I_{exp}). \end{aligned} \quad (5)$$

The instance error metric e_i depends on the application and the type of expected pipeline output instances I_{exp} and is chosen accordingly.

C. Pipeline Configuration

Now we can introduce the configuration of pipeline structure and parameterization as an optimization problem:

$$(P^*, \lambda^*) \in \underset{P \in \mathbb{P}, \lambda \in \Lambda_P}{\operatorname{argmin}} e_{\mathbb{D}}(P_{\lambda}, \mathbb{D}), \quad (6)$$

where Λ_P is the parameterization space of pipeline P . Typically, this can only be solved by derivative-free optimization methods due to missing derivative information for most operators and structure adaptations. In this general problem representation, the structural variety is encoded via \mathbb{P} . In the following section, we introduce structural elements and structure parameters to model and optimize the set of pipeline structures.

IV. APPROACH

A general overview of our approach is depicted in Fig. 1. The input is annotated data, a library of perception operators and a pipeline structure template that may be taken from a template library. The operator library contains the description of inputs, outputs, and parameters as introduced in the previous section. The pipeline template allows to narrow down the search space, based on prior engineering knowledge. Different templates can for instance cover different sensor and data fusion architectures [24].

A. Pipeline Template Model

We employ a pipeline template model, where structural elements of the pipeline are parameterized via structure parameters $\tilde{\lambda} \in \tilde{\Lambda}$, $\tilde{\Lambda} \subseteq \Lambda$ that encode the operator instances and their sequence of execution. The template we employed as working example is displayed in Fig. 3. We structure the pipeline template in three main structural elements: hypothesis generation, hypothesis refinement, and hypothesis scoring. The structural elements are configured via the respective structure parameters $\tilde{\lambda}_G$, $\tilde{\lambda}_R$, and $\tilde{\lambda}_S$.

During hypothesis generation, the resulting hypotheses of different pose estimation pipelines are accumulated to a common list of all initial hypotheses. In the following hypothesis refinement phase, the list of hypotheses is subsequently refined by a parameterized number of operators. Finally, all intermediate hypotheses are gathered and scored. This structure represents an instance of a hypothesize-and-test strategy. The actual operator instantiation of the structural elements is encoded via structure parameters, which represent the

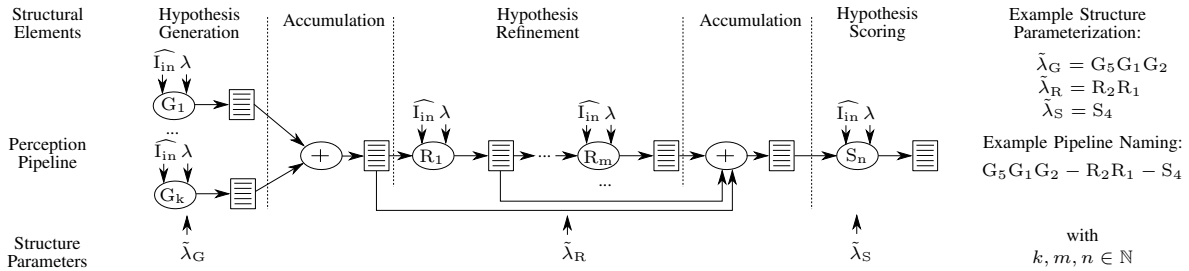


Fig. 3: Pipeline template that is used as working example. It is separated in three parameterizable structural elements, hypothesis generation, hypothesis refinement, and hypothesis scoring. The hypotheses can be generated by multiple operators. The refinement is a sequential procedure, where each refinement operator gets its input hypotheses from the preceding operator. The structural elements are parameterized via the structure parameters $\tilde{\lambda}_G$, $\tilde{\lambda}_R$, and $\tilde{\lambda}_S$.

employed operators as ordered lists. All different variants are encoded in the set of parameter values, which depends on the available operators $G, R, S \in \mathcal{O}$ for each structural element. For instance, given the two operators R_1 and R_2 , the set of parameterizations is

$$\tilde{\Lambda}_R(\{R_1, R_2\}) = \{\emptyset, R_1, R_2, R_1R_2, R_2R_1\}, \quad (7)$$

with $\tilde{\lambda}_R \in \tilde{\Lambda}_R(\{R_1, R_2\})$ and $\tilde{\lambda}_R \in \Lambda$.

The pipeline structure template additionally defines conditions on the inputs and outputs of the operators for different structural elements. In the working example, hypothesis generation operators G must produce a hypothesis list as output and may not require a hypothesis list as input. Refinement operators R require a hypothesis list as input and must produce a hypothesis list as output. The hypothesis scoring operators S transform hypotheses to scored hypotheses.

B. Optimization

In the following, we introduce two different optimization strategies on top of *sequential model-based optimization* (SMBO) [4]. In brief, SMBO performs black box optimization effectively by building a computationally efficient performance prediction model, that is used to evaluate the majority of parameter configurations.

With the different strategies we aim to gain insight, whether the joint optimization of all parameters is to be preferred over a strategy where operators are optimized individually as a prior step. We start with a description of common initializations and proceed with the strategies.

First, the available operators from the operator library are matched to the structural elements of the pipeline template, which is performed based on the template conditions for the different structural elements. Additionally, the required input of the operators has to be given within the dataset. The set of values for the structure parameters is initialized by generating the possible sequences of the operators for each structural element. Finally, operator and structure parameters are added to the optimization configuration space. The operator parameters are additionally conditioned on the structure parameters such that they are only considered when the operator is within the currently chosen operator sequence of the structure parameter.

The dataset \mathbb{D} is split into a training set $\mathbb{D}_{\text{train}}$ and a test set \mathbb{D}_{test} . The test set is used as a holdout dataset to assess the generalization of the pipeline structure and parameterization. We furthermore perform the training of data-driven operators, such as neural networks as initialization step. For this step only $\mathbb{D}_{\text{train}}$ as well as additional simulated data can be used.

1) *Joint-Optimization Strategy JointOpt*: Within the joint-optimization strategy, structure parameters and operator individual parameters are jointly optimized. Therefore, the responsibility to guide the search process is solely with the optimizer.

2) *Pre-Optimization Strategy PreOpt*: For the pre-optimization strategy the assumption is made that individually optimized operators are performing well within the pipeline structure. Therefore, prior to the optimization of structural parameters, the parameters of individual operators are optimized. This requires the following conditions to be fulfilled:

- input instances I_{in} and ground truth instances I_{gt} must be available for the input and output of the operator and
- the error metric $e_i(I_{\text{out}}, I_{\text{exp}})$ needs to be defined for the operator output types.

The operator individual optimization is followed by a joint optimization of the structural parameters and the remaining operator parameters. The previously optimized parameters of the individual operators are fixed.

C. Test Metric

In order to evaluate the results and stop the optimization we use a test error e_{test} where an acceptance threshold θ is applied on the instance error of the test dataset:

$$e_{\text{test}}(P_\lambda, \mathbb{D}_{\text{test}}) = \frac{1}{n} \sum_{j=1}^n \begin{cases} 1 & \text{if } e_i(P_\lambda(I_{\text{in}}), I_{\text{exp}}) > \theta \\ 0 & \text{otherwise} \end{cases}, \quad (8)$$

with $d_j = (I_{\text{in}}, I_{\text{gt}}, I_{\text{exp}}) \in \mathbb{D}_{\text{test}}$, $n = |\mathbb{D}_{\text{test}}|$. The threshold θ results from the application requirements. The test error reflects the ratio of data instance sets where the application requirements are not fulfilled.

V. EVALUATION

We evaluate the proposed pipeline configuration approach in different experiments. First, we use a subset of the T-LESS

Symbol	Name	Input	Output	Parameters	Description
O_M	MaskRCNN	RGB image	Object masks	Neural network weights, default hyper-parameters	The Matterport implementation [25] is used. The network is trained on all available object classes in the target scene.
O_P	Point Pair Feature Matcher (PPF)	Point cloud, Object masks	Pose hypotheses	29 parameters	Engineered pose estimation pipeline consisting of HALCON [26] 18.11 operators: <code>create_surface_model</code> , <code>find_surface_model</code> , <code>sample_object_model_3d</code> , and <code>surface_normals_object_model_3d</code> . The pipeline is applied on each input point cloud cluster, generated via object mask.
G_{MP}	MaskRCNN + PPF	RGB image, Point cloud	Pose hypotheses	Combined from O_M and O_P	Composed pipeline of the MaskRCNN and point pair feature matcher.
G_S	Single Shot Pose	RGB image	Pose hypotheses	Neural network weights, default hyper-parameters	The open source implementation of [23] is used. The output is restricted to top 5 candidates. The network is trained only for the target object class.
R_I	Iterative Closest Point	Point cloud, Pose hypotheses	Pose hypotheses	7 parameters	ICP implementation from Open3D library, version 0.3.0.0 [27] with optional down-sampling and normal estimation, using the following functions: <code>uniform_down_sample</code> , <code>estimate_normals</code> , and <code>registration_icp</code> . The runtime is restricted by the maximum number of iterations. A rendered point cloud of the hypothesis is used as object cloud.
R_H	Shape-based Refinement	RGB, Pose hypotheses	Pose hypotheses	26 parameters	Engineered pose refinement pipeline consisting of the following operators from HALCON [26] version 18.11 : <code>read_object_model_3d</code> , <code>create_shape_model_3d</code> , and <code>find_shape_model_3d</code> . The shape model is generated during runtime with a parameterizable depth and rotation delta around the initial hypothesis. Optionally, a region of interest can be applied around the initial pose hypothesis.
R_D	Depth Adaptation	Depth image, Pose hypotheses	Pose hypotheses	-	Position adaptation along the ray from optical frame to pose hypothesis using the depth delta between input depth image and rendered depth image. This operator allows to roughly compensate for erroneous distance estimations.
S_R	Depth Delta Score	Depth image, Pose hypotheses	Pose hypotheses	-	Annotates each hypothesis with a score based on the depth difference between rendered hypothesis object depth and actual depth.

TABLE I: Overview of the operators used within the experiments. Additionally required concepts such as CAD models and camera intrinsics are left out for the sake of simplicity.

Dataset [28] for the comparison of the optimization strategies and analysis of different aspects of the overall approach. In the second experiment, the approach is applied to an industrial assembly scenario, where the automatic configuration is used to determine a working pipeline and parameterization for the given assembly task. As the experimental setup is shared to significant parts, common elements are introduced first.

A. Common Experimental Setup

The set of operators used in the experiments is listed in Tab. I. The operators are chosen such that different approaches for 6D pose estimation and refinement are represented. The shape refinement operator R_H and depth adaptation operator R_D are custom engineered. The operator G_{MP} is composed of the MaskRCNN operator O_M and the Point Pair Feature Matcher O_P . Individually, the operators O_M and O_P are not matched within the pipeline template due to wrong output type and missing input. In these experiments, each operator may only be inserted once in each structural element. Therefore, the initialization of the structure parameters given the set of operators results in 64 distinct pipeline structures.

The overall dataset is split into 50% training and 50% test data. The pre-optimization is only performed on G_{MP}

according to the strategy definition. The parameter default values and ranges are initialized according to recommended values in the documentation, if available.

As error metric e , we use the visual surface discrepancy e_{VSD} as defined in the BOP benchmark [1], with the misalignment tolerance τ . We set $\tau = 0.02$ m, in accordance to the BOP benchmark. We evaluate the results for the acceptance thresholds $\theta = 0.3$ and $\theta = 0.15$. For the error calculation only the hypothesis with the highest score computed via S_R is considered.

The framework is written in Python. As sequential model-based black box optimizer we use PySMAC version 0.10.0 [4]. The experiments are computed on a Intel Core i7-4810MQ CPU. The training time of the neural networks is not considered in the following diagrams.

B. Strategy Comparison

In this first experiment, we apply our approach on a perception problem from the T-LESS dataset [28] and compare the different optimization strategies. Only the T-LESS scene *scene.09* and object *obj.03* are used for this experiment. The strategy evaluation is performed on a randomly sampled subset of the overall dataset with a size of 100 images, which is kept constant across all experiments. The strategies **JointOpt** and **PreOpt** are evaluated for 6 different opti-

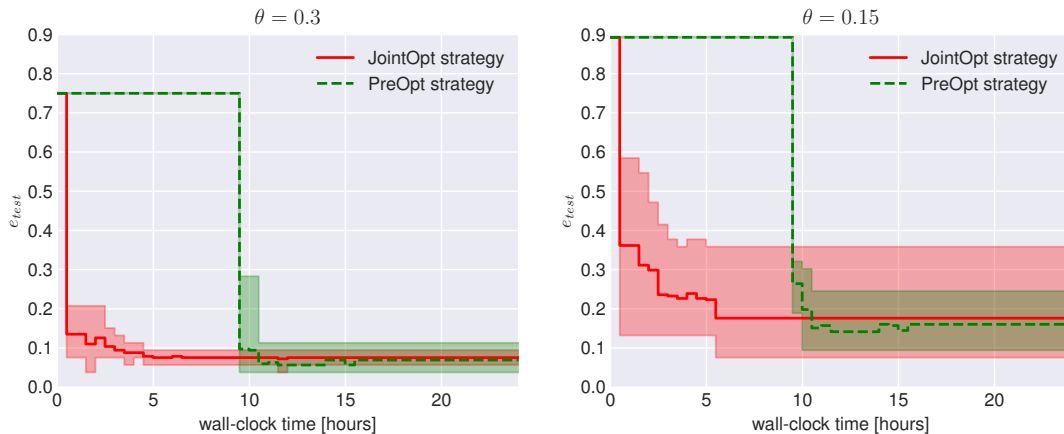


Fig. 4: Error trajectories for the strategies **JointOpt** and **PreOpt** for different error thresholds. For better visibility the outer hull and mean values of the individual trajectories of the 6 different seeds are displayed. The initial plateau for the **PreOpt** strategy represents the pre-optimization of individual operators. On average, the performance of both strategies is similar.

Strategy	Seed	Pipeline	Test Error $\theta = 0.15$
JointOpt	4	$G_S G_{MP} - R_H R_D - S_R$	0.08
PreOpt	5	$G_S G_{MP} - R_H R_D R_I - S_R$	0.09
PreOpt	2	$G_S G_{MP} - R_H R_D - S_R$	0.11
JointOpt	0	$G_S - R_H R_D - S_R$	0.13
PreOpt	0	$G_S G_{MP} - R_H R_D R_I - S_R$	0.13
JointOpt	2	$G_S G_{MP} - R_H R_D - S_R$	0.15
JointOpt	3	$G_{MP} G_S - R_H R_D - S_R$	0.15
PreOpt	1	$G_S G_{MP} - R_H - S_R$	0.17
JointOpt	1	$G_S G_{MP} - R_H R_D R_I - S_R$	0.19
PreOpt	4	$G_{MP} G_S - R_H R_D - S_R$	0.21
PreOpt	3	$G_S - R_H R_D - S_R$	0.24
JointOpt	5	$G_{MP} G_S - R_H - S_R$	0.36

TABLE II: Test error for the best results for 6 different seeds of the **PreOpt** and **JointOpt** strategies. Result are sorted by the test error with a threshold $\theta = 0.15$

mization seeds. The stopping criterion for the optimization is either a zero valued test error or a maximum runtime. For the **JointOpt** strategy the maximum runtime is set to 24 h. The **PreOpt** strategy consists in two optimization phases, the pre-optimization with a maximum runtime of 9 h and the joint optimization with a maximum runtime of 15 h, such that both strategies run within 24 hours.

In Fig. 4, the outer hull and mean of the resulting test error trajectory for the different optimization seeds are displayed for both strategies for a threshold θ of 0.3 and 0.15 respectively. The pre-optimization duration of maximum 9 h of the **PreOpt** strategy is displayed as default error plateau within the **PreOpt** strategy of the minimal default pipeline $G_S - S_R$. Additionally, the resulting pipelines for the different strategies and optimization seeds together with their test error are displayed in Table II.

Several characteristics can be observed. First, the resulting minimum absolute test error and error variance is higher for the more difficult case of $\theta = 0.15$, which is the expected

outcome. The best performing configuration for each seed is typically found within the first 6 hours of optimization, which leads to a flat error curve.

For $\theta = 0.3$ the **PreOpt** strategy has higher variance, whereas for $\theta = 0.15$ the error trajectories of the **JointOpt** strategy display higher variance. The mean error in both settings is slightly better for the **PreOpt** strategy. The **PreOpt** strategy also shows a faster convergence rate, as the parameter search space is smaller once the pre-optimization is finished. In this setup, no clear strategy preference can be deduced. In terms of resulting pipeline structures, there is a strong presence of the refinement operator R_H , especially at the first position. The refinement pipeline $R_H R_D$ is most frequent within the results.

The resulting pipelines generally are compatible with engineering intuition. The R_H operator improves the orientation estimate, but remains with erroneous depth estimation due to its RGB only input. The R_D operator may correct the depth estimation and is mostly placed afterwards. The ICP (operator R_I) is placed last in the top performing pipelines, which can be explained by the required accuracy in the initial guess. Interestingly it is not used in most of the pipelines. This is possible since the ICP may produce wrong hypotheses with good final depth score in the given setup.

Overall, there is a benefit of searching structure and parameter search space in order to improve the perception performance in the given setup. But the sensitivity with respect to the initial seed can be high.

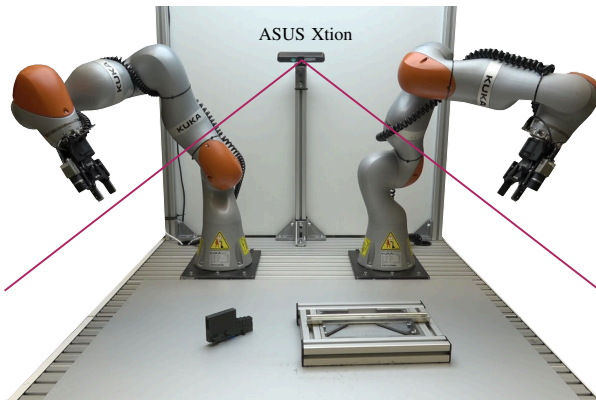
Additionally, it can be noted here that the authors tried to configure the perception system manually within the same configuration space. The results lacked behind in terms of perception performance and overall cost. Moreover, parameter tuning is a very tedious procedure where the perception engineer has to wait regularly for evaluation results on the dataset. But not all configuration space dimensions are explored in an automatic fashion. In the presented state of the

work the perception engineer still has to decide on structure templates and the hardware setup to be used. Also, the design of entirely new operators and perception algorithms is still out of scope.

C. Real Assembly Scenario

The second experiment is a real-world robotic assembly experiment. The task is to localize a control cabinet part and mount it on a hat rail. The part is a SIMATIC ET 200S terminal module. The manipulators are two seven-axis robots with parallel 2-finger grippers and the sensor is an ASUS Xtion PRO LIVE. The overall setup is displayed in Fig. 5. For the manipulation planning and assembly motion generation we use the approaches from [29] and [30].

Start configuration:



Target configuration:

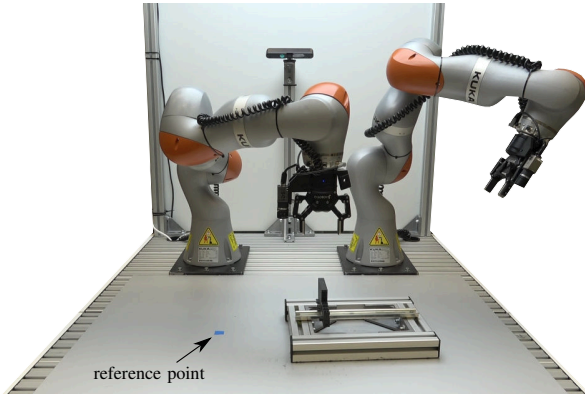


Fig. 5: Experimental setup for the pipeline evaluation in an assembly scenario. The object is placed in 16 different orientations on a reference point. The goal is to mount the object on the hat rail and success and failure are determined via successful snap-in. The experiment is repeated twice, resulting in 32 trials overall.

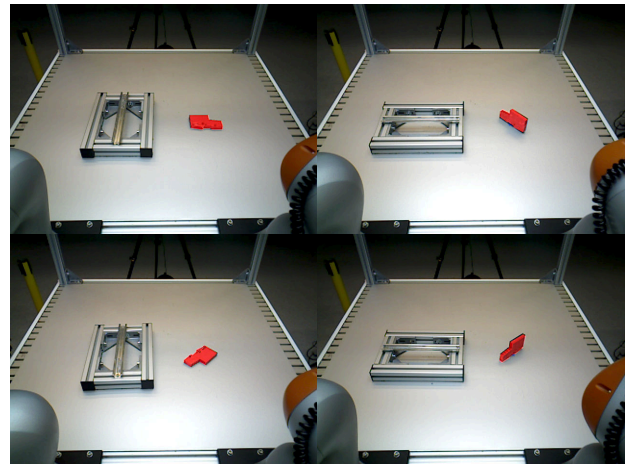
In order to assess the benefit of using the pipeline parameter and structure optimization, we compare the performance of different optimized pipelines based on the assembly success. The used dataset for the pipeline configuration consists of 100 images with multiple object instances that were acquired using a slow but accurate perception pipeline, where additional wrist-mounted cameras are placed close to the objects in order to ensure high accuracy. As configuration strategy the **JointOpt** strategy is used with a maximum

Pipeline	Assembly Success
$G_{MP}G_{S-RH}R_D R_I-S_R$	26 / 32
$G_{S-RH}R_D R_I-S_R$	22 / 32
$G_{MP}-S_R$	16 / 32
G_S-S_R	5 / 32

TABLE III: Pipeline performance in the assembly scenario, sorted by success rate.

runtime of 12 hours and different seeds. The choice for the **JointOpt** strategy is arbitrary according to the comparison results. The authors prefer it here, as structure and parameters are computed in the same optimization step.

Successful detections:



Non successful detections:

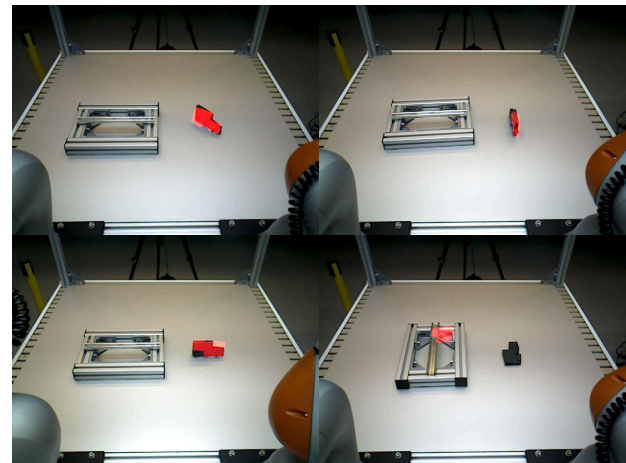


Fig. 6: Exemplary pose estimation results for successful and non-successful assembly operations. Best viewed in color.

For the actual experiments the part is placed on the reference point, see Fig. 5, either lying on the flat side or standing as depicted. These orientations correspond to the acquired training data. Additionally, after each run the object is rotated around the upright axis about 45° with respect to the previous pose. This yields 16 different poses overall. The experiment is performed twice, which results in 32 overall

assembly trials. The trial is counted as success, when the part snaps and remains on the hat rail. For the experiments the hat rail is localized with the same pipeline as used for the dataset generation.

The results are shown in Table III. The compared pipelines are the individually optimized basic hypothesis generation pipelines Mask-RCNN + PPF (G_{MP}) and Single Shot Pose (G_S) and two different resulting pipelines from the optimization $G_{MP}G_S-R_H R_D R_I-S_R$ and $G_S-R_H R_D R_I-S_R$. The hypothesis refinement pipeline $R_H R_D R_I$ showed to be good in this experimental setting, whereas $R_H R_D$ performs better in the T-LESS use case.

In summary, it can be deduced, that the joint optimization of pipeline structure and parameterization leads to higher success rates compared to the base operators G_{MP} and G_S . Still, there remains a performance gap, due to high process requirements, sensor noise, operator insufficiencies, and the distance to the object. It could be addressed, for instance, by improved generation of training data for the neural networks and a different choice and positioning of the sensor. The integration of these configuration space dimensions is conceptually feasible and within the scope of future work.

VI. CONCLUSION

In this paper, we present an approach to perform automatic configuration of perception pipelines based on structure templates and sequential model-based optimization. The approach allows to determine suitable parameters as well as a suitable pipeline structure in order to adapt to the specific task setting. In experiments on the T-LESS dataset as well as a real-world robotic assembly scenario we could demonstrate that large performance improvements can be achieved when both pipeline structure and parameterization are jointly configured. Overall, the presented approach shows promising results in reducing manual engineering effort in the design of perception systems.

REFERENCES

- [1] T. Hodan, F. Michel, E. Brachmann, W. Kehl, A. Glent Buch, D. Kraft, B. Drost, J. Vidal, S. Ihrke, X. Zabulis, *et al.*, "BOP: benchmark for 6D object pose estimation," in *European Conf. on Computer Vision*, 2018.
- [2] S. Falkner, A. Klein, and F. Hutter, "BOHB: Robust and efficient hyperparameter optimization at scale," *arXiv:1807.01774*, 2018.
- [3] R. S. Olson, R. J. Urbanowicz, P. C. Andrews, N. A. Lavender, L. C. Kidd, and J. H. Moore, "Automating biomedical data science through tree-based pipeline optimization," in *European Conf. Applications of Evolutionary Computation*, 2016.
- [4] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *Int. Conf. on Learning and Intelligent Optimization*. Springer, 2011.
- [5] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Autoweka: Combined selection and hyperparameter optimization of classification algorithms," in *ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*. ACM, 2013.
- [6] M. Feurer, A. Klein, K. Egensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," in *Advances in Neural Information Processing Systems*, 2015.
- [7] J. Bergstra, "Hyperopt: distributed asynchronous hyper-parameter optimization," <https://github.com/hyperopt/hyperopt>, 2016.
- [8] H. Jin, Q. Song, and X. Hu, "Auto-keras: Efficient neural architecture search with network morphism," *arXiv:1806.10282*, 2018.
- [9] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *arXiv:1808.05377*, 2018.
- [10] A. Quemy, "Data pipeline selection and optimization," in *Int. Workshop on Design, Optimization, Languages and Analytical Processing of Big Data*, 2019.
- [11] H. Niemann, G. F. Sagerer, S. Schroder, and F. Kummert, "ERNEST: A semantic network system for pattern understanding," *Trans. on Pattern Analysis and Machine Intelligence*, vol. 12, no. 9, 1990.
- [12] N. J. J. P. Koenderink-Ketelaars, "A knowledge-intensive approach to computer vision systems," Ph.D. dissertation, Delft University of Technology, Netherlands, 2010.
- [13] A. Fritze, U. Mönks, C.-A. Holst, and V. Lohweg, "An approach to automated fusion system design and adaptation," *Sensors*, vol. 17, no. 3, 2017.
- [14] M. Beetz, F. Bálint-Benczédi, N. Blodow, D. Nyga, T. Wiedemeyer, and Z.-C. Márton, "RoboSherlock: Unstructured information processing for robot perception," in *Int. Conf. on Robotics and Automation*. IEEE, 2015.
- [15] N. Hochgeschwender, M. A. Olivares-Mendez, H. Voos, and G. K. Kraetzschmar, "Context-based selection and execution of robot perception graphs," in *Conf. on Emerging Technologies & Factory Automation*. IEEE, 2015.
- [16] N. Hochgeschwender, S. Schneider, H. Voos, and G. K. Kraetzschmar, "Declarative specification of robot perception architectures," in *Int. Conf. on Simulation, Modeling, and Programming for Autonomous Robots*. Springer, 2014.
- [17] V. Dietrich, B. Kast, P. Schmitt, S. Albrecht, M. Fiegert, W. Feiten, and M. Beetz, "Configuration of perception systems via planning over factor graphs," in *Int. Conf. on Robotics and Automation*. IEEE, 2018.
- [18] S. Sarkar and S. Chavali, "Modeling parameter space behavior of vision systems using bayesian networks," *Computer Vision and Image Understanding*, 2000.
- [19] H. Hu and G. Kantor, "Efficient automatic perception system parameter tuning on site without expert supervision," in *Conf. on Robot Learning*, 2017.
- [20] M. Durner, S. Kriegel, S. Riedel, M. Brucker, Z.-C. Márton, F. Bálint-Benczédi, and R. Triebel, "Experience-based optimization of robotic perception," in *Int. Conf. on Advanced Robotics*. IEEE, 2017.
- [21] B. Kast, S. Albrecht, W. Feiten, and J. Zhang, "Bridging the gap between semantics and control for industry 4.0 and autonomous production," in *Int. Conf. on Automation, Science and Engineering*, 2019.
- [22] B. Kast, V. Dietrich, S. Albrecht, W. Feiten, and J. Zhang, "A hierarchical planner based on set-theoretic models: Towards automating the automation for autonomous systems," in *Int. Conf. on Informatics in Control, Automation and Robotics*, 2019.
- [23] B. Tekin, S. N. Sinha, and P. Fua, "Real-time seamless single shot 6d object pose prediction," *Conf. on Computer Vision and Pattern Recognition*, 2018.
- [24] W. Elmenreich, "An introduction to sensor fusion," *Research Report, Vienna University of Technology, Austria*, 2002.
- [25] W. Abdulla, "Mask r-cnn for object detection and instance segmentation on keras and tensorflow," <https://github.com/matterport/Mask-RCNN>, 2017.
- [26] MVTec HALCON, <https://www.mvtec.com/products/halcon/>.
- [27] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3D: A modern library for 3D data processing," *arXiv:1801.09847*, 2018.
- [28] T. Hodan, P. Haluza, Š. Obdržálek, J. Matas, M. Lourakis, and X. Zabulis, "T-LESS: An RGB-D dataset for 6d pose estimation of texture-less objects," in *Winter Conf. on Applications of Computer Vision*. IEEE, 2017.
- [29] P. S. Schmitt, W. Neubauer, W. Feiten, K. M. Wurm, G. v. Wichert, and W. Burgard, "Optimal, sampling-based manipulation planning," in *Int. Conf. on Robotics and Automation*. IEEE, 2017.
- [30] F. Wirschofer, P. S. Schmitt, W. Feiten, G. v. Wichert, and W. Burgard, "Robust, compliant assembly via optimal belief space planning," in *Int. Conf. on Robotics and Automation*. IEEE, 2018.

13

Data-Driven Synthesis of Perception Pipelines via Hierarchical Planning

Title	Data-Driven Synthesis of Perception Pipelines via Hierarchical Planning
Authors	Vincent Dietrich, Bernd Kast , Sebastian Albrecht, and Michael Beetz
ISBN/ISSN	978-3-030-48989-2
DOI	0.1007/978-3-030-48989-2_55
Status	published
Publisher	Springer
Contribution of Bernd Kast	I developed the underlying planning framework with the theoretical foundations for the hierarchization. Vincent Dietrich implemented the calibration scheme for the intermediate goals. We had close discussions about integrating this approach with the planner and aligning it with the theoretical foundations incorporated into this paper. The co-authors proofread and provided the scientific environment that enabled this work.
Summary	In this paper, we relaxed the structural limitations on the plan imposed by the plan templates used in the previous paper. We apply the generic hierarchical planning scheme to the domain of perception configuration planning. The challenge we solved in applying our approach in this domain was the difficulty of formulating hierarchical dependencies and derivation of intermediate goals. In this paper, we solved those problems with a data-driven calibration process that ensured that models were appropriately configured for the problem at hand.

Data-Driven Synthesis of Perception Pipelines via Hierarchical Planning

Vincent Dietrich¹[0000-0003-0568-9727], Bernd Kast¹[0000-0001-7838-3142],
Sebastian Albrecht¹[0000-0002-3647-4043], and
Michael Beetz²[0000-0002-7888-7444]

¹ Siemens Corporate Technology, Munich, Germany
`{first name.surname}@siemens.com`

² Institute for Artificial Intelligence, University Bremen, Germany
`beetz@cs.uni-bremen.de`

Abstract. Robotic systems in production environments have to adapt quickly to new situations and products to enable customization and short product cycles. This is especially true for the robot perception, which is sensitive to changes in environment and task. Therefore, we present an approach to quickly synthesize perception pipelines based on hierarchical planning. We calibrate the hierarchical model, such that it reflects condensed experience from historical data. We validate our approach in a simulated assembly scenario with objects from the Siemens Robot Learning Challenge, taking into account different possible sensor types and placements.

Keywords: Perception System Engineering · Robot Perception · Hierarchical Planning.

1 Introduction

Robotic systems, for instance in assembly applications, require reliable perception in order to take meaningful actions in partially unstructured environments. Unfortunately, changes in task or environment can degrade the perceptual performance, for instance when the product changes. This requires expensive and time consuming engineering effort to find a new system configuration that fulfills the task requirements of accuracy and speed. The engineering of perception systems is challenging and depends on task, environment, sensor, sensor placement and the available set of algorithms and their parameterization. A perception engineer uses a combination of knowledge and experience in order to steer a trial-and-error procedure to find a suitable configuration. For instance, as depicted in Fig. 1, different algorithms have distinct output characteristics that additionally depend on the quality of the input data. The engineer implicitly knows these characteristics and uses this knowledge to choose promising pipelines. Additionally, a human engineer can perform a hierarchical task decomposition and start with high level decisions such as the sensor choice, while roughly estimating the expected performance given the sensor's data quality.

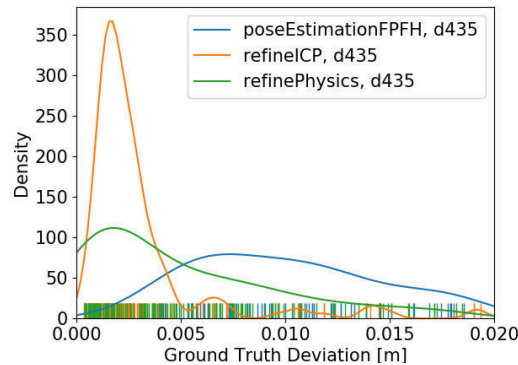


Fig. 1: Kernel density estimation of the positional output error with respect to ground truth of different perception and state estimation operators for a set of 200 perception pipelines. The operators have distinct error characteristics, which have to be taken into account in order to design a perception pipeline. Also, the inherent uncertainty and noise in the operator output is clearly visible.

Therefore, we present and investigate an approach to encode the engineering knowledge and experience in a hierarchical model, such that a hierarchical planner can perform a knowledge enabled search for suitable configurations.

The main contributions of this publication are:

- a hierarchical model that enables hierarchical planning for the synthesis of perception pipelines,
- a systematic handling of metrics that enables online planning, and
- an approach to encode engineering knowledge and experience in the hierarchical model via model calibration.

2 Related Work

In the computer vision community, the problem of automatic configuration or synthesis has a long standing history. For instance, Radig *et al.* [9] proposed a toolbox in 1992 for automated design of image understanding systems that uses the *FIGURE* system for knowledge based synthesis of pipelines by Messer [7]. The approach is based on logical rules and descriptions implemented in PROLOG, which are used to infer suitable pipelines. The more recent work of Nagato *et al.* [8] applies genetic programming and hierarchical program structuring in order to quickly adapt an image processing pipeline to a changing production environment. The approach was successfully demonstrated in a real production environment. Irgendfried *et al.* [4] address the automation of the design of entire inspection systems using accurate sensor simulation and uncertainty quantification. Another notable approach is the *RoboSherlock* framework by Beetz *et*

al. [1] which leverages unstructured information management and ontologies in order to generate perception pipelines based on semantic queries. Our own prior work includes automatic configuration of perception systems using single level planning and factor graphs for uncertainty representation [3] and the joint optimization of pipelines and parameters [2].

This work is grounded in robotics research and builds upon a hierarchical modeling and planning system that has successfully been demonstrated for task and motion planning [6]. This allows a tight integration between task planning and perception planning. Additionally, the target domain is not restricted to computer vision. General sensor fusion and state estimation algorithms, for instance using physics simulation, are targeted as well. Furthermore, a condensed model is trained that encodes experience and can be used for relatively fast planning of pipelines.

3 Approach

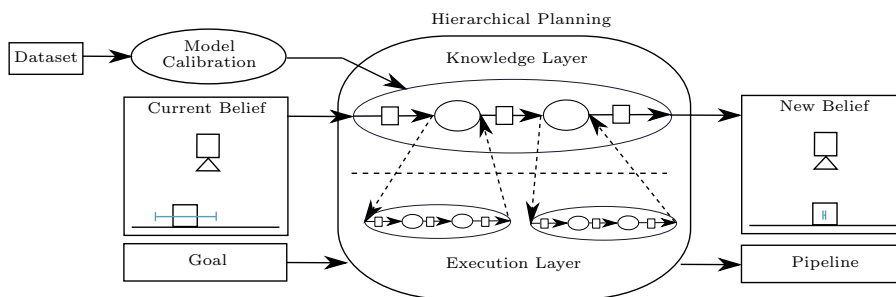


Fig. 2: Schematic overview of the approach. The core is a hierarchical planning system which acts on a 2-layer hierarchical model with a knowledge layer and an execution layer. The knowledge layer needs model calibration which is performed with a ground truth annotated dataset of pipeline executions. This planning system is able to determine a pipeline that converts a given belief to a belief with a target quality specified by a given goal.

Our approach is based on a hierarchical model combined with hierarchical planning and model calibration as depicted in Fig. 2. The hierarchical model is based on prior publications [5] [6] [2] and allows to model algorithms, denoted as *operators*, and instance classes, denoted as *concepts*, within the perception domain at different abstraction layers. This allows the hierarchical planner to search for solutions on cheap abstract domains and verify or refine the plans up to a real execution [6]. The model contains concepts such as pose, belief, point cloud and depth image. In this publication we employ only two different layers, the *knowledge layer* and the *execution layer*. The former encodes symbolic

knowledge and experience available for the domain. The latter provides the ability to actually execute the available perception algorithms on simulated or on real data.

The key aspect of this publication is the systematic use and calibration of so called *metrics* $m \in \mathbb{M}$. The metrics allow to assess properties, especially the quality of instances, in a compact manner. E.g., when a perception engineer looks at a point cloud he indirectly assesses its quality and determines by experience which algorithm he should try next in order to get the desired result. The operators on the knowledge layer can be parameterized such that they encode this experience and model the relationship of the input metrics and output metrics. We summarize the metric model as follows:

- A metric is single valued
- A metric can be associated with a distribution
- Multiple metrics can be associated with an instance
- We differentiate between *computable* and *estimated* metrics.

Computable metrics can be calculated at runtime based on the available data, as for instance the size of a point cloud. Estimated metrics can not be determined in an exact manner at runtime and need to be approximated. E.g., the positional distance with respect to ground truth is not known at pipeline runtime. Still, it has to be estimated if the planning goal is formulated as positional distance. The set of input and output metrics of an operator are denoted as \mathbb{M}_{IN} and \mathbb{M}_{OUT} respectively.

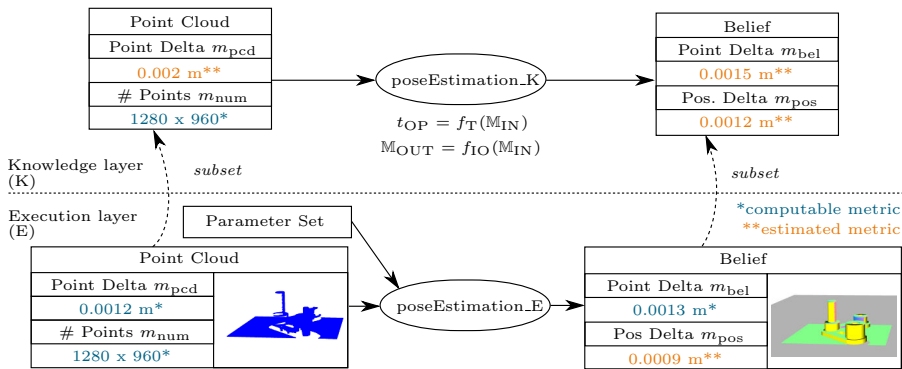


Fig. 3: Hierarchical view on a pose estimation operator on knowledge layer and execution layer. Detailed explanation in Sec. 3.

In order to work within the hierarchical modeling and planning approach, the metrics require a representation on different levels of abstraction. The role of the metrics within the hierarchy is therefore shown exemplary in Fig. 3. A pose estimation operator on the knowledge as well as the execution layer is

displayed alongside with input and output instances. Each instance is annotated with different metrics that describe the quality of the instance. On the knowledge layer these are mostly estimated metrics that approximate the actual value. On the execution layer actual values can be calculated for some of the metrics. Due to uncertainties in data and execution the operators on the execution layer not always yield results as predicted on the knowledge layer. A *subset* relationship is defined for all concepts, which is used during planning in order to identify such deviations and perform backtracking if the relationship does not hold. Therefore, model inaccuracies at the knowledge layer can be handled.

The approximations on the knowledge layer and for estimated metrics on the execution layer is encoded in the functions f_T and f_{IO} that model the runtime and the relationship between inputs and outputs of the operator, see Fig. 3. Both can be arbitrary functions that require parameterization. This step is performed initially based on a dataset of already executed perception pipelines, cf. Fig. 2. In this work we use linear models for the transfer functions and linear regression for the calibration.

For the hierarchical planning we use the planner presented in [6]. The framework is designed such that different planning or search algorithms can be used at different layers. We use a scheduler at the knowledge level in order to find the fastest pipeline that fulfills the application requirements. On the execution layer breadth first search is used. On the knowledge layer no distinction between different operator parameterizations is made. The parameterization is chosen based on a sub-problem planned within the execution layer. Additionally, only metrics are used during planning that can be calculated or estimated during runtime. Therefore, the presented system can be used out of the box for online decision making and active perception.

4 Evaluation

We address two core questions regarding the presented approach:

- Is the model and the calibration approach suitable for the given application?
- Is the planning system capable of synthesizing working pipelines in a reasonable amount of time?

The target application is an assembly scenario, where the robot perception system should give accurate 6D poses of the objects *base_plate* and *shaft_1* from the Siemens Robot Learning Challenge³, see Fig. 4. The setup is simulated and two different depth cameras are used, an Intel D435 consumer sensor equivalent and a Photoneo PhoXi M industrial sensor equivalent. Distance dependent longitudinal noise is added based on camera characteristics according to manufacturer data⁴. Additionally, the sensors are placed at six different poses in order to cover a range of different hardware setups, see Fig. 4.

³ <https://new.siemens.com/us/en/company/fairs-events/robot-learning.html>

⁴ https://www.intel.com/content/dam/support/us/en/documents/emerging-technologies/intel-realsense-technology/BKMs_Tuning_RealSense_D4xx-Cam.pdf and http://wiki.photoneo.com/index.php/PhoXi_3D_scanners_family

6 V. Dietrich et al.

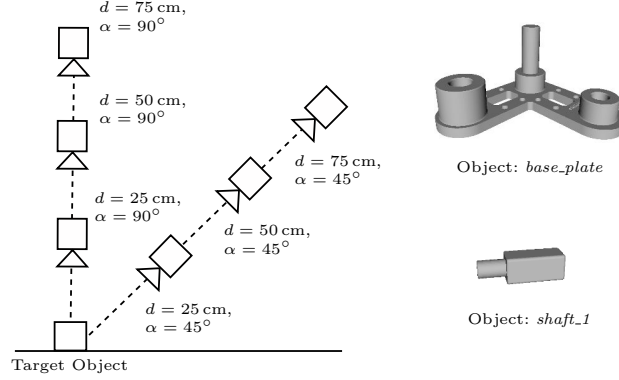


Fig. 4: Left: Application setup with 6 different view poses. Right: Target objects from the Siemens Robot Learning Challenge.

We use the following computable metrics on the execution layer:

- m_{num} : Number of points of the point cloud
- m_{pcd} : average distance between points that can be associated with known objects in the scene, such as a plane, and those objects
- m_{bel} : average distance between a depth rendering of the object estimate and the point cloud

The position distance m_{pos} between belief and ground truth can not be computed at runtime and has to be estimated. The mapping between m_{bel} and m_{pos} is therefore as well approximated with a linear model during the model calibration. The available operator set within the experiment is summarized in Tab. 1.

Table 1: Overview of used operators and their model calibration.⁵

Operator	Comment	# Param.	Sets	f_T	f_{IO}
getPointCloud*	point cloud rendering	0	-	0.030 s	$d \rightarrow m_{\text{pcd}}$
prepareModel	create object point cloud	1	1	0.050 s	-
removePlane	plane estimation and removal	3	1	0.013 s	$m_{\text{num}} \rightarrow m_{\text{num}}$
smoothTSDF	using Signed-Distance Functions	2	1	0.288 s	$\begin{pmatrix} m_{\text{num}} \\ m_{\text{pcd}} \end{pmatrix} \rightarrow \begin{pmatrix} m_{\text{num}} \\ m_{\text{pcd}} \end{pmatrix}$
poseEstimationFPFH	using Fast Point Feature Histograms	8	2	$m_{\text{num}} \rightarrow t$	$m_{\text{pcd}} \rightarrow m_{\text{bel}}$
refineICP	Iterative Closest Point	2	2	$m_{\text{num}} \rightarrow t$	$m_{\text{bel}} \rightarrow m_{\text{bel}}$
refinePhysics*	achieve physical plausibility with plane	6	1	2.060 s	$m_{\text{bel}} \rightarrow m_{\text{bel}}$

The model calibration is performed on a dataset of an exhaustive exploration of pipelines on 2 scenes, 2 cameras and 6 viewpoints of the object *base_plate*. The result of the calibration are parameterized models of the runtime behavior f_T

⁵ Operators marked with * are implemented using Bullet (<https://pybullet.org/>), otherwise using Open3D (<http://www.open3d.org/>)

and the input to output relation f_{IO} . In this work we use single-input, single-output linear models, but the approach supports models of arbitrary complexity and arbitrary number of inputs and outputs. The choice of input and output types for the models is done by manual examination of the correlation. The relationship between given and ground truth metrics is calibrated in addition to the operator behavior. In Tab. 1 the calibrated input-output relationships for the different operators are listed. Approximately constant values are directly shown. In the first row of Fig. 5 actual linear approximations for runtime and input-output relationship are displayed.

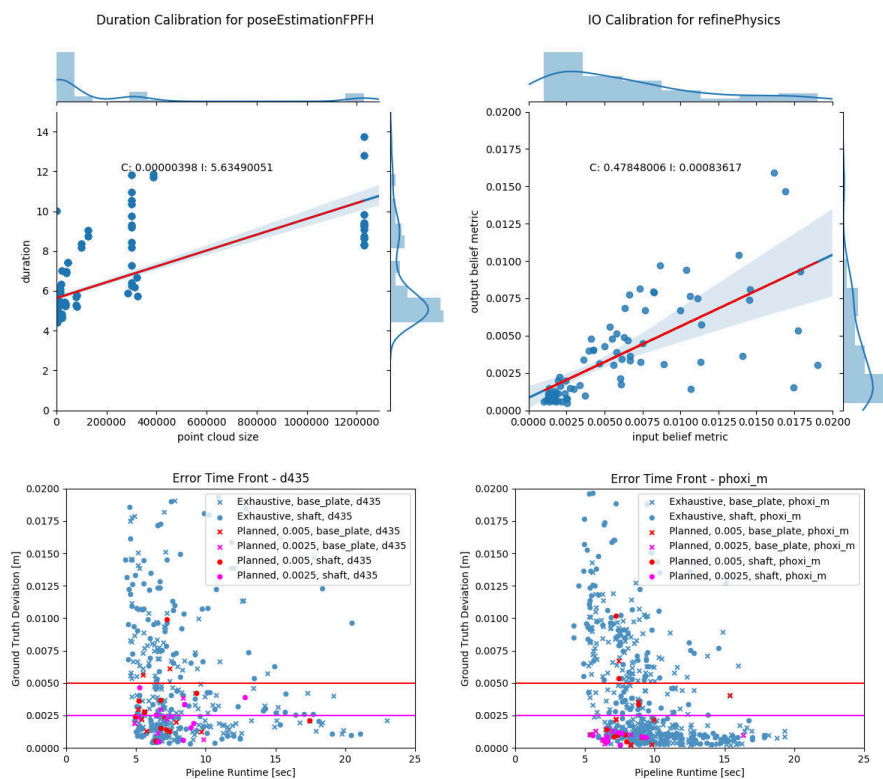


Fig. 5: Top left: Linear model fit for runtime depending on input cloud size for poseEstimationFPFH. Top right: Linear model fit for the output m_{bel} depending on the input m_{bel} . Bottom left: Comparison between baseline and planned pipelines for sensor D435. Bottom right: Comparison between baseline and planned pipelines for sensor Phoxi M.

In order to assess the approach we compare the planning results with the results of a greedy search, as shown in the bottom row of Fig. 5. The calibration has been performed only on *base_plate* and results are displayed for both ob-

jects. Furthermore, the planning was performed with two different goal settings: 0.005 m and 0.0025 m as indicated by the colors and the horizontal lines.

We can make a few observations. First of all, longer runtimes lead to lower errors, which is coherent with engineering intuition. Although this is not a general rule. For instance, the removal of the plane can lead to higher accuracy and shorter runtime due to a lower number of points for the pose estimation. Furthermore, we can observe that the ground truth of the majority of the planned results fulfills the goal requirements. During planning only runtime metrics are used, which can only approximate the ground truth. Therefore, the result may sometimes deviate from the ground truth due to uncertainty and insufficient calibration. The goal is reached with high success rates for the calibration object as well as the unseen object, which indicates that the model generalizes and is sufficiently complex. Most of the outliers are associated with the unseen and non calibrated *shaft_1* object, which conforms with our expectations. A further observation regards the runtime of the planned pipelines. On the knowledge level a scheduler is used in order to find the fastest pipelines that reach the goal. The experimental data shows that the planning system is actually capable of identifying a suitable compromise between speed and accuracy. This is achieved with a median planning time of about 51 s. Finally, we compare the results between the consumer camera and the industrial camera. The more expensive industrial camera can achieve higher accuracy overall. But while there are no outliers for 2.5 mm the model still leaves room for improvement as the outliers for a goal of 5 mm imply.

In summary, both questions formulated at the beginning of this section can be generally answered positively. However, there is still potential within further investigation to improve the model accuracy and decrease the synthesis times for actual use for active perception.

5 Conclusion

We presented an approach to synthesize perception pipelines using hierarchical planning, which allows to find a sequence of operators with sufficient accuracy for the task at hand. The hierarchical model consists of a knowledge layer, which encodes engineering experience from data and steers the search procedure, and an execution layer where actual operators are executed on the given data. Scheduling on the top level ensures that pipelines with short runtime are preferred. Experiments in simulation with different objects, sensor types and sensor placements show promising results. The knowledge layer is calibrated for one object and performs well for a unseen test object. The duration of the synthesis process is mostly less than a minute and therefore allows rapid adaptation, e.g., for a product change. Further speedup would also enable running the synthesis online during operation.

References

1. Beetz, M., Bálint-Benczédi, F., Blodow, N., Nyga, D., Wiedemeyer, T., Márton, Z.C.: RoboSherlock: Unstructured information processing for robot perception. In: Int. Conf. on Robotics and Automation. IEEE (2015)
 2. Dietrich, V., Kast, B., Fiegert, M., Albrecht, S., Beetz, M.: Automatic configuration of the structure and parameterization of perception pipelines. In: Int. Conf. on Advanced Robotics. IEEE (2019)
 3. Dietrich, V., Kast, B., Schmitt, P., Albrecht, S., Fiegert, M., Feiten, W., Beetz, M.: Configuration of perception systems via planning over factor graphs. In: Int. Conf. on Robotics and Automation. IEEE (2018)
 4. Irgenfried, S., Worn, H., Bergmann, S., Mohammadikaji, M., Beyerer, J., Dachsbacher, C.: Cad based workflow for semi-automatic design of optical inspection systems. *AT-AUTOMATISIERUNGSTECHNIK* **65**(6) (2017)
 5. Kast, B., Albrecht, S., Feiten, W., Zhang, J.: Bridging the gap between semantics and control for industry 4.0 and autonomous production. In: Int. Conf. on Automation, Science and Engineering. IEEE (2019)
 6. Kast, B., Dietrich, V., Albrecht, S., Feiten, W., Zhang, J.: A hierarchical planner based on set-theoretic models: Towards automating the automation for autonomous systems. In: Int. Conf. on Informatics in Control, Automation and Robotics (2019)
 7. Messer, T.: Model-based synthesis of vision routines. In: *Advances In Machine Vision: Strategies and Applications*, pp. 79–97. World Scientific (1992)
 8. Nagato, T., Koezuka, T.: Automatic generation of image-processing programs for production lines. *Fujitsu Sci. Tech. J* **52**(1), 27–33 (2016)
 9. Radig, B., Eckstein, W., Klotz, K., Messer, T., Pauli, J.: Automatization in the design of image understanding systems. In: Int. Conf. on Industrial, Engineering and Other Applications of Applied Intelligent Systems. Springer (1992)
-

Bibliography

- [1] V. Dietrich, B. Kast, P. Schmitt, S. Albrecht, M. Fiegert, W. Feiten, and M. Beetz, "Configuration of perception systems via planning over factor graphs," in *IEEE International Conference on Robotics and Automation*, 2018.
 - [2] B. Kast, V. Dietrich, S. Albrecht, W. Feiten, and J. Zhang, "A hierarchical planner based on set-theoretic models: Towards automating the automation for autonomous systems," in *International Conference on Informatics in Control, Automation and Robotics*. SCITEPRESS Digital Library, 2019.
 - [3] Siemens AG, fortiss GmbH, Baselabs GmbH, NXP Semiconductors, DFKI, and ALL4IP Technologies GmbH und Co KG, *Project Sada*, 2018 (accessed September 3, 2018), <http://www.projekt-sada.de/>.
 - [4] V. Dietrich, B. Kast, S. Albrecht, and M. Beetz, "Data-driven synthesis of perception pipelines via hierarchical planning," in *International Conference on Robotics in Alpe-Adria Danube Region*. Springer, 2020.
 - [5] B. Kast, V. Dietrich, S. Albrecht, W. Feiten, and J. Zhang, "Domain optimization for hierarchical planning based on set-theory," in *International Conference on Informatics in Control, Automation and Robotics*. SCITEPRESS Digital Library, 2020.
 - [6] A. Car and A. Frank, "General principles of hierarchical spatial reasoning-the case of wayfinding," in *6th International Symposium on Spatial Data Handling*, 1994.
 - [7] B. Kast, P. Schmitt, S. Albrecht, D. Meyer-Delius, and J. Zhang, "Augmenting working areas with action-relevant information for intuitive human-robot cooperation," in *International Convention on Rehabilitation Engineering and Assistive Technology*, 2018.
 - [8] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "Pddl-the planning domain definition language," Yale Center for Computational Vision and Control, Tech. Rep., 1998.
 - [9] M. Fox and D. Long, "Pddl2.1: An extension to PDDL for expressing temporal planning domains," *Artificial Intelligence Research*, vol. 20, 2003.
 - [10] M. Cashmore, M. Fox, D. Long, and D. Magazzeni, "A compilation of the full PDDL+ language into SMT." in *AAAI Workshop: Planning for Hybrid Systems*, 2016.
-

- [11] D. Höller, G. Behnke, P. Bercher, S. Biundo, H. Fiorino, D. Pellier, and R. Alford, “Hddl: An extension to PDDL for expressing hierarchical planning problems,” in *AAAI Conference on Artificial Intelligence*, vol. 34, no. 06, 2020.
 - [12] J. Hoffmann, “Ff: The fast-forward planning system,” *AI magazine*, vol. 22, no. 3, 2001.
 - [13] M. Helmert, “The fast downward planning system,” *Artificial Intelligence Research*, 2006.
 - [14] S. Richter and M. Westphal, “The lama planner: Guiding cost-based anytime planning with landmarks,” *Journal of Artificial Intelligence Research*, 2010.
 - [15] P. Kerschke, H. H. Hoos, F. Neumann, and H. Trautmann, “Automated algorithm selection: Survey and perspectives,” *Evolutionary computation*, vol. 27, no. 1, 2019.
 - [16] B. Kast, S. Albrecht, W. Feiten, and J. Zhang, “Bridging the gap between semantics and control for industry 4.0 and autonomous production,” in *International Conference on Automation Science and Engineering*, 2019.
 - [17] B. Ganter and R. Wille, *Formal concept analysis: mathematical foundations*. Springer Science & Business Media, 2012.
 - [18] D. L. McGuinness and F. Van Harmelen, “Owl web ontology language overview,” *W3C recommendation*, vol. 10, no. 10, 2004.
 - [19] B. De Schutter and T. Van Den Boom, “Model predictive control for discrete-event and hybrid systems,” in *IEEE Conference on Decision and Control*, 2003.
 - [20] M. Dogar, A. Spielberg, S. Baker, and D. Rus, “Multi-robot grasp planning for sequential assembly operations,” *Autonomous Robots*, vol. 43, no. 3, 2019.
 - [21] R. Moriyama, W. Wan, and K. Harada, “Dual-arm assembly planning considering gravitational constraints,” in *IEEE International Conference on Intelligent Robots and Systems*, 2019.
 - [22] P. Bercher, D. Höller, G. Behnke, and S. Biundo, “More than a name? On implications of preconditions and effects of compound HTN planning tasks.” in *European Conference on AI*, 2016.
 - [23] S. Edelkamp, “Limits and possibilities of PDDL for model checking software,” *Edelkamp, & Hoffmann*, 2003.
 - [24] F. Fourati, M. T. Bhiri, and R. Robbana, “Verification and validation of PDDL descriptions using event-b formal method,” in *International Conference on Multimedia Computing and Systems*. IEEE, 2016.
 - [25] R. Barták, A. Maillard, and R. C. Cardoso, “Validation of hierarchical plans via parsing of attribute grammars,” in *International Conference on Automated Planning and Scheduling*, 2018.
-

-
- [26] C. Garrett, T. Lozano-Pérez, and L. Kaelbling, “FFrob: An efficient heuristic for task and motion planning,” in *Algorithmic Foundations of Robotics XI*. Springer, 2015.
- [27] D. Beßler, M. Pomarlan, A. Akbari, M. Diab, J. Rosell, J. Bateman, and M. Beetz, “Assembly planning in cluttered environments through heterogeneous reasoning,” in *Joint German/Austrian Conference on Artificial Intelligence*, 2018.
- [28] K. Erol, J. Hendler, and D. Nau, “HTN planning: Complexity and expressivity,” in *AAAI Conference on Artificial Intelligence*, 1994.
- [29] L. Castillo, J. Fernández-Olivares, O. Garcia-Perez, and F. Palao, “Efficiently handling temporal knowledge in an HTN planner.” in *International Conference on Automated Planning and Scheduling*, 2006.
- [30] D. Nau, T.-C. Au, O. Ilghami, U. Kuter, J. Murdock, D. Wu, and F. Yaman, “Shop2: An HTN planning system,” *Artificial Intelligence Research*, 2003.
- [31] R. Goldman, “Durative planning in HTNs.” in *International Conference on Automated Planning and Scheduling*, 2006.
- [32] S. Kambhampati, A. Mali, and B. Srivastava, “Hybrid planning for partially hierarchical domains,” in *AAAI Conference on Artificial Intelligence*, 1998.
- [33] E. Plaku and G. D. Hager, “Sampling-based motion and symbolic action planning with geometric and differential constraints,” in *International Conference on Robotics and Automation*, 2010.
- [34] L. P. Kaelbling and T. Lozano-Pérez, “Hierarchical task and motion planning in the now,” in *IEEE International Conference on Robotics and Automation*, 2011.
- [35] E. Scioni, G. Borghesan, H. Bruyninckx, and M. Bonfè, “Bridging the gap between discrete symbolic planning and optimization-based robot control,” in *IEEE International Conference on Robotics and Automation*, 2015.
- [36] B. Kim, L. P. Kaelbling, and T. Lozano-Pérez, “Learning to guide task and motion planning using score-space representation,” in *IEEE International Conference on Robotics and Automation*, 2017.
- [37] S. Stock, M. Mansouri, F. Pecora, and J. Hertzberg, “Online task merging with a hierarchical hybrid task planner for mobile service robots,” in *IEEE International Conference on Intelligent Robots and Systems*, 2015.
- [38] C. R. Garrett, C. Paxton, T. Lozano-Pérez, L. P. Kaelbling, and D. Fox, “Online replanning in belief space for partially observable task and motion problems,” in *IEEE International Conference on Robotics and Automation*, 2020.
- [39] S. Awodey, *Category theory*. Oxford university press, 2010.
-

- [40] D. Hsu, J.-C. Latombe, and R. Motwani, "Path planning in expansive configuration spaces," in *International Conference on Robotics and Automation*, 1997.
- [41] V. Dietrich, B. Kast, M. Fiegert, S. Albrecht, and M. Beetz, "Automatic configuration of the structure and parameterization of perception pipelines," in *International Conference on Advanced Robotics*, 2019.
- [42] S. Wintermute, "Imagery in cognitive architecture: Representation and control at multiple levels of abstraction," *Cognitive Systems Research*, vol. 19, 2012.
- [43] J. S. Penberthy and D. S. Weld, "UCPOP: A sound, complete, partial order planner for adl." *International Conference on Principles of Knowledge Representation and Reasoning*, 1992.
- [44] A. Barrett and D. S. Weld, "Partial-order planning: evaluating possible efficiency gains," *Artificial Intelligence*, vol. 67, no. 1, 1994.
- [45] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "Sampling-based methods for factored task and motion planning," *International Journal of Robotics Research*, vol. 37, no. 13-14, 2018.
- [46] Z. Zhao, Z. Zhou, M. Park, and Y. Zhao, "Sydebo: Symbolic-decision-embedded bilevel optimization for long-horizon manipulation in dynamic environments," *IEEE Access*, vol. 9, 2021.
- [47] Y. Zhu, J. Tremblay, S. Birchfield, and Y. Zhu, "Hierarchical planning for long-horizon manipulation with geometric and symbolic scene graphs," in *IEEE International Conference on Robotics and Automation*, 2021.
- [48] B. Nurimbetov, M. Issa, and H. A. Varol, "Robotic assembly planning of tensegrity structures," in *IEEE/SICE International Symposium on System Integration*, 2019.
- [49] R. Glück, A. Hoffmann, L. Nägele, A. Schierl, W. Reif, and H. Voggenreiter, "Towards a tool-based methodology for developing software for dynamic robot teams," 2018.
- [50] J. Leuvenink, K. Kruger, and A. Basson, "Architectures for human worker integration in holonic manufacturing systems," in *International Workshop on Service Orientation in Holonic and Multi-Agent Manufacturing*. Springer, 2018.
- [51] N. Nikolakis, K. Sipsas, and S. Makris, "Cockpit: a portal for symbiotic human-robot collaborative assembly," in *Advanced Human-Robot Collaboration in Manufacturing*. Springer, 2021.
- [52] R. Müller, R. Peifer, and O. Mailahn, "Objectification of assembly planning for the implementation of human-robot cooperation," in *International Conference on Applied Human Factors and Ergonomics*. Springer, 2018.
-

-
- [53] A. A. Malik and A. Bilberg, "Complexity-based task allocation in human-robot collaborative assembly," *Industrial Robot-The International Journal of Robotics Research and Application*, 2019.
- [54] M. Rizwan, V. Patoglu, and E. Erdem, "Human-robot collaborative assembly planning using hybrid conditional planning," in *FAIM/ISCA Workshop on Artificial Intelligence for Multimodal HRI*, 2018.
- [55] Z. Kingston, M. Moll, and L. E. Kavraki, "Sampling-based methods for motion planning with constraints," *Annual Review of Control, Robotics, and Autonomous Systems*, 2018.
- [56] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Robotics Research*, vol. 30, no. 7, 2011.
- [57] J. Kuffner and S. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *International Conference on Robotics and Automation*, 2000.
- [58] E. Aertbeliën and J. De Schutter, "eTaSL/eTC: A constraint-based task specification language and robot controller using expression graphs," in *International Conference on Intelligent Robots and Systems*, 2014.
- [59] P. S. Schmitt, F. Wirschofer, K. M. Wurm, G. v. Wichert, and W. Burgard, "Modeling and planning manipulation in dynamic environments," in *International Conference on Robotics and Automation*, 2019.
- [60] T. Hasegawa, T. Suehiro, and K. Takase, "A model-based manipulation system with skill-based execution," *Transactions on Robotics and Automation*, vol. 8, no. 5, 1992.
- [61] A. Wahrburg, S. Zeiss, B. Matthias, J. Peters, and H. Ding, "Combined pose-wrench and state machine representation for modeling robotic assembly skills," in *IEEE International Conference on Intelligent Robots and Systems*, 2015.
- [62] S. Brunner, F. Steinmetz, R. Belder, and A. Dömel, "Rafcon: A graphical tool for engineering complex, robotic tasks," in *IEEE International Conference on Intelligent Robots and Systems*, 2016.
- [63] C. Schlegel and R. Worz, "The software framework smartsoft for implementing sensorimotor systems," in *IEEE International Conference on Intelligent Robots and Systems. Human and Environment Friendly Robots with High Intelligence and Emotional Quotients*, 1999.
- [64] A. Lotz, A. Hamann, I. Lütkebohle, D. Stampfer, M. Lutz, and C. Schlegel, "Modeling non-functional application domain constraints for component-based robotics software systems," *arXiv preprint arXiv:1601.02379*, 2016.
-

- [65] A. Lotz, A. Hamann, R. Lange, C. Heinzemann, J. Staschulat, V. Kesel, D. Stampfer, M. Lutz, and C. Schlegel, "Combining robotics component-based model-driven development with a model-based performance analysis," in *IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, 2016.
- [66] H. Sirkin, M. Zinser, and J. Rose, "How robots will redefine competitiveness," *Boston Consulting Group*, <https://www.bcg.com/publications/2015/lean-manufacturing-innovation-robots-redefine-competitiveness.aspx>, 2015.
- [67] S. Landscheidt and M. Kans, "Method for assessing the total cost of ownership of industrial robots," *Procedia Cirp*, vol. 57, 2016.
- [68] F. Sadeghi, A. Toshev, E. Jang, and S. Levine, "Sim2real viewpoint invariant visual servoing by recurrent control," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.
- [69] Q. Bateux, E. Marchand, J. Leitner, F. Chaumette, and P. Corke, "Training deep neural networks for visual servoing," in *IEEE international conference on robotics and automation*, 2018.
- [70] H. Wang, B. Yang, J. Wang, X. Liang, W. Chen, and Y.-H. Liu, "Adaptive visual servoing of contour features," *IEEE/ASME Transactions on Mechatronics*, vol. 23, no. 2, 2018.
- [71] E. Muñoz, Y. Konishi, V. Murino, and A. Del Bue, "Fast 6D pose estimation for texture-less objects from a single RGB image," in *IEEE International Conference on Robotics and Automation*, 2016.
- [72] C. Song, J. Song, and Q. Huang, "Hybridpose: 6d object pose estimation under hybrid representations," in *IEEE/CVF conference on computer vision and pattern recognition*, 2020.
- [73] H. Zhang and Q. Cao, "Detect in RGB, optimize in edge: accurate 6D pose estimation for texture-less industrial parts," in *International Conference on Robotics and Automation*, 2019.
- [74] F. Wirschofer, P. Schmitt, W. Feiten, G. v. Wichert, and W. Burgard, "Robust, compliant assembly via optimal belief space planning," in *International Conference on Robotics and Automation*, 2018.
- [75] F. Wirschofer, P. S. Schmitt, G. von Wichert, and W. Burgard, "Controlling contact-rich manipulation under partial observability," in *Robotics: Science and Systems*, 2020.
- [76] P. S. Schmitt, F. Wirschofer, K. M. Wurm, G. v. Wichert, and W. Burgard, "Modeling and planning manipulation in dynamic environments," in *International Conference on Robotics and Automation*. IEEE, 2019.
- [77] T. Migimatsu and J. Bohg, "Object-centric task and motion planning in dynamic environments," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, 2020.
-

-
- [78] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "Strips planning in infinite domains," *arXiv preprint arXiv:1701.00287*, 2017.
- [79] S.-H. Leitner and W. Mahnke, "OPC UA-service-oriented architecture for industrial applications," *ABB Corporate Research Center*, vol. 48, no. 61-66, 2006.
- [80] M. Hause, "The SysML modelling language," in *European Systems Engineering Conference*, 2006.
- [81] R. Drath, A. Luder, J. Peschke, and L. Hundt, "AutomationML-the glue for seamless automation engineering," in *IEEE International Conference on Emerging Technologies and Factory Automation*, 2008.
- [82] D. Stampfer, A. Lotz, M. Lutz, and C. Schlegel, "The SmartMDSO toolchain: An integrated mdsd workflow and integrated development environment (ide) for robotics software," *Journal of Software Engineering for Robotics*, vol. 7, no. 1, 2016.
- [83] C. Schlegel, A. Lotz, M. Lutz, and D. Stampfer, "Composition, separation of roles and model-driven approaches as enabler of a robotics software ecosystem," in *Software Engineering for Robotics*. Springer, 2021.
- [84] I. Beltagy, K. Lo, and A. Cohan, "Scibert: A pretrained language model for scientific text," *arXiv preprint arXiv:1903.10676*, 2019.
- [85] M. Galanis, V. Dietrich, B. Kast, and M. Fiegert, "Rtfd: Towards understanding source code using natural language processing," in *International Conference on Informatics in Control, Automation and Robotics*. SCITEPRESS Digital Library, 2020.
- [86] C. Wang, L. Wang, J. Qin, Z. Wu, L. Duan, Z. Li, M. Cao, X. Ou, X. Su, and W. Li, "Path planning of automated guided vehicles based on improved a-star algorithm," in *IEEE International Conference on Information and Automation*, 2015.
- [87] G. Qing, Z. Zheng, and X. Yue, "Path-planning of automated guided vehicle based on improved dijkstra algorithm," in *Chinese control and decision conference*, 2017.
- [88] T. Zheng, Y. Xu, and D. Zheng, "Agv path planning based on improved a-star algorithm," in *IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference*, 2019.
- [89] B. Kim, L. Shimanuki, L. P. Kaelbling, and T. Lozano-Pérez, "Representation, learning, and planning algorithms for geometric task and motion planning," *International Journal of Robotics Research*, 2021.
- [90] M. Chakraborty, S. K. Biswas, and B. Purkayastha, "Recursive rule extraction from NN using reverse engineering technique," *New Generation Computing*, vol. 36, no. 2, 2018.
-

- [91] J. R. Zilke, E. L. Mencía, and F. Janssen, “Deepred–rule extraction from deep neural networks,” in *International Conference on Discovery Science*. Springer, 2016.
 - [92] R. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec, “GNNExplainer: Generating explanations for graph neural networks,” *Advances in neural information processing systems*, vol. 32, 2019.
 - [93] Z. Wang, C. R. Garrett, L. P. Kaelbling, and T. Lozano-Pérez, “Learning compositional models of robot skills for task and motion planning,” *International Journal of Robotics Research*, vol. 40, no. 6-7, 2021.
 - [94] G. Konidaris, L. P. Kaelbling, and T. Lozano-Perez, “From skills to symbols: Learning symbolic representations for abstract high-level planning,” *Journal of Artificial Intelligence Research*, vol. 61, 2018.
 - [95] M. Guo and M. Bürger, “Geometric task networks: Learning efficient and explainable skill coordination for object manipulation,” *IEEE Transactions on Robotics*, 2021.
 - [96] M. Sharma, J. Liang, J. Zhao, A. LaGrassa, and O. Kroemer, “Learning to compose hierarchical object-centric controllers for robotic manipulation,” *arXiv preprint arXiv:2011.04627*, 2020.
 - [97] M. Saha and P. Ito, “Multi-robot motion planning by incremental coordination,” in *IEEE International Conference on Intelligent Robots and Systems*, 2006.
 - [98] G. Wagner and H. Choset, “M*: A complete multirobot path planning algorithm with performance bounds,” in *IEEE international conference on intelligent robots and systems*, 2011.
 - [99] J. Liu and R. K. Williams, “Submodular optimization for coupled task allocation and intermittent deployment problems,” *IEEE Robotics and Automation Letters*, vol. 4, no. 4, 2019.
 - [100] S. D. Han and J. Yu, “Ddm: Fast near-optimal multi-robot path planning using diversified-path and optimal sub-problem solution database heuristics,” *IEEE Robotics and Automation Letters*, vol. 5, no. 2, 2020.
 - [101] M. Toussaint, “Logic-geometric programming: An optimization-based approach to combined task and motion planning.” in *International Joint Conference on Artificial Intelligence*, 2015.
 - [102] A. Schierl, A. Hoffmann, L. Nagele, and W. Reif, “Integrating reactive behavior and planning: Optimizing execution time through predictive preparation of state machine tasks,” in *IEEE International Conference on Robotic Computing*, 2018.
 - [103] L. Nägele, A. Hoffmann, A. Schierl, and W. Reif, “Legobot: Automated planning for coordinated multi-robot assembly of lego structures,” in *IEEE International Conference on Intelligent Robots and Systems*, 2020.
-

-
- [104] R. Smits, T. De Laet, K. Claes, H. Bruyninckx, and J. De Schutter, “iTASC: a tool for multi-sensor integration in robot manipulation,” in *IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*, 2008.
- [105] W. Decré, H. Bruyninckx, and J. De Schutter, “Extending the iTaSC constraint-based robot task specification framework to time-independent trajectories and user-configurable task horizons,” in *International Conference on Robotics and Automation*, 2013.
-

Eidesstattliche Versicherung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Dissertationsschrift selbst verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

München, den _____ Unterschrift: _____