Modular Models for Multi-Phase Robotic Manipulation Tasks

Dissertation
zur Erlangung des akademischen Grades
Dr. rer. nat.
an der Fakultät
für Mathematik, Informatik und Naturwissenschaften
der Universität Hamburg

Eingereicht beim Fachbereich Informatik
von Michael Görner

April 2025

Erstgutachter:
Prof. Dr. Jianwei Zhang

Zweitgutachter:
Prof. Dr. Michael Beetz

Vorsitz der Prüfungskommission:
Prof. Dr. Janick Edinger

Stellvertretender Vorsitz der Prüfungskommission:
Prof. Dr. Stefan Wermter

Tag der Disputation:
6.10.2025

Document Identifier:
urn:nbn:de:gbv:18-ediss-133898

Abstract

The research literature on trajectory generation for robotics comprises a plethora of methods in different scenarios — each with their respective advantages and disadvantages in terms of algorithmic complexity, computational cost, dynamic constraints, adaptivity, and predictability. However, in actual robotic use, individual motions seldom stand on their own. Instead, they are part of larger manipulation sequences which are necessary to interact with the environment and complete a given task. Such prototypical sequences include object relocation, navigating a mobile robot to a different location (potentially opening and passing through doors along the way), constraint manipulation actions, such as opening hinged doors or drawers, or achieving other multimodal effects of a wide variety through physical interaction, such as pouring liquids from containers, torch welding, or pressing buttons.

This thesis contributes a framework to formulate and solve such manipulation sequences through segment factorization by relying on non-invasive information forwarding between dedicated modules. The framework, named *Task Construction*, features (1) explicitly modularized subproblems, isolating different motion phases during manipulation tasks and enabling specialized solvers and expert knowledge to affect them individually, (2) accounting for inter-dependencies that arise between different motions, e.g., grasping a tool and subsequent tool-use, through structured information propagation, and (3) comprehensive introspection to identify problems in separate motion phases — supporting users in setting up manipulation tasks. After considering a series of example uses, the thesis further investigates how a robotic system can acquire specialized effect models for motion phases. To this end, it investigates the example of haptic interaction with chordophones, where motions to produce individual musical sounds can be isolated and parameterized in a small number of parameters. Self-supervised exploration in the physical world is demonstrated through explicit constraints on observed effects. The thesis presents a *Gaussian Process*-based active exploration strategy which emphasizes observable validity as a constraint. Robot experiments demonstrate the system's ability to characterize instrument interactions after sufficient self-exploration.

Zusammenfassung

Die Forschungsliteratur zur Trajektoriengenerierung in der Robotik umfasst eine Vielzahl von Methoden für unterschiedliche Szenarien — jede mit spezifischen Vor- und Nachteilen im Hinblick auf algorithmische Komplexität, Rechenaufwand, dynamische Randbedingungen, Adaptivität und Vorhersagbarkeit. In der praktischen Anwendung stehen einzelne Bewegungen jedoch selten allein. Stattdessen sind sie Teil längerer Manipulationssequenzen, die erforderlich sind, um mit der Umgebung zu interagieren und eine Aufgabe zu erfüllen. Solche typischen Abläufe beinhalten etwa das Umplatzieren von Objekten, das Navigieren eines mobilen Roboters an einen anderen Ort (gegebenenfalls inklusive Öffnen und Passieren von Türen), die Manipulation unter Nebenbedingungen wie etwa das Öffnen von Türen mit Scharnieren oder Schubladen, oder das Auslösen verschiedener multimodaler Effekte durch physische Interaktion — beispielsweise das Ausgießen von Flüssigkeiten, das robotische Schweißen, oder das Betätigen von Schaltern. Diese Dissertation trägt ein Framework namens *Task Construction* bei, das solche Manipulationssequenzen durch die Faktorisierung von Phasen formuliert und löst, indem es sich auf nichtinvasive Informationsweiterleitung zwischen dedizierten Modulen stützt. Das Framework zeichnet sich durch (1) explizit modularisierte Teilprobleme aus, die verschiedene Bewegungsphasen während Manipulationsaufgaben isolieren und spezialisierte Lösungsansätze sowie Expertenwissen gezielt in diesen Phasen zur Wirkung kommen lassen, (2) die Berücksichtigung von Abhängigkeiten, die zwischen unterschiedlichen Bewegungen entstehen — etwa beim Greifen eines Werkzeugs und dessen anschließender Verwendung — durch eine strukturierte Informationsweitergabe, und (3) umfassende Introspektion, um Probleme in den einzelnen Bewegungsphasen zu identifizieren und so Anwender bei der Einrichtung von Manipulationsaufgaben zu unterstützen. Nach der Betrachtung verschiedener Anwendungsbeispiele wird weiterhin untersucht, wie ein robotisches System spezialisierte Wirkmodelle für einzelne Bewegungsphasen erwerben kann. Hierfür dient das Beispiel der haptischen Interaktion mit Chordophonen, bei denen die Bewegungen zur Erzeugung einzelner musikalischer Töne isoliert ist und in wenige Parameter gefasst werden können. Selbstüberwachtes Explorieren in der physischen Welt wird unter Berücksichtigung expliziter Beschränkungen der beobachteten Effekte untersucht. Die Arbeit stellt eine auf *Gauß-Prozess Methodik* basierende Strategie zur aktiven Exploration vor, die die erwartete Validität von Explorationsversuchen als Nebenbedingung hervorhebt. In Roboterversuchen wird schließlich demonstriert, dass das System nach hinreichender Selbstexploration in der Lage ist, die Effekte von Interaktionen mit dem Instrument erfolgreich zu charakterisieren.

Acknowledgements

I wish to thank my supervisor for the rare opportunity to work with a diverse array of advanced robotic hardware and for involving me in important ongoing cross-cultural and cross-disciplinary dialog. Both are experiences I deeply treasure that enhanced my understanding of the field and its social implications. Thank you to my senior and longtime room neighbor Norman Hendrich who always presented me with challenges beyond my capabilities — accompanied by well-thought-out insights I appreciate. Thank you to the MoveIt maintainer team and specifically Robert Haschke with whom I share years of discussions to get things right. Thank you to the MoveIt user community for their interest and continuous stream of feedback. Thank you to my colleagues and work group. They provided endless streams of technical questions to resolve and trivia to research when distractions were urgently needed. Thank you Sarah. I am beyond grateful.

Contents

Chapter 1

Introduction

1.1 Motivation

Robots are built to move. But rather than moving in isolation, they are expected to manipulate their environment. Across the literature, many planning and trajectory generation methods can be found to compute trajectories that define motions of robotic systems between different target states, according to specific motion profiles, all the while possibly respecting additional constraints. Generated trajectories can eventually be executed through trajectory tracking controllers on physical robots. Research on trajectory planning for extended manipulation actions, in contrast though, is often limited to few scenarios and simplified in one of several ways due to the inherently increased complexity.

In *grasp prediction systems*, e.g., PointNetGPD [87], full grasping actions are simplified to a single grasp pose, leaving all other aspects of the involved motions, including reaching the grasp point and actually retrieving the grasped object to the engineer. Where specific grasp types are required for subsequent actions, predicted grasp poses are further classified into *affordance categories* to select compatible grasps.

Multi-Modal Planning (MMP) approaches, as introduced by Hauser et al. [54], build on path and motion planning algorithms to include manipulation motions, but require sufficient abstraction of individual motion phases to sample from and thus trade individual generation of phases for alignment. The execution of the resulting trajectories is challenging, and requires significant work to support the required abstractions on the robot meant to execute the trajectories.

Lastly, Task and Motion Planning (TAMP) approaches which explicitly target extended manipulation scenarios, such as for example presented by Srivastava et al. [135], rely on higher-level action abstractions, such as a modular directive for object pick-up to allow the combination of motions on the abstraction level. These approaches aim for ontology-guided solving of artificial puzzle-like scenarios and their plans tend to be less predictable in mundane tasks.

This thesis addresses the gap between MMP and TAMP approaches by proposing a modular and highly-customizable framework termed *Task Construction* for offline planning of feasible ma-

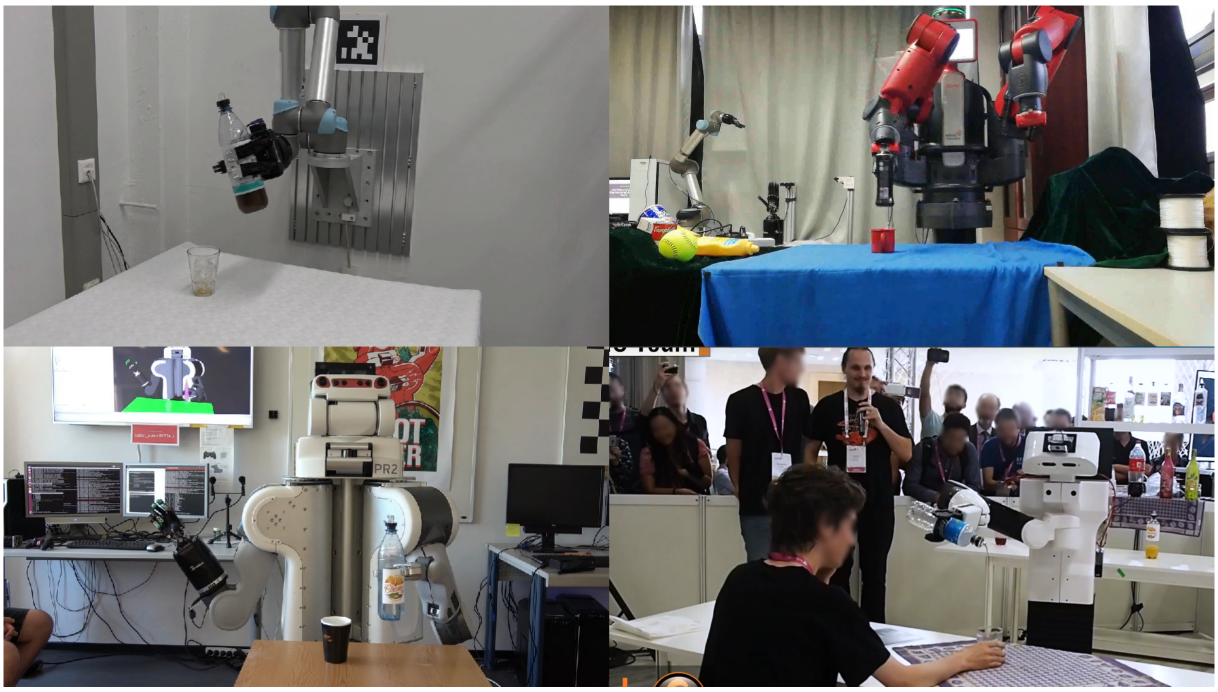


Figure 1.1 – Four different physical robots executing a manipulation plan which involves picking up a target bottle, pouring into a target container, and eventually placing the bottle on a table again. All four systems utilize the same designed manipulation sequence with adapted parameterization.

nipulation trajectories with unspecified action parameters. It allows for the explicit separation of different aspects of manipulation planning while reusing various methods within modules. By explicitly limiting the framework to pre-specified manipulation sequences, instead of covering all sequences of applicable actions, the structure of generated solution trajectories remains predictable and can be tailored to the specific requirements of individual tasks at hand.

Manipulation designs are robot-agnostic and designs can be reused with adjusted parameterization on different robots. Generated manipulation plans can be executed successfully on physical hardware. Figure 1.1 demonstrates the physical execution of manipulation plans generated from a designed task structure for multiple independent robotic systems with only parameter adjustment. The complete structure of this task is discussed in detail in subsection 3.9.2 and involves generic object manipulation as well as a task-specific local motion phase.

The second part of this thesis investigates the data-driven acquisition of trajectory profiles for local motion phases in task-specific manipulation. Many approaches to data-driven motion generation either utilize data *from human demonstration or teleoperation* [38, 49, 157, 166], or rely on *simulation-based training* [141, 161]. By contrast, this work focuses on the autonomy of the physical robotic system, which is excluded from consideration in the aforementioned lines of research. Instead, it turns towards a third available avenue of *physical self-exploration* [67, 117, 127]. While online exploration by the physical robot offers the unique opportunity to perceive real-world effects across modalities under the real dynamic behavior of commanded actions, it comes with substantial hurdles in terms of safety and efficiency.

Human demonstration of example behaviors enforces the desired behavior a-priori through the human operator, who explicitly limits the covered state space to adequate actions during data collection. Approaches for simulation-based training embrace an unrestricted exploration space through primitive actions and there is no risk of breaking the robot or parts of its environment. Due to the basic nature of individual actions, the resulting space of trajectories exceeds realistic exploration times on a real system and low sample efficiency is usually compensated by parallelization and increased real-time factors in the simulation.
Building on the results of the first part of this work, which demonstrate global manipulation planning with modular generators for motion phases, the described methodology for physical self-exploration focuses on the exploration of local motions through motion primitive parameterization. In this manner, well-defined motion ranges can be explored with functional coverage in an anytime manner. As the set of acceptable motions within this range is still unknown, the method suggests to apply a validity measure on the observed effects of each action as perceived across modalities. Through an incremental extension of trust regions in the parameter space, modelled in a Gaussian Process setting, the exploration can be guided towards the most informative regions of the parameter space while maintaining constraints on the estimated validity measure. To exemplify the methodology, experiments demonstrate the successful exploration of plucking interactions with a Chinese stringed musical instrument, where effects are perceived in the proprioceptive, auditory, and haptic modalities.

1.2 Scientific Contributions

The thesis contributes to the following two research areas:

Modular Planning Representations: What are effective modular representations and planning strategies for manipulation planning in semi-controlled robotics? The thesis proposes the concept of *Task Construction*, isolating different phases of manipulation in separate modules. These modules support phase-specific and adjustable planning strategies while remaining robot agnostic. Various aspects of the concept are demonstrated through the MoveIt Task Constructor (MTC) framework implementation and exemplary use cases.

Physical Self-Exploration in Vulnerable Environments: How can a robotic system acquire models of motion-phase trajectory primitives and their multimodal effects through constraint active exploration on the physical platform? This work demonstrates an active exploration strategy for robotic motion primitive exploration that explicitly models action validity through constraints on the observed effects and can thus reduce constraint violations during the exploration process. The method is validated on the physical exploration of plucking interactions with a stringed musical instrument.

1.3 Structure of the Thesis

Chapter 2 presents a background for robot motion and manipulation planning approaches providing a basis for all involved areas which are built on in the rest of the work. It briefly defines basic terminology and robot modeling approaches. An overview of approaches to trajectory generation and motion planning is given, exemplarily discussing various common techniques with their capabilities and trade-offs. A number of well-known publicly available software frameworks for trajectory generation and manipulation planning are discussed, focusing on the framework *MoveIt* utilized in this work. As the main component of this thesis focuses an intermediate level between the fields of *Motion Planning* and *Task and Motion Planning*, dominant approaches in those fields are structured and discussed.

Chapter 3 presents the main concept of *Task Construction* and details the implementation in the *MoveIt Task Constructor* framework. The chapter discusses the major components as well as planning dynamics, and several advanced mechanisms applicable to the structure. Multiple exemplary scenarios are used to illustrate the concept's various capabilities in applications.

Focusing on a particular motion type which can be part of a wider manipulation sequence, the subsequent **Chapter 4** investigates the above-mentioned second research question of physical robotic self-exploration in vulnerable environments. This chapter presents the proposed methodology using the example of plucking interactions with a musical chordophone instrument. Required model definitions and exploration steps are formulated and a multi-stage approach to exploration is detailed which focuses on model reconstruction and dynamics exploration. The approach is validated through a series of experiments.

Eventually **Chapter 5** summarizes the achievements of the thesis and discusses promising future directions for research in the investigated fields.

Chapter 2

Background

The capabilities and applications of the Task Constructor framework interface with various aspects of robotic motion generation. To provide a basis for the work, this chapter provide a general background over related areas, prominently *Robot and Motion Models* (section 2.1), approaches to *Trajectory Generation and Planning* (section 2.3), *Frameworks for Robotic Trajectory Planning* (section 2.4), and methods for *TAMP* as well as the closely related field of *MMP* (section 2.5).

2.1 Geometric World Models

Robot Models To plan and evaluate any motion on a robotic system without directly interacting with the physical world, a representation of the robot is required. Such representations can be defined with regards to different aspects and for different purposes.
Simple kinematic reasoning is already enabled with a minimal set of Denavit-Hartenberg parameters [30] and joint limits describing the geometric relationship between joints. These parameters are sufficient to reason about the pose of geometric frames on a robot, such as the end effector, but they do not describe the volumetric information required to reason about physical contacts. Such volumetric information can be represented at different levels of detail and some examples are depicted in Figure 2.1. The right illustration shows a *sphere approximation* of a PR2 robot barely recognizable from the image. While such primitive representations do not represent the real object shape, they are designed as application-specific over-approximations and are a typical means to accelerate computation. When a representation of the robot surface is required, meshes or *convex mesh approximations*, such as the Panda robot model illustrated in the center of the figure, provide a middle ground between primitive and detailed geometry. Lastly, visualization and photorealistic rendering require a detailed appearance of the robot model which is often described through textured meshes. Such a *photorealistic representation* is illustrated for the Sawyer robot on the left side of the figure. For motion planning, in contrast to visualization, simpler descriptions are preferred to enable fast detection of collisions between different geometries and systems commonly use mixtures of representation types.

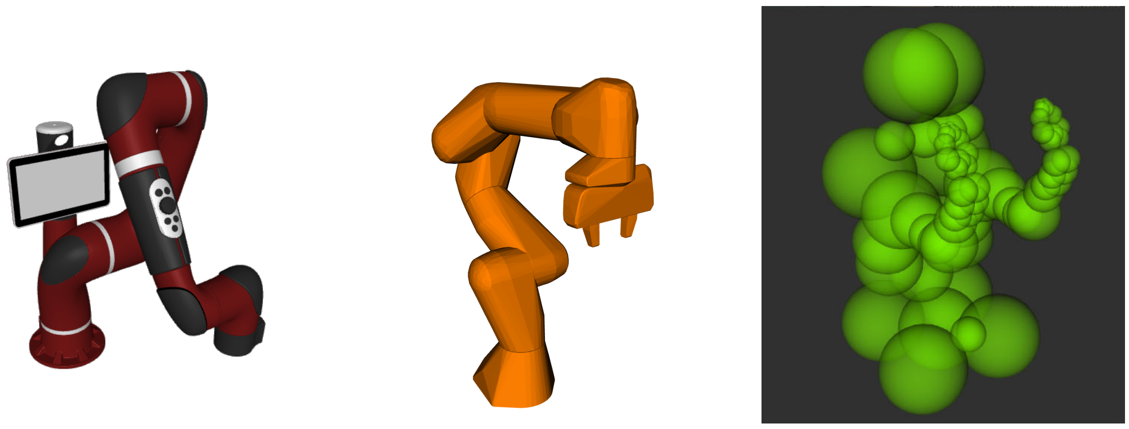


Figure 2.1 – Geometric robot models illustrating shape approximations of varying complexity: *(left to right)* a Rethink Robotics Sawyer [W24], a Franka Emika Panda [W9], and a PR2 [106].

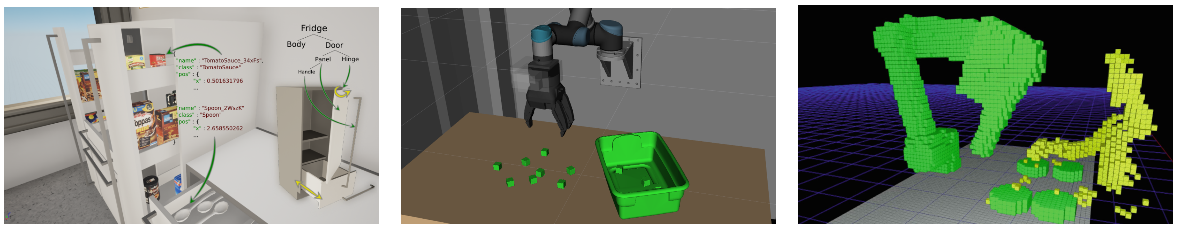


Figure 2.2 – *(left)* Articulated environment model with semantic object labels and appearance for forward simulation and reasoning [6]. *(center)* Tabletop geometric object representations as task-specific model in a collection scenario. *(right)* Uniform volumetric voxel representation without scene semantics [69].

Another level of description, which is mostly independent of the volumetric representation, is the dynamic model of the robot. To generate motions that can be actuated by the robot's motors and gear system, the joint torques required for the motion must be limited accordingly. In practice, it often suffices to consider box constraints on the velocity and acceleration of each joint as long as use-cases remain inside the usual operating ranges of the physical system.
When robot motions should not just be planned, but simulated under plausible physics, all forces and joint torques need to be considered and the robot model needs to define explicit torque limits, the mass and inertia of individual robot links, as well as friction properties for potential contacts. Identifying accurate parameters for each model aspect can be challenging. The robot link geometry is often available through the manufacturer in the form of CAD files and can be simplified as needed. Most dynamics parameters beyond velocity limits, however, are not directly available and adequate values are difficult to obtain. The process of identifying sufficiently accurate parameters for each aspect is described as calibration and *system identification*.

Environment Models When the robot should act in its physical environment, relevant aspects of this environment must be described as well. Depending on the intended use of the environment representation, the level of detail can vary here as well and ranges from simple volumetric

representations, as illustrated through a voxel grid model shown in Figure 2.2 (right), over specific modeling of manipulation-related objects (center), to detailed general-purpose ontologies with articulated object models, such as shown on the left side of the figure.
There are multiple crucial differences between robot and environment models: While model parameters for the robot are often calibrated offline and the robot model can be described compactly, the environment model might contain a varying set of object categories, depending on the intended application. As individual category instances can vary significantly in parameters, offline calibration is not feasible. In addition to unknown and possibly varying dynamics parameters such as an object's inertia, there is usually no ground truth for the geometric description of objects and their position in the model available and the estimates generated through perception systems are subject to epistemic uncertainty. Lastly, the semantics within a robot model are defined by design, but the environment constitutes an *open world* where any level of semantics might be relevant. To this end, semantic annotation can be added to the environment model, reaching from simple object labels to complex ontologies, involving relations between object categories and manipulation affordances of individual object instances.

Definition. *In the context of this work, the robot and environment models will be collectively referred to as the* ***world model****. A world model with concrete parameters for all relevant model aspects, especially the positions of all objects and movable joints, is referred to as a* ***world state****.*

There are various established formats to describe world models. The Unified (former *Universal*) Robot Description Format (URDF) format [W21] is the most common format for robot models and static environment elements which in many cases can be considered to be part of the model geometry. Many formats beyond URDF are bound to specific simulators and describe world models adjusted to the focus and the capabilities of the simulator. The Gazebo simulator utilizes the *Simulation Description Format (SDF)* [W18], which evolves over time as new capabilities become available. The *MJCF* model, native to the MuJoCo simulator [W12], is a world model format for articulated objects and robots with a focus on physics simulation. The *Universal Scene Description* format, prominently supported by the Nvidia Omniverse platform [W13], was originally developed for computer graphics and character animation, but was later extended to support physical aspects relevant for robotics. In addition to their direct application as geometric world models, information can also be extracted from them for ontological reasoning [110].
Beyond explicitly specified world models, a recently successful robotic paradigm aims to learn world models from direct world interaction [97, 156]. These models usually do not provide explicit geometric information, but can be used given actions to predict world states in the domain of the learned model, e.g. camera images. As they do not represent feasible state spaces explicitly, they are not directly applicable for the task planning and motion generation approaches described in this work. However, any system exploring such learned models also requires a geometric world model as described above to ensure feasibility of physically attempted motions and to provide a basis for different action representations and reward structures.

2.2 Motion Description

Based on a geometric robot model, one can consider the description of motions of the robot next. In planning contexts, robot motion generation is often decomposed into *path finding* — where to go — and *trajectory parameterization* — when to be there.

Definition. *A **path** is a continuous function $\gamma : [0, 1] \to \mathcal{S}$ which maps a domain parameter to a state space $\mathcal{S}$. A **trajectory** is a path domain-parameterized in time, such that $\gamma : [0, T] \to \mathcal{S}$ is indexed through time until an end time T. In contrast to paths, trajectories thus provide a physical interpretation of their derivatives such as velocity and acceleration.*

There are two state spaces $\mathcal{S}$ particularly relevant for robot motion planning:

- The *Cartesian pose space $SE(3)$*, *Special Euclidean Group 3*, describes all possible poses in 3D space in a reference frame with position and orientation. Trajectories which map onto this space describe the intuitive motion of physical objects and links of the robot, including the end effector, in the world.

- The *Joint Space $\mathcal{J} \subseteq \mathbb{R}^n$*, where each dimension corresponds to the position of a Degree of Freedom (DOF) of the robot. The individual dimensions are restricted to the valid joint ranges of the robot.

In order to describe continuous trajectories in few uniform parameters, they are usually represented as a series of waypoints which parametrize a spline function, such as B-splines or Hermite splines [27]. In robotics, the *quintic Hermite spline* is a common choice to represent trajectories as it provides continuous acceleration profiles along the path and is explicitly based on the position, velocity, and acceleration.
The computation of a feasible trajectory for execution given a joint path and a robot model estimates the time, velocity and acceleration for the path, defined for each waypoint. This process is referred to as *time parameterization.* As this step already involves explicit higher-order constraints on the motion to satisfy dynamics limits, might include modifications of the path within permissible bounds, and in some cases also supports constraints in both Cartesian and joint space together, it is sometimes considered a part of *trajectory planning* and will be discussed in the next section.

2.3 Approaches to Trajectory Generation

Generating feasible robot trajectories in an environment is among the most general problems in robotics and a wide variety of approaches has been implemented over time to compute applicable trajectories in different contexts, under varying constraints, and with different requirements w.r.t. smoothness, computational cost, and optimality.

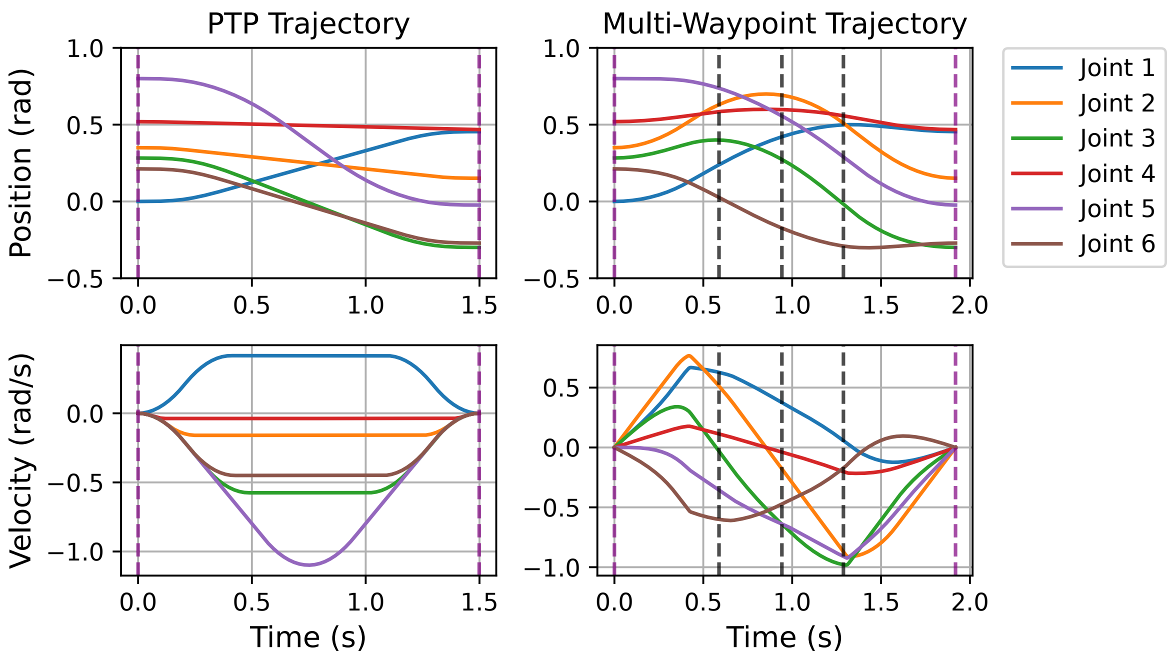


Figure 2.3 – Comparison of time parameterization for a path segment between two waypoints *(left)* and a path through 5 waypoints (vertical lines) *(right)*. Both paths use the same start and end points. The top plots show joint positions, the bottom plots joint velocities; *(bottom left)* s-curve velocity profile for path segment, *(bottom right)* time-optimal velocity profile for multi-waypoint path.

Primitive Industrial Motions Most industrial robots support different primitive motion types for standard path specification. The most basic of these are *PTP (Point-to-Point)* motions describing line segments between two points in joint space. The path between these points is usually time-parameterized with an s-curve velocity profile as shown in Figure 2.3 (left), jerk-limited with continuous acceleration, or a trapezoidal velocity profile, acceleration-limited with continuous velocity [92]. Both types accelerate in all joints in the direction of the target, maintain a constant travel velocity, decelerate towards the end.

The complementary *LIN (Cartesian Linear)* motion type describes a straight-line path from the Cartesian pose of a given robot configuration, to a pose in Cartesian space. As many industrial applications, such as welding or painting, require constant velocities along the tool path, these motions are time-parameterized with profiles similar to PTP motions, but apply to Cartesian velocities. A similar Cartesian motion type is the *CIRC (Circular)* motion, which describes an arc towards a target pose. In this case, the path itself is parameterized in a via point.

But in contrast to PTP trajectories, Cartesian motions represent highly non-linear paths in joint space for most common robot kinematic structures, especially so in the vicinity of joint singularities. As such, the corresponding joint trajectories do not follow a simple time profile in joint space anymore, require additional checks to ensure feasibility with respect to dynamics constraints, and are implemented through multi-waypoint joint space path planning.

Time-Parameterization Going beyond s-curve parameterization between two position waypoints, one can consider joint dynamics and parameterize paths with non-zero velocities and accelerations in both waypoints. Algorithms have been developed for *time-optimal* jerk-limited parameterization where goal states can specify velocities [80] and accelerations [10]. An important aspect of these generalizations is the synchronization of all active joints required for the coordinated motion of a kinematic chain.
These approaches consider a *single* path segment and do not readily generalize to paths defined through *multiple* waypoints. While it is possible to chain trajectory segments with zero velocity and acceleration at the waypoints, or blend between them, most sequences of waypoints can be traversed significantly faster with higher velocities and accelerations at intermediate waypoints. The coupling of these intermediate values under second-order constraints (in the form of dynamics limits) adds computational complexity to the problem and, while optimization problem formulations perform well, no general analytical solution is known. Kunz et al. [82] introduced a time-optimal algorithm based on forward integration with continuous velocities and bounded acceleration. Later, Pham et al. [116] presented a more efficient algorithm TOPP-RA based on a two-pass scheme on the discretized trajectory. Figure 2.3 (right) shows a prototypical time parameterization for a path composed of five waypoints with a time-optimal parameterization generated through TOPP-RA. Discontinuous acceleration can be observed at switching points as sharp changes in slope in the velocity profile.

Collision and Feasibility Checking The aforementioned trajectory generation approaches only consider the kinematic and dynamic properties of the robot and ignore geometric constraints. However, an essential requirement of practical motion generation is that the robot must avoid unwanted contact with the environment and between its own links. To this end, volumetric world models, as described in section 2.1, are commonly used: By assuming a candidate world state, the Cartesian poses of all shapes can be determined and tested for unwanted intersections [45, 101], commonly referred to as *collision checking*. In addition to individual world states, motions between two states can also be validated individually through *continuous collision detection* [120, 121]. Beyond collision detection, feasibility checks can also involve further aspects depending on the scenario, such as distances to human operators [93] or constraints on end effector orientation for welding tasks or container transport.

Sampling-Based Planning Relying on the path parameterization and feasibility testing methods above, motion planning problems can be reframed as search problems for *feasible joint space paths* instead of trajectories and time-parameterize them in a second step. In this regard, sampling-based approaches form a broad category of general-purpose path planners which are robust across different planning spaces. Fundamentally, such approaches utilize sets of samples in the planning space, i.e. world states, to approximate and explore the continuous feasible state space in order to connect given start and goal configurations. Approaches differ in their

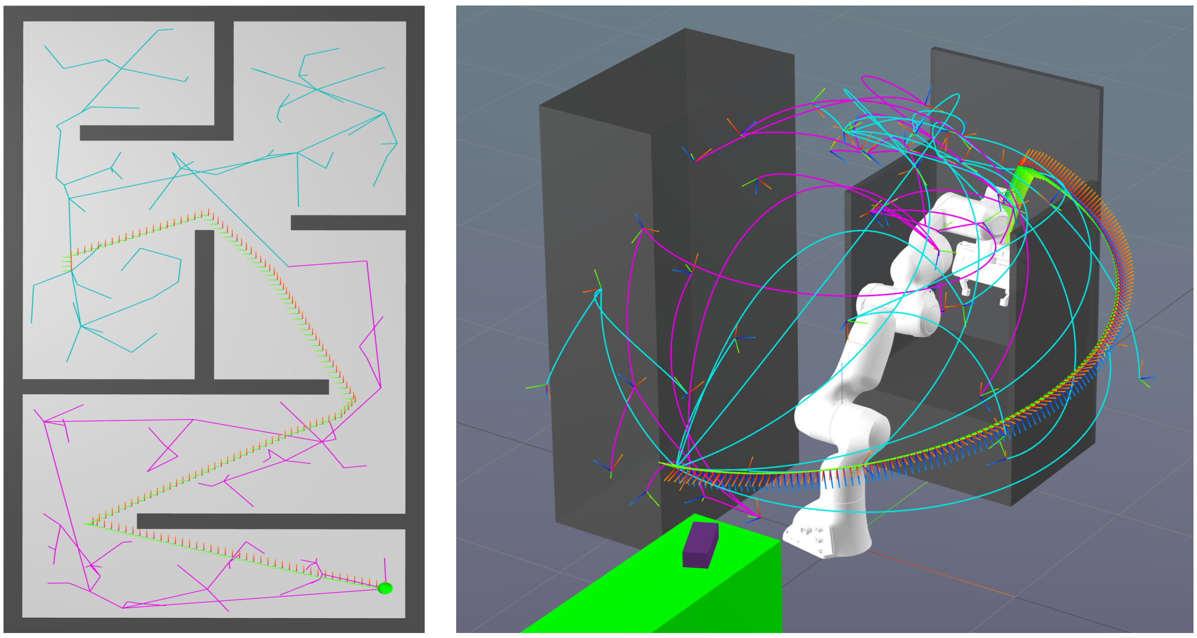


Figure 2.4 – Examples for planning trees generated through RRT-Connect: *(left)* abstract green sphere moving in a 2D labyrinth, *(right)* Franka Emika Panda robot with a 7D joint space in a warehouse scenario. The full initial world state is shown. Search trees are visualized in end effector space in pink (start tree) and turquoise (goal tree). Successful solutions after shortcutting are illustrated through consecutive coordinate frames.

exact strategies to sample the space, connect selected samples, and eventually refine or optimize discovered solutions.

The most traditional algorithm in this category is *Probabilistic Roadmaps (PRM)* [74] which builds a search graph by randomly sampling the space and connecting feasible samples in a neighborhood. The growing (or pre-generated) discrete graph is then used to find a path between start and goal configurations through regular A* graph search and can be reused with different start and goal configurations in the same space as long as these states can be directly connected to other graph nodes.

By contrast, the *Rapidly-exploring Random Trees (RRT)* [84] algorithm exploits the individual start and goal configurations during the sampling process. Each sampling step proceeds by selecting a random sample in the space. But instead of incorporating it directly into a search graph, the algorithm locates the nearest node in the current search tree (initially only the start configuration) and extends it *towards* the random sample by a maximum step size. This process asymptotically covers the full feasible state space reachable from the start configuration and generates tree branches which represent feasible paths from the start. To bias the search towards the goal, the goal configuration itself can be selected as random sample with an adjustable *bias probability*. The equally established *expansive-spaces tree (EST)* algorithm [61] extends comparable search trees through a different expansion strategy, where random samples are selected in low-density neighborhoods of existing nodes in the tree.

RRT-Connect [81] further extends the RRT approach and improves its efficiency through two major modifications: First, the algorithm utilizes a *bi-directional* formulation: It generates separate *start* and *goal* trees and attempts to connect them in each expansion step. This adaptation, originally proposed with EST, became a common feature in many sampling algorithms over time [61, 125, 136, 138, 139]. Second, while RRT originally extends the tree with a maximum step size, RRT-Connect extends the tree "aggressively" by repeatedly expanding new branches towards the random sample as long as the branch remains collision-free. These choices lead to drastically faster connections in practice. Figure 2.4 illustrates RRT-Connect planning trees for a 2D and a 7D scenario.
The planner remains a popular choice for path planning due to its comparably sparse trees and fast runtime in practical use and it also remains a well-performing competitor, when comparing the planning time to first solution, in current research benchmarks [40, 138, 153]. The contributions described in this thesis trivially generalize to arbitrary concrete path planners and thus most path planning examples in this work utilize RRT-Connect with shortcut refinement, detailed below.

Shortcutting Initial feasible paths from sampling-based planners directly connect random samples from the space and are thus very inefficient in almost all cases. As a simple post-processing step, one can shorten most solution paths significantly through simple approaches: In the case of *partial shortcut* [44], the algorithm repeatedly samples two random points on the path — which are not necessarily original waypoints — and checks the feasibility of a directly connecting path between them. If found feasible, the intermediate path is removed from the solution in favor of the new direct connection and the process repeats. The solution paths visualized in Figure 2.4 are produced through this process. Additionally, it can also happen that sampled points cannot be shortcut directly, but can still shortcut in multiple dimensions. By optionally sampling also a subset of dimensions for each pair, these cases can be at the cost of an overall increased runtime due to more required attempts. The *rope shortcut* [115] algorithm provides a deterministic alternative to partial shortcutting at the cost of an explicit path discretization.

Optimal Sampling-Based Planning Beyond simple post-processing of initial solutions, which are proven to be almost-surely *suboptimal* for the planners mentioned above [73], various planners were developed that integrate optimization strategies during the exploration process, ensuring *almost-surely asymptotically optimal (ASAO)* solutions. The *RRT** algorithm [73] extends regular RRT by maintaining the path cost from the start node to each tree node. For every new sample added to the tree, the algorithm attempts to connect it through the node with the lowest cost and additionally updates all nodes in the neighborhood of the new sample whenever the new connection results in a lower path cost. The FMT* and BFMT* algorithms [66, 136], as examples of non-incremental planners, assume a set of valid sampled states and implement a dynamic wavefront expansion on them, guaranteeing ASAO solutions as the sample size is increased over multiple planning attempts.

As a combination and generalization of both approaches, BIT* [40] alternates between incrementally sampling *batches* of states and exploring each batch for improved solutions paths. By additionally exploiting a user-provided heuristic function, the algorithm evaluates sample batches by their potential solution cost. To improve the efficiency of exploration beyond the initial solution, further sampling is restricted to regions of the space which might support better solutions according to the heuristic.
Lastly, the AIT* and EIT* planning algorithms [138] further build on the scheme of BIT* and introduce the concept of *asymmetric bidirectional planning*. Instead of building two equal search trees and aiming to connect them, the goal planning tree is expanded without dense edge validation and computes heuristic estimates to inform the search process during expansion of the start tree. The goal tree is maintained beyond computation of the heuristic values, and can be efficiently updated when invalid edges are detected during the forward search. EIT*, additionally introduces effort heuristics, estimating the amount of computation required to validate paths, as well as inadmissible but informative heuristics that can be used to guide the forward search.
As most optimization-based planners are based on an *anytime improvement scheme*, all uses of such planners as black-box invocations pose the additional question of when to stop the search and report the final trajectory. Best practices to resolve this challenge are predetermined timeouts or a convergence test on the solution cost during planner iterations [139].
All sampling-based planners operate in well-defined world models, and several of the required steps can be modelled through data-parallel operations. Thus performance of these algorithms can be significantly improved through parallelization and vectorization of key methods, notably feasibility testing of samples and connections between samples. Several recent projects independently demonstrated drastic speedups of established as well as novel planning algorithms through CPU parallelization [106], GPU offloading [140], and SIMD adaptation [142, 153].

Optimization-Based Planning The second established category of motion planning algorithms is based on numeric optimization of trajectory parameters. Contrasting the strength of sampling-based planners to quickly find valid global paths of a-priori unknown length, optimization methods excel at gradient-based local improvement of paths at the cost of susceptibility towards local minima. While the underlying optimization methods can differ significantly, most variety in approaches lies is the design of the problem space and the scope of the problem to be solved. The remainder of this section reviews several scopes at which optimization-based planning is applied.

Inverse Kinematics The most common use of optimization found in robotics is its application to find solutions for the *Inverse Kinematics (IK)* problem [154]. Formally, it describes the problem to invert forward kinematics equations and, given a desired Cartesian end effector pose, characterize joint space configurations which realizes this Cartesian pose. While various specific IK problems can be addressed analytically [146], optimization based methods support arbitrary

kinematic structures, additional constraints for feasibility testing and redundancy resolution, and facilitate *approximate* solutions when facing infeasible poses.
While individual kinematic solutions do not constitute robot motions, the optimization of joint configurations can be iterated to *roll out* complete paths or trajectories. To this end, dense incremental Cartesian waypoints can be solved to track Cartesian paths [5], regularization constraints such as minimal displacement terms for paths can be added to converge on goal poses over multiple iterations [123], or one can add the assumption that each solved iteration corresponds to a fixed trajectory time step, such that explicit higher-order constraints on velocity, acceleration, and jerk can be supported through numeric differentiation of past states [149]. A major shortcoming of these latter approaches however is that such higher-order constraints are only optimized for the current state while the effect of this state diminishes with the constraint order and the constraint function becomes mostly determined by previously computed states.

Model Predictive Control While single-state rollouts exhibit low problem complexity per instance because the requested solutions are very close to the initial guess, they also involve an inherent tradeoff between a preferably low step size to ensure consistent behavior between the steps for a given dynamics model and a preferably high forethought along the future trajectory to optimize on the effects of considered motions. One way to circumvent this inconsistency is to optimize multiple states together over several time steps into the future and increase the investigated planning horizon to short multi-waypoint trajectories. By optimizing over a fixed horizon for each step in turn, but only fixing the current step in the final trajectory, one arrives at a *model predictive control* scheme.
These schemes can be rolled out offline for trajectory generation, as implemented for example in the Giskard framework [137]. Alternatively, many implementations aim at online use in reactive scenarios. This becomes especially relevant with stochastic system dynamics and dynamic motions, where low step sizes are necessary for better physics approximations, but effects such as breaking motions and future environment contacts can only be evaluated at a longer horizon. Typical approaches in this area include Quadratic Programming on linear MPC formulations [56, 137] and *model predictive path integral* approaches exploiting fast sampled rollouts [11].

Global Trajectory Optimization This last class of optimization approaches aims to solve for globally optimal trajectories over the full trajectory at once. Such approaches come at the cost of much higher computational complexity with high vulnerability to local minima in constraint collision environments.
Among these, *CHOMP* [168] leverages functional gradient descent to iteratively refine a trajectory under smoothness and collision cost objectives. *STOMP* [70] employs Monte Carlo updates to reduce the risk of getting stuck in poor local minima, sampling noisy trajectory perturbations and favoring paths that lower user-defined costs. *TrajOpt* [130] formulates trajectory planning as a sequence of convex subproblems, allowing for efficient collision handling through con-

vex relaxation of obstacle constraints. In contrast, *GPMP2* [105] models motion planning via Gaussian processes and factor graphs, offering continuous-time trajectory representations and probabilistic reasoning over collisions. The recent *Graph of Convex Sets* (GCS) approach [95] assumes a partition of the feasible motion space into a finite set of convex feasible regions and connects them in a graph structure. This decomposition enables globally optimal solutions by combining convex trajectory optimization with mixed-integer graph search problems, which can still be solved through tight convex relaxations.

As sampling- and optimization-based planning techniques complement each other's strengths in finding global solutions and optimizing costs, their integration in combined approaches can improve planning speed and solution quality, prominently demonstrated in the BITKOMO planner [71].

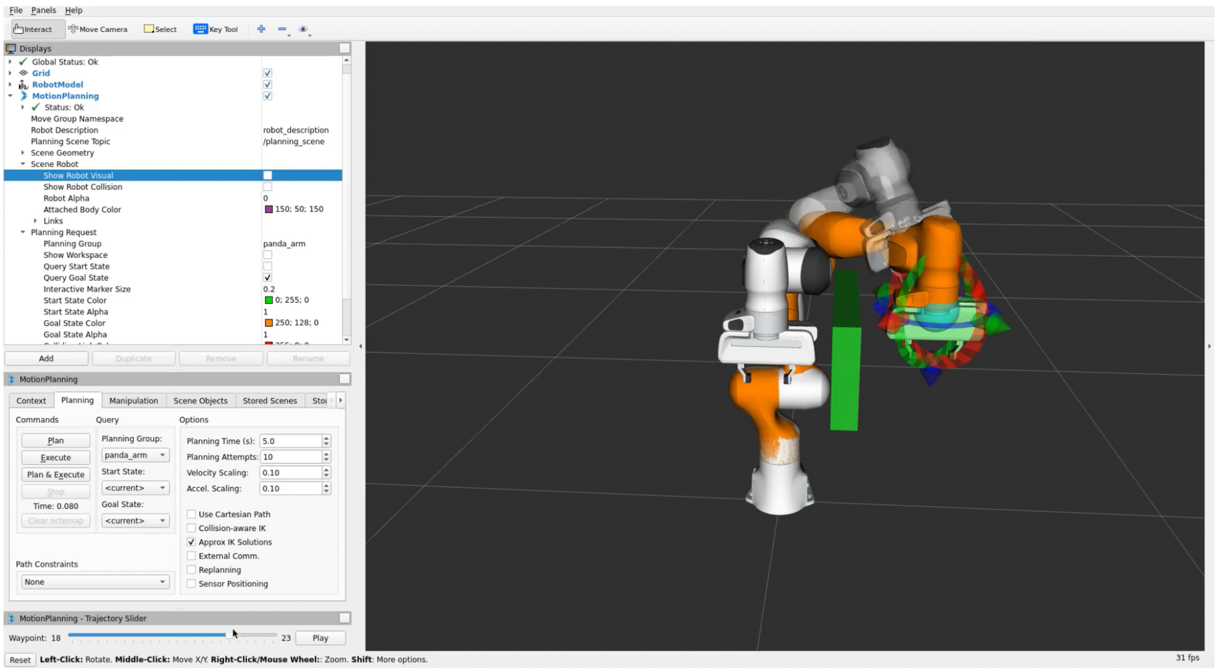
2.4 Software Frameworks for Trajectory Planning

Any use of the described approaches to generate executable trajectories for a physical robotic system hinges on a software implementation and the eventual performance of such use depends on the accessibility and quality of the specific implementation. For many proposed algorithms, software implementations are provided together with their respective publication, but these implementations do not necessarily integrate well with a regular robotic software ecosystem. To facilitate this integration with many common as well as with new systems, a number of groups maintain robotics frameworks that bundle various methods in a common software stack.

Multiple commercial robot vendors provide frameworks for their specific robot family, including Universal Robot's Polyscope [W16] and KUKA's KSS [W7]. Beyond these, software solutions for general process automation which aim for multi-vendor support include Optonic Mikado [W8], Energid Actin [W1], Intrinsic FlowState [W6], and PickNik MoveIt Pro [W10], among others.

Orthogonal to such systems targeting solution deployment, various OpenSource frameworks are available for research with different foci and levels of abstraction. These include low-level libraries, such as *Orocos KDL* [W20] and *Pinocchio* [19], for efficient computation of kinematics and dynamics, educational frameworks such as *PyRoboplan* [W4] emphasizing comprehensive implementation over efficiency, and frameworks specifically focused on optimization, such as *Horizon* [124], *Crocoddyl* [96], and *EXOTica* [65], for efficient definitions of optimization problems, *VAMP* [142] for SIMD parallelized sampling-based planning, and *cuRobo* [140] for GPU-accelerated planning.

Beyond these intermediate-level frameworks, there are several projects which aim for more general applicability across the robot motion planning and control domain. Prominent systems in this area include *MoveIt* [24], *Drake* [W19], *Tesseract* [W2], *Klamp't* [52], *HPP* [99], and further wrapper systems like the *Robowflex* project [77].



Global Status: Ok
Grid ✓
RobotModel ✓
MotionPlanning ✓
Status: Ok
Move Group Namespace
Robot Description: robot_description
Planning Scene Topic: /planning_scene
Scene Geometry
Scene Robot
Show Robot Visual ☐
Show Robot Collision ☐
Robot Alpha: 0
Attached Body Color: 🟪 150; 50; 150
Links
Planning Request
Planning Group: panda_arm
Show Workspace ☐
Query Start State ☐
Query Goal State ✓
Interactive Marker Size: 0.2
Start State Color: 🟩 0; 255; 0
Start State Alpha: 1
Goal State Color: 🟧 250; 128; 0
Goal State Alpha: 1

Time: 0.080
Planning Group: panda_arm
Start State: <current>
Goal State: <current>
Path Constraints: None
Planning Time (s): 5.0
Planning Attempts: 10
Velocity Scaling: 0.10
Accel. Scaling: 0.10
☐ Use Cartesian Path
☐ Collision-aware IK
✓ Approx IK Solutions
☐ External Comm.
☐ Replanning
☐ Sensor Positioning

Waypoint: 18 23

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:**: Zoom. **Shift**: More options.

Figure 2.5 – MoveIt user interface as part of the rviz visualization framework [W17].

The work presented in chapter 3 was implemented in *MoveIt* [24] and constitutes a major part of the frameworks's current manipulation capabilities. MoveIt is one of the largest OpenSource frameworks for robot path and manipulation planning and it provides a robot-agnostic abstraction from concrete robot controllers based on Robot Operating System (ROS) communication as well as general interfaces for planning with world models. While the framework is designed as a set of software libraries, it also provides a graphical user interface integrated in the *rviz* visualization framework [W17], as shown in Figure 2.5. The general framework supports alternative backend libraries implementing different methods required for path and motion planning. The core of the system is designed around a robot model based on URDF together with a flexible environment of explicit objects around the robot. This combined world model is termed *planning scene* within MoveIt. The core model description is augmented by the concept of *planning groups*, which constitute groups of actively controllable DOF of the robot considered together in kinematic chains and motion planning problems.
Another annotation, common across other frameworks as well, explicitly enumerates *allowed collisions* between links of the world model. Concerning planner performance, this explicit formulation foregoes a significant burden of computation during planning. Usually, many pairs of bodies in a model can never be in collision due to geometric constraints or will never be in collision without other pairs colliding. Such pairs can be explicitly neglected during collision checking through inductive priors. Additionally, the dynamic definition of allowed collisions between links can modify the feasibility of object contact as part of the world model state. For example, the annotations can describe whether a grasped object is permitted to come into contact with a specific surface or not — a decision which depends on the intention of the planned motion.

2.5 Task and Motion Planning

Task Planning Contrasting individual motions, where clear start and goal regions in the state space are specified, robot action generation can also consider sequences of actions for long-term action plans. On a symbolic level, such plans combine abstract actions such as *pick object X*, *place held object on X*, and *move to location X*. Given a target objective, such as *move the apple onto the table*, or *prepare dinner*, and an ontology describing the preconditions and effects of individual actions, *Task Planners* can generate a sequence of instantiated action symbols, such as ⟨*pick object apple*, *move to table*, *place held apple on table*⟩ to achieve the requested high level objective.
Various models to specify and solve such problems were proposed in the past, dating back to the pioneering *STRIPS* planner [34], and involving popular specifications such as *Hierarchical Task Networks* [33] and *Planning Description Domain Language (PDDL)* [37].
A representative example for a formal action definition in PDDL, relying on assumed logical predicates `near`, `on`, `gripper_free`, and `holds`, is shown in the following.

```
(
 :action pick_object
 :parameters (?obj - pickable ?loc - location)
 :precondition (and (near ?loc) (on ?obj ?loc) (gripper_free))
 :effect (and (holds ?obj) (not (on ?obj ?loc)) (not (gripper_free)))
)
```

In this example, the definition of the action `pick_object` is formally parameterized in an object `?obj` to be picked and the location `?loc` of the object, where both parameters are instantiated in each applied action. The action can only be applied in a world state where the robot is near the location of the object, and the robot's gripper is currently available, as determined by boolean predicates. Within the symbolic world model, applying the action will modify the truth values of the predicates `holds`, `on`, and `gripper_free` in the world state to reflect the intuitive consequences of picking up the object: The robot now holds the object, the gripper becomes unavailable for further grasps, and the object does not reside at its location anymore.
Since the recent proliferation of *Large Language Models (LLMs)* in generating intuitive sequences, different research directions proposed to generate task plans directly through natural language prompting without explicit model verification [63, 155], or to translate natural language goals into PDDL descriptions [90, 131].

Task and Motion Planning (TAMP) In general, all task planners are limited in their application with actual robotic systems where they have to ensure the *downward refinement property* [3]: While the abstract world model used for planning supports generated action sequences, the generation of robot trajectories to actuate these actions can fail due to kino-dynamic constraints that

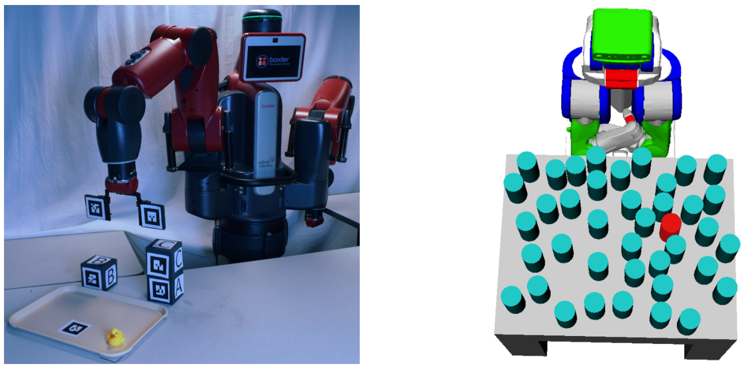


Figure 2.6 – Examples for TAMP scenarios [28, 135]. *(left)* Implementation of a blocks-world scenario with a robotics setting: To arrange blocks A–C as an ordered stack on the tray, the system must plan multiple symbolic actions to stack individual blocks and generate motion plans to lift and place the blocks. *(right)* With the goal to retrieve the marked (red) cylinder from the cluttered table, multiple cylinders block the direct path to reach the target and various pick and place operations need to be considered to rearrange the table and eventually retrieve the target object without collisions.

are not described in the symbolic model. One example of this limitation is the potential for a `pick_object` action to fail due to environment clutter: Collision-aware inverse kinematics to realize potential Cartesian grasps for the target object can fail due to infeasible robot collisions with the world model. To circumvent this limitation, the field of TAMP considers approaches to integrate motion planning within the task planning process [42]. Figure 2.6 illustrates representative example problems in this area.

A traditional and intuitive approach to combine solvers for both aspects lies in lazy satisfiability modulo theory solving [4]. In these approaches, as for example demonstrated by Srivastava et al. [135], task planners are invoked to solve the symbolic planning problem. The resulting action sequences, which are also called *plan skeletons*, are then further mapped to executable trajectories by solving motion planning problems that aim to instantiate the abstract actions with concrete robot motions. Upon planning failure for a motion, planners might backtrack and consider alternative motion parameters. When no instantiation for an action sequence can be found even with parameter backtracking, the task planner is informed that the provided solution is infeasible, possibly providing hints to guide the task planner towards better symbolic plans. Dantam et al. [28] add an iterative deepening approach to this scheme over the coupled symbolic plan length and the duration of planning attempts for sampling-based planners to achieve probabilistic completeness. PDDLStream [43] extends the pddl specification through additional *stream* predicates which enumerate motion parameters, such as object grasps, and planning proceeds in a similar iterative deepening fashion over found symbolic plan skeletons.

The main shortcomings of TAMP approaches in general are their overall complexity, impractical runtimes, and various crucial method parameters. These parameters are often treated as implementation details but strongly affect whether approaches might yield solutions in practice.

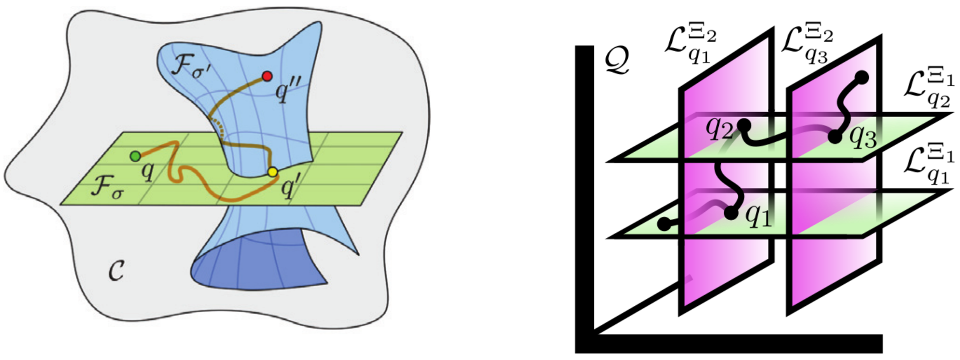

$\mathcal{F}_{\sigma'}$
q''
q
$\mathcal{F}_{\sigma}$
q'
$\mathcal{C}$
$\mathcal{Q}$
$\mathcal{L}^{\Xi_2}_{q_1}$
$\mathcal{L}^{\Xi_2}_{q_3}$
$\mathcal{L}^{\Xi_1}_{q_2}$
$\mathcal{L}^{\Xi_1}_{q_1}$
q_2
q_3
q_1

Figure 2.7 – Schematic illustrations of multiple intersecting planning modes as manifolds in the complete system state space [54, 78]. *(left)* Intersecting manifolds $\mathcal{F}_\sigma, \mathcal{F}_{\sigma'}$. Both manifolds intersect in a well-defined subspace containing state q'. *(right)* Modes with continuous parameters, such as an object pose, can characterize an infinite amount of manifolds as *mode foliations* $\mathcal{L}^{\Xi}_{\star}$. Two manifolds are shown for two intersecting mode foliations Ξ_1, Ξ_2.

They include determining when to backtrack to investigate different task plans, when to consider alternative choices for continuous parameters, when to assume (only asymptotically complete) motion planning attempts failed, how to configure planners to find feasible mode transitions, as well as what concrete feedback to provide to the task planner to guide the symbolic planning process and what aspects of the considered problem class to encode within the symbolic theory. Additionally, the entangled nature of TAMP algorithms hinders introspection when designing problems and critical modelling mistakes can be overshadowed by other attempted solution branches and heuristics.

Multi-Modal Planning (MMP) The field of MMP provides a complementary view to problem domains similar to TAMP, but it approaches the field from the perspective of motion planning. Under its interpretation, the planning spaces of each individual motion in a multi-step plan describe *mode manifolds* in the complete system state space. These manifolds are set up to intersect in well-defined subspaces, where a solution path can transition between different modes through states in the intersection. Figure 2.7 (left) illustrates the concept. Two often-examined modes are planning *(1)* with a free gripper and *(2)* with an object held in the gripper [78, 108, 126]. Each individual connection state in the intersection of these manifolds implicitly describes the complete process of the gripper grasping the stationary object and achieving a stable grasp (or releasing the gripper to place the object).
As start and goal regions are simply defined in the union of mode manifolds, the planning problem can be treated like any other motion planning problem with the additional scope extension that all relevant modes must be considered. Hauser et al. [54] introduce an intuitive approach *Multi-Modal-PRM* for finite sets of manifolds: The algorithm extends traditional PRM to sample each mode manifold *and each mode intersection* in a round-robin fashion which guarantees probabilistic completeness.

Beyond finite sets of manifolds, most manipulation problems actually involve continuous parameters, such as the set of gripper poses that can be used to grasp an object. These *co-parameters* can be seen to index an infinite family of manifolds, a *mode foliation*, where elements are *leaf manifolds* and only vary in their characterized co-parameters [55]. Leaf manifolds for the same foliation do not intersect in prehensile manipulation planning, as for example a grasp on an object cannot be modified without first releasing the held grasp, thus switching to a different mode before regrasping. Consequently, a main challenge in MMP is to identify promising foliation co-parameter values to explore, as only a finite subset of leaf manifolds can be considered.
Hauser [53] proposes a tree-of-roadmaps structure where they explore mode sequences through EST-inspired exploration of an infinitely-branching tree, each branch constitutes a PRM planning problem in itself. Schmitt et al. [126] globally sample a set of co-parameters together with intersection states while creating a PRM* graph across modes, altogether foregoing forward search through manifold sequences.
Thomason et al. [143] build on the iterative deepening structure of Dantam's TAMP approach, but treat planning as MMP. They generalize the optimal sampling approach of AIT* path planning to explore different manifolds once samples close to an intersection state can be reached.
Kingston et al. [78] propose a single-mode planning approach for mode foliations that permits connections between samples of different leaf manifolds in an *augmented foliated space*. During actual planning in a concrete manifold, many samples from this space can be projected to the manifold at hand, and inform the search process. Additionally, a second method guides the selection of concrete manifold sequences and co-parameters through statistics on previous attempts to plan through modes.

A shortcoming of all discussed approaches to Multi-Modal Planning is their simplistic representation of the semantics in mode intersections. While it can serve as a plausible abstraction for the considered problems to assume a single state space sample characterizes a transition, especially grasping and placing remain complex actuation domains in autonomous robotics even with simple parallel grippers [83] and generated solution paths in MMP approaches fall short to represent them. Additionally, all methods enforce motion planning through a single homogeneous mechanism across modes, which complicates plan adaptations for physical robotic hardware as well and does not exploit the potential for specialized motion planning algorithms for different modes. Chapter 3 contributes to this field through a modular and customizable architecture for solving given plan skeletons with known alternative paths and explicit mode-specific introspection capabilities.

Chapter 3

Task Construction

I presented the original design of the MoveIt Task Constructor at two PhD student schools (Interdisciplinary College 2017 and CITEC Summer School 2017). Afterwards the project was further developed in collaboration with Dr. Robert Haschke, who introduced containers, local ranking of Interface States during planning, pruning, and rviz Task introspection. Eventually, we presented the further developed framework together in a conference publication [46] at multiple conferences. The chapter is based on the conference publication with major additions to the original content.

This chapter presents the concept of *Task Construction* as implemented in the MTC software framework [46]. The concept provides means for the specification, planning, and introspection of manipulation trajectories in a modular fashion.

3.1 Manipulation Phases

Task Construction draws heavily on the definition of separable motion phases in manipulation actions. The concept of such phases is intuitive and prevalent in the literature of various research fields. Still, its definition and use differs significantly between fields with some overlap. For example, research in mammal and human prehensile manipulation [41, 68] characterizes modes of neural activity with task stages such as `Approach`, `Contact`, `Grasp`, `Lift`, `Hold`, `Lower`, and `Relax`. Studies in human reaching motions, such as Nieuwenhuizen et al. [111], split up reaching motions into at least a `Ballistic` and a `Correction` phase — introducing additional fine-grained phases as needed. Motion perception studies, such as De Stefani et al. [29], characterize phases such as `Initial Finger Opening`, `Maximal Finger Aperture`, and `Final Finger Closing`. For control in robotic grasping, fine-grained motion phases during finger closing are useful to adjust controller behavior for phases such as `Closing`, `Establish Force Closure`, and `Grasp Force Stabilization` used by Lach et al. [83].

In contrast, research in manipulation planning tends to split phases between different planning modes. Hauser et al. [55] distinguish phases by which DOFs are activate during trajectories and Kingston et al. [78] essentially describe only two parameterized modes for object interaction, which are `Held` and `Placed`.
Work in optimization-based trajectory generation, such as the Crocodyll framework [96], aims to solve for joint torques along fixed time steps assuming known inverse-dynamics models. As additional constraints are active for each physical contact between bodies, these methods define phases by the set of currently active contacts. The phase structure and duration of each phase is typically determined before optimization as *predefined contact sequence* and is not part of the optimization.
In all cases, the exact decomposition of phases depends on the aspects relevant to the individual research and focus points and there is no unique level of abstraction.
As there is no unique phase decomposition and useful decompositions highly depend on the required manipulation actions, the framework does not enforce any specific motion structure. Instead, it provides the means to specify motion phases and their characteristics in terms of computable units, providing introspection and the implicit propagation of solutions throughout these units.

3.2 Modularization of Manipulation Planning

As presented in section 2.3, the robotics research field provides a multitude of methods to generate feasible trajectories for any concrete motion. To allow for manipulation planning with hybrid methods, the driving design criterion of Task Construction is the black-box abstraction of motion phases in *Stages*.

Concept. *A **Stage** represents a computable motion phase of any duration that is part of a manipulation action. It comprises (1) a **Method** to compute robot trajectories and expected world model changes for the represented phase, (2) an **Interface** describing what information is required to apply the method, (3) a set of method parameters described in Stage-specific **Properties**, and (4) it associates with each generated trajectory relevant **Introspection objects** in the form of motion-specific geometric annotations. Computed trajectories, together with world model changes and Introspection objects, are collectively referred to as **Stage solutions**.*

In this formulation, each Stage encapsulates an isolated expert solver (a parameterized method) that is able to generate solutions for its motion phase. While the alternative design choice to utilize a homogeneous method to yield all phase trajectories can benefit from a uniform representation and partly shared inference, it also limits the compositionality of the complete system to utilize the chosen method. The investigated black-box abstraction of Stages chosen in this work allows for the integration of any expert solver, as long as it can provide an *Interface* and yields trajectories for a designated phase.

A focus of Task Construction lies in the local introspectability of each Stage. To this end, Stages usually generate additional *Introspection objects* together with computed trajectories. This introspection can include text annotations, but prominently includes geometric visualization objects[1] to highlight key aspects of a Stage parameterization, such as goal poses, waypoints, or selected link frame, which can be inspected together with the generated trajectories. Additionally, Stages can explicitly yield *failed* trajectories, together with further Introspection objects. While failures are not considered in further computation, they are essential for the introspection of unexpected failures.
Separate motion phases for a single manipulation action of course interrelate and in order to generate consistent trajectories for the whole manipulation action, information has to propagate between the individual solvers. To this end, the *Task* orchestrates all Stages.

Concept. *A* ***Task*** *represents a well-specified structured manipulation action and comprises* ***a set of child Stages****, representing motion phases of the action. During planning, the Task* ***explores solution paths*** *by scheduling its children to generate trajectory samples and providing a* ***structured information flow*** *between these children. A* ***Task solution*** *is a connected sequence of Stage solutions, reaching from the initial to the final Stage of the Task.*

A major framework design decision addresses the potential structure of a Task to exchange information between Stages. As the Stage concept presupposes solvers without intermediate communication, Stages do not affect each other during computation, but only through completed solving attempts.
To allow for maximum flexibility in a heterogenous environment, any kind of information might be shared among any expert through a blackboard or noSQL database format. Such approaches are common in cognitive architectures such as the Cognitive Robot Abstract Machine [7, 8], the ArmarX memory system [114], or the RACE blackboard [59], which act over long time horizons and support generic extensions for research projects.
This flexibility comes at great costs though. Passing unstructured information drastically *limits composability of Stages*. As every Stage needs to extract the relevant information from passed data, any Task instantiation would decide *per communication* on their own data types and contents. As a result, Stages cannot be directly reused for different Tasks.
Secondly, modularity and the ability to introspect plan generation degrade due to global side effects. As information in a global state can affect any expert, the planning problem cannot be reliably decomposed along the different Stages and the full current planning state of all Stages is required to explain the behavior of a single Stage.

Instead, Task Construction relies on a concept similar to *manifold intersections* or *mode transitions* which describe an established concept in the domain of multi-modal planning (see section 2.5). Following the observation that temporally adjunct motion phases relate to each other

[1]MTC utilizes `visualization_msgs::Marker` ROS messages

Structure	Compoundability	Flexibility	Locality
Blackboard	No	Full	No
Local World State Passing	Yes	Flow-Structured	Yes

Table 3.1 – Comparison of information passing schema for black-box modules.

only through a single world state within a low-dimensional manifold in which the state can be considered an element of both motion phases, Stages can be limited to communicate explicit world state samples from this manifold.

By communicating only these states to designated other Stages, *locality* is established, as each Stage computation can be understood in isolation given the received individual states the computation is based on. As another consequence, Task structures are directly compoundable, as only interfaces between different compounds need to be specified without risking global side effects. Table 3.1 summarizes these aspects.

Additionally, a small amount of information not represented in the world state can be relevant for general manipulation strategies. Considering a manipulation scenario of a bimanual system picking up a small object, an initial Stage of a Task could consider picking the object with either arm and would generate trajectories to move the selected arm towards the object. The decision which arm is moved in each trajectory constitutes a planning commitment not explicitly represented in the world state. Such decisions are referred to as (local) plan commitment, because continuations of the generated solution path should commit to the same decision. Alternative considered trajectories within the same global planning attempt might stray from this commitment though. As the type of such decisions varies between Task structures, the concept of typed *Properties* can be reused here.

Put together, the information communicated between neighboring states is defined as follow:

Concept. *An **Interface State** comprises (1) the kinodynamic state of a world model including the robot and objects and (2) a set of typed **Properties** defining non-geometric aspects of manipulation actions. Interface States mark both end points of robot trajectories generated by Stages.*

Using this concept, one can consider the structure of Stage communication. The Task Construction framework aims to abstract over various methods to generate trajectories in a Stage. To support any computational approach without custom additions, all Stage interaction happens through *directed passing of explicit Interface States*. Thus, the only assumption on applicable approaches is that, given a set of inputs, they can produce a number of trajectories.

Stages that generate trajectories associate these trajectories with Interface States representing their respective temporal start and end. Moving forward, these Interface States are referred to as *start state* and *end state* of the respective phase. Interface States which are newly generated in this process are passed on to their neighboring Stages. As new states arrive in other Stages, these

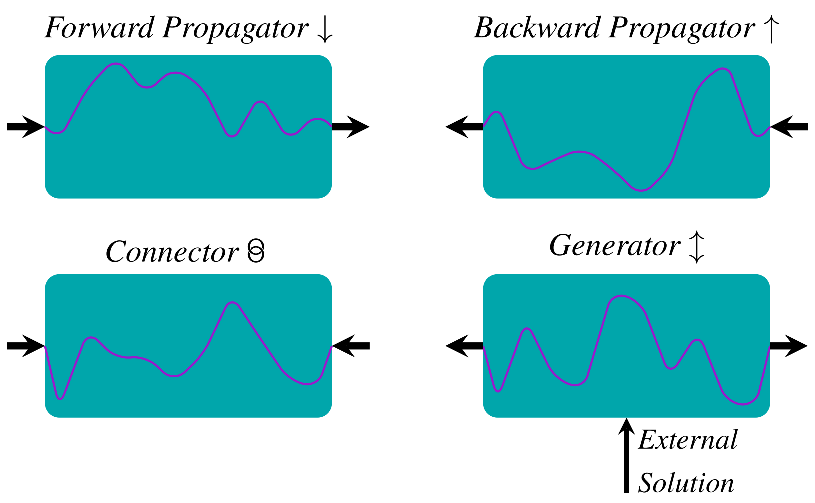


Figure 3.1 – The four Stage Interface types. Each Stage supports at least one type. Arrows indicate from which temporal direction an Interface State is available (either of the start or end state for the represented phase might be provided) for computation and which world states are generated as part of computed trajectories.

will initiate further planning steps to generate trajectories for their respective phase extending from the received Interface States.

Crucially, it does not suffice to pass Interface States forward in time, as this can leave Stages with insufficient information to compute trajectories. A simple example of this restriction can be observed with motion generation based on *grasp planning systems* [87, 109]. The basic framing of these systems is to compute Cartesian grasp positions for a gripper from perception independent of the rest of the robot system. Integrating such an approach into a Task structure and starting the planned motion from the current state of the system, the subsequent transit trajectory can only be computed *after* a grasp has been selected in the later part of the Task. Once the grasp -and the finger motions related to it- are determined though, the start state of a grasping trajectory can propagate backwards in time such that the transit can be computed based on the current robot state as start and the passed Interface State as end of the transit trajectory.

The decision to direct the communication between Stages entails a set of four potential types of interfaces, as illustrated in Figure 3.1. These are detailed with their respective use-cases in the following sections.

3.2.1 Propagators

The most intuitive interface type is the *Forward Propagator* ↓ which describes goal-guided computation forward in time. It assumes a start state and computes trajectories resulting in different end states. As the concrete end state is not known in advance, this covers many use-cases where the Stage defines the end state through constraints, such as the motion towards a Cartesian goal pose, and solution trajectories can be computed based on these constraints.

Additionally, it encompasses motion primitives which can be rolled out from an initial state. In such a case, the relevant part of the motion phase is not the reached end state, but the motion profile itself. Examples include *affordance templates* [50] and motions to pour liquid from a held container into a target container. This later example is further detailed in subsection 3.9.2. Propagators are generally not limited to compute a single trajectory from each start state, but can also fork trajectories to multiple new end states. This way, Tasks can consider multiple alternative trajectories within a Stage, such as different ways to approach a goal pose and multi-modal goal constraint specifications. In this way Stages are not bound to a single continuation of each initial Interface State.
In addition to trajectory generation, Propagators can also describe semantic changes in the world model, as they can freely modify the new world states they yield based on the received state.

While forward inference agrees in execution direction and the direction of inference, the *Backward Propagator* $\uparrow$ explicitly inverts this relationship, rendering it much less intuitive at first. It models situations where the end state of the motion phase is known and the motion itself together with the start state can be inferred. The final solution of the Stage will nevertheless be executed from the start to the end state. Although seemingly rare, it is quite common in manipulation planning, as approach motions towards a known goal pose are a common use-case:
In many systems, end effector grasp poses to manipulate objects are directly determined from visual perception and robot motions towards these poses are considered *transit* motions. The validity of the transit trajectory, however, is determined in the known or perceived environment model, and thus validity can only be guaranteed up to the accuracy of the model to describe the real robot workspace. As this model gap can span between few millimeters up to few centimeters[2], transit motions computed using generic motion planning methods often do not yield sufficient clearance with the actual environment to allow for successful execution and are likely to get the end effector in contact with the object which should be approached. To provide more robustness and predictability for execution of these approach motions, it is often beneficial to compute a short linear Cartesian motion approaching the goal pose. An additional benefit of such a well-defined approach motion is the increased predictability of the robot for observers which is often a requirement in human-robot co-working spaces. This motion phase is well-modeled through the Backward Propagator, as the Cartesian motion can only be computed based on the targeted goal pose and the trajectory yields the temporal start of the approach phase, which can be used as the end of a generic transit motion preceding the Stage.
Although the mirrored direction of inference is a crucial difference between both Propagator types, the underlying computation of the trajectory can often be generalized between them by simple inversion of the generated trajectory and inverting the effects on generated Interface States. In the case of a Cartesian motion Stage computing an approach motion, it suffices to

[2]Discrepancies arise from perception accuracy, robot calibration, robot trajectory following behavior, and the overall efforts of the responsible engineers to improve the model quality for the requirements of the respective task.

plan a *retract* path γ_{ret} which starts in the end state and follows a Cartesian path for a specified distance. The resulting path (or time-parameterized trajectory) can be inverted afterwards to yield the final approach γ_{app}:

$$\gamma_{\text{app}}(t) = \gamma_{\text{ret}}(1 - t)$$

When Propagators implement invertible unidirectional logic, such as in this case, implementations can generalize them to *bidirectional* interface types, where the second direction inverts the trajectory generation process and effects on the model state. Task structures can infer the required direction of bidirectional Propagators from context during initialization.

3.2.2 Connectors

The *Connector* ⊜ interface characterizes the most common type of motion planning problem: Given a start and an end state, the Stage has to generate a robot trajectory connecting them. As Interface States always include a complete robot state with all joint positions, this request constitutes regular (constraint-aware) motion planning between joint space targets.
Its functions in the context of the Task structure come with additional considerations. As it provides the bridge between forward and backward inference directions, it will collect various Interface States from both directions during planning. Structure-wise, any pair of these states can be combined to form a valid planning problem. However, all states constitute different planning attempts, and state pairs can differ significantly beyond traditional joint space planning. For example in scenarios involving the consecutive movement of multiple objects, a Connect Stage may encounter state pairs in which not just the robot joint state differs, but the pose of objects unrelated to the current motion phase as well. Considering the problem to move between such state pairs, it — in itself — describes a separate manipulation planning problem where objects at different positions have to be moved around through a connecting robot trajectory. To circumvent such a recursive problem decomposition, Connectors introduce *compatibility checks* between state pairs to ensure that states which might be connected coincide in all aspects which cannot be derived through traditional joint space planning. This includes the existence and pose of all environment objects, but also the current poses of all joints which are not expected to move in the current motion phase. The exact requirements for compatibility can differ depending on the concrete Task. For example, in scenarios with rotationally-invariant objects, object poses that differ only around its axis might be considered compatible. In these cases, custom compatibility criteria can be specified.
In practice, Connectors often represent *transit motions*, i.e., motions that do not directly engage the robot in manipulation with its environment, but instead move the robot between relevant poses at which manipulation actions are executed. In this role, there are two implicitly different types of Connectors: *(1)* In the situation where both Interface States are computed independent of each other in the same environment state, essentially all $M \times N$ state pairs are compatible. This is

the case in the example later presented in subsection 3.9.3. *(2)* In many other situations Interface States on one side are effectively computed from individual states on the other side. When there are essential differences between the different source states, such as the grasp applied to an object, these differences propagate to the other side as well. This effectively reduces the amount of compatible state pairs to a much smaller subset $M \times N'$, where N' describes the average number of states produced for each of the M source states. This applies to the transit motions in *pick and place* tasks, which are detailed in subsection 3.9.1. In either case, the Connector constitutes the main bottleneck in exhaustive Task planning, as the combinatorial explosion between state pairs leads to a significant increase in required planning attempts. Consequently, good heuristics need to be employed to attempt promising state pairs first.
Lastly, it can happen that planning requests between compatible states are *intrinsically infeasible*. That is to say, there exists no feasible path between these states that could be found by a motion planner. This can happen in scenarios where the model geometry entails disjoint feasible subspaces of the joint space, such as in the case of the UR5 robot arm [W14]. But more commonly paths between states can become infeasible in the context of additional path constraints. An orientation constraint on the end effector, for example, renders all state pairs infeasible, which intrinsically require the end effector to rotate. This example is further detailed in subsection 3.9.2. In the general case these situations are part of the intrinsic complexity of motion planning. For theoretical guarantees, *complete* planning approaches resort to semi-finite planning strategies to ensure paths for all feasible state pairs are found eventually. As Tasks in practice utilize finite resources and individual Stage computations are expected to complete, this approach is not feasible for Task Construction. While a technical alternative can implement interruptible Stage computations or iterative deepening of planning attempts with incremental timeout parameters, these techniques introduce additional parameters and more computational overhead when reattempting infeasible planning problems. Instead, the MTC implementation balances completeness with attempt-wise runtime and utilizes user-configurable timeouts for individual planning attempts and marks unsolved planning problems as failures.

3.2.3 Generators

In contrast to the other interface types, the *Generator* $\updownarrow$ interface does not describe a traditional type of motion planning problem. Instead, it represents the situation where neither the start nor the end of a Stage phase are required to compute a complete trajectory with associated Interface States. As Task planning forwards Interface States between Stages, Generators act as local seeds for Interface States which initiate the planning process.
Initially, Generators within a Task can only perform computation based on information provided to them outside the Task Construction framework. Such information can prominently include specific world states to assume at some point during the manipulation, including the current state as start state in an online planning system, as well as perceived object poses or perception

data. They are thus restricted to either generate fixed Stage solutions, or sample solutions from available information.
While multiple Generators can be initialized independently, it simplifies the use of a Task to instead restrict external information seeding to few Generators and provide another information source for others. To this end, Task Construction enhances Generator Stages with the ability to *monitor* another designated Stage within the Task and receive all generated external solutions. This additional information channel also allows to forward planning commitments, such as the end effector used in a previous phase, or the exact grasp used to pick up an object, *beyond* Connectors (which connect states, but do not yield new information) to subsequent steps.
Three crucial Generators regularly used in Task Construction are introduced in the next section.

Task Interfaces Notably, as complete Tasks are composed of individual connected Stages, where each Stage possesses a designated Interface type, the Task itself will also possess an Interface. There is no strict formal reason to designate a specific Interface for the Task structure, as all Interface types provide their respective use-cases in general manipulation planning. However, Task Construction aims for a uniform structural representation of Tasks. To this end, it simplifies Task interfaces to avoid leaking the concept of Interface States beyond the Task structure. As the only Interface that does not require Interface States to be passed *into* the structure is the Generator $\updownarrow$, the Task Interface is, without loss of generality, defined to expose a Generator Interface.

3.2.4 Instantaneous Stages

In general, solutions generated by Stages describe robotic trajectories connected to world states. The special case of *instantaneous trajectories*, representing only a single moment in time, is of particular interest for reasoning here. Utilizing this concept, Generators within Task Construction can isolate computation on *individual model states* instead of extended trajectories. Such Stages are referred to as *instantaneous Generators*. They often determinate key decisions along individual planning attempts about joint space regions by generating an individual world state along a Task plan and passing it on to neighboring Stages to plan the motion around this point of the final manipulation plan.
Relevant instantaneous Generator semantics include the following:

- *Assume a fixed robot model state*. To provide the Task with a model state to plan with, it is possible to assume a fixed state in an instantaneous Generator. Examples for these include potential start states during Task evaluation and well-defined end states such as the robot's home position at the end of a motion plan.

- *Acquire a model of the running system's current state*. As most Tasks are planned for execution on a real robot system, the trajectories have to relate to the current system state.

They will usually start in this *Current State* or optimize states for minimal displacement from it. In the context of MoveIt planning scenes, updating the model requires reading the system's proprioceptive joint state as well as externally estimated visually-estimated transformations of environment objects. While this Generator may be fully implemented through the fixed Generator above, both differ sufficiently in semantics to warrant separate definitions for intuitive bridging between off-robot analysis and online planning for execution, avoiding explicit data structure handling outside the Task.

- *Inverse Kinematics*. Arguably the most essential type of reasoning about instantaneous robot states during manipulation planning is the computation of joint poses from Cartesian target poses or constraints. Such reasoning is required, for example, to reach Cartesian grasps with a kinematic chain. Generally in robotics application, the inverse kinematics problem is often solved in a separate step from trajectory generation. Task Construction encapsulates this logic as an instantaneous Generator. However, inverse kinematics can be decomposed beyond the Generator concept and Task Construction models it as a *Wrapper Container*. This structure will be detailed in the respective section subsection 3.4.2.

Additionally, instantaneous Stages model a second important aspect of manipulation planning: In multi-modal planning, motion phases often neighbor in states which change the behavior or interpretation of the geometric world model. When a robot grasps an object, the kinematic model has to be adjusted to account for the object's presence in the gripper and avoid unwanted collisions of the object with the environment in the follow-up motion. Instead of capturing these changes implicitly or outside of the framework, as is typically done in multi-modal planning (compare review in section 2.5), Task Construction explicitly models them through *instantaneous Propagators*. In the context of MoveIt's planning scene used with the MTC framework, they include the following:

- *Attach and detach objects*. In order to represent the successful grasp or release of an object from a manipulator, environment objects can be set to maintain a fixed transformation relative to a specific link in the robot's structure. Attaching an object to a link in the robot's end effector thus allows to continue planning with the object as part of the robot's kinematic model, while detaching the object will remove it from the robot model again, assuming it is placed in the environment again.
- *Update permissible collisions between model bodies*. Traditionally, most contacts between robot links and the environment are considered hazardous and should be avoided by generic motions. Before key motion phases such as closing an end effector for grasping, however, the intended individual contacts need to be permitted. To this end, MoveIt provides an annotation (the *allowed collision matrix*) to represent allowed contacts between individual bodies as part of the model. These contacts are not limited to the robot's

links, as for example placing an object on the table requires an allowed contact between the pair, while likely no robot links should be involved in contact.

- *Update collision geometry*. Generally, it can also be useful to adjust the model geometry instead of only the model state's collision matrix. In manipulation planning, most transit motions can be validated with coarse collision geometries to speed up feasibility checks and increase the minimal distance between the environment and the moving robot. For precise approach motions though, the robot's end effector has to be represented with its full geometry to ensure a safe approach. To implement this, intermediate Stages can adjust environment padding or provide simpler object models for transit phases. An example for this use-case will be discussed in subsection 3.9.3.

3.2.5 Properties

As described in section 3.2, methods implemented in Stages often expose parameters, which vary between different instantiations of the same Stage. Such parameters can include *(1)* declarative aspects, such as the *name* of the Stage within the Task structure or the *reference to a monitored Stage* in Generators, *(2)* geometric parameters, such as *which object to grasp*, or *how far to move*, as well as *(3)* algorithmic parameters, such as *how long to attempt planning*, *how many solutions to consider per attempt*, or *how many intermediate waypoints to include in solution trajectories*. To formalize these parameters in Task Construction, they are introduced as *Stage Properties* which affect the Stage's computation.

Properties are explicitly declared by each Stage and can support values from a set of predetermined types. In this way, a Stage `MoveTo` can, for example, support a `goal` property which is either a Cartesian pose or a Joint pose. It is the Stage implementation's responsibility to interpret the property values and adjust the computation accordingly.

All required properties of Stages must be explicitly specified during Task initialization, to provide all information required for Task inference. However, this leads to a significant amount of redundant specification as many Stages utilize the same parameters, such as the set of joints which should be controlled during parts of a Task. Additionally, not all properties can be uniformly specified for all attempts within a Stage as they might differ between different plans. Assuming one of several objects can be grasped to fulfill the Task, there is no unique object for which grasps should be proposed and different planning attempts might vary in this Property. To enable these cases, Properties might also be specified with different initialization sources. Specifics on these initialization sources are presented with their respective context of use in subsection 3.3.2 and section 3.4.2.

3.2.6 Decoupling Where and How

Many relevant target trajectories modelled by Propagators specify motion goals relative to coordinate systems in a given world state (which can be either start or end state), and either define the motion's goal pose, or a constraint on the motion. Examples for these include moving a link along a vector or twist in some reference frame, or moving joints to offset positions. To avoid redundancy, these specifications can be condensed into two common stages `MoveTo`, for target goal poses, and `MoveRelative`, for movement directions.
Similarly, Connectors match incoming Interface States on both ends, and generate planning problems with model states that should be connected through a trajectory.
While all of these Stages describe *where* the robot should move, many of them do not specify *how* the robot should move there and none specify *how*, as in *by which method*, the trajectory should be generated.
To decouple the latter aspects from the Stage specification, Task Construction utilizes a separate `planner` property associated with them, which can be specified as any expert planner. Depending on the Stage, the selected planning method must either support generating connecting trajectories given two compatible world states or trajectories to a Cartesian goal given an initial world state. Planners are expected to determine the validity of their generated trajectories internally, as they act as black-box experts, but are explicitly allowed to return relevant infeasible solution attempts marked as failures. Returning such infeasible, or *approximate*, plans strongly supports introspection into the planning process, as these planning failures often indicate constraints violated in the goal specification.
The MTC implementation provides several default planners derived from the MoveIt subsystem which are available for direct use:

- Direct *Joint Interpolation* implements a simple joint-space stepped waypoint path, which is time-parameterized. While computationally very efficient and optimal in joint space distance, it obviously does not adapt trajectories to the robot environment.
- *Cartesian Path* generation, which steps Cartesian waypoints and utilizes configured IK solvers for the specified joint model group to project the resulting path to a consistent joint space path.
- A MoveIt-specific *Pipeline Planner* which utilizes configured MoveIt planning pipelines to generate trajectories. Available planners that have been used include various sampling-based planners available through OMPL [139], such as `RRTConnect` [81] and `RRT`* [73], implementations of `CHOMP` [168], and `STOMP` [70], as well as industry-endorsed `PTP` and `LIN` planner implementations.

While either one-shot or optimizing planners can be selected for individual Stages, each planning problem will only be considered once throughout Task planning and the resulting trajectories can

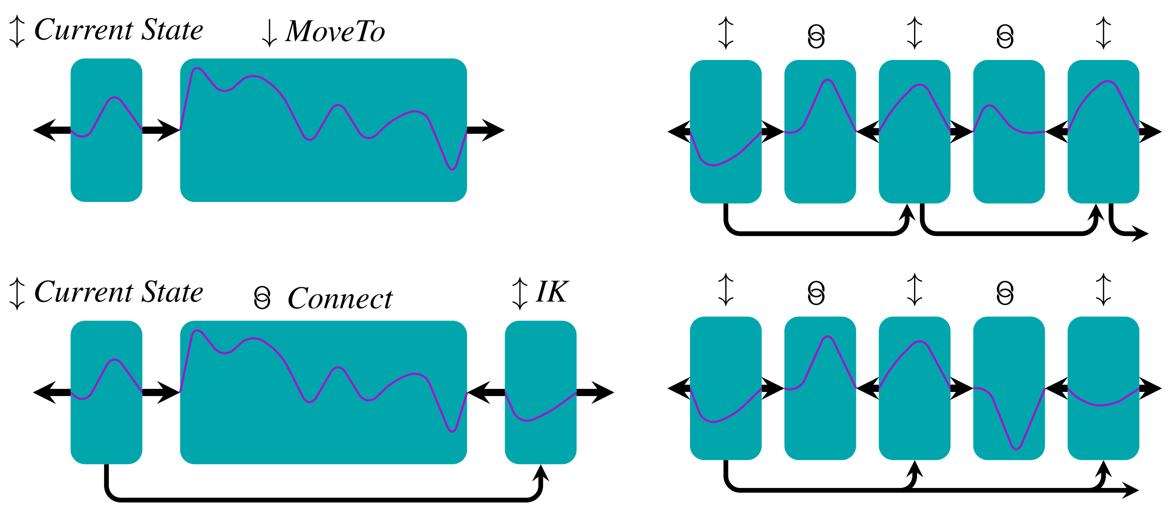


Figure 3.2 – Illustrations of alternative Task layouts. *(left)* A trajectory to reach a Cartesian pose can be generated by a `MoveTo` Stage parameterized in the target pose or through independent IK computation connected through joint space planning in a `Connect` Stage. *(right)* In a Task with multiple independent Generators, Monitoring Generators can either monitor the preceding Stage or the first relevant Stage in the sequence.

contribute to the final Task solution. As one-shot sampling planners return suboptimal paths with probability one [73], this leaves room for potential further improvements.

3.2.7 On Alternative Modeling Choices

As the framework does not ordain a specific method to solve Tasks, it also enables different structures to achieve similar manipulation trajectories depending on the utilized methods. The simplest example of such a choice is how to design a Task for the elementary request to move the end effector to a Cartesian pose. Some methods, guided by local gradients from the system's current state, support this as a native query (e.g., Jacobian-based steering), but many other approaches (such as PRM and RRT) rely on IK and only solve for trajectories between joint poses.

For the first group, it suffices to model a `MoveTo` Stage which receives the current state as its start joint position and reads the Cartesian target pose as a parameter, forwarding it to the utilized method. The respective structure is shown in Figure 3.2 (top-left).

The second group of methods can be modeled as Tasks by adding an explicit monitoring IK Stage which computes joint poses for the end of the trajectory, such that a `Connect` Stage can compute trajectories based on known start and goal states, as illustrated in Figure 3.2 (bottom-left).

Neither approach is strictly superior to the other. Methods with the former approach can exploit the explicitly passed Cartesian pose, e.g., to optimize the joint values for the Cartesian pose in the kinematic chain's null space with respect to the last part of the trajectory. The latter approach, on the other hand, allows for explicit reasoning over considered joint space targets, directly supports target introspection and a potentially parallel search for trajectories with different global kinematics solutions. As the joint values in the IK Stage are computed based on the joint pose in `Current State`, kinematic optimization w.r.t. the start state of the trajectory (in contrast to the whole trajectory) is still possible in the second structure.

A second notable design choice in structuring Tasks is specific to Monitoring Generators, as their corresponding monitored Stages have to be defined. In Tasks where multiple phases of a motion sequence are mostly independent—such as for reaching multiple waypoints in succession—and each phase is built around a Monitoring Generator, different monitoring strategies can be employed. The right sketches in Figure 3.2 illustrate this scenario. Each involved Monitoring Generator can *either* monitor a Stage from the preceding phase (illustrated in the top-right) — effectively serializing most of the solving process —, *or* monitor the first applicable Stage in the sequence (illustrated in the bottom-right). And again, neither approach is strictly superior to the other. The former approach retains a more natural branching structure and can profit from local information of the preceding motion phase, e.g., to seed IK from a state that is reached in at least one solution in the preceding motion phase. The latter approach, in contrast, can parallelize planning of phases and utilizes the respective interweaved Connectors to generate transit motions between close compatible solutions.

3.3 Containerization

While basic structures can be represented well with sequences of individual Stages, Task expressiveness and modularity can be increased by combining multiple Stages into Container structures, which represent complex Stages themselves. The Task Constructor framework provides three fundamental ways to containerize Stages, namely *Serial*, *Parallel*, and *Wrapper* structures. The following sections detail the semantics of these containers and their respective use-cases.

3.3.1 Serial Container

Up until now, this chapter considered Tasks as connected sequences of Stages which split a manipulation action into temporally separated motion phases. In the context of containerization, this composition can be expressed through a *Serial Container*, rephrasing the structure of previously discussed Tasks as a Generator-Interface Serial Container.

A Serial Container primarily serves to group consecutive Stage sequences into a single, semantically meaningful module — such as a complete Task, or a high-level motion like *pick object* which is composed of multiple Stages within a Task. Within the Container, every child generates its own solutions based on its respective specification, and these partial solutions are then aggregated to form complete overall solution paths for the Container.

From the perspective of Stage Interfaces, the Serial Container *maps* its interface based on the

respective Interfaces of its first and last child. As such, a Container's Interface type is not predetermined, but inferred from the composition of the Task and the Container's children.
Because child Stages feed directly into one another, the Serial Container produces *fully connected* solution paths: the end state of one child Stage becomes the start state of the next. In this way, every possible path from a start state of the first child to an end state of the last child represents an individual solution for the container as a whole. Thus, a single new child solution can also lead to multiple new Serial Container solutions if multiple diverging paths become available through the new solution. An example for this situation might be a new partial solution to place an object at a location, when the preceding part of the container already maintains multiple partial solutions to pick the object. While different geometric object grasps will usually lead to *incompatible* solution paths in this situation, the *same* grasp with *different* joint space solutions can be compatible. As both solutions to pick and place the object represent significantly different robot manipulation trajectories, with different characteristics, they are considered separate solutions in the Container.

3.3.2 Container Properties

As Serial Containers combine multiple Stages into a single semantic unit, their children frequently share common Properties. Multiple child Stages might for example consistently use the same robotic end effector, the same reference frames, or the same object identifiers. While Task Construction enforces the explicit definition of all required Properties during specification, this leads to excessive repetition when specifying identical Properties for each Stage and risks mismatching configurations upon changes.
Container-level Property definitions address these complications by allowing a Container to centralize repeated Properties. When a Stage is instantiated, it can declare that a particular Property (e.g., `joint_group`, `planner`, or `table`) should be initialized from its parent Container. As a result, any child Stage that requires the same parameter inherits the single, consistent parent definition while all parameters are still explicitly specified.
In many cases, it is not the *exact* Property name that is shared between Stages though, but the *value* which is reused in different Properties. For example, one Stage might utilize a robotic finger link identifier, e.g., `index_finger` as a target frame for an inverse kinematics computation, while another Stage uses the same identifier as part of a collision body pair for which geometric collisions should be allowed. To accommodate this use, the framework allows to initialize Properties from *different names* and, more generally, through arbitrary functions of parent properties. For example, a Container might declare a `transit_planner` Property from which the `planner` property of relevant transit Stages can be initialized — effectively sharing the planning method, its timeout configuration, and other planner settings between these Stages.
In other cases, it is also not the exact *value* that is shared between Stages, but the values are *semantically related*. For example, a Stage might require the name of an end effector to eval-

uate potential grasps, while another Stage requires the names of that end effector's finger links for kinematics computations. While it *is* possible to include the relevant mapping logic in a Task-specific implementation of the utilized Stage which uses the given value, the functional initialization mechanism instead facilitates reuse of more general Stages through a shallow interpretation layer.
These mechanisms ensure that even when common parameters share a single source, each Stage can receive the exact form or name required for its internal computation.
Lastly, more advanced Container structures can include several layers of nested Containers with parent-initialized properties. To avoid excessive re-declaration of shared Properties on each Container level, Properties will be searched upwards through the Container hierarchy when they are explicitly marked to be initialized from a parent Container and the immediate parent does not declare the required Property name.

3.3.3 Parallel Containers

In addition to the Serial Container described above which concatenates child Stage solutions, one can further consider Container structures that instead interpret multiple child Stages *in parallel*. In contrast to the Serial Container interpretation of children though, parallel Stages will not directly correspond to temporal motion phases anymore, leaving room for at least three different semantics useful in manipulation planning.

Alternatives

The first Interpretation of parallel Stages replicates the role of sets of Stage solutions on the level of the Task structure. As each solution produced by a Stage represents a valid trajectory for the corresponding manipulation phase, they already represent a natural parallel interpretation as *competing alternatives*. The *Alternatives* Container structure thus lifts this interpretation to Containers and interprets all solutions of children Stages as valid alternatives, independent of their underlying structure. Many discrete choices, such as *which object to grasp*, can be modelled within a single Serial Container structure through forwarded Properties and compatibility checking. Alternatives offer a second simpler way to model choices where different solution branches can be clearly described during Task specification. This is for example the case in *bimodal grasping* scenarios where either of two robot arms can be used to grasp an object, with no specific preference. Additionally, Container children can again be Serial Containers, thus supporting independent Stage structures and Stage parameterizations between solution branches.

Fallbacks

While the *Alternatives* Container treats all child solutions as equally valid, in some scenarios a clear ordering of preference among solution paths is more desirable. Consider, for example, the

hook task investigated by Toussaint et al. [144] where a robot must retrieve an object placed at some distance. The robot can either reach and pick the object directly —if it is within feasible range— or resort to a more complex solution of first picking an additional “hook” tool and using it to pull the object closer before picking. Using an *Alternatives* structure for such a Task explores both approaches in parallel, supporting the tool-based hooking strategy even when a simpler direct grasp might suffice.
For these situations, *Fallbacks* provide a specialized variant of the *Alternatives* Container that interprets the order of children as *unique preference*. By design, only the first child Stage that can produce a valid solution for each attempt contributes its solution as a Container solution, while subsequent children only attempt planning if the preceding child failed to produce a successful motion. Thus, if direct picking is feasible in the hook scenario and the respective motion is described as the first child of a Fallbacks container, the system will never attempt to plan a motion involving the hook, described in a second child. If, on the other hand, direct picking proves infeasible —conceptually because the object is out of the arm’s reach, technically because the involved Inverse Kinematics Stage fails to produce a solution—, the hook strategy is planned as a fallback. A similar Task concept, involving single or multi-step motions, is given in Appendix B.
Notably, Fallbacks is thus the only type of a Container that constrains *whether* a child Stage may contribute to a Container solution at all: In contrast to Serial Containers, which enable new planning attempts through forwarding Interface States of *successful* child solutions, Fallbacks enable new planning attempts of children not just through newly-received InterfaceStates, but also through *failed* child planning attempts.
The fallback mechanism applies to any Interface type with slightly different semantics:

- **Generator Fallback:** The only possibility for a Generator to fail is to produce no solution at all. Only in this case, the next child Generator is attempted.
- **Propagator Fallback:** As Propagators initiate a planning attempt for each received Interface State, fallbacks apply to each Interface State separately.
- **Connector Fallback:** For each state pair that must be connected, one child Stage attempts direct motion planning. If that fails, the next child attempts the hooking approach.

This leaves the question when to attempt planning for each Fallback child when attempts are available for multiple children. While the general structure of planning in Tasks is further addressed in section 3.6 and the abstract Task Construction structure does not specify a unique planning order, the MTC implementation for the Fallback container utilizes different strategies for different Interface types. For the Propagator Interface, planning attempts are ordered *per available Interface State*. Thus, the Propagator will attempt planning of all possible fallbacks for a single Interface State before considering the next Interface State. The Connector Interface, on the other hand, inverts this logic and always prioritizes the first child Stage in order with unattempted Interface State pairs. The rationale behind this inversion lies in the computational complexity of the different Interface fallbacks: Propagator Fallbacks typically describe different geometric approaches to continue a received Interface State and the number of attempts is linear in the number of incoming Interface States. Connector Fallbacks, on the other hand, often utilize methods with increasing complexity and an expected longer solution duration, such as initially attempting direct joint-space *point-to-point* motions, and falling back to generic sampling-based planning strategies with increasing timeouts. As the number of planning attempts can grow quadratically in the number of Interface States, these fallback planners constitute an increasingly large proportion of open computation nodes over time but are also more expensive to compute.

Merging Independent Components

A last potential interpretation of trajectories described in parallel is to consider them as *executing in parallel*, which effectively combines them to a single solution trajectory. In this interpretation, child Stages plan trajectories for disjoint sets of robot joints independent of each other, and the Container merges solutions from all children for each Container planning attempt into a new Container solution.
Naturally, this approach is only feasible if planning for different sets of joints is indeed independent. As e.g., moving two arms in parallel in the same workspace will often lead to collisions. This limits the use of this structure in manipulation planning and merged trajectories need to be validated. However, for specific motions, such as moving a left and a right arm in the same motion phase to spatially separated locations, this structure provides a useful model. For additional consideration, there are different options available to merge robotic trajectories. While synchronous resampling and synchronization of the trajectories by time warping are possible, the default MTC implementation combines trajectory waypoints followed by path-reparameterization for simplicity. If direct merging in this way proofs infeasible, a second interpretation acts as a Serial Container instead. Depending on the exact planning problems, this approach might still yield infeasible (i.e., *failure*) trajectories because each executed trajectory modifies the geometric context of subsequent trajectories.
Thus far, the *Merger* Container is the only Container structure that does not directly reference the solutions of its children, but instead generates a new solution based on the solutions of its children.

3.4 Wrappers

The *Wrapper* container constitutes a special case of the Serial and Parallel Containers. It is designed to encapsulate a single child Stage only, addressing the ability to *modify* child solutions, as mentioned above. While the main purpose of Containers is to structure the relationship between their children and their solutions, their structure also allows to receive and modify the solutions of their children before propagating them to the parent Task. In order not to convolve these different aspects beyond the capabilities of the Merger container, the Wrapper structure provides an explicit mechanism to reevaluate (and potentially modify) the solutions of a wrapped Stage.

A simple modification a Wrapper can perform is to reject solutions based on specific criteria, or *predicates*, such as to ensure assumptions on the robot's state or the environment are met in the initial world state through a *Predicate Filter*. This includes, for example, the requirement that an object which is designated to be grasped is actually available in the model state. Other direct modifications through Wrappers might include the addition of further Introspection Objects or additional Properties to the solution.

Two advanced specialized Wrapper implementations which significantly modify child solutions are detailed in the following subsections.

3.4.1 Path Reparameterization

The dynamics of the Interface States communicated between Stages pose a general challenge in the independent computation of motion phases. While these states do encompass the dynamic state of the system in terms of the velocities and accelerations of all actuated robot joints, the effects of non-zero dynamics are not considered beyond the phase boundaries and add complexity and potential failures to neighboring Stages in the sample-based Task planning approach presented here. Additionally many methods for path generation assume a *stationary* start/goal state and do not necessarily respect dynamics at the boundaries. When computing multi-phase trajectories using these methods, the final trajectory returns to resting states with zero velocities between different motion phases as illustrated in the velocity plot in Figure 3.3 (top).

To remove such artificial stops eventually, Container solutions can be reparameterized in a Wrapper using a path parameterization technique (compare chapter 2). Two additional considerations are necessary for reparameterization though. First, such an approach slightly modifies valid paths which were generated by the expert Stages, and the modified trajectories need to be validated again. Secondly, key states such as the arm pose to reach before the closing motion of an end effector during grasping should be reached exactly to ensure robust grasping in practice.

To this end, the process splits the full plan into groups of phases which are processed together, keeping their respective boundaries fixed. Crucially, these groups do not correspond to the previously discussed grouping through Serial Containers: While Containers usually group Stages around manipulation-centric aspects, such as grasping an object, the phase grouping here imposes an orthogonal association based on semantic model changes and may for example merge a motion lifting an object with a successive transit motion.

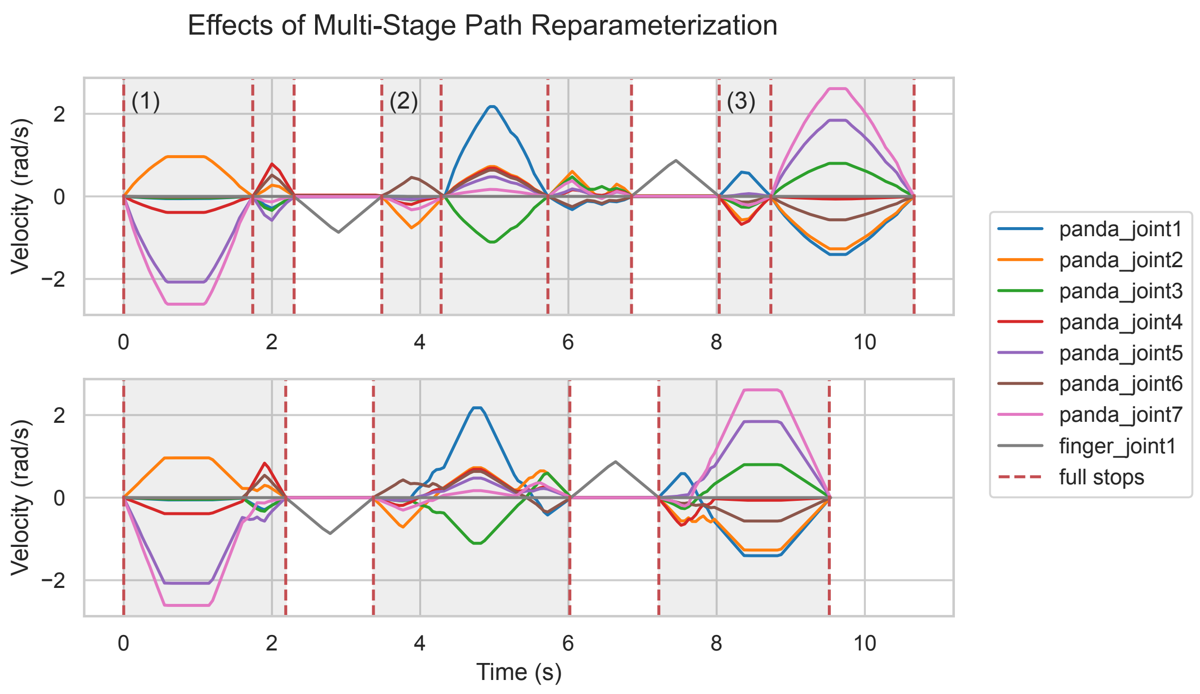


Figure 3.3 – Comparison of reparameterized velocity profiles for pick and place trajectories of a panda robot arm generated by the Task further detailed in subsection 3.9.1. The original trajectory *(top)* contains clear halting points between all generated Stage solutions. The reparameterized trajectory *(bottom)* avoids full stops within grouped motion phases, i.e., (1) reaching the object, (2) porting the grasped object, and (3) retracting after object placement, while retaining the original trajectory profile to an adjustable degree.

An example reparameterization of a multi-phase plan, generated using the previously discussed Time-Optimal Trajectory Generation approach [82], can be seen in Figure 3.3 (bottom). While the concrete parameterization depends on method parameters, prominently the application-specific maximum deviation from the original path and the resulting velocity limit curves for the trajectory, the shown processed trajectory profile explicitly avoids full stops within phase groups and exhibits typical time-optimal velocity profiles near constraints through maximum path deviation. Consequently, the resulting plan is shortened by 10 % in execution time over the originally generated plan.

3.4.2 Inverse Kinematics Wrapper

The second specialized use-case for a Wrapper structure is the dedicated computation of inverse kinematics solutions. While IK is a well-defined inverse function definition and robotics systems typically isolate it from other computation steps, its representation in Task Construction as described above does not provide the same level of isolation. As Stages always receive and reason with complete Interface States, there is no dedicated method interface to provide a logic module that computes Cartesian target poses (or other kinematic constraints) during planning. Consequently, each IK-specific Generator, as for example depicted in Figure 3.2, would convolve inverse kinematics computations with methods to generate target poses, such as grasp points.

As Wrappers are expected to yield new Container solutions based on their child's solutions, the separation of Cartesian target pose generation from world states in a child and the following joint space computation from these poses in a Wrapper is a plausible instantiation of the Wrapper structure. In combination, both modules provide an Instantaneous Generator with separated concerns. This leaves the question of how to represent the intermediate Cartesian poses as part of the child solutions and leads to the last specification of Properties mentioned in subsection 3.2.5.

Property Initialization from Interface States

As already argued for in the concept descriptions in the beginning of the chapter, it does not suffice to communicate only world model states between Stages. Additional *planning commitments* can be necessary to ensure a consistent planning process beyond individual Stages and these are explicitly modelled as Properties in the sense of Stage Properties. As such, planning commitments are a viable source for Property initialization in Stages, similar to the initialization from parent Containers described in subsection 3.3.2, and Stages can explicitly define Properties to be initialized from the Interface State (or States) that enable the respective Stage computation which uses these Properties. In contrast to all other definitions though, these initializations are explicitly dynamic and depend on the implementation of pose generators to provide the correct Interface Properties. This cannot be verified before planning without code analysis of the utilized Stages, adding complexity to the Task design.

IK Wrapper Structure

Applying this Interface initialization for the IK Wrapper yields the structure shown in Figure 3.4. In order to propose Cartesian reaching poses in a given Task context, a *Pose Generator* Stage can monitor solutions from a different phase of the Task, e.g., successful manipulation plans picking up a tool for manipulation. For each candidate (as there might be many candidate poses for each received solution context), the Generator will yield a new Stage solution, representing either the world state received or an adapted one. Each such solution additionally contains the estimated parameters required for the IK computation. Continuing the tool scenario, the Stage might infer the tool's operation frame, e.g., a screwdriver tip, specifying it as target frame for the IK computation, and a target pose for the tool tip, e.g., on a screw head.
While both properties can be used dynamically, either or both of them might also be declared statically in Task structures when their variability is not required. This is the case in regular robotics application were the robot end effector and its tool frame are fixed in a Task but the target pose varies.

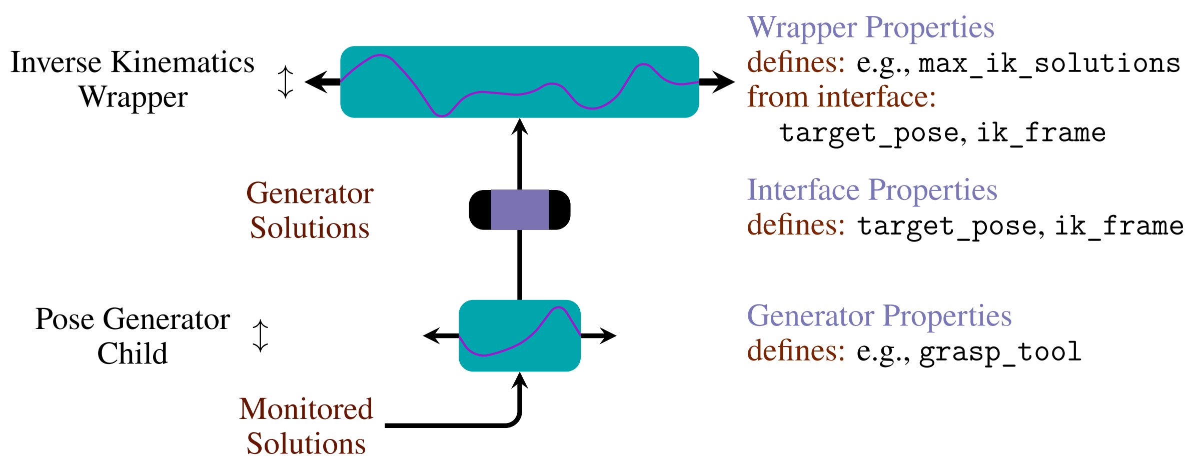


Figure 3.4 – Conceptual structure of a Generator Wrapper for Inverse Kinematics wrapping a Generator that yields compatible solutions. The Inverse Kinematics Wrapper is specified to utilize dynamic Properties for the computation defined in the child solution's Interface State Properties.

The generated solutions are then passed to the Wrapper, which extracts the required Properties for each computation and utilizes the provided world state as initial kinematic reference and environment for validating the feasibility of potential solutions. Eventually, the resulting complete world states generated by the Wrapper are passed to neighboring Stages for further planning.

Inverse Kinematics in Task Construction

Stages in Task Construction yield sets of discrete solutions representing individual robot states and this particularly affects the requirements of IK computation. As is the nature of inverse function problems in general, IK problems can yield either (1) no solutions, (2) a finite set of solutions, or (3) describe a manifold of infinite solutions. In the first case, the IK Wrapper will yield no successful solution, though potentially failures for introspection. In the second case, the Wrapper can enumerate all generated solutions separately. As a maximum relevant number of solutions can often be specified in this case, a Property can be used to limit searching for solutions per request beyond this limit.

In the third case though, manifolds of solutions can not be communicated through Interface State samples of individual world states. But these emerge when kinematic chains with 7 or more DOF are considered. A simple mechanism to handle the redundancy is discrete sampling of IK solutions[3]. Given a limit on relevant solutions and a minimal weighted joint space distance between solutions, this approach can represent the solution space effectively at the cost of discretization. An advanced alternative to this approach can exploit additional inductive biases and consider optimization of IK solutions within different discrete null-spaces in the kinematic structure, such as those investigated by Bongardt [14]. Thus, an informed IK solver can effectively represent each

[3]also used in some contexts within MoveIt under the term of *locked redundant joints*

such discrete subspace through few well-defined samples when auxiliary objectives can resolve each discrete null-space locally to few samples, e.g., by enforcing a specific elbow direction or requiring the closest joint space solution *from each discrete manifold*.

3.5 Cost Terms

With Stages generating many solutions over the course of planning, it is essential to rank them. Such a ranking is required to select the final solution from a planned Task for execution. But even more so, it is essential during the planning process itself to guide exploration towards promising solutions and delay the full computation of less promising partial paths.
There is no unique way to rank plans and any choice of ranking expresses an application-specific preference. For many motions in intelligent automation, it is reasonable to prefer plans with a shorter overall execution time for high throughput. When the world model state includes significant geometric uncertainty, or human observers should be considered, the minimal distance between the robot and obstacles in its environment can be a relevant criterion. When picking up an object in a tool-use Task, trajectories with minimal tool motion might be preferred over trajectories optimized for shortest joint path or shortest execution time, e.g., enabling longer preparatory motions without penalties before grasping.
Instantaneous Stage solutions, in contrast to trajectories, cannot be ranked by trajectory-criteria as they do not represent a motion. Still, they can be evaluated in the kinematic state they describe, for example by favoring results closer to a specific known state, or through the use of one of various *manipulability measures* [147] which estimate the flexibility of individual robot joint configurations.
To meet these varying requirements, Stages can be associated with user-specified functions to evaluate the cost of each generated solution. Cost terms, as used within the concepts of this thesis, do not inform the Stage-internal planning process — as configured Stage methods are considered to be black box expert solvers — but are used for solution ranking within the Task structure. As Stages directly compute the associated cost terms with each solution they generate, there is no need for differentiable function implementations with this approach.
The MTC implementation provides several prepared cost terms for convenience:

- *Path Length*, describing the total joint space L_1 distance spanned by a solution,
- *End Effector Distance*, describing the total Cartesian path length of a link moving with the robot during the plan,
- *Trajectory Duration* which is the full estimated execution time of the solution as determined through dynamics profiles, time parameterization, and the decision of the expert method,

- *Trajectory Clearance*, which computes the minimum — or path-aggregated minimal — distance between robot links and the environment anywhere along the trajectory,
- *Distance to Reference*, which measures the path-aggregated weighted distance of a solution to an application-specific reference state,
- the trivial *Constant* and *ConstantOffset* terms, which can be used to prioritize solutions of different parallel Stages, either considering or neglecting the cost child Stages associated with it,
- as well as a *Uniform Random* cost term with randomization seed, which effectively implements a *permute-by-sorting* variant [26] and shuffles solutions available at the same time, thus supporting an unbiased exploration of the search space locally around ranked solutions.

As user-definable functions, cost terms can be combined and weighted for further configuration. The cost of a solution s_{serial} of a Serial Container, unless explicitly specified through a different cost term, is defined as the sum of the costs of all its child solutions.

$$\text{cost}(s_{\text{serial}}) = \sum_{s_{\text{child}} \in s_{\text{serial}}} \text{cost}(s_{\text{child}})$$

Lastly, all Stage computation is initiated from Interface States, which are connected to a graph of Stage solutions, and not from complete solutions. In order to rank Interface States and provide heuristics for their further exploration, costs need to be associated to states based on the costs of the solutions they connect to. This assignment is based on cost aggregation, similar to Serial Container solutions. But the solution graph can branch and merge along the Task structure, e.g., after connecting to multiple solutions in a Connector. To represent only the best solution attempt, the cost is restricted to the *minimum sum of solution costs* along a single path in the set of directional paths $\mathcal{P}(i)$ connecting to i through the graph.
In addition, solution paths of different states will often exhibit different lengths during Task planning. This entails a bias towards *shorter* paths getting assigned lower costs, whereas *longer* paths are closer to complete Task solutions and should be preferred. To overcome this bias, the cost of an Interface State is eventually defined as a prioritized pair of values, where the first value describes the inverse of the maximum solution path length from the state and the second is the minimum cost sum over all maximum length paths connecting to it:

$$\text{cost}(i) = \left\langle \min_{p \in \mathcal{P}(i)} |p|^{-1}, \quad \min_{\substack{p \in \mathcal{P}(i) \\ \forall p' \in \mathcal{P}(i).\, |p| >= |p'|}} \sum_{s \in p} \text{cost}(s) \right\rangle$$

The ordering of Interface State costs then follows the lexicographic order of the pair and ensures long solution paths are always preferred over shorter paths with lower cost sum.

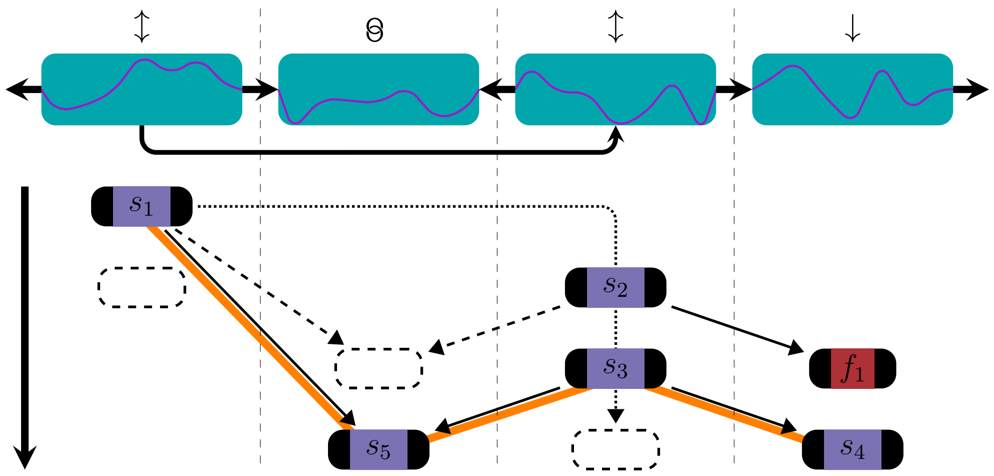


Figure 3.5 – Conceptual illustration of Task planning by graph exploration. Each Stage in the representative Task structure at the top yields solutions as planning progresses (vertical axis). Initially the first Generator computes a solution s_1 which is monitored by the second Generator and used to generate multiple solutions there s_2, s_3. Afterwards both Generators might compute additional attempts (dashed solutions). Planning failures such as f_1 effectively reduce the search space. A complete Task solution (orange) connects a solution of the first Stage through the whole Task to the end of the last Stage.

3.6 Compute Graph Exploration

Stages within a Task structure represent individual compute locations and each unit can be solved independently in a particular instance as soon as the required Interface States become available. Interpreting each compute instance as a node in a graph, and each passed Interface State (or monitored solution) that triggers new compute instances in other Stages as an edge, any Task planning attempt can thus be interpreted as a directed acyclic graph, with depth bounded by the number of Stages in the Task. In contrast to the graph depth, its width is not always bounded as Stages per se are unrestricted in the number of solutions they generate.

During Task planning this compute graph expands through each computation, similar to traditional search trees. But in contrast to regular graph search, a Task solution is *not* defined as a walk from a root node to a terminal node. Instead, there can be many root nodes, as any solution from a non-monitoring Generator constitutes its own root. The Task solution is defined as a connected path of solutions from the start of its first child Stage to the end of its last child Stage. This progress of Task Planning and a complete solution are illustrated in Figure 3.5.

While the perspective of *compute graphs*, or *task graphs* (where the traditional use of *task* refers to a single compute unit in a graph), is useful to consider Task Constructor planning behavior, the graph structure in this work expands dynamically as Stages can generate multiple (or no) solutions. Existing frameworks for task graph modelling, such as TaskFlow [62], on the other hand are designed around static task definitions and thus not directly applicable models for planning.

The compute graph is updated with available compute nodes to expand at each step in the form of Stage-bound Interface States and their costs. These represent a regular search frontier and search policies for node selection can be defined to guide the planning process. Early Stages can generate many initial solutions which may fail to expand into a full Task solution though, but Stages may also generate an infinite number of solution candidates in each local step. Consequently, neither depth-first nor breadth-first can be meaningfully applied in this structure.

Instead, the search policy in the MTC implementation utilizes a round-robin selection of all Stages with available compute nodes, where lower-cost Interface States (and pairs of such states for Connectors) are preferred. This results in a form of iterative deepening in the search graph, which readily overcomes bad initial expansions and infinite child branching. Alternative exploration heuristics may be considered. As one example, Toussaint et al. [145] recently proposed a heuristic based on the expected computational effort required for each node, which can be used to guide the search process. While this approach was proposed independent of the Task Construction framework, it is sufficiently abstract to be applicable here as well.

As another native aspect of the compute graph structure, many compute nodes can be *pruned* over the course of Task planning, once they will never become part of a full solution path through the Task. As planning can continue both forward *and* backward for a single solution sequence, a failure on one end, e.g., to retract an end effector during a placement motion, implies that placement in this configuration is not feasible and no transit motion or approach towards this placement configuration needs to be computed. The failed planning attempt f_1 in Figure 3.5 illustrates this pruning behavior. Assuming no alternative path from s_2 to the end of the Task is possible, further computation of a connecting solution from s_2 to s_1 is not necessary.

Lastly, while the main limitation of Task Construction lies in its explicit sampling of solution trajectories, the explicit control over sampled approaches is also where its main strength can be found. The explicit specification of Task planning spaces can be tailored to the needs of the scenario. To this end, instantaneous Generators can be parameterized through Properties and individual IK computation can restrict the number of attempted solutions per considered Interface State. A noteworthy parameterization here to point out is *a single IK solution* per Interface State. Assuming regular behavior of the selected IK implementation, this parameterization implies the unique solution found by gradient-projection of the reference state, if one exists. Utilizing this strategy for a selected part of a Task effectively prunes solution paths already early on when (unwanted) large motions would become necessary to reach the current target pose.

In addition to this Stage-internal tuning, *solution commitment* can be enforced for individual Stages (or Containers) on the Task level, by restricting the maximum number of solutions this Stage might generate throughout the planning process. While this artificially prunes the search space and can even lead to planning failures for otherwise feasible Tasks, it also allows to direct resources during planning towards different parts as planning progresses. Assuming for example a Task which should rearrange multiple objects, it can be beneficial to restrict the number of considered solutions to pick and place the first object and afterwards focus on the second object.

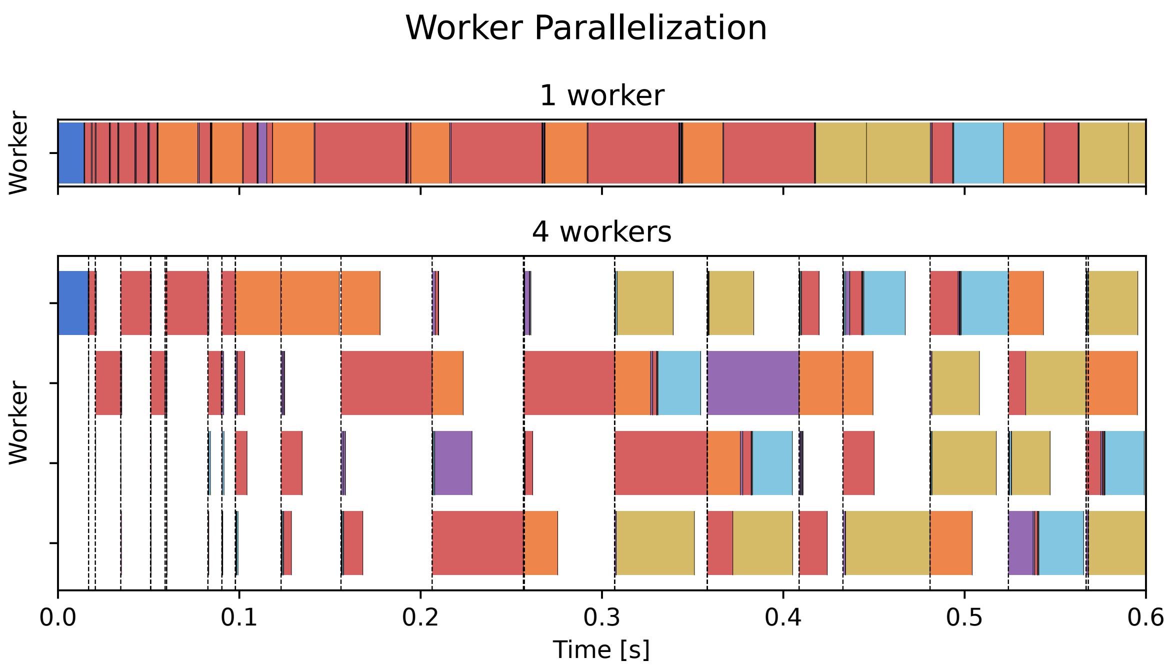


Figure 3.6 – Initial 600 ms of computation traces for sequential and parallelized Task planning. Colors indicate different Stages. *(top)* Sequential planning as described in section 3.6. *(bottom)* Round-based parallelization with four workers, dashed lines indicate synchronization points between rounds. The shown parallelized trace computed 1 300 ms in the time window with higher speedup after an initial bootstrapping phase.

3.6.1 Planning Parallelization

With the clean separation of computation units in separate Stages comes the opportunity to parallelize the planning process. In addition to the default sequential expansion of compute nodes in the MTC software implementation, a second parallel expansion strategy was implemented and is demonstrated to reduce planning times for multiple solutions in section 3.9.

Given a pool of workers, planning attempts in all Stages can be processed without dependencies between them. However, the required compute time for each Stage varies drastically and continuous computation of fast planning Stages leads to a strongly imbalanced solution set between Stages when complete solutions still need to connect to the slowest Stage as well. To balance the amount of exploration between Stages explicitly, the parallelization strategy retains the iterative deepening behavior discussed before through round-based synchronization. Within each scheduling round, the strategy distributes all Stages with open compute nodes to the available workers. As such, the upper limit on the maximum speedup within a round can be determined as the number of Stages active during planning, but severely lower due to the very different compute times between Stages.

An optional exception to the synchronization balancing was made for Connectors. As explained in subsection 3.2.2, these constitute the main bottleneck for *exhaustive* Task planning (assuming

finite branching in all Stages) and clearly dominates the later part of the planning process. As they are usually also the slowest Stages, this effectively reduces the number of active workers in the later planning stages to the number of Connectors. To facilitate better exploitation of workers in the later part of planning, it can thus be useful to increase the amount of compute nodes from Connectors per synchronization round to match the number of idle workers.
Figure 3.6 illustrates an exemplary compute trace of a sequential and 4-worker parallelized planning process for a Pick and Place scenario with 20 Stages. Initially, during the first 300 ms only few compute nodes are available in the graph and there is very little parallelization possible. The speedup in this particular scenario and phase is limited to approximately 1.4. After this point, the search frontier expanded to include more available compute nodes in different Stages and a speedup of up to 3.4 is achieved.

3.7 Task Specification

All functional planning aspects of a Task are implemented in an established programming language[4] and complete Tasks can be composed through dynamic instantiation of Stages and Containers in this style as well. The Task structure itself, as well as the Property definitions for all Stages, are static information, however, and can be mostly structured in a declarative format as well. A YAML-based [W23] implementation for Task specifications was demonstrated as a proof of concept with the MTC implementation, and is further detailed below.
A single Stage can be readily represented through a map block, as illustrated in this example:

```yaml
stage:
  name: "approach object"
  stage: [library/]MoveRelative
  group: left_arm
  direction:
    frame: l_gripper_tool_frame
    vector: [1.0, 0.0, 0.0]
  distance:
    min: 0.01
    max: 0.05
  planner:
    CartesianPath:
      step_size: 0.005
```

Two basic keys are required for each Stage: a `name` key, which provides a human-readable reference for the Stage, and a `stage` key to reference the functional software implementation through a plugin mechanism. As MTC is designed for ROS, plugins from custom libraries can be referenced through an optional prefix and can be loaded at runtime through ROS's `class_loader` mechanism. Additional parameters are specific to the respective Stage, and interpreted by the

[4]MTC utilizes mostly C++ with Python Task instantiations also supporting Python.

Stage implementation. In the example, the properties specify a relative motion of the `left_arm` MoveIt joint model group towards the positive x-axis of the `l_gripper_tool_frame` reference frame, with a distance of at least 1 cm and ideally 5 cm.

Additionally, as described in subsection 3.2.6, in some cases motion specification and planner implementation can be decoupled and the `planner` key is reserved in the relevant Stages `MoveTo` and `MoveRelative` to describe the applied planning method and its parameters, in this case describing a Cartesian path solver with a step size of 5 mm.

Sequences of Stages are simply represented as a list of Stage maps. The YAML specification also readily generalizes to container structures by introducing another dedicated key `stages` (and `wrapped` for Wrappers) which contain such lists. A fallback structure with different solving strategies to reach a pre-specified end effector pose goal might thus be described as follows:

```yaml
name: "move to goal pose"
stage: Fallbacks
stages:
  - name: "Cartesian linear"
    stage: MoveTo
    planner: pilz_industrial_motion_planner:LIN
    group: left_arm
    goal: &goal
      frame: base_link
      position: [0.5, 0.15, 0.5]
      orientation: [0.0, 0.0, 3.14159]
  - name: "point-to-point joint space"
    stage: MoveTo
    planner: pilz_industrial_motion_planner:PTP
    group: left_arm
    goal: *goal
  - name: "sampling-based planning"
    stage: MoveTo
    planner: ompl:RRTConnect
    group: left_arm
    goal: *goal
```

As used above, YAML anchors (`&goal`) and aliases (`*goal`) can avoid verbose repetition of the goal specification [W22]. The thus shortened structure nevertheless contains redundancy and the repeated properties can be considered as part of the fallback container. To condense parameters into the container, subsection 3.3.2 introduced alternative property initialization from parent containers and the following example specification uses a novel syntax `<PARENT:property>` to reference parent properties.

```yaml
name: "move to goal pose"
stage: Fallbacks
group: left_arm
target_pose:
  frame: base_link
  position: [0.62, 0.15, 0.57]
  orientation: [0.0, 0.0, 1.5708]
stages:
  - name: "Cartesian linear"
    stage: MoveTo
    planner: pilz_industrial_motion_planner:LIN
    group: <PARENT>
    goal: <PARENT:target_pose>
  - ...
```

Property initialization from interfaces (subsection 3.4.2) is similarly supported through the aligned syntax `<INTERFACE:property>`. Property initialization in Task Construction is not strictly limited to properties with different names, but as described in subsection 3.3.2 actually supports *arbitrary functions* based on other Properties. While this is possible to implement in addition to the custom notation through another scheme of referencing user-provided functions, such as, `group: <INTERFACE:arm_of(gripper)>`, this is not implemented in the MTC prototype and would add substantial complexity to the specification beyond the YAML definition, as it introduces another code referencing scheme beyond Stages.

The notion of container properties becomes even more prominent in the declarative specification, as it allows to fully specify the Task structure, but forward the necessary input parameters of a Task, e.g., *which object to grasp*, *how many Task solutions to consider*, *in what order to manipulate objects*, to the Task's top-level container, where they can be readily specified after a Task is instantiated from the specification.

A last essential addition that is required for full Task specification is the definition of Monitors. As Generators often rely on the solutions of specific other Stages, they need to refer to them in the specification. While YAML anchors might be exploited as absolute references for this purpose, they are explicitly not part of the semantics in a YAML specification, but an implementation detail of the parser. Instead, the specification supports relative local referencing of Stages through a Unix-style path syntax in a new property `monitors:`. Using "`..`", to raise to the parent container scope and the Stage names as references, an example specification looks like `monitors: "../attach cube"`. This style of local referencing supports modular definition such that the same specification can be reused (and YAML-referenced) in multiple parts of a specification.

An example of a complete Task specification for a custom manipulation Task can be found in Appendix B.

3.8 Task Execution

The explicit focus of Task Construction lies in the design of the Task structure and the planning process. Nevertheless, the framework generates robot trajectories (and expected changes to the semantic world model) that are meant to actuate a physical robot. Trajectories can be adjusted in detail in each Stage through the Task structure to adhere to hardware requirements, just as robotics engineers can traditionally adapt individual motions. Executing solutions online requires the initial Stage of the Task to be the *Current State* described in subsection 3.2.4, as execution prominently relies on trajectory following controllers starting from the initial waypoint of a solution. As planning progresses, applications receive feedback on solutions and can decide to stop planning and forward the lowest-cost solution (or any other) to the lower-level controllers for immediate execution. When planning approaches are utilized for online planning and execution, the open question remains at which point the additional waiting time required for a better solution does not justify the potential improvement in execution quality. In practice, it is often beneficial to plan beyond the first solution before starting execution, but the decision is application-specific and also relies heavily on the heuristic effect of the utilized cost terms.

Another aspect of robotic execution for manipulation is that different motion phases often require different low-level controller behavior. While a transit motion will usually rely on position control and a well-calibrated trajectory-following implementation, an interaction motion which for example should press a button, requires Cartesian force or Impedance control. To this end the motion phase specification of MTC Tasks can be annotated with controller parameters, such that the correct controller for each phase can be selected during execution based on the Stage specification. This approach was demonstrated in two external Task scenarios [118, 132].

An orthogonal approach to direct execution of solutions is to focus on reactive policy training instead and utilizes imitation learning approaches. In such a paradigm, online policies can be trained based on large sets of offline solutions used as expert demonstrations [22, 36].

Except for this last approach, a major concern for any system that plans complete motions before execution is the handling of execution failures. In practical applications, there are various potential failure sources, from mechanical and electric safety of the system, over assumed passivity of the system and Cartesian force limitations, insufficient calibration accuracy, to safe behavioral responses on human interruption or unexpected changes in the environment. Depending on the concrete failure, relevant resolution strategies can include state machine and behavior tree-based [25] flow control, plan repair and replanning [58], online system identification [1, 134], failure analysis based on language models [91], and others. In contrast to single-trajectory execution, the phase decomposition of solutions with explicit low-level controller references and the involved model changes applied to the monitored world state provide additional structure that can be utilized during failure handling. In general, the exact handling of failures constitutes a heterogenous variety of approaches which clearly extends beyond the scope of the Task Constructor framework and individual mechanisms are beyond the scope of this work.

3.9 Exemplary Task Applications

Task Construction provides building blocks for custom Task structures following different objectives. To illustrate multiple use cases, the following sections present three concrete Tasks implemented through the MTC implementation and evaluate them under different aspects. All planning-time evaluations discussed in this section were performed on an AMD Ryzen PRO 5965WX CPU Desktop system.
The presented Tasks were chosen and designed to demonstrate different aspects of the framework with different analyses. In multiple independent works, researchers presented advanced integration scenarios for Tasks with longer time horizons, e.g., for food preparation [132] and laboratory automation [118]. With increasing time horizon of manipulation Tasks the number of potential solutions grow exponentially with the number of concatenated "subtasks" and the heuristics and exploration techniques discussed in sections 3.5 and 3.6 gain prominence to explore relevant Task solution spaces.

3.9.1 Pick & Place

Following the original release of the MTC framework, it was adopted as the default system for MoveIt's pick and place planning stack with an extended Pick & Place Task structure. Many industry applications for adaptive robotics are fundamentally composed of object picking followed by either an object drop in a specific Cartesian location (*"pick and drop"*) or a more fine-grained requirement to place the object in specific poses in the environment. Accordingly, this Task specification represents a wide range of applications and was often requested by the MoveIt user community.

Task Visualization An earlier implementation of pick and place functionality was provided as a parameterized black-box C++ library interface [W11]. As its implementation constitutes a monolithic pipeline design, it could not reasonably be extended to other use cases. Additionally, while demonstrated in integration scenarios with a PR2 robot, most user feedback pointed out missing introspection facilities and attempts to integrate it with different robotic setups suffered from unguided parameter choices made by the user.
In contrast, an exemplary application of the pick and place Task implemented through Task Construction is shown in Figure 3.7. The scenario is based on a Franka Emika Panda robot picking up a cylindrical object from a table in front, moving it to a different location, and placing it on the table again. Any implementation for a pick and place scenario must provide a method to compute grasp candidates for the target object. As the target object in the demonstration example is a geometric cylinder and the scenario is meant to provide a simple demonstration, grasps are sampled uniformly around the object's vertical axis, following a prototype reference grasp. In general, any grasp generation method can be applied here and more elaborate generators based

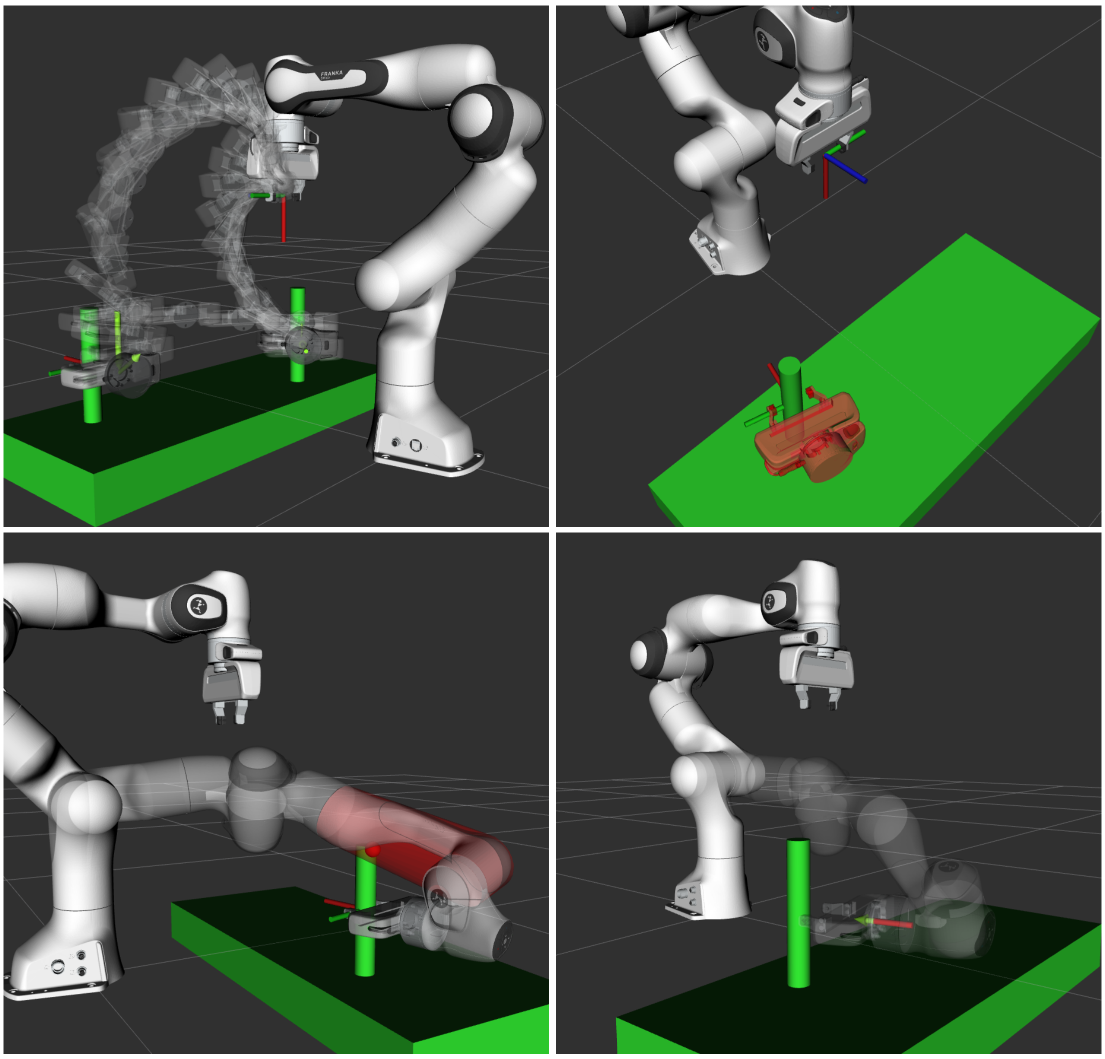


Figure 3.7 – Task application for MoveIt's default pick and place pipeline modeled in MTC. *(top left)* A successfully planned trajectory, shown as partial waypoints of the end effector for the generated trajectory. *(others)* Failure introspection: Illustrations of different representative Stage failures during Task planning. Further explanations are provided in the text.

on DexNex 4 [94] and GPD [113] were demonstrated with MTC Tasks separately [W5].
Task solutions for the scenario are visualized in RViz [W17]. The top left part of the figure shows a successful Task plan. While only the end effector trajectory is shown in the figure for better visibility, usually the full joint trajectory is inspected over time. During dynamic inspection, respective motion phases are highlighted and phase-specific introspection information, such as approach and retract directions and the Cartesian end effector pose, are displayed.
In addition to inspecting successful Task solutions, succeeded and failed attempts for each Stage can be investigated separately with introspection support. The top right part shows a failed attempt to find a valid IK solution for a particularly inaccessible grasp attempt on the opposite side of the object, and the failure condition is visible through the additional visualization of the red end effector in the grasp attempt. The bottom left part shows a successful IK solution for a different grasp which is however in collision with the object to be grasped. Again the adequate introspection information is provided through the red visualization of the collision to indicate the failure. Lastly, the bottom right part of the figure shows a failure of the Cartesian approach trajectory (planned backwards), where the arm cannot reach far enough to provide a minimum clearing distance to the object, indicated by the approach vector being coded red for the unsolvable part of the path. These visualizations are essential for developing and adapting Tasks to new scenarios and provide guidance for adapting Property definitions. Due to the nature of MTC Properties, all relevant parameters of the pipeline, including planner parameters, frame offsets, object approaches, target objects and end effector grasp configuration can thus be adjusted to the scenario requirements.

Task Structure To illustrate the fine-grained level of detail supported in Task Construction, the complete structure of the pick and place Task is shown in Figure 3.8. For better overview, a more concise notation is used to outline Task structures in the example Tasks, leaving out individual property definitions. Individual Stages are presented in separate lines and indentations indicate the nesting of Stages in Containers. Each Stage is specified through its Interface type by symbol, the Stage name, and optionally an outgoing arrow indicating that solutions of the Stage are forwarded to the pointed-to monitoring Stage.
To elaborate on the semantics of each Stage, a single planning attempt for a successful Task solution in the given scenario is detailed below. As explained before, many such attempts are explored in the complete compute graph, and can partially affect each other.
The Task structure does *not* directly start with a *Current State* Stage, but instead wraps the Stage in a predicate filter to validate applicability of the Task with regards to the current model state. For the scenario, the only checks validated are the presence of the object to be grasped in the world model and whether the end effector might already hold it. While both checks are technically not necessary as the Task will fail planning if the object is not present or the gripper is currently holding it, these sanity checks are included to provide more informative failure messages in these cases. After this start Stage, the robot's gripper will be opened.

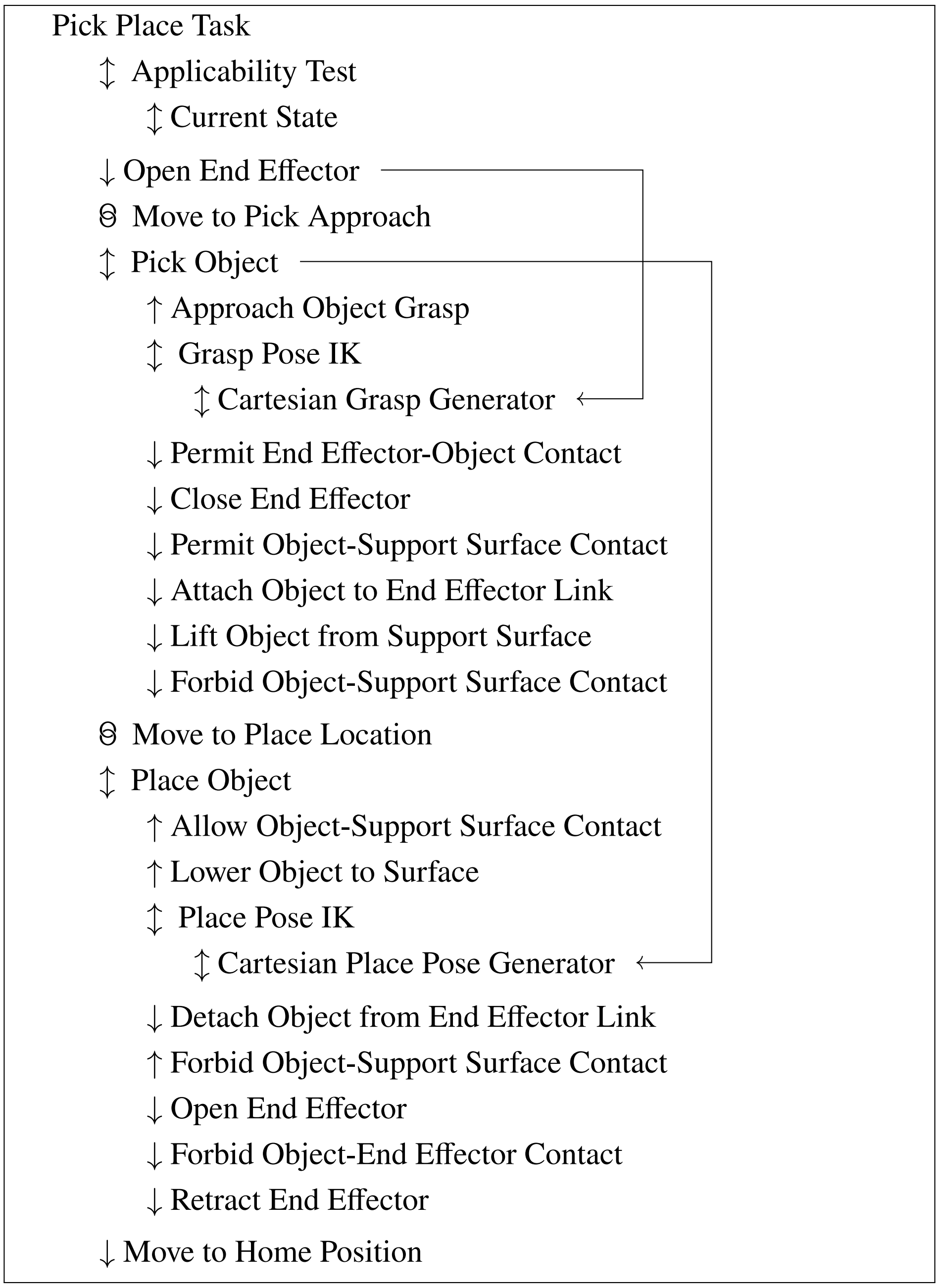

Pick Place Task
↕ Applicability Test
↕ Current State
↓ Open End Effector
Move to Pick Approach
↕ Pick Object
↑ Approach Object Grasp
↕ Grasp Pose IK
↕ Cartesian Grasp Generator
↓ Permit End Effector-Object Contact
↓ Close End Effector
↓ Permit Object-Support Surface Contact
↓ Attach Object to End Effector Link
↓ Lift Object from Support Surface
↓ Forbid Object-Support Surface Contact
Move to Place Location
↕ Place Object
↑ Allow Object-Support Surface Contact
↑ Lower Object to Surface
↕ Place Pose IK
↕ Cartesian Place Pose Generator
↓ Detach Object from End Effector Link
↑ Forbid Object-Support Surface Contact
↓ Open End Effector
↓ Forbid Object-End Effector Contact
↓ Retract End Effector
↓ Move to Home Position

Figure 3.8 – Task structure of MoveIt's Pick&Place pipeline.

As the next motion phase towards a pick approach cannot be planned by the Connector without a valid approach direction and no end states are available initially, the only Stage that can continue to plan is the *Cartesian Grasp Generator* which monitors an early Stage. As stated above, the grasp generator for this scenario samples a grasp prototype around the object's vertical axis. The cost of generated grasp proposals could be guided to specific directions if preferences are given, but is randomized for this the demonstration scenario. The grasp poses are forwarded through the `target_pose` Interface Property to its inverse kinematics Wrapper, which attempts to solve the individual kinematics requests and provides sets of up to 8 solutions (excluding additionally reported failures). As the Panda robot has 7 degrees of freedom and a continuous solution manifold for most Cartesian poses, an additional minimal solution distance of 1.0 (L_1 norm in joint space) is enforced for a representative sample set of the manifold.
Individual solutions of this *Grasp Pose IK* Stage are assigned costs based on their L_1 norm to the monitored solution from which they were generated, leading to faster exploration of grasp solutions close to the initial configuration of the robot. From this point, exploration within the *Pick Object* Container branches slightly and planning attempts for approach trajectories towards the individual robot state for a grasp are planned backwards, while the actual grasping motion of the gripper is computed forwards.
In order to support semantic collision avoidance during the manipulation, grasping the object and lifting it off the table adjust feasible collisions between model geometry. Before closing the fingers, contacts between the gripper geometry and the object are explicitly allowed. As MoveIt's world model does not consider physics interaction between the gripper and the object, but further planning should consider the object to be held in the gripper, the next two Stages rearrange the model geometry accordingly. The object is rigidly attached to the gripper link through their current relative transform and allowed to contact the table surface. Afterwards a lifting motion is planned to raise the object from the table surface, followed by another explicit change in feasible collisions, as the object is not allowed to contact the table surface during subsequent transit motions.
When forward and backward inference within the *Pick Object* Container are successful for a single grasp attempt, the Container eventually yields a complete serial solution for the (object-local) pick motion. This solution, on the one hand, provides an end state for the intermediate transit motion *Move to Pick Approach* which can now be planned through a general collision-aware motion planner (in the scenario RRT-Connect), and on the other hand, is forwarded to the monitoring *Cartesian Place Pose Generator*. The later Stage utilizes the known world state after successfully picking, and in particular the object pose inside the gripper, to generate a world model state in which the object is placed in a particular pose on the other side of the table. As the object is expected to be placed *in contact* with the table, the Generator must also adjust the allowed contacts between the object and the table to allow for this. In the general case, final object poses often vary between attempts or are sampled on unoccupied surfaces and the Stage can be readily adapted for this. This target *object* pose is again forwarded to the inverse kinematics

Wrapper which computes joint state solutions for the pose. Based on the resulting world states in which the robot holds the grasped object on the table, planning can proceed forward, adapt the semantic model again to release the object from the gripper, and retract the gripper from it. Additionally, planning from the place model state also proceeds backwards, generating a motion to approach the table. Once the local placement motion is successfully planned, the *Place Object* Container yields a serial solution for it, providing an end state for the *Move to Place Location* transit motion. This end state will not only match the monitored solution from which the place pose was generated, but also any other kinematic solution for the same grasp pose.
Lastly, the demonstration scenario includes a final transit motion back to a predefined home position, as is common in many industrial applications.
While the above description details the inference for a single Task solution, interpretation of the resulting solution—and thus its execution—instead proceeds in the reading direction of the entire Task structure Figure 3.8.

Planning Performance To investigate the planning behavior of this scenario, the default setup is evaluated with regards to planning time and relative solution cost with different numbers of target solutions and parallel workers. Conditions were set to plan for 1, 10, and 100 different Task solutions with increasing numbers of workers, as well as with the default sequential planning mode (*seq*), without parallelization overhead. For parallel evaluation, Connector computation attempts per round were kept at one and statistics were computed over 15 Task planning attempts. The analyzed solution cost for complete Task solutions in this case evaluates the total duration of the generated trajectories, while intermediate costs for the solutions of the IK Stages (not shown) used joint displacement of the solution w.r.t. the monitored state as explained above. As any analysis of absolute values concerning the planning process depends heavily on the exact model description, configured planners and the detailed Task Property definitions, further results are at best indicative for different scenarios.
A benchmark evaluation is shown in Figure 3.9. As most partial plans can be successfully extended to Task solutions, the Task can quickly find an initial solution within $200\,\mathrm{ms}$, by completing a full inference cycle as described above. Parallel solving attempts incur additional delays, partly because of data structure overhead, but predominantly because synchronization between rounds enforces the planning time of the slowest Stage per round. For the first solution in this scenario, this leads to an average time to the first solution of $350\,\mathrm{ms}$, independent of the number of parallel workers. In contrast, when planning continues to enumerate more solutions, which vary in grasp pose or the joint space poses for grasp and object release, parallelization significantly speeds up planning, enumerating 10 solutions in less than $800\,\mathrm{ms}$ (over $1.12\,\mathrm{sec}$ for the sequential case). With the investigated Task structure and parallelization approach, a maximum speedup of 1.5 is achieved enumerating 10 solutions and a maximum speedup of 2.3 for 100 solutions. As parallelization is limited by the maximum amount of computation available in each round, no significant improvement can be observed beyond four workers.

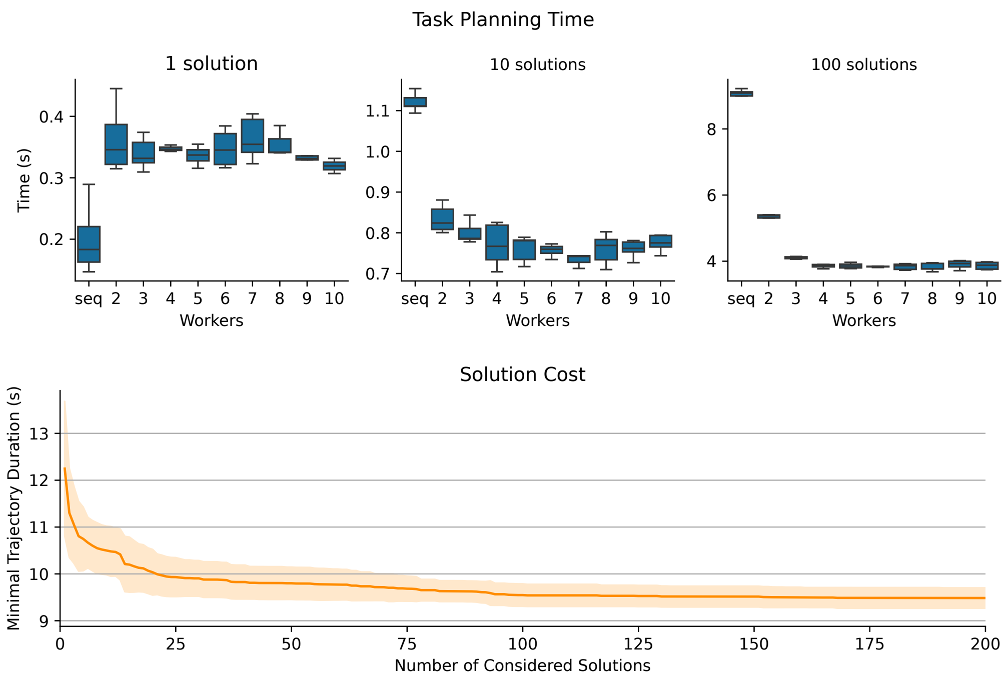


Figure 3.9 – Planning time and solution cost of the default pick and place pipeline for a Franka Emika Panda robot setup, with varying number of parallel workers. *(top)* Planning times for 1/10/100 different Task solutions. *(bottom)* Minimum trajectory duration cost among generated solutions. As more solution paths are explored, the expected best cost decreases from initially 12.25 sec to 9.5 sec.

From the perspective of solution quality, one can investigate the expected minimum cost, here trajectory duration, among the generated solutions as a function of enumerated Task solutions. As sampled Cartesian grasp attempts are randomized, the initial solution path is unlikely to consider the best grasp pose among sampled poses. Additionally, while Inverse Kinematics Stages heuristically prioritize solutions *closer* to the previous state, better complete plans might be achieved with more distant solutions whenever subsequent motions are restricted by joint limits. Indeed, the expected solution duration for the first solution evaluates to approximately 12.25 sec, but as more solutions become available, the best observed plan duration decreases to around 60% with 9.5 sec.

While the grasp pose and joint space solutions determined by the Generator describe key constraints on the plan duration, for this experiment all transit motions in the Task are planned through a simple uninformed RRT-Connect planner, which yields suboptimal paths between joint space solutions. As such, while the initial improvement of the best observed cost is due to the exploration of different grasp poses, the tail improvement of the cost curve mainly describes the stochastic minimum over solutions sampled for the involved transit motions.

Summary The described Pick and Place Task application illustrates the flexibility of Task Construction to modularly adjust details in the structure of a manipulation action and inspect in-

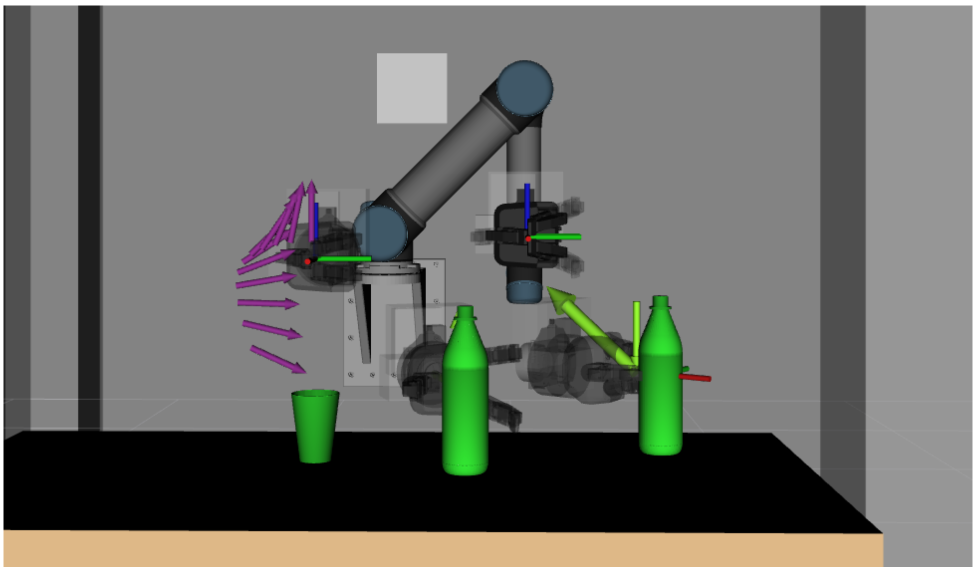


Figure 3.10 – Visualized phases of a manipulation sequence for a UR5 robot. The Task is composed of *(a)* picking up a bottle from the table (in the center), *(b)* pouring liquid from it into a cup (on the left), and *(c)* placing the bottle in a different location (on the right). Colored arrows, coordinate frames, and end effector phantoms indicate key aspects of a planned sequence, including approach and lift directions during *(a)*, bottle poses for *(a)* and *(c)*, and the upright direction of the bottle during *(b)*. Motions between picking the bottle and placing it are orientation-constraint in the bottle pose to avoid spillage.

dividual planning attempts through comprehensive introspection support. The associated performance evaluation demonstrates that, with sufficient heuristic costs or unrestricted Tasks, whole manipulation actions can be planned fast (through single or few inference runs through the complete Task structure), while the exploration of alternative solutions leaves room for large improvements on user-selected cost functions. When exploring a larger part of the planning space, the proposed parallelization scheme can yield a significant speedup, with good parallelization parameters depending on the specific Task structure.

3.9.2 Bottle Pouring

Adding complexity to the previous Task, one can introduce more unspecified degrees of freedom than only the applied grasp and kinematic place pose. This section details a scenario around a UR5-based robot setup manipulating a bottle and pouring from it, illustrated in Figure 3.10. The Task opens with the same grasp container as before, using the same horizontally-rotating grasp generator, to pick up the bottle. It proceeds with an orientation-constraint transit motion above a target cup and produces a specified Cartesian pouring motion. Following the actual pouring, the Task ends with placing the bottle in a specified position on the table, but leaves the horizontal orientation of the bottle unspecified and samples it instead. The corresponding Task structure is presented in Figure 3.11.

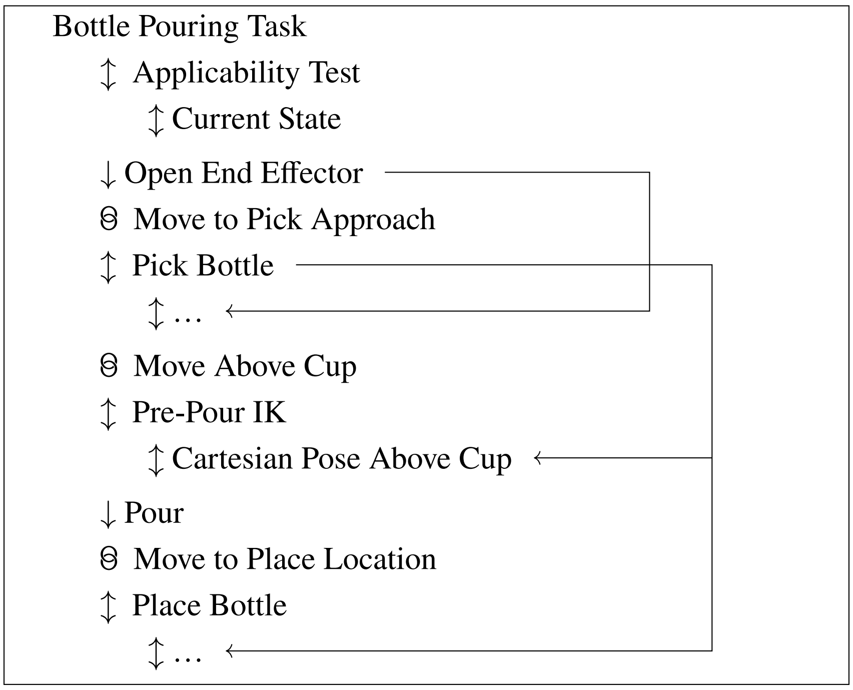


Figure 3.11 – Structure of the Bottle Pouring Task. Ellipses shorten variants of their corresponding container from the Pick Place Task structure in Figure 3.8 for brevity.

Although the illustration in Figure 3.10 suggests an intuitive solution through a straight Cartesian end effector path between the key poses, such a path does not necessarily exist in joint space. Different constraints, including joint limits, self-collisions, and the respective initial robot state render many attempts to map these paths into joint space infeasible. In practice-guided robotics applications, robotics engineers resolve these cases through workspace modifications and explicitly selected joint-space solutions — taking into account the feasibility of pouring in a later step and knowledge on potential self-collisions arising in the wrist during bottle placement. Instead, Task Construction supports the specification of different solution space complexities, and can rely on the planning process to yield successful Task solutions in complex spaces.

Kinematic Solution Space The Universal Robots UR5 robot design utilized in the scenario uses a variation on the well-studied 6-revolute joints, ortho-parallel-basis offset-wrist structure, which is known to yield up to eight analytically solvable discrete joint space solutions for the inverse kinematics problem within its core workspace [57, 146]. The UR5 system comprises a notable extension over this structure in that all six joints support an operating range of $2\,\tau$, resulting in a total of up to 512 theoretical joint space solutions for a given Cartesian pose and an equally expanded search space for trajectory planning. For the utilized setup this set is significantly reduced. Due to inherent self-collisions of the robot links and the mounted end effector, the third and fifth joint are effectively limited to a range of $1\,\tau$. Additionally, the first two joints in the chain (base pan and lift) are limited to $[-\tau/2, \tau/2]$ and $[-\tau/2, \tau/30]$, respectively. These choices avoid full revolutions of the base joint which cannot provide additional flexibility in the scenario, as well as require the lift link to point upwards. These restrictions effectively limit the set of potential joint solutions to 16, with 4 different Cartesian robot link poses. For the concrete Task instantiation, 20 grasp angles are sampled for the initial grasp attempt. The pose above the cup before pouring is given as a single Cartesian pose, which yields another 16 potential joint solutions. Eventually, the demonstration evenly samples 20 orientations for the bottle to place it on the table again. This contrasts the first exemplary Task, which defined a single place pose. These considerations are crucial, as the Task structure can consider *global* planning solutions, potentially evaluating all paths between the described joint space solutions where feasible transit motions between them can be found. An effective upper limit of solution paths for this Task on the level of discrete choices discussed above can thus be given as

$$\mathcal{U}(\mathcal{T}_{\text{pour}}) = 20 \times 2^4 \times 2^4 \times 20 \times 2^4 = 1\,638\,400.$$

Directly enumerating these alternatives is not feasible online and Task planning prunes solution paths where possible through collision checking of inverse kinematics solutions and evaluating the feasibility of Cartesian motions. Nevertheless the remaining valid space exceeds realistic online evaluation and Task planning relies on the incremental graph search for described in section 3.6 to explore many solutions through anytime search.

Pouring Path The *Pour* Stage illustrates the use of a custom engineered Cartesian motion profile as part of a Task. The pouring motion is specified through an analytical Cartesian path γ_{pour} of the bottle tip in the plane $(x, y, \theta) : \mathcal{SE}(2)$ orthogonal to a specified tilting axis, with y and $\theta = 0$ pointing up. It is parameterized to achieve a collision-free interpolation motion between the starting pose (x_s, y_s, θ_s) and an application-specific tilted pouring pose (x_p, y_p, θ_p) of the bottle lid, with the lid close to the rim of the cup.

$$\gamma_{\text{pour}}(t \in [0;1]) = \begin{pmatrix} x_s + & (t^{0.2} + (1-t) \cdot t \cdot c) & \cdot\,(x_p - x_s) \\ y_s + & t^2 & \cdot\,(y_p - y_s) \\ \theta_s + & t & \cdot\,(\theta_p - \theta_s) \end{pmatrix}$$

An example of the described path can be seen in Figure 3.10 in the tip of the purple waypoint arrows. The custom x term in this definition can explicitly overshoot depending on a constant c to avoid collisions between the bottle and the cup and the y term uses quadratic interpolation to move the tilted bottle lid closer to the cup and avoid spillage when the bottle is almost full. The complete pouring trajectory is then computed by concatenating the resulting parameterized trajectory with the reverse motion after a custom pause, which controls the pouring duration.
As demonstrated in further work [51], a data-driven alternative approach to the discussed analytical path definition can also rely on recorded human pouring demonstrations and utilize the smoothened Cartesian paths of the tracked bottle.

Additionally, *(1)* the end effector orientation above the cup before pouring and *(2)* the horizontal axis orientation around which the bottle is tilted can be considered free parameters as well. However, most orientations result in slow and inefficient joint-space trajectories, unintuitive to human observers, and the volume of valid solutions in these parameters drastically changes with slightly changing cup positions due to self-collisions [51]. For more predictable behavior during pouring and to limit further expansions of the solution space detailed above, the Task is restricted to one fixed orientation for both parameters.

Orientation-Constraint Transit As the Task describes handling of an open bottle with liquid, transit motions while the bottle is held must be subject to an orientation constraint rejecting upside-down handling of the bottle to avoid spillage. To this end, RRTConnect planning is performed in the Cartesian space of the end effector, where each sampled pose is projected to the joint space seeded from the extended sample[5]. The utilized constraint rejects states where the orientation of the bottle deviates by more that $\frac{1}{8}\tau$ from an upright world frame.
In practice, almost all *unconstrained* path planning problems between robot states are designed to have solutions, as most utilized planners are unable to proof the absence of a solution. As such, problems are considered ill-designed if their robot states live in two disjoint sets of feasible states in the joint space. As a consequence of the constraint however, planning attempts between independently generated robot states are not guaranteed to be solvable because robot states might require explicit wrist rotations which violate the constraint. In these cases, the planning Stages will always return after exhausting the full allocated planning time, slowing down the overall planning process. In the case of the experiments below, both affected Connectors, *Move Above Glass* and *Move to Place Location* use an empirically-determined explicit timeout of $500\,\mathrm{ms}$ to avoid severe impact on the planning time, while retaining solutions for almost all solvable instances.

Task Variants The following analysis discusses the effects of minor variations in Task specification and different strategies to design appropriate solution spaces. To this end, three different variations of the Task illustrated in Figure 3.10 are considered:

- *(pour left)*: In contrast to the illustration, the first variant specifies the bottle to tilt towards the left of the image, with no other modification.
- *(pour right)*: This represents the original illustration.
- *(with obstacle)*: A small block is placed between the bottle and the cup, intuitively restricting grasps and lengthening transit motions between the two.

The three variants are visualized in Figure 3.12 along with representative solutions.

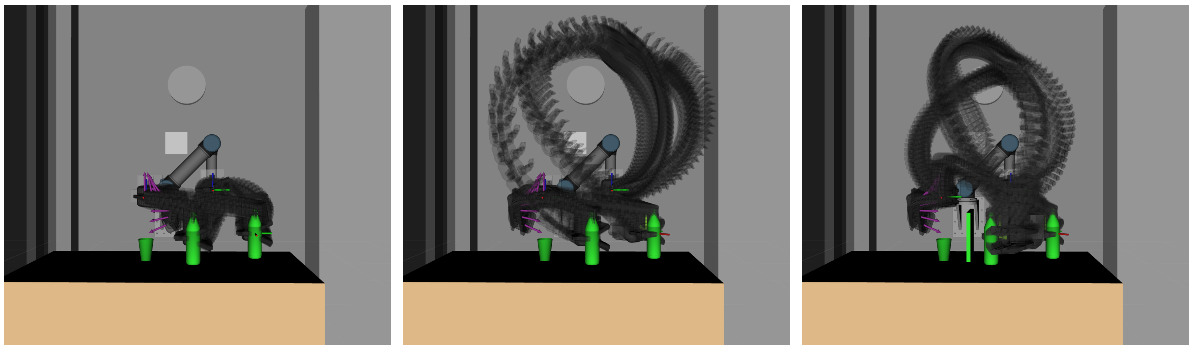


Figure 3.12 – Three visualized Task variants of the bottle pouring Task with representative solutions. Only the end effector trajectory is shown for better visibility. Results illustrate how small changes can entail the need for different global solutions. *(left) pour left*: The tilt direction for pouring is specified towards the left of the image, leading to short transit motions. *(center) pour right*: With a tilt direction towards the right, a long preparatory transit motion is necessary before grasping the bottle. *(right) with obstacle*: A small obstacle between the bottle and cup restricts feasible grasps and necessitates a global transit motion *after* grasping the bottle.

IK strategies There are various Task properties that can be adjusted to significantly affect Task planning. These include the choice of planners, maximum planning times, intermediate cost terms, number of workers, the number and sampling strategy for Grasp and Place poses, and the choice of Inverse Kinematics solutions. For this evaluation all but the last aspect are kept constant, utilizing Cartesian-space RRTConnect planning, randomized Grasp and Place pose costs, 10 workers with 5 compute nodes per Connector, and 20 evenly spaced Cartesian grasp and place poses. The following investigates different possible strategies for resolving the space of inverse kinematics solutions across the IK Stages of the Task.

The complexity of the solution space for the variants increases, as illustrated in the solution trajectories in Figure 3.12, and increasingly large solution spaces need to be considered to solve them. To start out, none of the three variants can be solved through direct Cartesian gradient following. Already the first motion to approach the bottle requires a large gripper rotation in two wrist joints to reach grasp poses without collisions with the table.

(Closest) To extend the set of solvable motions, one can instead utilize an advanced IK solver to compute the closest *feasible* joint space solution for each of the Cartesian poses. In this case the first scenario variant can be solved: A single outstretched grasp pose, from the sampled 20 poses, yields a joint solution that can be extended throughout the Task and can be combined with different place poses. The other two scenarios, however, only succeed to grasp (through the same pose), but fail to map the successive pouring motion into joint space due to a link collision in the wrist.

(All Grasps) Going one step further, one can widen the search space to consider all feasible *initial* joint space solutions for the first grasp, while still keeping the subsequent Inverse Kine-

[5]This is MoveIt's default implementation for Constraint-based sampling planning through OMPL [139].

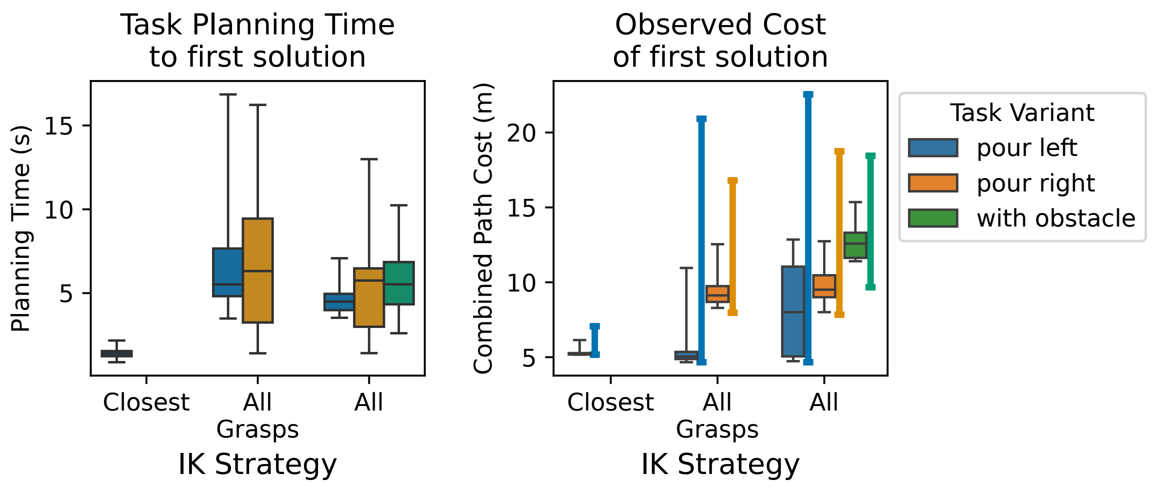


Figure 3.13 – Planning times and solution costs for the first solution in the three Task variants under different IK strategies. Whiskers indicate absolute spread of solutions. Additionally, colored bars next to each boxplot indicate the overall range of solution costs for each condition, evaluated through randomized planning of up to 1000 solutions for Tasks.

matics requests to utilize only the closest feasible joint space solution. In this way, the problem becomes a *global* search problem for *a single kinematic state*, rating grasp states as feasible if they can be extended to a full Task solution without further branching. This approach can be seen as an automatic search for feasible initial robot poses for the Task with direct continuation. Through this technique, the second scenario variant can be solved as well, with the first two variants yielding over 100 solutions through exhaustive Task planning. However, the third scenario variant remains without solutions.

(All) Whereas the introduced obstacle between the bottle and the cup appears to be a minor change, it significantly restricts the set of feasible grasp poses through new potential collisions with the gripper and wrist geometry. As a consequence only a single joint space manifold for grasping remains feasible. But solutions from this manifold cannot be extended through the *Pour* Stage when only close-by robot poses above the cup are considered. By including alternative joint space solutions for *all* inverse kinematics Stages in the Task, the third scenario can be solved by exploring the full previously described solution space. The resulting solutions require global transit motions with the bottle in hand, as illustrated in Figure 3.12 (right).

Planning Performance The evaluation of planning to the first full Task solution for the three variants under the different IK strategies is shown in Figure 3.13. Box statistics were computed over 30 planning attempts. To assess the quality of the generated solutions, costs are computed as the traveled distance of the robotic end effector. To additionally penalize long transit motions while holding the bottle, compared to long motions without the bottle before grasping, the traveled distance of the bottle during manipulation is added to the cost.

Each IK strategy supports an additional variant and only the *All* strategy can solve all three. The *Closest* strategy outperforms the others in planning time for the first *pour left* variant, as almost no combinatorial search is necessary, and it computes an initial full solution in 1.4 s on average. As no global transit is considered at all, the expected solution cost for the first solution, as well as the full range of solution costs for this condition, are also low compared to the other strategies. In contrast, the *All Grasps* strategy requires 6.5 s on average to find the first solution, as it explores the significantly larger space of joint space poses for all grasps through iterative deepening. Additionally, initial global transit motions are considered towards the grasp and the range of solution costs for this condition consequently increases dramatically, with many solutions containing long transit motions towards the grasp, which are feasible, but not required in the first variant. As the intermediate cost terms of the IK Stages still prioritize local motions between the key states, the expected solution cost for the first solution remains low. However, longer solutions can be generated as well, depending on stochastic exploration and they constitute a significant tail of the expected cost distribution for the first solution.
Trading off the increased planning time and solution cost on the first Task variant, the *All Grasps* strategy can solve the second variant *pour right* as well, which requires long preparatory motions before grasping the bottle. While planning times between the two Task variants are comparable for this strategy, the range of solution costs for the second variant is systematically higher, as the variant requires solutions with long preparatory transit motions towards the grasp.
Lastly, the *All* strategy, which considers all IK solutions for the three IK Stages of the Task, produces solutions for all variants, with comparable planning times and solutions costs increasing by variant. Again, the range of solution costs for the variant *with obstacle* systematically increases due to the necessity of global transit motions with the bottle in hand.
An obvious conclusion to drawn from the results is that single-solution exploration through a *Closest* IK strategy significantly reduces the planning time and ensures a low solution cost when the strategy supports solutions for the modelled scenario. Additionally, a trade-off can be observed between the *All Grasps* and *All* strategies: Where the former yields initial solutions with significantly lower costs in easier Task variants due to the local continuity of solutions after grasping, the latter strategy can resolve a more complex Task variant at comparable planning times without compromising on the solution quality for the *pour right* scenario. This can be attributed to the intermediate cost term heuristics which remain unchanged between the strategies.

Summary The described scenario demonstrates the use of specific Cartesian motion profiles in Tasks and investigates planning performance with an extended number of free variables. Whereas manipulation actions which require large transit motions are usually avoided in robotic setups, the described scenario variants demonstrate that such trajectories can quickly become necessary with minor changes to easily-solvable scenarios. Explicitly considering the set of supported solutions can significantly affect planning performance and the expected quality of generated solutions.

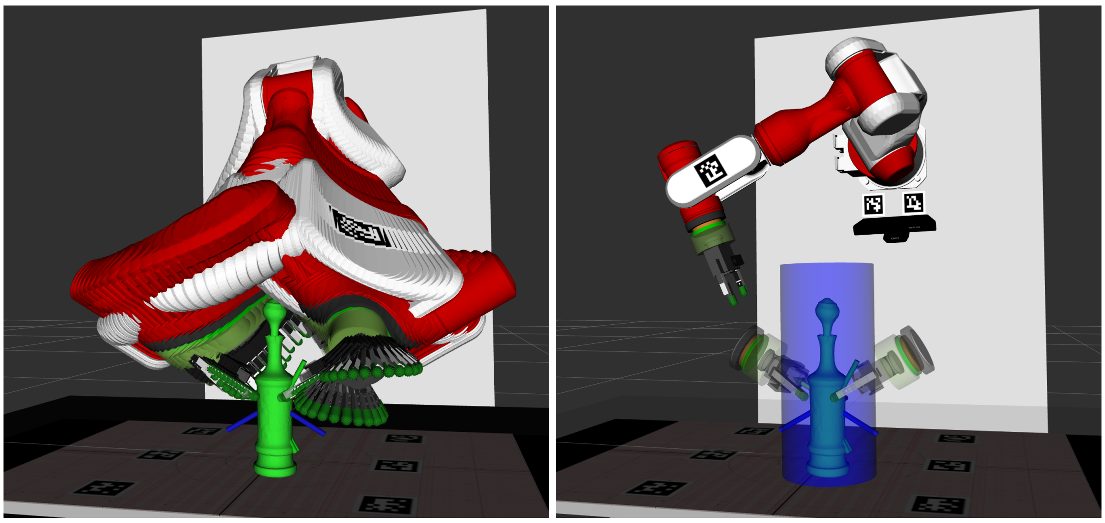


Figure 3.14 – *(left)* An example trajectory for a Mitsubishi PA10-based robot setup between two poses in close proximity to the central object with an undesirably low object clearance during transit. *(right)* Related Task visualization: Two opposing Cartesian poses are in close proximity to the object. The Task includes a Cartesian retract from one pose, a transit around the object, and a Cartesian approach to the other pose. A blue padding cylinder can be added during transit to enforce higher clearance during the phase.

3.9.3 Retract – Transit – Approach

This example represents a common motion pattern found in many domains, including industrial welding and surface blending, as well as mundane object manipulation. In such scenarios the robot regularly performs motions between different poses where an end effector is close to an object of interest. To avoid accidental interactions of the robot with the environment, such motions are often implemented to include an initial retraction motion away from the object, followed by a transit closer to the new pose and a final well-defined approach ending at the target pose. Whereas these trajectories are usually defined for known joint space regions, the presented structure encapsulates the idea in a Task structure, allowing for the automatic generation and analysis of such motions in arbitrary contexts.
Additionally, the transit phase can involve global trajectory planning with potential self- and obstacle collisions. In contrast to the precise approach/retract motions, where detailed environment geometry should be considered, the transit phase is a free-space motion and is not expected to get close to the known environment, although paths close to the environment are not a-priori excluded. Figure 3.14 (left) illustrates an exemplary transit trajectory passing in closed proximity to the central object which is feasible from the perspective of path planning.
The closest distance between robot links and the environment during a trajectory is referred to as the trajectory's *clearance*. To enforce sufficient clearance, one can either rely on optimizing trajectory planners to yield solutions above a certain clearance — trading generic definitions for computational complexity and potential local-minima —, or specify intuitive padding volumes

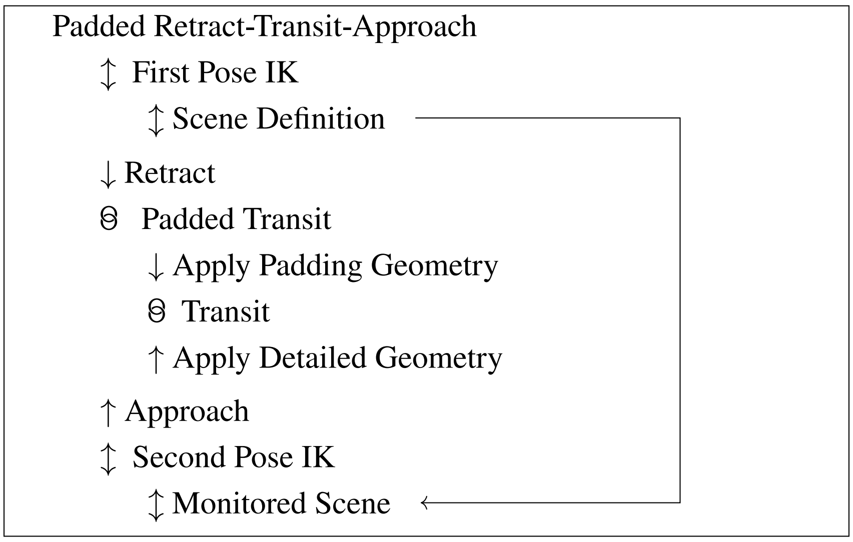


Figure 3.15 – Generic Task structure of Retract–Transit–Approach motions with padded transit.

as auxiliary collision bodies for the transit. Depending on the intended level of control, these volumes can either be specified by hand or computed, in the case of Figure 3.14 as 5 cm inflated bounding cylinders. As the added bodies are considered during state validity checks, they render trajectories which pass through them infeasible during the initial global path search, leaving planners to search the reduced feasible space. This latter approach is described in the Task structure presented in Figure 3.15 and an instantiation of the Task for a 6 DOF PA10 robot setup including a blue padding object considered only during transit is shown in Figure 3.14 (left).

Structure In contrast to the previous examples, the Task opens with an externally provided scene definition, wrapped in an inverse kinematics Stage. While tasks meant for direct actuation on the robot need to start from the *current state* of the system, the approach used here can be used to reason over solutions starting from different robot states. The IK wrapper is parameterized in a target pose, thus generating the different discrete null-space solutions as different joint space states. This is followed up by a forward-planned relative Cartesian retract motion. Mirroring these first Stages, the Task ends in a second IK Stage utilizing the same scene, and a backward-planned relative Cartesian approach motion. To connect both parts, the `Padded Transit` container is introduced. In addition to the expected connecting Stage *Transit* which employs an RRT-Connect planner, the container bookends the Stage with two instantaneous Stages to adapt the collision geometry, where `Apply Padding Geometry` replaces the detailed geometry with larger padding geometry and `Apply Detailed Geometry` reverts the change. As explained before, only *solutions* to the Stage actually revert the change when interpreted in the temporal direction, whereas the Stage is planned backwards and thus causally applies the same modification as the prior Stage to provide compatible Interface States for the transit Stage. The relevant parameters of the Task to be adapted as necessary are thus the application-specific

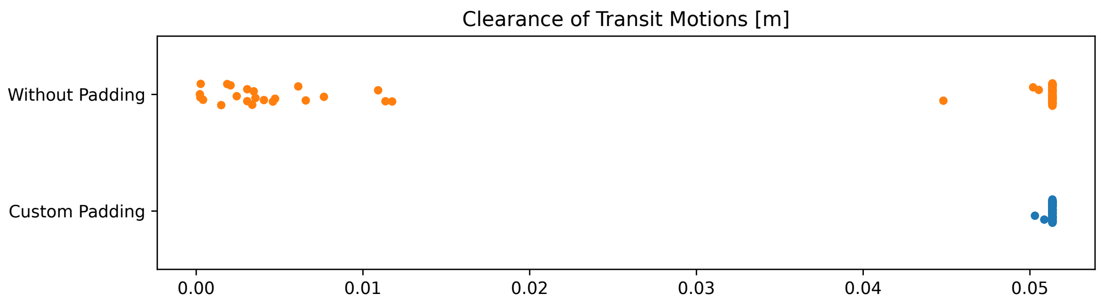


Figure 3.16 – Minimum object clearance for representative trajectories from sampling-based planning over all 96 combinations of joint space start and end transits for the Task shown in Figure 3.14. Without custom padding no clearance to the object is enforced and planners can generate solution paths in close proximity. The additional custom padding ensures a safe transit around the object with at least 5 cm clearance.

Cartesian poses, their approach/retract directions and the padding geometry. In many cases the approach poses can be specified in the end effector's frame per scenario, leaving the Cartesian poses of interest and the scene geometry as the only free parameters.

Evaluation To demonstrate the effect of the additional padding, consider the concrete scenario with fixed object and targe poses illustrated in Figure 3.14 (right). With this concrete geometry, the PA10 robot kinematic structure and the configured joint limits support 12 different joint space solutions for the first Cartesian pose and 8 for the second pose, resulting in a total of 96 potential planning queries between the different joint states. Figure 3.16 illustrates the minimum object clearance for all solutions with and without the custom padding after exhaustive Task planning. Adhering to the specification, the padded transit ensures a minimum clearance of 5 cm, while 23 % of the solution trajectories for the unpadded transit pass in close proximity of less than 1.5 cm to the object. The visible clusters of solutions with minimum clearance of 5.1 cm corresponds to the pre-determined poses after retract/before approach, indicating that stochastically many solutions move around the object with more clearance than is determined by the fixed Cartesian end effector poses.
Table 3.2 shows the planning times for the Task with and without custom padding. Although the padding cylinder slightly simplifies individual collision checks over the detailed object mesh, it also reduces the feasible space for the transit planner, requiring significantly more samples to find valid solutions in some cases.

Discrete Kinematic Evaluation of Motions Considering different initial joint states for a Cartesian pose and the multi-Stage motion to a second Cartesian pose, the Task relates different joint space solutions for the two poses. The combinatorial space can be evaluated offline for analysis. While this strategy does not apply to extended Tasks due to the exponential growth of the solution space with each variable, it can provide insights into relevant plan segments.

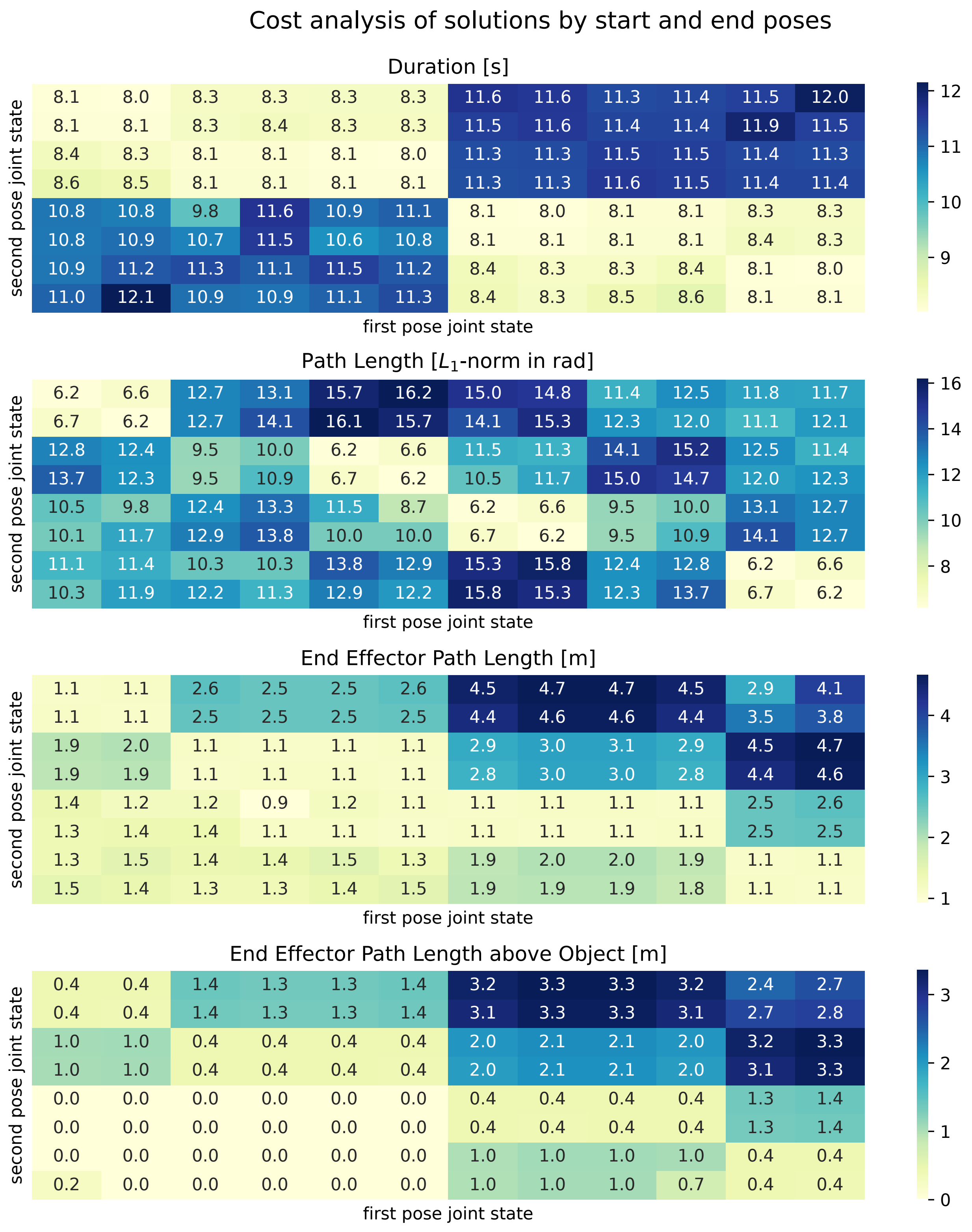


Figure 3.17 – Cost grid analysis between sampled solutions of the padded Retract–Transit–Approach Task starting and ending in the different discrete joint space poses. Different cost terms illustrate potential trade-offs.

	First Solution	Exhaustive Planning (10 workers)
Without Padding	0.154s ± 0.011	1.047s ± 0.081
Custom Padding	0.314s ± 0.082	3.385s ± 0.404

Table 3.2 – Benchmark of planning durations for the Retract–Transit–Approach Task with and without custom padding.

This evaluation relies on an uninformed sampling-based RRT-Connect planner, thus the exact solution trajectories differ between runs and are not optimized. However, much larger variations in trajectories can be observed between different IK solution pairs than between multiple invocations of sampling-based planning for individual pairs. In the following, transit motions were planned through uninformed RRT-Connect to the first solution and the best cost out of 10 attempts is considered. While this approach can only provide an upper bound for a cost-optimized path between the poses, it nonetheless provides insights into the trade-offs between different cost functions when selecting joint space targets. Depending on the Task, optimizing planners informed through the respective cost function can be used to refine estimated values. In the general case, this involves a trade-off between refined transit motions for all considered paths and a higher per-attempt computation time — leading to accumulating Stage timeouts in case the explored Task includes infeasible planning requests, such as observed in subsection 3.9.2.
Figure 3.17 illustrates exemplary analyses for multiple contradictory cost terms that can be relevant in different applications. First, it can be noted that the pairwise solution space splits into two subspaces, where traveling between these spaces takes around 40 % longer. This dichotomy is due to two distant solution types for the robot's two base joints (*S1* and *S2*) which use significantly slower velocity limits. Within each subspace, the joint space path length analysis indicates several clearly preferable solutions. Investigating the end effector path length reveals more comparable solutions. Finally, application-specific cost terms such as objectives to keep the end effector at the level of the object, can fully conflict with an objective to minimize the trajectory duration. Such competing objectives are not uncommon in practice and the analysis can help with informed decision-making.

Summary The example Task design illustrates functionality in Task Construction to model varying planning spaces throughout Task solutions, which apply to complex planning Stages. Adapted models can add guarantees to generated solutions, such as forcing clearance from relevant parts of the environment. Independently, Task structures can be used to relate different solutions for joint states to each other, allowing for offline analysis of the modelled solution space with regards to different objectives.

3.10 Whole-Stack Benchmarking

This section is based on a workshop publication [48] I presented together with David Pivin at the IROS2022 workshop for Evaluating Motion Planning Performance. The MoveIt Benchmark Suite software implementation was written by David Pivin with close supervision by me and Felix von Drigalski during a Google Summer of Code 2021 project [W15].

Beyond the type of kinematic analysis presented in the previous section, the Task Construction framework can also be used to evaluate the performance of a whole motion planning stack. In the context of software development and maintenance for a complete planning system, it is crucial to assess performance changes over time across different system configurations and between competing feature implementations, in order to identify bottlenecks and regressions.

Many approaches have been proposed in the past to assess the performance of motion planning systems. These include prominent examples, such as the *PlannerArena* [100], *Robowflex* [77], and the *MotionBenchMaker* [21], as well as specific analysis approaches, such as investigated by Liu et al. [89]. Noteworthy to single out, the *Box and Blocks Test for robotic systems* [102] even targets an established rehabilitation benchmark for assessing the manual dexterity of humans and adapts it for complete robotic systems, including motion planning, but also the kinodynamic design of the robot.

However, all but the last of these solutions focus on the integrative performance analysis of a range of motion planning tasks and thus only address a subset of the complete planning stack required for manipulation actions with a robot. Many motion planning systems utilize various modularized routines, prominently including collision checking (and more generally feasibility checking), and inverse kinematics, but also projection operators, constraint solvers, time parameterization algorithms, kinematic datastructure operations, and dynamic environment representations such as voxel grids and octrees [60], among others.

Seemingly minute changes in any of these operations, which are often provided through standalone software dependencies, can have strong effects on the complete framework performance in specific situations. While traditional unit testing and microbenchmarks, as for example available through the *Benchmark* [W3] project, can generate detailed insights into the performance of individual components, including low-level significance testing and code path warm-ups, their coverage is typically limited to artificial and reduced cases and by their very nature, in isolation from the complete system the component is used in.

The MoveIt Benchmark Suite [48] fills the resulting gap between pure motion planning benchmarks and low-level microbenchmarks specifically for the MoveIt framework. Along with several MoveIt-specific benchmark categories, including the initialization of datastructure and performance of collision checks between different object representations, the suite also prominently includes benchmarking through Tasks. As the Task structure is a modular representation of the

planning process, it can be used to evaluate the performance of the planning stack on complete manipulation actions as a whole *as well as* providing performance characteristics for many operations in isolated modules. Given a Task specification and a world model state, in a similar but extended scheme as the one used with MotionBenchMaker, Task benchmarks can plan and benchmark complete manipulation actions. At the same time, they maintain Stage-wise statistics for the planning time as well as the cost of yielded Stage solutions.
Figure 3.18 illustrates exemplary measurements generated in a Task benchmark. These characterize the performance of the complete planning stack and can be compared between versions, where data is either inspected visually, or automatically evaluated through statistical significance testing. As benchmark statistics in software are likely to change distribution properties under logic changes, e.g., spread of runtime measurements might increase for partial failures in caching mechanisms, non-parametric tests, such as the Mann–Whitney U test are usually preferred over parametric significance tests [W3]. For the purpose of framework benchmarking, the exact values of these statistics are not as important as the relative changes between different configurations or versions. Indeed, it actually enhances the representational value of assessed modules if Stages are explicitly parameterized to utilize different planner configurations and other module settings as more code paths are relevant between different modules.
The MoveIt Benchmark Suite has been effectively used to evaluate performance differences of different build configurations and versions of the MoveIt framework and supports the investigation of introduced regressions.

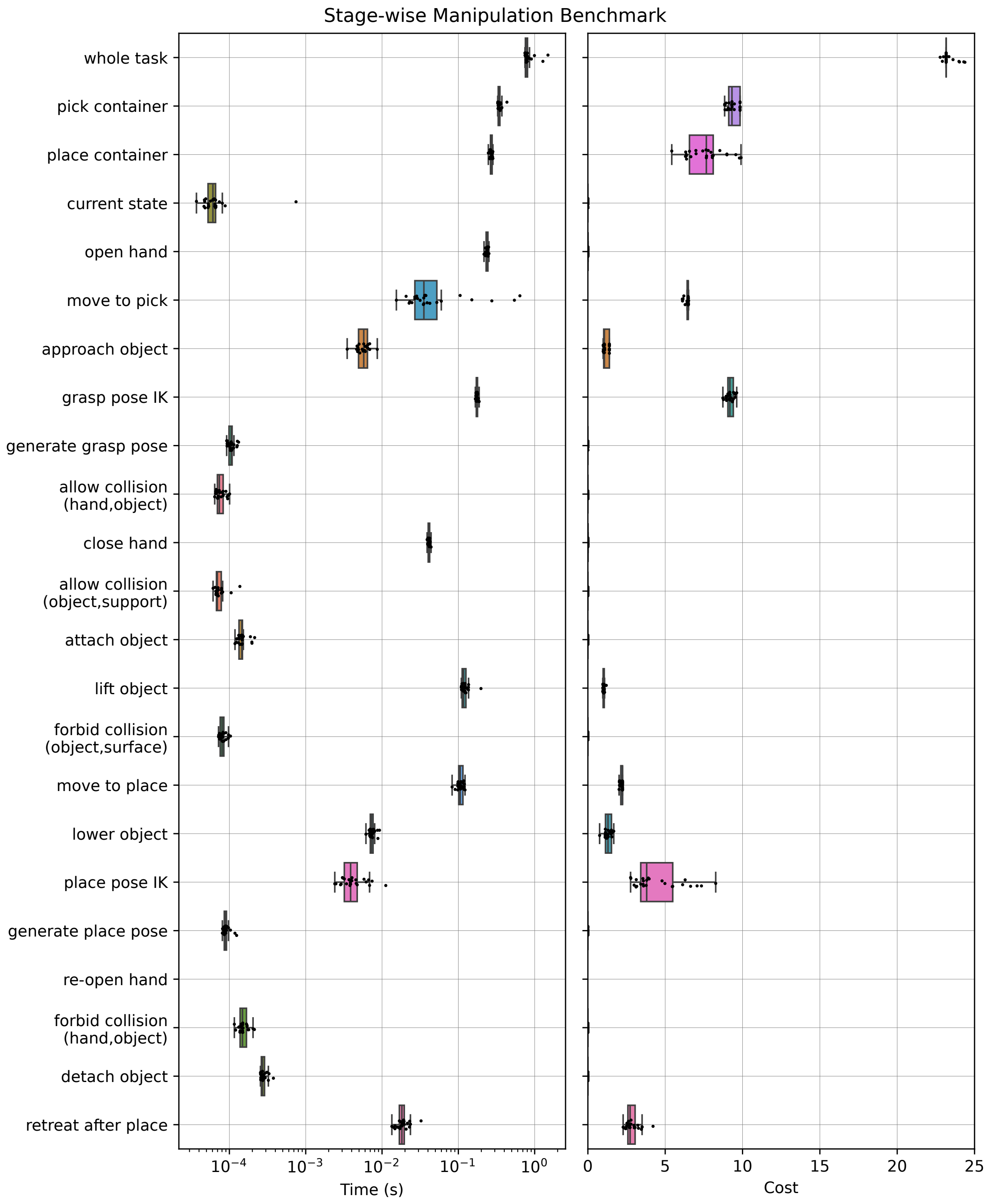


Figure 3.18 – Planning characteristics for a Pick and Place Task similar to subsection 3.9.1. Plotted statistics are computed over 25 planning attempts to the first solution with planning time and reported solution costs of all involved Stages.

Chapter 4

Validity-Constraint Motion Primitive Exploration

This section is based on the conference publication "Pluck and Play: Self-supervised Exploration of Chordophones for Robotic Playing" [47] presented at the IEEE International Conference on Robotics and Automation in 2024.

4.1 Physical Self-Supervised Exploration

Based on the framework detailed in the third chapter, it is possible to design complete manipulation sequences where different parts can be easily specified. For many applications a core challenge that remains is the design and implementation of key stages, which are typically difficult to engineer by hand. In the context of object bin picking, for example, the crucial aspect to select a successful grasp pose in the workspace has seen decades of research [13, 79]. To this end, data driven methods provide good solutions, but require significant amounts of data to be trained. A common approach to generate this data is the procedural analysis of offline datasets [87], optionally augmented by synthetic scenes [94]. While the obvious benefit of such an approach is the automatic generation of large datasets through parallel computation, the data often does not correctly represent the real-world setting and integration efforts on the physical system can suffer from sim-to-real artifacts.
An alternative approach to data collection is the paradigm of self-supervised exploration where data is generated through motion attempts on the physical robot. Pioneering such an approach at a larger scale, Pinto et al. [117] demonstrate a software system for a Baxter robot in a table-top scenario. The robot autonomously explored $50\,000$ top-grasp attempts on various objects, labeling each attempt based on whether the robot could lift the object.
In a reinforcement learning paradigm, Zeng et al. [163] demonstrate a system to push and sub-

sequently grasp objects on a tabletop, demonstrating combined action parameter learning.
Berscheid [9] further extends these approaches to include autonomous training of grasp poses in $SE(3)$ and utilizes the trained models to subsequently collect data for complete pick and place sequences. They prominently generalize the concept of *Motion Primitive* to complete manipulation sequences parameterized in few trained parameters. While this general scope suits their examples, this thesis focuses the notion of *Motion Primitive* on the local motion generation of a single motion, which might be modularly integrated into larger sequences.
Lastly, Schneider et al. [127] focus on dynamic motions involving ball balancing on a robotic end effector. They could demonstrate drastic reductions in training time for a reinforcement learning paradigm by including an *Active Exploration* strategy: Instead of optimizing motions to maximize only the reward given their current model, they include an additional information gain term, steering the system behavior to explore actions which are uncertain in the current model.
A common aspect of all mentioned self-exploration approaches is their design of an *inherently safe* action space that can be explored almost without human intervention. However, such a design is not always feasible. Focusing on a multi-modal scenario where such a definition is not feasible, this chapter investigates autonomous exploration of a fragile musical instrument, combining the self-supervised exploration paradigm with methods from the field of safe active learning.

4.1.1 Scenario

Numerous robotic setups were demonstrated playing musical instruments, such as piano, guitar, and drums [16]. Approaches for this tend to focus on the higher-level programming/composing layer, whereas the actual actuation happens through precise servo motors or custom-built special-purpose hardware [107]. Other approaches augment musical instruments to enhance human playing or track the system's physical state [112]. The emphasis is often put on intelligent musical behavior with the declared vision for competent robotic musicians to interact with others using musical theory as a common ground [16, 151].
Stringed instruments, *chordophones*, in particular are challenging for robots: they afford many different and expressive playing techniques, triggered through slight variations of playing motions and fully pre-programming motions for varying compositions is not realistic, as discussed below.
The scenario in this chapter considers a Guzheng (古筝) depicted in Figure 4.1, a traditional Chinese chordophone from the family of zithers. The predominant variant features 21 strings in pentatonic tuning, with the strings spaced approximately 1.5 cm apart. Human performers usually manipulate the strings through four plectra taped to all except the little finger of each hand [39, 148]. Figure 4.2 illustrates the mounted plectra on the anthropomorphic hand of the utilized robot.

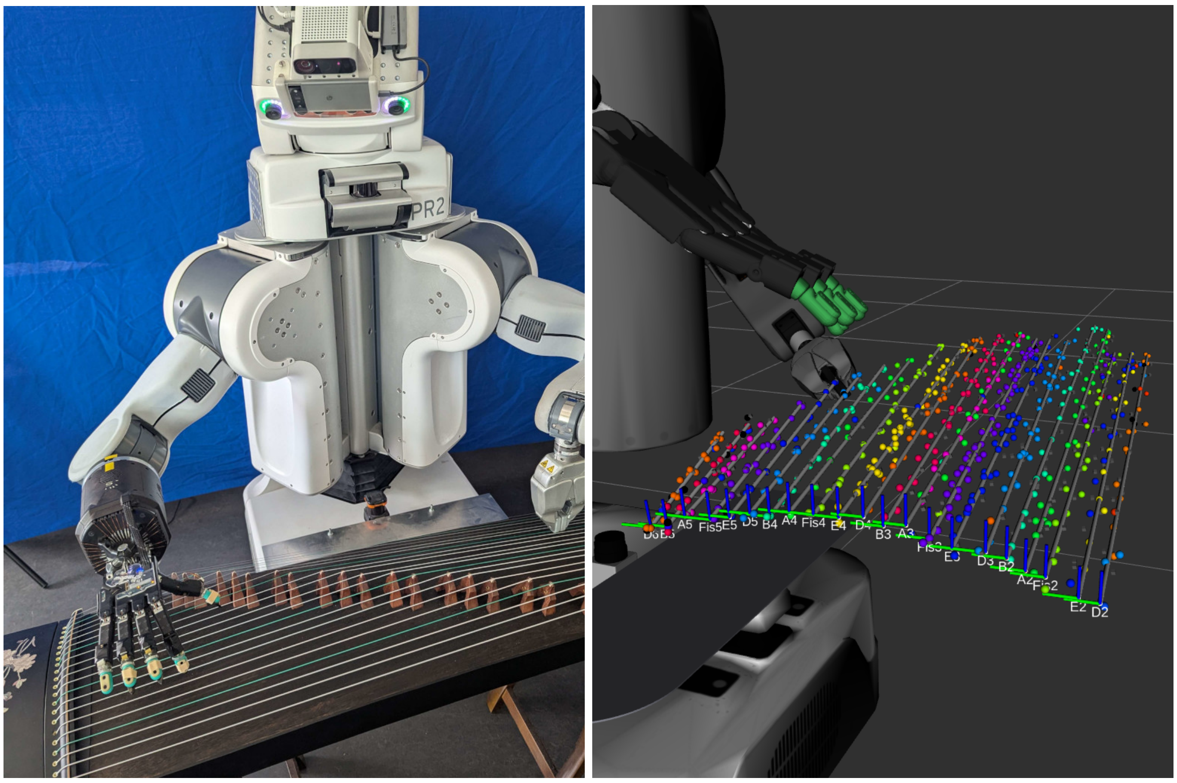


Figure 4.1 – *(left)* Modified PR2 robot with Shadow Dexterous Hand and SynTouch BioTac fingertip sensors in front of a Chinese Guzheng. *(right)* Projected audio note onsets (spheres) and reconstructed string representation (lines and coordinate systems) generated by self-supervised geometry exploration. Colors encode the musical note recognized from the audio channel.

Compared to most other chordophones, which usually feature strings running in parallel and equidistant on a plane, the Guzheng features a more complicated geometry. The strings are mounted on a wooden body with a doubly curved surface and run over bridges at different distances from the instrument head, changing the height and angle of each string. As the bridges are mobile and moved during tuning, the strings are not positioned identically in different instances. Different artists also prefer different levels of string tension, which affects the placement of the bridges as well as the produced sound [31].

Lastly, the scenario utilizes the modified PR2 robot also shown in Figure 4.1 (left) which features spring-compensated shoulder joints and belt transmissions behind the position sensors of the arm joints. This kinematic structure reduces the force necessary to move the joints but impedes precise position control and calibration. During operation, significant systematic end effector path errors above $5\,\mathrm{mm}$ are common. As these can strongly impede the quality of motion primitive models, the learning process must be able to incorporate them.

Overall, this chapter presents a three-stage approach to actively explore the instrument:

- Assuming an initial estimate of the string positions, the first stage estimates a stable environment model of the instrument geometry through repeated heuristic motion attempts.

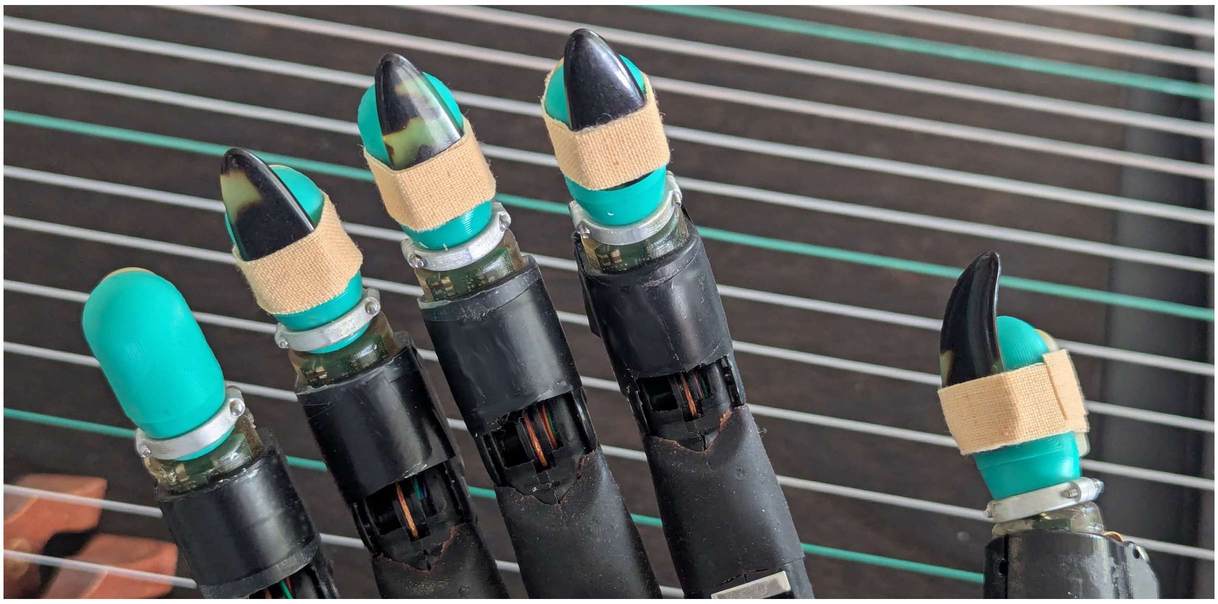


Figure 4.2 – Close-up view of the Shadow Hand fingertips with SynTouch BioTac tactile sensors (turquoise), prepared with traditional fingernail plectra for Guzheng playing. Each plectrum is taped to the fingertip to reach several millimeters beyond the tip with stable contact to the sensor surface. The thumb is taped with a regular curved thumb plectrum for a more relaxed hand posture during playing; little finger without a plectrum, as traditional in Guzheng playing.

- Based on the refined model, active exploration strategies are presented to estimate the audio response of plucking motions based on audio, tactile, and proprioceptive feedback by growing trust regions around valid motion parameters. A *Gaussian Process* based model formulation implicitly accounts for controller behavior and physical string characteristics.
- Lastly, an inverse model formulation can utilize the explored models to play intended notes with specified loudness.

4.2 Multimodal Manipulation of Chordophones

4.2.1 Background

Robot Musicians Kapur [72] discusses the early history of robotic instrument playing, covering robots playing standard musical instruments as well as instruments explicitly designed for robotic actuation. Similarly, Sobh and Wang [133] discuss the development of a full band of robot musicians that perform on real musical instruments.
Several works have concentrated on piano playing. Zhang et al. [164] describe both the "robotic" and "musical" playing of (entry-level) piano pieces with the highly complex Anatomically Correct Testbed hand. A much simpler robot, designed to mimic the soft skeleton and elastic properties of a human finger, was presented more recently by Hughes et al. [64].

Expressive Play "Musical" playing aims to invoke a large variety of emotions and impressions through the multifarious use of playing techniques. The scenario investigated here focuses on exploring the dynamic range of plucking strings, but expressive play also encompasses various other techniques such as damping and modulation of notes as described by Murphy et al. [107].

Guzheng Despite the important role of the Guzheng in Chinese musical tradition, only comparatively few scientific studies of the instrument and its playing style have been published. A vibroacoustic study of the Guzheng soundboard was presented by Xiaowei et al. [159]. Zhang et al. [165] presents a detailed modal sound synthesis model of the Guzheng. Recent work by Mazurek et al. [98] describes an optical sensor setup to track string plucking and vibration on the Guzheng. The harp is generally considered the closest Western-style equivalent to the Guzheng. The properties of a harp-plucking robotic finger were studied by Chadefaux et al. [20], with detailed motion and vibration analysis as a function of fingertip material, shape, and reference trajectory.

Audio Analysis In the context of this chapter, algorithms are needed for automatic and robust detection of note onset and note pitch as presented for example by Duxbury and Zhou [23, 167]. As with other stringed musical instruments, this is complicated by the transient vibration patterns of the instrument until a stable vibration with distinct spectrum is reached, as discussed by Mounir [103, 104]. Wang and Cao [150] present an algorithm to match Guzheng audio data with a musical score based on dynamic time warping.

Deep Learning Beyond the earlier analytical and physics-based works, deep-learning systems demonstrate advanced performance. For example, Byambatsogt et al. [18] presented a deep neural network for automatic guitar chord detection and classification. The network was pre-trained on a human-played but highly biased dataset [158] and a self-supervised exploration setup was constructed to collect a larger augmented robotic dataset including all possible chords.

Reinforcement Learning for Instrument Playing Xu et al. [160] proposed a Bullet-based simulation setup with an Allegro hand model for learning to play short sequences of notes on a piano from music notation with fingering. The trained agents can control the model in a four-dimensional action space to press multiple keys with specific attack velocities from online input. Zakka et al. [162] presented a MuJoCo-based piano simulation with two Shadow Dexterous Hand models. Agents need to be trained on individual music scores with fingering but demonstrate impressive advanced skills in playing well-known Western piano compositions. Neither work investigates a physical instrument, though sim-to-real transfer might be within reach with sufficient engineering. Additionally, both works forego raw audio signals through explicit MIDI `note_on` events.

4.2.2 Problem Formulation

A fundamental requirement of the presented approach is the ability to detect and characterize plucks of tuned strings in the audio and tactile modality. To this end, the developed setup utilizes dedicated unimodal detectors. Aligned multimodal detections are then integrated to estimate the geometric model $\mathcal{S}$. After introducing a motion primitive $\mathcal{P}$ for plucking motions and its associated action space Φ, the chapter details the exploration stages to autonomously estimate and refine the model components $\langle \mathcal{S}, \mathcal{V}, \mathcal{D} \rangle$ over time.
Given a chordophone with unknown geometry located in the reachable environment of the robot, the objectives pursued by the discussed approach are to

1. reconstruct the instrument geometry as part of the environment model using mostly self-supervised motions,
2. explore the action space of plucking motions with estimated strings to characterize and model the produced audio responses, and
3. reduce unwanted robot motions that might damage or wear out the instrument and robot.

To isolate the first two objectives and to achieve a stable exploration of the valid action space, these objectives are implemented through two decoupled exploration stages.
The first stage reconstructs the relevant geometry of the instrument. As a generic formulation for chordophones, the model describes a set of strings that can be manipulated. Each such physical instrument string $s \in \mathcal{S}^*$ is described through a reachable string segment of length $\hat{\ell}_s$ between the Cartesian pose $\hat{T}_s \in SE(3)$ and $\hat{T}_s \cdot [\hat{\ell}_s, 0, 0, 1]^T$ and annotated with its fundamental note N_s (a musical scale element) in a set $\mathcal{S}$:

$$\mathcal{S} = \left\{ \langle \hat{N}_s, \hat{T}_s, \hat{\ell}_s \rangle \mid s \in \mathcal{S}^* \right\},$$

The size of the ground-truth set of strings is a-priori unknown and thus initial information needs to be provided to bootstrap the estimation of the set $\mathcal{S}$.
The action space Φ of the system is defined relative to the individual estimated strings s and utilizes a parameterized motion primitive $\mathcal{P}$ that describes Cartesian trajectories Γ.

$$\mathcal{P} : (T_s : SE(3)) \times (\ell_s : \mathbb{R}) \times (\phi : \Phi) \to (\gamma_s^\phi : \Gamma),$$

The system assumes that at least one parameter vector $\phi_s^{\text{init}} \in \Phi$ can be heuristically inferred for each element in $\mathcal{S}$ such that the execution of the associated trajectory $\gamma_s^{\phi_s^{\text{init}}}$ yields the expected audio note onset $\hat{N}_s$. Once a stable geometry $\mathcal{S}$ is reconstructed, the second exploration stage proceeds to explore the action space and audio onset responses of the instrument as two related functions:

$$\mathcal{V} : \mathcal{S} \times \Phi \to [0; 1], \text{ and}$$
$$\mathcal{D} : \mathcal{S} \times \Phi \to \Psi.$$

In this formulation $\mathcal{V}$ estimates a probability of ϕ to be *valid*. Any criteria observable during the execution of the motion primitive might define validity in the general case. For the purposes of the scenario, a parameter vector ϕ is determined to be valid, if and only if the physical execution of γ_s^ϕ causes a single string pluck observable in multiple modalities, where the observed musical note onset matches the note N_s associated with the string $s \in \mathcal{S}$. $\mathcal{D}$ estimates the audio onset characteristics $\psi \in \Psi$ assuming ϕ were valid. This investigation defines Ψ as the measured loudness of the fundamental note of an audio onset in A-weighted decibels [dBA]. The general approach supports any measurable metric on note onsets, though, including overtone profiles and temporal envelope, both related to the timbre of the instrument and playing technique.
This decomposition into *functional* and *discriminative* functions originates in the field of *Safe Active Exploration* approaches using Gaussian processes (used, e.g., by Schreiter and Li [85, 129]) and facilitates reasoning over a set of safe action parameters $\Phi_s^\alpha = \{\phi \in \Phi | \, \mathcal{V}(s, \phi) > \alpha\}$, where $1 - \alpha$ describes the remaining risk of an invalid action. As there is no concept of safety involved in the scenario, the term *validity* is used instead.
Eventually, the combined system model $\langle \mathcal{S}, \mathcal{V}, \mathcal{D} \rangle$, which comprises all three data-driven components, is utilized to infer motion parameters that produce audio onsets for note N^t with characteristics close to ψ^t. Parameters can be derived through an inverse model approach on $\mathcal{D}$ within the estimated valid action space through loss optimization of the forward model:

$$\begin{aligned}\phi_s^t &= \underset{\substack{s\in\mathcal{S}\\ \phi\in\Phi}}{\arg\min}\, (\mathcal{D}(s,\phi) - \psi^t)^2 \\ &s.t.\ N_s = N^t, \text{and } \phi \in \Phi_s^\alpha\end{aligned}$$

4.2.3 Modality Analysis

Audio Analysis

The audio noise floor near the PR2 robot reaches 60–68 dBA SPL, caused mainly by cooling fans. As audio onsets should be captured from around 40 dBA SPL, the physical setup includes a contact microphone attached to the resonating body of the instrument. However, this yields an uncalibrated audio signal, without a direct interpretations as sound pressure level. Audio evaluation instead considers dB above signal noise floor.

Because the audio signal originates from a tuned musical instrument, the frequency analysis can be limited to frequencies associated with the western music scale. A well-established way for fast spectrum analysis in this field is described by the *constant Q-transform* (CQT) [17, 128], which can be computed based on the classical Fast-Fourier transform. The guzheng, known for its complex timbre [86], usually encompasses a playing range from D2 (73.42 Hz) to D6 ($\approx$ 1.17 kHz). To include essential harmonics for all notes, the applied CQT considers two additional octaves above yielding 84 semitones above D2. The microphone signal is sampled with 44.1 kHz and the analysis uses 512 samples per hop, yielding an effective temporal resolution of 12 ms for further analysis.

To locate note onsets, the detector builds on Böck et al.'s work [12] and extracts maxima in the spectral flux envelope of the CQT in overlapping processing windows of 500 ms. The magnitude representation of the CQT exhibits maxima at the fundamental frequencies associated with each string. However, with instrument resonance, simple maximum detection yields several systematic false positives to classify onsets by note. For more robust classification in the required range, the *CREPE* network [75] is applied to estimate the fundamental note of each onset.

Tactile Pluck Detection

The PR2 robot used with this scenario is equipped with a right Shadow Robotic Hand and *BioTac* tactile fingertips [152] (Figure 4.2). Because the plectra are directly taped to the sensor, contact of a plectrum with a string of the instrument clearly reflects in the measured tactile feedback. The fingertip sensor provides a number of signals that can be used to evaluate contact position and state. For the purpose of perceiving plucking events and plucking strength, the absolute pressure values reported by the BioTac at 100 Hz suffices to estimate deflection and release of the string. A plucking event is detected as a sudden drop in pressure, detected through a tunable threshold on the numeric derivative. To filter noise and invalid burst detections during onset transients, a 100 ms suppression time after each detection is enforced.

Pluck Validation

All plucking attempts by the system are assigned a binary validity score $\nu \in \{1, -1\}$ as labels for the subsequent online exploration. To evaluate the performance and location of an attempted pluck and reduce ambiguities in either detector, the two modalities are aligned and used to evaluate events together.

Audio onsets are associated with detected tactile plucks if they occur in a short time frame of 50 ms after the tactile pluck event. Any audio onset that does not match a tactile pluck is assumed to be unintentional or externally caused and yields a negative score. Additionally, plucking attempts that fail to trigger any onset at all, trigger multiple onsets, or multiple tactile events, yield negative scores as they either missed the intended string, or unintentionally made contact with other parts of the instrument. To avoid excessive stress on individual strings, observed plucks are additionally assigned negative scores if they exceed fixed pressure thresholds perceived by the tactile sensors. Lastly, a plucking attempt that aims to trigger a specific note onset through a targeted string receives a positive score $\nu = 1$ if and only if a single tactile-grounded audio onset is observed during the pluck.

4.3 Model Reconstruction

Kinematic Projection

Assuming a known position of the plectrum tip w.r.t. the finger it is attached to, each audio onset event, which is grounded in a tactile detection, can be associated with a Cartesian position of this tip using the known forward kinematics of the robot. While the exact plectrum positions are usually unknown, the described exploration stages are robust enough to cope with discrepancies of several millimeters.
The resulting point projections of haptically-validated audio onsets can be seen in Figure 4.1 (right). Relevant sources for the observable noise in this projection include (1) uncertainties in the plectrum position, (2) the deflection of the string during the plucking motion that differs per pluck, (3) an unmeasured deformation of the plectrum itself, (4) temporal skew between the different modalities, and (5) systematic errors in joint measurements and robot calibration.

String Fitting

To estimate the string coordinate frames $\hat{T}_s$ and length $\hat{\ell}_s$ for all elements of $\mathcal{S}$, the projected pluck events grouped by observed note $\hat{N}_s$ provide evidence for line models fitted through Random Sample Consensus (RANSAC) [35]. Model fitting utilizes an inlier threshold of 1 cm. An outlier-resistant line fitting is required because in practice false-positive detections still occur sporadically due to harmonics, and background noise. To yield a stable frame estimate, the coordinate frame T_s for each string is defined to originate at the RANSAC model's right- and bottom-most inlier projection and points along the string in its x- and upwards in its z-axis.
Lastly, one can optionally assume a planar bridge for the mounting of all strings on the instrument and fit a 2D line on the regression plane of all inliers of string models that fits the origins of the computed coordinate systems. To align all frame origins, they can be redefined at the intersections of the string model and the 3D plane associated with the fitted bridge line that is orthogonal to the regression plane. The resulting frames and fitted string segments can be seen in Figure 4.1 as well.

Motion Primitive for Plucking

There are various plucking techniques for chordophones depending on the intended dynamic expression and musical context. Exemplary techniques for a modern playing style were reviewed by Fu [39]. As a formal definition of the described motions must essentially include unconstrained motion profiles, the respective action space Φ to parameterize plucks via a motion primitive $\mathcal{P}$ is effectively intractable, and the automatic evaluation of their execution safety in the partially unknown physical world is very limited.
Instead, the scenario in this chapter considers a strongly reduced parametrization for $\mathcal{P}$ that can be specified as a position on the string and a trajectory of the plectrum tip in the two-dimensional

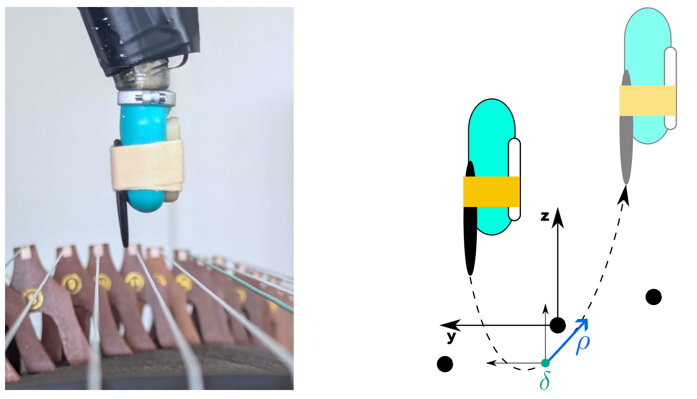


Figure 4.3 – *(left)* A finger of the Shadow Dexterous Hand with BioTac sensor in playing position, with the plectrum between the 11th and 10th string tuned to *D4* and *E4*, respectively. *(right)* Planar sketch of the parameterized pluck primitive in the local coordinate frame of a string. Each motion starts on one side of the string at a fixed position, dives between two strings, and accelerates upwards to reach the waypoint δ with velocity ρ. The last waypoint specifies a position above and behind the string to achieve the actual pluck.

Cartesian coordinate system orthogonal to the string. This primitive connects fixed 2D start- and end poses for either plucking direction d (inward or outward) with a parameterized intermediate position/velocity waypoint $\langle \delta, \rho \rangle$ close to the string, as illustrated in Figure 4.3. Note that any such pluck model includes an implicit spring component, as the string is deflected. In the case of the Shadow Dexterous Hand, the finger joints exhibit sufficient controlled spring compliance to store all additional potential energy until the plectrum releases the string. The fixed start position is selected closer to the string and the goal position slightly further away to allow for a broader range of transit motions to smoothly reach the start point and have more flexibility in continuing the trajectory after the physical pluck occurs. To generate smooth trajectory profiles with intuitive dynamics limits, the Ruckig trajectory generator [10] is applied in Cartesian space. In summary, this primitive defines action parameters $\phi \in \Phi$ as

$$\langle d, \eta, \delta, \rho \rangle \in \{\textbf{in}, \textbf{out}\} \times [0; 1] \times \mathbb{R}^2 \times \mathbb{R}^2,$$

where $\eta \cdot \hat{\ell}_s$ defines the string position to pluck.

To actuate the described Cartesian trajectory γ_s^{ϕ}, it needs to be mapped to the robot's joint space. Various analytical and optimization methods can be applied here, as discussed in chapter 2. The experimental implementation utilizes an IK-based iterative tracking loop based on the *bio_ik* solver [123]. The solver tightly integrates with the MoveIt system [24] and can thus avoid collisions with the modeled environment also for longer motions when moving between strings.

4.3.1 Geometric Exploration

Combining the reconstruction system for $\mathcal{S}$, and the pluck parametrization $\mathcal{P}$, the system gathers spatial evidence autonomously and refines $\mathcal{S}$ over time. Still, $\mathcal{P}$ relies on estimates $\hat{T}_s, \hat{\ell}_s$ for the string geometry as Cartesian reference space. Such initial estimates may originate from a previous reconstruction, visual estimations without associated fundamental note annotations, or kinesthetic demonstrations. In the performed experiments, the latter approach is adopted by demonstrating plucks at the reachable ends of the strings. Thus, these initial demonstrations determine the length of reconstructed string models $\hat{\ell}_s$ and autonomous samples can be collected between them. During autonomous operation during this phase, the framework repeatedly samples a direction, a string $s \in \mathcal{S}$, and a string position $\eta \cdot \hat{\ell}_s$, executes the associated pluck with predetermined δ, ρ and evaluates the validity of the attempt. On invalid plucks, attempts are repeated with heuristically-modified parameters up to a fixed number of tries, where δ is systematically lowered and moved along the pluck direction on missing onsets, and moved opposite the direction with ρ step-wise rotated upwards on invalid detected onsets. Valid plucks are integrated to update $\mathcal{S}$.
Experimentally, Halton sampling for η improved model stability over uniform sampling, as the latter is known to yield spatial clusters in few samples. Such clusters effectively cause model fitting to diverge as not enough spatially spread evidence initially exists. While the best geometric sampling points to ground the string geometry are near the ends, these points are also most prone to invalid detections due to the stronger excitation of harmonics. Attempted plucks along the whole length of the string remain more robust.
To account for long transit motions, samples from $\mathcal{S}$ are taken with a shaped distribution with the motivation to encourage strings closer to the current finger position and penalize strings with sufficient gathered evidence. While accelerating overall exploration, this choice is entirely optional, and target strings might also be determined by sheet music at the cost of a less balanced overall exploration.

4.4 Active Motion Primitive Exploration (AMPE)

After successful geometric construction of $\mathcal{S}$, the second exploration stage actively selects informative and likely-valid parameters from Φ to improve approximations of $\mathcal{V}$ and $\mathcal{D}$. The estimated model $\mathcal{S}$ acts as a basis for both functions and remains fixed as reference geometry for the string-centric pluck primitive. To allow for exhaustive exploration of the parameter space Φ in the evaluation, the number of free parameters Φ are further reduced by freezing δ_z and ρ in the respective direction. Thus, the motion primitive effectively describes actions through a *target string* s, a *binary direction* d, a *normalized string position* η, and δ_y, labelled the *deflection offset*. Given sufficient priors, plucks can theoretically contribute information on other strings and the opposite plucking direction through through model transfer. However, no clear data association

could be recovered during experimental analysis as controller responses of the PR2 robot arm vary significantly between directions. Thus, independence is assumed in s and d, and $\mathcal{V}, \mathcal{D}$ are modelled as independent functions $\mathcal{V}^s_{\textbf{in}}/\mathcal{D}^s_{\textbf{in}}$, $\mathcal{V}^s_{\textbf{out}}/\mathcal{D}^s_{\textbf{out}}$ per string s.
With each attempted pluck with parameters ϕ, the collected datapoint describes the validity and audio response for this attempt:

$$\langle \phi, \nu, \psi \rangle \in \Phi \times \{1, -1\} \times \Psi.$$

The tuple $\langle \phi, \nu \rangle$ is added to a set $X^s_{\mathcal{V}}$ collecting evidence for the validity landscape $\mathcal{V}$. Whenever $\nu = 1$, $\langle \phi, \psi \rangle$ is additionally added to a set X^s_{Ψ} providing evidence for $\mathcal{D}$ within the set of valid action parameters. Following existing safe active learning paradigms, as proposed for example by Schreiter et al. [129], both functions are independently modelled as a Gaussian Process (GP) with stationary squared exponential kernel and zero mean with normalized X^s_{Ψ}.
$\mathcal{D}$ is directly modelled through GP regression over X^s_{Ψ}:

$$\begin{aligned} \mathcal{D}(s, \phi) &= \mathbb{E}[g | X^s_{\Psi}(\phi)];\ g \sim GP(rbf(\lambda_d)) \\ &= \mathbb{E}[\mathcal{N}(x; \mu_\psi, \sigma_\psi)] = \mu_\psi \end{aligned}$$

As $\mathcal{V}$ describes a probability space, it is modelled through probit regression [119] over $X^s_{\mathcal{V}}$, assuming noisy measurements:

$$\begin{aligned} \mathcal{V}(s, \phi) &= p(\mathbb{E}[g | X^s_{\mathcal{V}}(\phi)] > 0);\ g \sim GP(rbf(\lambda_v)) \\ &= \int_{-\inf}^{0} \mathcal{N}(x; \mu_\nu, \sigma_\nu) dx \\ &= \frac{1}{2} + \frac{1}{2} \operatorname{erf}\left(-\frac{\mu_\nu}{\sigma_\nu \sqrt{2}} \right) \end{aligned}$$

The kernel scale for the validity λ_v remains fixed. This step is required to avoid unsafe extrapolation of trust regions because the regression, by the nature of the safe exploration paradigm, is based on unstratified sample sets. To determine the next action parameters in each exploration step, an information criterion is required which indicates plausible areas of the parameter space for exploration. While Bottero et al. [15] recently proposed a mutual information criterion on the discriminative function, this work uses the simpler differential entropy of the onset characteristics Gaussian Process modelling $\mathcal{D}$, which can be maximized through its posterior predicted variance σ_ψ and can be optimized well through Monte Carlo methods. Thus, the next best pluck to execute for a given string and direction is iteratively selected as

$$\begin{aligned} \phi^* = &\arg\max_{\phi \in \Phi} H[g | X^s_{\Psi}(\phi)];\ g \sim GP(rbf(\lambda_s)) \\ &s.t.\ \mathcal{V}(s, \phi) > \alpha \end{aligned}$$

Parameter selection could also include an optimization over all strings s, at the cost of an increased computational burden. However, this definition would not consider time for transit

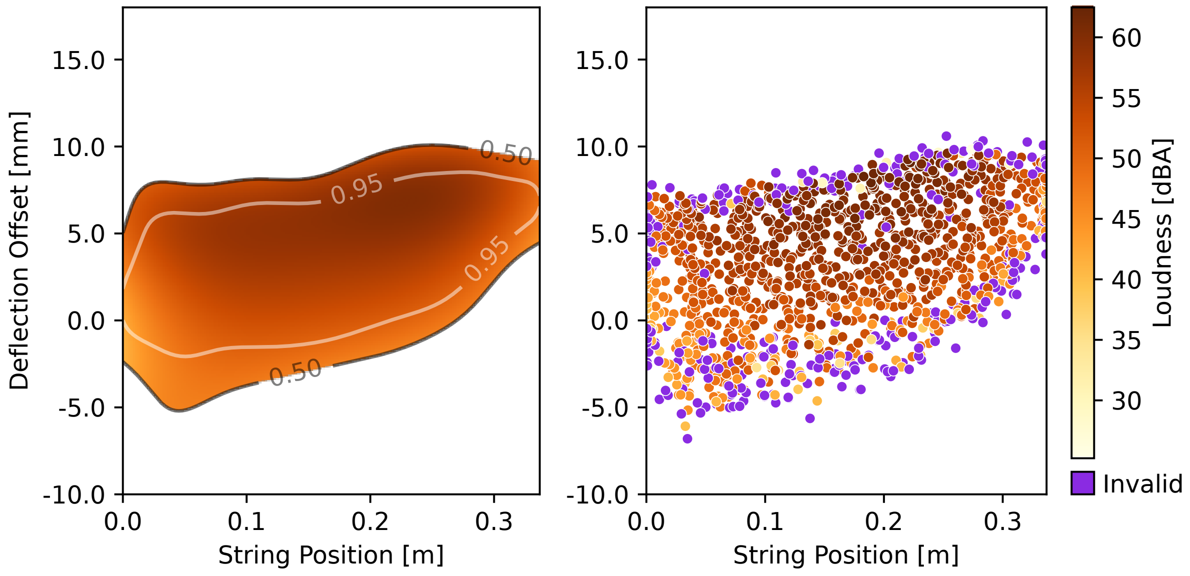


Figure 4.4 – *(left)* Exemplary pluck dynamics $\langle \mathcal{V}, \mathcal{D} \rangle$ fit for 1500 explored F♯4 plucks. Contours indicate $\Phi_s^{0.5}$ and $\Phi_s^{0.95}$. *(right)* All associated explored plucks, including those evaluated as invalid highlighted in purple.

motions and thus lead to significantly increased exploration times. Instead, the target string to explore in each step is selected through the same shaped sampling distribution used in subsection 4.3.1. To accelerate exploration at the cost of slightly more invalid plucking attempts, a lower validity threshold of $\alpha = 0.7$ is applied during active exploration. An exemplary $\langle V, D \rangle$ model and the underlying dataset are illustrated in Figure 4.4 and 4.5.

4.4.1 Experimental Validation

This section presents an evaluation of the high-level exploration stages.

Typical error conditions of the lower-level modules include (a) soft onsets being missed or misclassified when strong overtones are present or the audio-tactile modalities are misaligned, (b) false positive pluck detections when a plectrum slides along the string, (c) temporal drift between the robot sensor data and the audio channels due to drifting clocks. This last problem was mitigated through occasional manual recalibration.

To illustrate the behavior of the exploration stages, several representative experiments are detailed below, each limited to one finger and a subset of instrument strings.

Geometric Reconstruction

The quality and efficiency of the string reconstruction stage was evaluated as follows. A series of five individual strings were repeatedly initialized through two kinesthetic demonstrations while varying the instrument position within the reachable workspace of the robot arm. Subsequently,

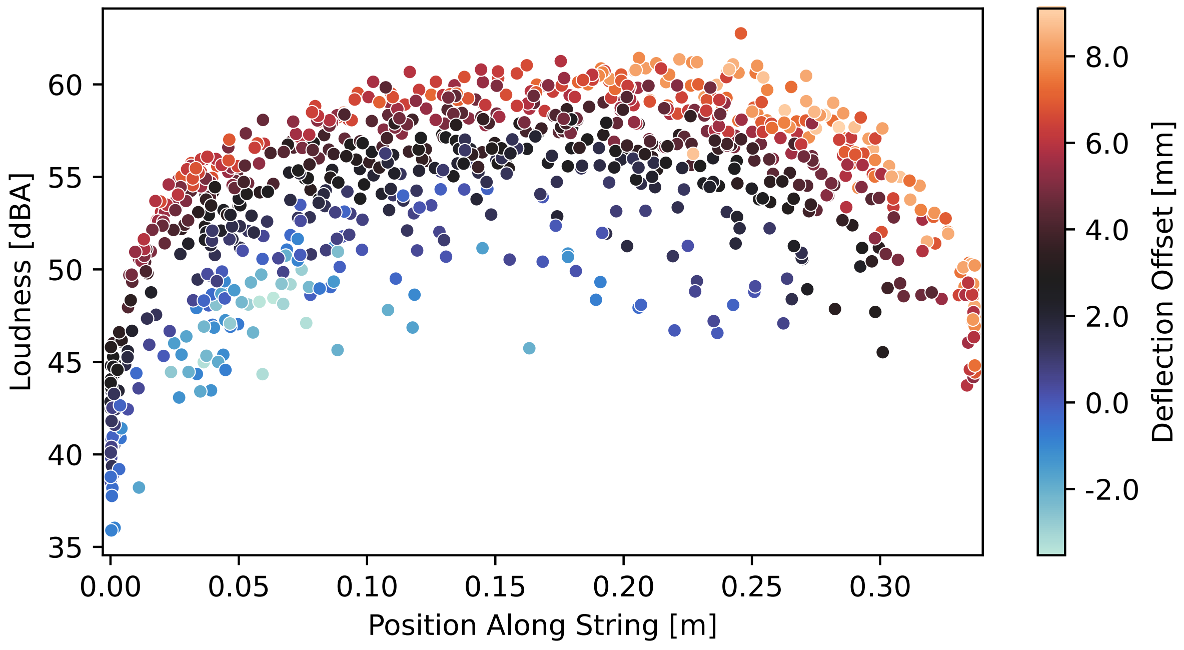


Figure 4.5 – Dynamic range of valid explored pluck motions for F♯4 along the string. As expected, plucks at the center of the string can produce significantly louder onsets with similar deflection offsets.

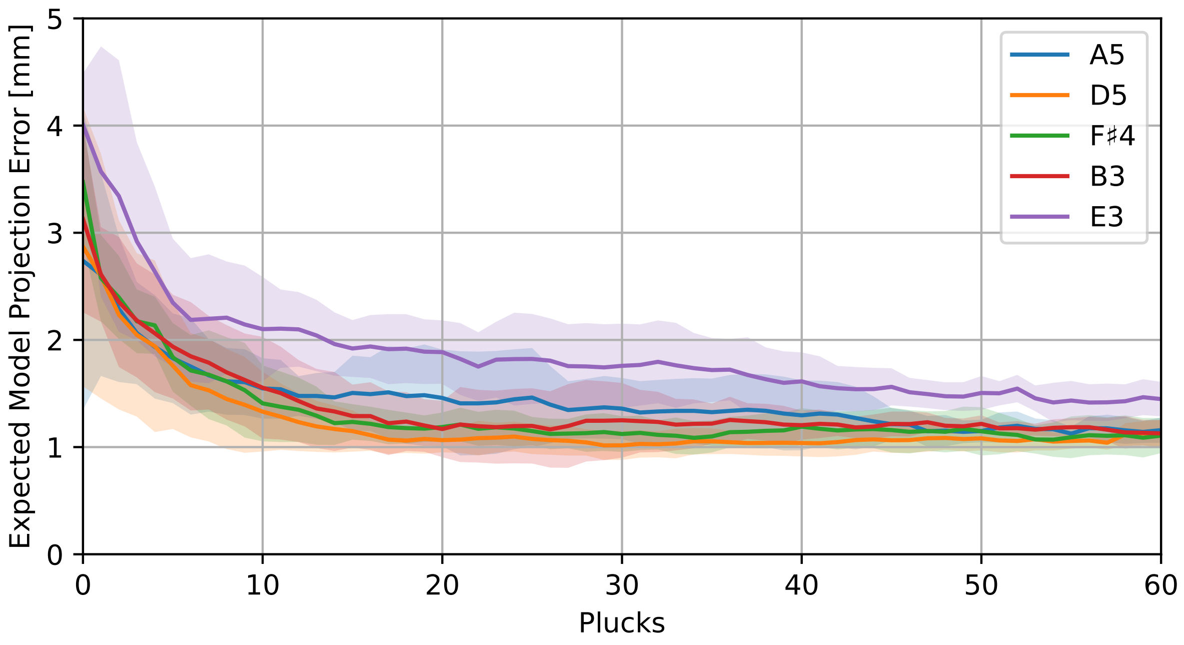


Figure 4.6 – Expected Euclidean model error between kinematic onset projections p and their projection $m(p)$ onto the current RANSAC line fit as the number of attempted plucks increases. Expectations are approximated through the next 30 plucks at each point. Five episodes were collected per string.

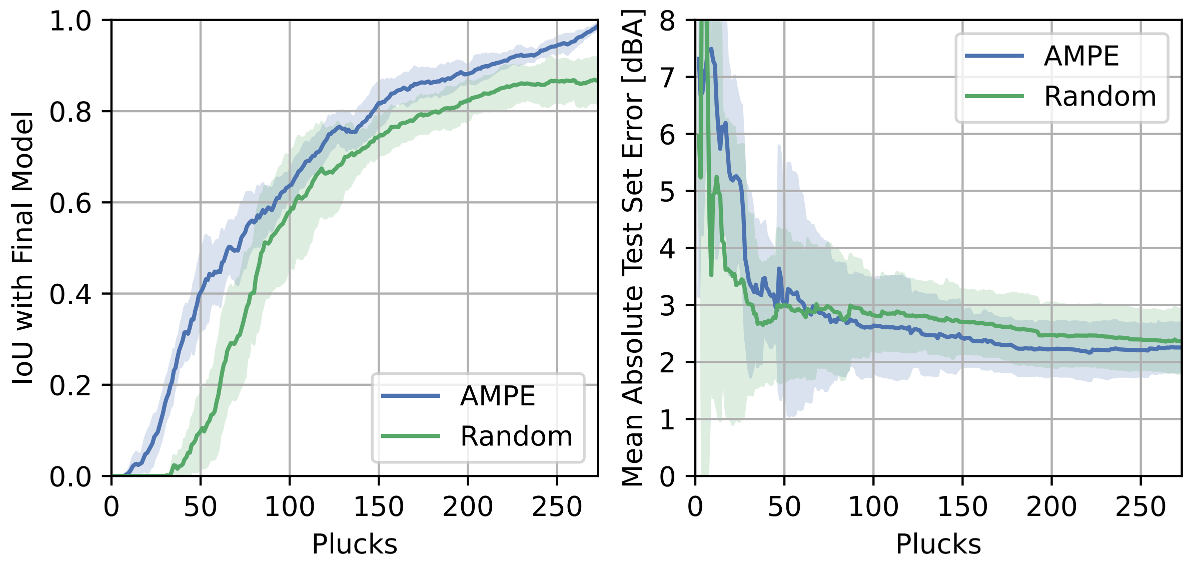


Figure 4.7 – *(left)* IoU of valid action space volumes $\Phi_s^{0.75}$ after each new pluck and the final valid volume. Volumes were estimated through Monte Carlo integration. *(right)* Mean absolute error of $\mathcal{D}$ on test set over number of recorded plucks. Both compare AMPE and a random baseline.

the stage explored each string autonomously through 90 plucks. For the analysis the expected Euclidean projection error for new plucks onto the current string model m_i: $E[\|m_i(p) - p\|_2]$ was utilized and estimated through the next 30 valid plucks after each point. Figure 4.6 illustrates the results over five episodes per string. All models converge to an expected projection error of about 1 mm. This limit corresponds to the kinematic inaccuracies of the utilized robot system, limited reproducibility of trajectory execution, and the unaccounted deflection of plectrum and string. Almost all episodes converge between 10 and 20 exploratory plucks, where the exact behavior does not strictly depend on the length of the string. As basis for the motion primitive exploration in the next section, each string was plucked at least 15 times during the geometric exploration stages to balance the best achievable precision and swift exploration.

Pluck Exploration

To access the success of the described active valid pluck exploration process, the growth and evolution of the valid parameter set throughout the exploration is determined through an intersection over union (IoU) measure with the corresponding final valid parameter set. As shown in Figure 4.7, a significant portion of the final set is approximated within the first 70 samples where IoU exceeds 0.5. Progress near the border regions takes significantly more attempted plucks. A hypothetical baseline exploration, which integrates random samples from the same dataset, performs significantly worse, because it does not systematically grow the initial valid set. As it is not bound by the unknown validity of sampled plucks, it provides global information on $\mathcal{D}$ with less sampled plucks, but the mean absolute test set error still falls below 3 dBA for both policies within 50 plucks.

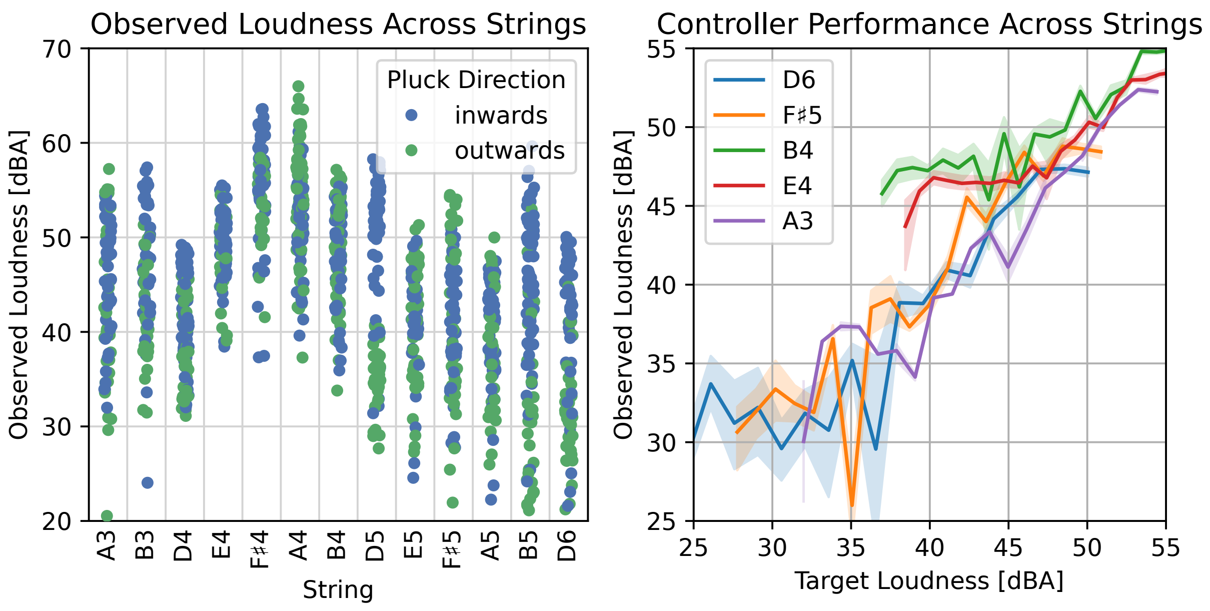


Figure 4.8 – *(left)* Loudness distribution of evaluated plucks for 13 strings after 1300 plucks of autonomous exploration. All reconstructed strings could be plucked within a dynamic range of at least 20 dBA, corresponding to a four- to eightfold subjective doubling of the loudness. *(right)* Experimental performance curve of predicted pluck parameters to achieve a target loudness.

System-Level Integration

In a final integration test, estimates for the 13 upper strings of the instrument are initialized kinesthetically and the system is left to explore both stages for two hours. The switch between the two exploration stages was heuristically determined to happen after 200 successful plucks during the first stage. Figure 4.8 (left) presents the observed loudness distribution of note onsets in the collected set of samples. All strings were successfully explored with a dynamic range of 20–40 dBA. The significant shift in measured loudness between central notes appears as an artifact of the instrument's resonance and the contact microphone.

Eventually, the generated model can be utilized to infer parameters for a target loudness the plucking motion should achieve. To this end, a motion parameter set can be optimized within $\Phi^{0.95}$ of the explored model through Monte Carlo methods as described in subsection 4.2.2. Testing reproducibility, 20 evenly spaced loudness responses across the range of previously observed values for each string were evaluated in five attempts each through physical execution. As shown in Figure 4.8 (right), response predictions are usually accurate within 3 dB, but flatten out where weak onsets could not be reproduced sufficiently well due to stochastic system noise.

4.4.2 Summary

This chapter investigates a two-stage active exploration approach that allows a physical robot to reconstruct the geometry of chordophones and explore associated plucking motions for expressive play through motion primitive exploration. As the musical instrument and the dexterous robot components are prone to break through repeated inappropriate contacts, a validity-constraint exploration paradigm was derived. Validity of attempted motions was determined through observed effects across multiple modalities. Starting from a weak prior of the instrument geometry provided through kinesthetic demonstrations, the robot autonomously and successfully explored the doubly-curved geometry of a Chinese Guzheng, refining an environment model, and explores the action space of a motion primitive formulation to interact with the instrument. The effects of motion primitives to generate note onsets could be characterized with a small margin of error and the generated model can be utilized to generate local motions to manipulate the instrument and yield desired note onsets.

Chapter 5

Conclusion

Summary This dissertation advances both the fields of description and planning of manipulation actions, as well as the field of data-oriented self-exploration for local motions during manipulation. It introduces a modular, phase-based approach to robotic manipulation design and planning and demonstrates its effectiveness and versatility using the MoveIt Task Constructor software framework, supported by experimental studies.

By segmenting complex manipulation tasks into distinct motion phases — each solvable by a dedicated planning module — the methods can be adapted to different scenarios, robots, and application domains. This approach combines modular reuse of solvers with special-purpose trajectory generation, while facilitating introspection and debugging: failures can be localized in individual phases and traced back to the corresponding solver or planning request. This modular, phase-oriented structure simplifies integration in real-world deployments.

In straightforward motion-planning contexts, dedicated planners generate collision-free transit motions, while separate logic identifies and evaluates multiple inverse-kinematics solutions. In more advanced settings, such as tool use, the framework cleanly delimits subproblems — including approach, tool attachment, and specialized object handling — so that each can incorporate task-specific computation. A key advantage of this phase-based delineation is the explicit handling of phase transitions, allowing the system to define and adjust the expected world state and possibly switch trajectory controllers between phases.

A complementary contribution of this work is a method for physical self-exploration of multimodal motion-phase primitives, demonstrated through robotic interactions with a Chinese chordophone. By limiting the exploration to a constrained set of motion parameters and tracking feedback across multiple sensing modalities, the robot can efficiently map the range of achievable effects. To minimize risk in environments where inappropriate movements may cause damage, the approach employs a validity-constrained exploration strategy, growing a trust region through attempted motions. Applied in an anytime fashion, the system collects sufficient experience to build predictive models of expected motion outcomes and can then reproduce desired effects through model inversion.

Impact beyond publications During the development of the software implementations for this dissertation, substantial contributions were made to the global Open Source robotics community. Numerous software patches were contributed to community projects in the ROS ecosystem, including over 1000 patches to software repositories directly related to maintenance efforts for the MoveIt manipulation framework. The released core package of MoveIt received over 1.3 million binary package downloads from between 2020 and 2025. The released software framework *MoveIt Task Constructor*, developed in collaboration with Robert Haschke, received over 19000 binary package downloads between 2023 and 2025. Further enhancing exposure in the community, an early version of the framework was utilized as part of a significantly larger system in the winning entry of the *Mobile Manipulation Hackathon Competition* at the International Conference on Intelligent Robots and Systems (IROS) 2018 [122].

Outlook The primary goal of this work is to enable planning and exploration of local motions, providing essential building blocks and planning schemas for complex manipulation tasks. While the Task Construction design philosophy simplifies the overall planning structure and supports black-box methods for subproblems, information propagation between phases remains a challenge for further research. In contrast to the purely local approach developed in this work, fully optimization-based methods for complete manipulation sequences — such as the recent *Shadow Program* approach by Alt et al. [2] — aim to optimize entire manipulation processes, but demand significant offline computation and are sensitive to hyperparameter selection. Open potential remains in the sample selection for co-parameters through integration of information over alternative planning attempts. Informed sampling strategies for phase parameters, such as cross-entropy methods and scene-conditioned discrete parameter prediction inspired by Driess et al. [32], might provide a feasible approach for faster selection of performant parameters and learning of heuristics beyond individual manipulation planning attempts.
Task Construction with standard Stage and Container structures explicitly accommodates textual task specifications which are tied to clear stage definitions and properties. Research into automatically generating and adapting these specifications for new scenarios holds promise for future work.
With the recent success of imitation learning through diffusion mechanisms [22, 88] and vision–language–action model training [76], Task Construction provides an adaptable mechanism for the synthetic generation of expert demonstrations, generalizing the approach of the *MotionBenchMaker* system [21] to extended manipulation tasks.
Finally, as a middle ground between motion planning and TAMP with a practical focus, Task Construction could contribute to introspectable TAMP approaches through an additional interface layer for task planning. Initial ideas in this direction where presented by Siburian et al. [132] through integration with PDDLStream [43].

References

[1] Sina Aghli and Christoffer Heckman. “Online System Identification and Calibration of Dynamic Models for Autonomous Ground Vehicles”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018, pp. 4933–4939. DOI: 10.1109/ICRA.2018.8460691.

[2] Benjamin Alt, Claudius Kienle, Darko Katic, Rainer Jäkel, and Michael Beetz. *Shadow Program Inversion with Differentiable Planning: A Framework for Unified Robot Program Parameter and Trajectory Optimization*. 2025. DOI: 10.48550/arXiv.2409.08678.

[3] Fahiem Bacchus and Qiang Yang. “The downward refinement property”. In: *Proceedings of the 12th International Joint Conference on Artificial Intelligence - Volume 1*. IJCAI’91. Sydney, New South Wales, Australia: Morgan Kaufmann Publishers Inc., 1991, pp. 286–292. ISBN: 1558601600.
URL: https://dl.acm.org/doi/10.5555/1631171.1631214.

[4] Clark Barrett and Cesare Tinelli. “Satisfiability Modulo Theories”. In: *Handbook of Model Checking*. Ed. by Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem. Cham: Springer International Publishing, 2018, pp. 305–343. ISBN: 978-3-319-10575-8. DOI: 10.1007/978-3-319-10575-8_11.

[5] Patrick Beeson and Barrett Ames. “TRAC-IK: An open-source library for improved solving of generic inverse kinematics”. In: *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*. 2015, pp. 928–935. DOI: 10.1109/HUMANOIDS.2015.7363472.

[6] Michael Beetz, Daniel Beßler, Andrei Haidu, Mihai Pomarlan, Asil Kaan Bozcuoğlu, and Georg Bartels. “Know Rob 2.0 — A 2nd Generation Knowledge Processing Framework for Cognition-Enabled Robotic Agents”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018, pp. 512–519. DOI: 10.1109/ICRA.2018.8460964.

[7] Michael Beetz, Gayane Kazhoyan, and David Vernon. *The CRAM Cognitive Architecture for Robot Manipulation in Everyday Activities*. 2023. DOI: 10.48550/arXiv.2304.14119.

[8] Michael Beetz, Lorenz Mösenlechner, and Moritz Tenorth. “CRAM — A Cognitive Robot Abstract Machine for everyday manipulation in human environments”. In: *2010*

IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). 2010, pp. 1012–1017. DOI: 10.1109/IROS.2010.5650146.

[9] Lars Berscheid. "Self-supervised Learning of Primitive-based Robotic Manipulation". PhD thesis. Karlsruher Institut für Technologie (KIT), 2023. 181 pp. DOI: 10.5445/IR/1000164562.

[10] Lars Berscheid and Torsten Kröger. "Jerk-limited Real-time Trajectory Generation with Arbitrary Target States". In: *Robotics: Science and Systems XVII* (2021). DOI: arXiv:2105.04830.

[11] Mohak Bhardwaj, Balakumar Sundaralingam, Arsalan Mousavian, Nathan D. Ratliff, Dieter Fox, Fabio Ramos, and Byron Boots. "STORM: An Integrated Framework for Fast Joint-Space Model-Predictive Control for Reactive Manipulation". In: *Proceedings of the 5th Conference on Robot Learning*. Ed. by Aleksandra Faust, David Hsu, and Gerhard Neumann. Vol. 164. Proceedings of Machine Learning Research. PMLR, Nov. 2022, pp. 750–759.

[12] Sebastian Böck and Gerhard Widmer. "Maximum filter vibrato suppression for onset detection". In: *Proc. of the 16th Int. Conf. on Digital Audio Effects (DAFx). Maynooth, Ireland (Sept 2013)*. Vol. 7. 2013, p. 4.
URL: https://www.dafx.de/paper-archive/2013/papers/09.dafx2013_submission_12.pdf.

[13] Jeannette Bohg, Antonio Morales, Tamim Asfour, and Danica Kragic. "Data-Driven Grasp Synthesis—A Survey". In: *IEEE Transactions on Robotics* 30.2 (2014), pp. 289–309. DOI: 10.1109/TRO.2013.2289018.

[14] Bertold Bongardt. "Inverse Kinematics of Anthropomorphic Arms Yielding Eight Coinciding Circles". In: *Computational Kinematics*. Ed. by Saïd Zeghloul, Lotfi Romdhane, and Med Amine Laribi. Cham: Springer International Publishing, 2018, pp. 525–534. ISBN: 978-3-319-60867-9.

[15] Alessandro Bottero, Carlos Luis, Julia Vinogradska, Felix Berkenkamp, and Jan R Peters. "Information-Theoretic Safe Exploration with Gaussian Processes". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 30707–30719. DOI: 10.48550/arXiv.2212.04914.

[16] Mason Bretan and Gil Weinberg. "A Survey of Robotic Musicianship". In: *Commun. ACM* 59.5 (Apr. 2016), pp. 100–109. DOI: 10.1145/2818994.

[17] Judith C. Brown and Miller S. Puckette. "An efficient algorithm for the calculation of a constant Q transform". In: *The Journal of the Acoustical Society of America* 92.5 (Nov. 1992), pp. 2698–2701. ISSN: 0001-4966. DOI: 10.1121/1.404385.

[18] Gerelmaa Byambatsogt, Lodoiravsal Choimaa, and Gou Koutaki. "Guitar Chord Sensing and Recognition Using Multi-Task Learning and Physical Data Augmentation with Robotics". In: *Sensors* 20.21 (2020). DOI: 10.3390/s20216077.

[19] Justin Carpentier, Guilhem Saurel, Gabriele Buondonno, Joseph Mirabel, Florent Lamiraux, Olivier Stasse, and Nicolas Mansard. "The Pinocchio C++ library : A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives". In: *2019 IEEE/SICE International Symposium on System Integration (SII)*. 2019, pp. 614–619. DOI: 10.1109/SII.2019.8700380.

[20] Delphine Chadefaux, Jean-Loïc Le Carrou, Marie-Aude Vitrani, Sylvère Billout, and Laurent Quartier. "Harp plucking robotic finger". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2012, pp. 4886–4891. DOI: 1 0.1109/IROS.2012.6385720.

[21] Constantinos Chamzas, Carlos Quintero-Peña, Zachary Kingston, Andreas Orthey, Daniel Rakita, Michael Gleicher, Marc Toussaint, and Lydia E. Kavraki. "MotionBenchMaker: A Tool to Generate and Benchmark Motion Planning Datasets". In: *IEEE Robotics and Automation Letters* 7.2 (2022), pp. 882–889. DOI: 10.1109/LRA.2021.3133603.

[22] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin CM Burchfiel, and Shuran Song. "Diffusion Policy: Visuomotor Policy Learning via Action Diffusion". In: *Proceedings of Robotics: Science and Systems*. Daegu, Republic of Korea, July 2023. DOI: 10.15607/RSS.2023.XIX.026.

[23] Chris Duxbury and Mark Sandler and Mike Davies. "A Hybrid Approach to Musical Note Onset Detection". In: *Proc. of the 5th Int. Conference on Digital Audio Effects (DAFx-02), Hamburg, Germany*. Sept. 2002, pp. 33–38.
URL: https://dafx.de/papers/DAFX02_Duxbury_Sandler_Davis_note_onset_detection.pdf.

[24] David Coleman, Ioan A. Şucan, Sachin Chitta, and Nikolaus Correll. "Reducing the barrier to entry of complex robotic software: A MoveIt! case study". In: *Journal of Software Engineering for Robotics* 5.1 (May 2014), pp. 3–16. DOI: 10.6092/JOSER_2 014_05_01_P3.

[25] Michele Colledanchise and Petter Ögren. *Behavior trees in robotics and AI: An introduction*. CRC Press, 2018. ISBN: 978-1138593732.

[26] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. 3rd. Cambridge, MA, USA: MIT Press, 2009. ISBN: 978-0-262-03384-8.

[27] John J. Craig. *Introduction to Robotics: Mechanics and Control*. 3rd. Pearson, 2013. ISBN: 978-1-292-04004-2.

[28] Neil T. Dantam, Zachary Kingston, Swarat Chaudhuri, and Lydia E. Kavraki. “An Incremental Constraint-Based Framework for Task and Motion Planning”. In: *The International Journal of Robotics Research, Special Issue on the 2016 Robotics: Science and Systems Conference* 37.10 (2018), pp. 1134–1151. doi: 10.1177/0278364918761570.

[29] Elisa De Stefani, Alessandro Innocenti, Doriana De Marco, and Maurizio Gentilucci. “Concatenation of Observed Grasp Phases with Observer’s Distal Movements: A Behavioural and TMS Study”. In: *PLOS ONE* 8.11 (Nov. 2013), null. doi: 10.1371/journal.pone.0081197.

[30] Jacques Denavit and Richard S. Hartenberg. “A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices”. In: *Journal of Applied Mechanics* 22.2 (June 1955), pp. 215–221. issn: 0021-8936. doi: 10.1115/1.4011045.

[31] Hailei Ding, Hao Zhang, Bingqiang Yan, Junjun Jiang, Min Huang, and Zhongzhe Xiao. “Automatic Recognition of Basic Guzheng Fingering Techniques”. In: *Proceedings of the 8th Conference on Sound and Music Technology*. Springer Singapore, 2021, pp. 66–77. doi: 10.1007/978-981-16-1649-5_6.

[32] Danny Driess, Jung-Su Ha, and Marc Toussaint. “Deep Visual Reasoning: Learning to Predict Action Sequences for Task and Motion Planning from an Initial Scene Image”. In: *Proceedings of Robotics: Science and Systems*. Corvalis, Oregon, USA, July 2020. doi: 10.15607/RSS.2020.XVI.003.

[33] Kutluhan Erol, James Hendler, and Dana S. Nau. “HTN planning: complexity and expressivity”. In: *Proceedings of the Twelfth AAAI National Conference on Artificial Intelligence*. AAAI’94. Seattle, Washington: AAAI Press, 1994, pp. 1123–1128.

[34] Richard E. Fikes and Nils J. Nilsson. “Strips: A new approach to the application of theorem proving to problem solving”. In: *Artificial Intelligence* 2.3 (1971), pp. 189–208. issn: 0004-3702. doi: 10.1016/0004-3702(71)90010-5.

[35] Martin A. Fischler and Robert C. Bolles. “Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography”. In: *Commun. ACM* 24.6 (June 1981), pp. 381–395. doi: 10.1145/358669.358692.

[36] Adam Fishman, Adithyavairavan Murali, Clemens Eppner, Bryan Peele, Byron Boots, and Dieter Fox. “Motion Policy Networks”. In: *Proceedings of The 6th Conference on Robot Learning*. Ed. by Karen Liu, Dana Kulic, and Jeff Ichnowski. Vol. 205. Proceedings of Machine Learning Research. PMLR, Dec. 2023, pp. 967–977. doi: 10.48550/arXiv.2210.12209.

[37] Maria Fox and Derek Long. “PDDL2.1: An extension to PDDL for expressing temporal planning domains”. In: *Journal of Artificial Intelligence Research* 20 (2003), pp. 61–124. doi: 10.1613/jair.1129.

[38] Zipeng Fu, Tony Z. Zhao, and Chelsea Finn. *Mobile ALOHA: Learning Bimanual Mobile Manipulation with Low-Cost Whole-Body Teleoperation*. 2024. doi: 10.48550/arXiv.2401.02117.

[39] Zixuan Fu. "A Brief Analysis of the Performance Skills and Treatment of Guzheng's". In: *Proceedings of the 2021 3rd International Conference on Literature, Art and Human Development (ICLAHD 2021)*. 2021, pp. 733–741. doi: 10.2991/assehr.k.211120.134.

[40] Jonathan D. Gammell, Timothy D. Barfoot, and Siddhartha S. Srinivasa. "Batch Informed Trees (BIT*): Informed asymptotically optimal anytime search". In: *The International Journal of Robotics Research (IJRR)* 39.5 (Apr. 2020), pp. 543–567. doi: 10.1177/0278364919890396.

[41] Esther P. Gardner, K. Srinivasa Babu, Shari D. Reitzen, Soumya Ghosh, Alice S. Brown, Jessie Chen, Anastasia L. Hall, Michael D. Herzlinger, Jane B. Kohlenstein, and Jin Y. Ro. "Neurophysiology of Prehension. I. Posterior Parietal Cortex and Object-Oriented Hand Behaviors". In: *Journal of Neurophysiology* 97.1 (2007). PMID: 16971679, pp. 387–406. doi: 10.1152/jn.00558.2006.

[42] Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. "Integrated Task and Motion Planning". In: *Annual Review of Control, Robotics, and Autonomous Systems* 4.Volume 4, 2021 (2021), pp. 265–293. issn: 2573-5144. doi: 10.1146/annurev-control-091420-084139.

[43] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. "PDDLStream: Integrating Symbolic Planners and Blackbox Samplers via Optimistic Adaptive Planning". In: *Proceedings of the International Conference on Automated Planning and Scheduling* 30.1 (June 2020), pp. 440–448. doi: 10.1609/icaps.v30i1.6739.

[44] Roland Geraerts and Mark H. Overmars. "Creating High-quality Paths for Motion Planning". In: *The International Journal of Robotics Research* 26.8 (2007), pp. 845–863. doi: 10.1177/0278364907079280.

[45] Elmer G. Gilbert, Daniel W. Johnson, and Sathiya S. Keerthi. "A fast procedure for computing the distance between complex objects in three-dimensional space". In: *IEEE Journal on Robotics and Automation* 4.2 (1988), pp. 193–203. doi: 10.1109/56.2083.

[46] Michael Görner, Robert Haschke, Helge Ritter, and Jianwei Zhang. "MoveIt! Task Constructor for Task-Level Motion Planning". In: *2019 IEEE International Conference on Robotics and Automation (ICRA)*. 2019, pp. 190–196. doi: 10.1109/ICRA.2019.8793898.

[47] Michael Görner, Norman Hendrich, and Jianwei Zhang. "Pluck and Play: Self-supervised Exploration of Chordophones for Robotic Playing". In: *2024 IEEE International Con-*

ference on Robotics and Automation (ICRA). 2024, pp. 18286–18293. doi: 10.1109/ICRA57147.2024.10610120.

[48] Michael Görner, David Pivin, François Michaud, and Jianwei Zhang. "The MoveIt Benchmark Suite for Whole-Stack Planner Evaluation". In: *Workshop on Evaluating Motion Planning Performance at IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2022.
url: https://motion-planning-workshop.kavrakilab.org/papers/moveit-benchmark.pdf.

[49] Ankur Handa, Karl Van Wyk, Wei Yang, Jacky Liang, Yu-Wei Chao, Qian Wan, Stan Birchfield, Nathan Ratliff, and Dieter Fox. "DexPilot: Vision-Based Teleoperation of Dexterous Robotic Hand-Arm System". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 2020, pp. 9164–9170. doi: 10.1109/ICRA40945.2020.9197124.

[50] Stephen Hart, Paul Dinh, and Kimberly Hambuchen. "The Affordance Template ROS package for robot task programming". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. 2015, pp. 6227–6234. doi: 10.1109/ICRA.2015.7140073.

[51] Jeremias Hartz. "Adaptive Pouring of Liquids Based on Human Motions Using a Robotic Arm". MA thesis. University of Hamburg.

[52] Kris Hauser. "Robust Contact Generation for Robot Simulation with Unstructured Meshes". In: *Robotics Research: The 16th International Symposium ISRR*. Ed. by Masayuki Inaba and Peter Corke. Cham: Springer International Publishing, 2016, pp. 357–373. isbn: 978-3-319-28872-7. doi: 10.1007/978-3-319-28872-7_21.

[53] Kris Hauser. "Task planning with continuous actions and nondeterministic motion planning queries". In: *Proc. of AAAI Workshop on Bridging the Gap between Task and Motion Planning*. 2010.

[54] Kris Hauser and Jean-Claude Latombe. "Multi-modal Motion Planning in Non-expansive Spaces". In: *The International Journal of Robotics Research* 29.7 (2010), pp. 897–915. doi: 10.1177/0278364909352098.

[55] Kris Hauser and Victor Ng-Thow-Hing. "Randomized multi-modal motion planning for a humanoid robot manipulation task". In: *The International Journal of Robotics Research* 30.6 (2011), pp. 678–698. doi: 10.1177/0278364910386985.

[56] Jesse Haviland and Peter Corke. "NEO: A Novel Expeditious Optimisation Algorithm for Reactive Motion Control of Manipulators". In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 1043–1050. doi: 10.1109/LRA.2021.3056060.

[57] Kelsey P. Hawkins. *Analytic Inverse Kinematics for the Universal Robots UR-5/UR-10 Arms*. Technical Report. Institute for Robotics and Intelligent Machines (IRIM). Georgia Institute of Technology, 2013.
URL: http://hdl.handle.net/1853/50782 (visited on 11/17/2024).

[58] Frederik W. Heger and Sanjiv Singh. "Robust robotic assembly through contingencies, plan repair and re-planning". In: *2010 IEEE International Conference on Robotics and Automation (ICRA)*. 2010, pp. 3825–3830. DOI: 10.1109/ROBOT.2010.5509274.

[59] Joachim Hertzberg, Jianwei Zhang, Liwei Zhang, Sebastian Rockel, Bernd Neumann, Jos Lehmann, Krishna S. R. Dubba, Anthony G. Cohn, Alessandro Saffiotti, Federico Pecora, Masoumeh Mansouri, Štefan Konečný, Martin Günther, Sebastian Stock, Luis Seabra Lopes, Miguel Oliveira, Gi Hyun Lim, Hamidreza Kasaei, Vahid Mokhtari, Lothar Hotz, and Wilfried Bohlken. "The RACE Project". In: *KI - Künstliche Intelligenz* 28.4 (Nov. 2014), pp. 297–304. ISSN: 1610-1987. DOI: 10.1007/s13218-014-0327-y.

[60] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. "OctoMap: an efficient probabilistic 3D mapping framework based on octrees". In: *Autonomous Robots* 34.3 (Apr. 2013), pp. 189–206. ISSN: 1573-7527. DOI: 10.1007/s10514-012-9321-0.

[61] David Hsu, Jean-Claude Latombe, and Rajeev Motwani. "Path Planning in Expansive Configuration Spaces". In: *International Journal of Computational Geometry & Applications* 09.04n05 (1999), pp. 495–512. DOI: 10.1142/S0218195999000285.

[62] Tsung-Wei Huang, Dian-Lun Lin, Chun-Xun Lin, and Yibo Lin. "Taskflow: A Lightweight Parallel and Heterogeneous Task Graph Computing System". In: *IEEE Transactions on Parallel and Distributed Systems* 33.6 (2022), pp. 1303–1320. DOI: 10.1109/TPDS.2021.3104255.

[63] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. "Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents". In: *Proceedings of the 39th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato. Vol. 162. Proceedings of Machine Learning Research. PMLR, July 2022, pp. 9118–9147. DOI: 10.48550/arXiv.2201.07207.

[64] Josie A. E. Hughes, Perla Maiolino, and Fumiya Iida. "An anthropomorphic soft skeleton hand exploiting conditional models for piano playing". In: *Science Robotics* 3.25 (2018), eaau3098. DOI: 10.1126/scirobotics.aau3098.

[65] Vladimir Ivan, Yiming Yang, Wolfgang Merkt, Michael P. Camilleri, and Sethu Vijayakumar. "EXOTica: An Extensible Optimization Toolset for Prototyping and Benchmarking Motion Planning and Control". In: *Robot Operating System (ROS): The Complete Reference (Volume 3)*. Ed. by Anis Koubaa. Cham: Springer International Pub-

lishing, 2019, pp. 211–240. isbn: 978-3-319-91590-6. doi: 10.1007/978-3-319-9159 0-6_7.

[66] Lucas Janson, Edward Schmerling, Ashley Clark, and Marco Pavone. “Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions”. In: *The International Journal of Robotics Research* 34.7 (2015). PMID: 27003958, pp. 883–921. doi: 10.1177/0278364915577958.

[67] Shuo Jiang and Lawson L.S. Wong. “Active Tactile Exploration using Shape-Dependent Reinforcement Learning”. In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2022, pp. 8995–9002. doi: 10.1109/IROS47612.2022.99 82266.

[68] Roland S. Johansson and J. Randall Flanagan. “Coding and use of tactile signals from the fingertips in object manipulation tasks”. In: *Nature Reviews Neuroscience* 10.5 (May 2009), pp. 345–359. issn: 1471-0048. doi: 10.1038/nrn2621.

[69] Christian Juelg, Andreas Hermann, Arne Roennau, and Rüdiger Dillmann. “Efficient, Collaborative Screw Assembly in a Shared Workspace”. In: *Intelligent Autonomous Systems 15*. Ed. by Marcus Strand, Rüdiger Dillmann, Emanuele Menegatti, and Stefano Ghidoni. Cham: Springer International Publishing, 2019, pp. 837–848. isbn: 978-3-030-01370-7. doi: 10.1007/978-3-030-01370-7_65.

[70] Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, Peter Pastor, and Stefan Schaal. “STOMP: Stochastic trajectory optimization for motion planning”. In: *2011 IEEE International Conference on Robotics and Automation (ICRA)*. 2011, pp. 4569–4574. doi: 10.1109/ICRA.2011.5980280.

[71] Jay Kamat, Joaquim Ortiz-Haro, Marc Toussaint, Florian T. Pokorny, and Andreas Orthey. “BITKOMO: Combining Sampling and Optimization for Fast Convergence in Optimal Motion Planning”. In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2022, pp. 4492–4497. doi: 10.1109/IROS47612.202 2.9981732.

[72] Ajay Kapur. “A History of robotic Musical Instruments”. In: *Proceedings of the 2005 International Computer Music Conference, ICMC 2005, Barcelona, Spain, September 4-10, 2005*. Michigan Publishing, 2005.
url: http://hdl.handle.net/2027/spo.bbp2372.2005.162.

[73] Sertac Karaman and Emilio Frazzoli. “Sampling-based algorithms for optimal motion planning”. In: *The International Journal of Robotics Research* 30.7 (2011), pp. 846–894. doi: 10.1177/0278364911406761.

[74] Lydia E. Kavraki, Petr Svestka, Jean-Claude Latombe, and Mark H. Overmars. “Probabilistic roadmaps for path planning in high-dimensional configuration spaces”. In: *IEEE*

Transactions on Robotics and Automation 12.4 (1996), pp. 566–580. doi: 10.1109/70.508439.

[75] Jong Wook Kim, Justin Salamon, Peter Li, and Juan Pablo Bello. "Crepe: A Convolutional Representation for Pitch Estimation". In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2018, pp. 161–165. doi: 10.1109/ICASSP.2018.8461329.

[76] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P Foster, Pannag R Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. "OpenVLA: An Open-Source Vision-Language-Action Model". In: *Proceedings of The 8th Conference on Robot Learning*. Ed. by Pulkit Agrawal, Oliver Kroemer, and Wolfram Burgard. Vol. 270. Proceedings of Machine Learning Research. PMLR, June 2025, pp. 2679–2713. doi: 10.48550/arXiv.2406.09246.

[77] Zachary Kingston and Lydia E. Kavraki. "Robowflex: Robot Motion Planning with MoveIt Made Easy". In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2022, pp. 3108–3114. doi: 10.1109/IROS47612.2022.9981698.

[78] Zachary Kingston and Lydia E. Kavraki. "Scaling Multimodal Planning: Using Experience and Informing Discrete Search". In: *IEEE Transactions on Robotics* 39.1 (2023), pp. 128–146. doi: 10.1109/TRO.2022.3197080.

[79] Kilian Kleeberger, Richard Bormann, Werner Kraus, and Marco F. Huber. "A Survey on Learning-Based Robotic Grasping". In: *Current Robotics Reports* 1.4 (Dec. 2020), pp. 239–249. issn: 2662-4087. doi: 10.1007/s43154-020-00021-6.

[80] Torsten Kröger. "Opening the door to new sensor-based robot applications—The Reflexxes Motion Libraries". In: *2011 IEEE International Conference on Robotics and Automation (ICRA)*. 2011, pp. 1–4. doi: 10.1109/ICRA.2011.5980578.

[81] James J. Kuffner and Steven M. LaValle. "RRT-connect: An efficient approach to single-query path planning". In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*. Vol. 2. 2000, 995–1001 vol.2. doi: 10.1109/ROBOT.2000.844730.

[82] Tobias Kunz and Mike Stilman. "Time-Optimal Trajectory Generation for Path Following with Bounded Acceleration and Velocity". In: *Robotics: Science and Systems VIII*. The MIT Press, July 2013. isbn: 9780262315722. doi: 10.7551/mitpress/9816.003.0032.

[83] Luca Lach, Séverin Lemaignan, Francesco Ferro, Helge Ritter, and Robert Haschke. "Bio-Inspired Grasping Controller for Sensorized 2-DoF Grippers". In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2022, pp. 11231–11237. doi: 10.1109/IROS47612.2022.9981819.

[84] Steven LaValle. “Rapidly-exploring random trees: A new tool for path planning”. In: *Research Report 9811* (1998).

[85] Cen-You Li, Barbara Rakitsch, and Christoph Zimmer. “Safe active learning for multi-output gaussian processes”. In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2022, pp. 4512–4551. doi: 10.48550/arXiv.2203.14849.

[86] Ning Li and Dandan Li. “Acoustic Measurement and Modeling of the Traditional Chinese Instrument Guzheng in Digital Transformation: A Case Study of Spectral and Resonance Analysis of Standard Pitch A440”. In: *Open Journal of Acoustics* 12 (Jan. 2024), pp. 17–30. doi: 10.4236/oja.2024.122002.

[87] Hongzhuo Liang, Xiaojian Ma, Shuang Li, Michael Görner, Song Tang, Bin Fang, Fuchun Sun, and Jianwei Zhang. “PointNetGPD: Detecting Grasp Configurations from Point Sets”. In: *2019 International Conference on Robotics and Automation (ICRA)*. 2019, pp. 3629–3635. doi: 10.1109/ICRA.2019.8794435.

[88] Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. “Flow Matching for Generative Modeling”. In: *The Eleventh International Conference on Learning Representations*. 2023. doi: 10.48550/arXiv.2210.02747.

[89] Shuai Liu and Pengcheng Liu. “Benchmarking and optimization of robot motion planning with motion planning pipeline”. In: *The International Journal of Advanced Manufacturing Technology* 118.3 (Jan. 2022), pp. 949–961. issn: 1433-3015. doi: 10.1007/s00170-021-07985-5.

[90] Yuchen Liu, Luigi Palmieri, Sebastian Koch, Ilche Georgievski, and Marco Aiello. “Delta: Decomposed efficient long-term robot task planning using large language models”. In: *2025 IEEE International Conference on Robotics and Automation (ICRA)*. To Appear. May 2025. doi: 10.48550/arXiv.2404.03275.

[91] Zeyi Liu, Arpit Bahety, and Shuran Song. “REFLECT: Summarizing Robot Experiences for Failure Explanation and Correction”. In: *7th Annual Conference on Robot Learning*. 2023.
url: https://openreview.net/forum?id=8yTS_nAILxt.

[92] Kevin M. Lynch and Frank C. Park. *Modern Robotics: Mechanics, Planning, and Control*. 1st. USA: Cambridge University Press, 2017. isbn: 1107156300.

[93] Jianzhi Lyu, Philipp Ruppel, Norman Hendrich, Shuang Li, Michael Görner, and Jianwei Zhang. “Efficient and Collision-Free Human–Robot Collaboration Based on Intention and Trajectory Prediction”. In: *IEEE Transactions on Cognitive and Developmental Systems* 15.4 (2023), pp. 1853–1863. doi: 10.1109/TCDS.2022.3215093.

[94] Jeffrey Mahler, Matthew Matl, Vishal Satish, Michael Danielczuk, Bill DeRose, Stephen McKinley, and Ken Goldberg. "Learning ambidextrous robot grasping policies". In: *Science Robotics* 4.26 (2019), eaau4984. DOI: 10.1126/scirobotics.aau4984.

[95] Tobia Marcucci, Mark Petersen, David von Wrangel, and Russ Tedrake. "Motion planning around obstacles with convex optimization". In: *Science Robotics* 8.84 (2023), eadf7843. DOI: 10.1126/scirobotics.adf7843.

[96] Carlos Mastalli, Rohan Budhiraja, Wolfgang Merkt, Guilhem Saurel, Bilal Hammoud, Maximilien Naveau, Justin Carpentier, Ludovic Righetti, Sethu Vijayakumar, and Nicolas Mansard. "Crocoddyl: An Efficient and Versatile Framework for Multi-Contact Optimal Control". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 2020, pp. 2536–2542. DOI: 10.1109/ICRA40945.2020.9196673.

[97] Yutaka Matsuo, Yann LeCun, Maneesh Sahani, Doina Precup, David Silver, Masashi Sugiyama, Eiji Uchibe, and Jun Morimoto. "Deep learning, reinforcement learning, and world models". In: *Neural Networks* 152 (2022), pp. 267–275. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2022.03.037.

[98] Przemyslaw Mazurek and Dorota Oszutowska-Mazurek. "String Plucking and Touching Sensing using Transmissive Optical Sensors for Guzheng". In: *16th International Conference on Control, Automation, Robotics and Vision (ICARCV)*. 2020, pp. 1143–1149. DOI: 10.1109/ICARCV50220.2020.9305480.

[99] Joseph Mirabel, Steve Tonneau, Pierre Fernbach, Anna-Kaarina Seppälä, Mylène Campana, Nicolas Mansard, and Florent Lamiraux. "HPP: A new software for constrained motion planning". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2016, pp. 383–389. DOI: 10.1109/IROS.2016.7759083.

[100] Mark Moll, Ioan A. Şucan, and Lydia E. Kavraki. "Benchmarking motion planning algorithms: An extensible infrastructure for analysis and visualization". In: *IEEE Robotics & Automation Magazine* 22.3 (Sept. 2015), pp. 96–102. DOI: 10.1109/MRA.2015.2448276.

[101] Louis Montaut, Quentin Le Lidec, Vladimír Petrík, Josef Sivic, and Justin Carpentier. "Collision Detection Accelerated: An Optimization Perspective". In: *Proceedings of Robotics: Science and Systems*. New York City, NY, USA, June 2022. DOI: 10.15607/RSS.2022.XVIII.039.

[102] Andrew S. Morgan, Kaiyu Hang, Walter G. Bircher, Fadi M. Alladkani, Abhinav Gandhi, Berk Calli, and Aaron M. Dollar. "Benchmarking cluttered robot pick-and-place manipulation with the box and blocks test". In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 454–461. DOI: 10.1109/LRA.2019.2961053.

[103] Mina Mounir, Peter Karsmakers, and Toon van Waterschoot. "CNN-based Note Onset Detection using Synthetic Data Augmentation". In: *2020 28th European Signal Pro-*

cessing Conference (EUSIPCO). 2021, pp. 171–175. DOI: 10.23919/Eusipco47968.2020.9287621.

[104] Mina Mounir, Peter Karsmakers, and Toon van Waterschoot. “Musical note onset detection based on a spectral sparsity measure”. In: *EURASIP Journal on Audio, Speech, and Music Processing* 2021 (July 2021), p. 30. DOI: 10.1186/s13636-021-00214-7.

[105] Mustafa Mukadam, Jing Dong, Xinyan Yan, Frank Dellaert, and Byron Boots. “Continuous-time Gaussian process motion planning via probabilistic inference”. In: *The International Journal of Robotics Research* 37.11 (2018), pp. 1319–1340. DOI: 10.1177/0278364918790369.

[106] Shohin Mukherjee, Sandip Aine, and Maxim Likhachev. “MPLP: Massively Parallelized Lazy Planning”. In: *IEEE Robotics and Automation Letters* 7.3 (2022), pp. 6067–6074. DOI: 10.1109/LRA.2022.3157544.

[107] Jim Murphy, James McVay, Paul Mathews, Dale A. Carnegie, and Ajay Kapur. “Expressive Robotic Guitars: Developments in Musical Robotics for Chordophones”. In: *Computer Music Journal* 39.1 (2015), pp. 59–73. DOI: 10.1162/COMJ_a_00285.

[108] Sankaranarayanan Natarajan and Frank Kirchner. “Multi-Modal Manipulation Planning for an Upper-Torso Humanoid System”. In: *2022 IEEE-RAS 21st International Conference on Humanoid Robots (Humanoids)*. 2022, pp. 134–140. DOI: 10.1109/Humanoids53995.2022.10000144.

[109] Rhys Newbury, Morris Gu, Lachlan Chumbley, Arsalan Mousavian, Clemens Eppner, Jürgen Leitner, Jeannette Bohg, Antonio Morales, Tamim Asfour, Danica Kragic, Dieter Fox, and Akansel Cosgun. “Deep Learning Approaches to Grasp Synthesis: A Review”. In: *IEEE Transactions on Robotics* 39.5 (2023), pp. 3994–4015. DOI: 10.1109/TRO.2023.3280597.

[110] Giang Hoang Nguyen, Daniel Beßler, Simon Stelter, Mihai Pomarlan, and Michael Beetz. “Translating Universal Scene Descriptions into Knowledge Graphs for Robotic Environment”. In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. 2024, pp. 9389–9395. DOI: 10.1109/ICRA57147.2024.10611691.

[111] Karin Nieuwenhuizen, Lei Liu, Robert van Liere, and Jean-Bernard Martens. “Insights from Dividing 3D Goal-Directed Movements into Meaningful Phases”. In: *IEEE Computer Graphics and Applications* 29.6 (2009), pp. 44–53. DOI: 10.1109/MCG.2009.121.

[112] Takumi Ogata and Gil Weinberg. “Robotically Augmented Electric Guitar for Shared Control”. In: *Proceedings of the International Conference on New Interfaces for Musical Expression*. Copenhagen, Denmark: Aalborg University Copenhagen, 2017, pp. 487–488. DOI: 10.5281/zenodo.1176326.

[113] Andreas ten Pas, Marcus Gualtieri, Kate Saenko, and Robert Platt. “Grasp Pose Detection in Point Clouds”. In: *The International Journal of Robotics Research* 36.13-14 (2017), pp. 1455–1473. DOI: 10.1177/0278364917735594.

[114] Fabian Peller-Konrad, Rainer Kartmann, Christian R.G. Dreher, Andre Meixner, Fabian Reister, Markus Grotz, and Tamim Asfour. “A memory system of a robot cognitive architecture and its implementation in ArmarX”. In: *Robotics and Autonomous Systems* 164 (2023), p. 104415. ISSN: 0921-8890. DOI: https://doi.org/10.1016/j.robot.2023.104415.

[115] Louis Petit and Alexis Lussier Desbiens. “RRT-Rope: A deterministic shortening approach for fast near-optimal path planning in large-scale uncluttered 3D environments”. In: *2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. 2021, pp. 1111–1118. DOI: 10.1109/SMC52423.2021.9659071.

[116] Hung Pham and Quang Cuong Pham. “A New Approach to Time-Optimal Path Parameterization Based on Reachability Analysis”. In: *IEEE Transactions on Robotics* 34 (June 2018), pp. 645–659. DOI: 10.1109/TRO.2018.2819195.

[117] Lerrel Pinto and Abhinav Gupta. “Supersizing self-supervision: Learning to grasp from 50K tries and 700 robot hours”. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. 2016, pp. 3406–3413. DOI: 10.1109/ICRA.2016.7487517.

[118] David Pivin. “Pipette automatisée et pince comme effecteurs d’un bras robotique collaboratif pour l’automatisation de procédés expérimentaux en biologie synthétique”. MA thesis. Université de Sherbrooke, 2022.

[119] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, Nov. 2005. ISBN: 9780262256834. DOI: 10.7551/mitpress/3206.001.0001.

[120] Stéphane Redon, Abderrahmane Kheddar, and Sabine Coquillart. “Fast Continuous Collision Detection between Rigid Bodies”. In: *Computer Graphics Forum* 21.3 (2002), pp. 279–287. DOI: https://doi.org/10.1111/1467-8659.t01-1-00587.

[121] Stephane Redon, Ming C. Lin, Dinesh Manocha, and Young J. Kim. “Fast Continuous Collision Detection for Articulated Models”. In: *Journal of Computing and Information Science in Engineering* 5.2 (Feb. 2005), pp. 126–137. ISSN: 1530-9827. DOI: 10.1115/1.1884133.

[122] Maximo A. Roa, Mehmet R. Dogar, Jordi Pages, Carlos Vivas, Antonio Morales, Nikolaus Correll, Michael Görner, Jan Rosell, Sergi Foix, Raphael Memmesheimer, and Francesco Ferro. “Mobile Manipulation Hackathon: Moving into Real World Applications”. In: *IEEE Robotics & Automation Magazine* 28.2 (2021), pp. 112–124. DOI: 10.1109/MRA.2021.3061951.

[123] Philipp Ruppel, Norman Hendrich, Sebastian Starke, and Jianwei Zhang. “Cost Functions to Specify Full-Body Motion and Multi-Goal Manipulation Tasks”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018, pp. 3152–3159. doi: 10.1109/ICRA.2018.8460799.

[124] Francesco Ruscelli, Arturo Laurenzi, Nikos G. Tsagarakis, and Enrico Mingo Hoffman. “Horizon: A Trajectory Optimization Framework for Robotic Systems”. In: *Frontiers in Robotics and AI* 9 (2022). issn: 2296-9144. doi: 10.3389/frobt.2022.899025.

[125] Gildardo Sánchez and Jean-Claude Latombe. “A Single-Query Bi-Directional Probabilistic Roadmap Planner with Lazy Collision Checking”. In: *Robotics Research*. Ed. by Raymond Austin Jarvis and Alexander Zelinsky. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 403–417. isbn: 978-3-540-36460-3. doi: 10.1007/3-540-36460-9_27.

[126] Philipp S. Schmitt, Werner Neubauer, Wendelin Feiten, Kai M. Wurm, Georg V. Wichert, and Wolfram Burgard. “Optimal, sampling-based manipulation planning”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 3426–3432. doi: 10.1109/ICRA.2017.7989390.

[127] Tim Schneider, Boris Belousov, Georgia Chalvatzaki, Diego Romeres, Devesh K. Jha, and Jan Peters. “Active Exploration for Robotic Manipulation”. In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2022, pp. 9355–9362. doi: 10.1109/IROS47612.2022.9982061.

[128] Christian Schörkhuber and Anssi Klapuri. “Constant-Q transform toolbox for music processing”. In: *7th sound and music computing conference, Barcelona, Spain*. 2010, pp. 3–64.

[129] Jens Schreiter, Duy Nguyen-Tuong, Mona Eberts, Bastian Bischoff, Heiner Markert, and Marc Toussaint. “Safe Exploration for Active Learning with Gaussian Processes”. In: *Machine Learning and Knowledge Discovery in Databases*. Ed. by Albert Bifet, Michael May, Bianca Zadrozny, Ricard Gavalda, Dino Pedreschi, Francesco Bonchi, Jaime Cardoso, and Myra Spiliopoulou. Cham: Springer International Publishing, 2015, pp. 133–149. isbn: 978-3-319-23461-8. doi: 10.1007/978-3-319-23461-8_9.

[130] John Schulman, Yan Duan, Jonathan Ho, Alex Lee, Ibrahim Awwal, Henry Bradlow, Jia Pan, Sachin Patil, Ken Goldberg, and Pieter Abbeel. “Motion planning with sequential convex optimization and convex collision checking”. In: *The International Journal of Robotics Research* 33.9 (2014), pp. 1251–1270. doi: 10.1177/0278364914528132.

[131] Keisuke Shirai, Cristian C. Beltran-Hernandez, Masashi Hamaya, Atsushi Hashimoto, Shohei Tanaka, Kento Kawaharazuka, Kazutoshi Tanaka, Yoshitaka Ushiku, and Shinsuke Mori. “Vision-Language Interpreter for Robot Task Planning”. In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. 2024, pp. 2051–2058. doi: 10.1109/ICRA57147.2024.10611112.

[132] Jeremy Siburian, Cristian C. Beltran-Hernandez, and Masashi Hamaya. "Practical Task and Motion Planning for Robotic Food Preparation". In: *2025 IEEE/SICE International Symposium on System Integration (SII)*. 2025, pp. 1229–1234. doi: 10.1109/SII59315.2025.10870896.

[133] Tarek M. Sobh, Bei Wang, and Kurt W. Coble. "Experimental Robot Musicians". In: *Journal of Intelligent and Robotic Systems* 38.2 (Oct. 2003), pp. 197–212. issn: 1573-0409. doi: 10.1023/A:1027319831986.

[134] Junlin Song, Pedro J. Sanchez-Cuevas, and Miguel Olivares-Mendez. "Towards Online System Identification: Benchmark of Model Identification Techniques for Variable Dynamics UAV Applications". In: *2022 International Conference on Unmanned Aircraft Systems (ICUAS)*. 2022, pp. 590–598. doi: 10.1109/ICUAS54217.2022.9836134.

[135] Siddharth Srivastava, Eugene Fang, Lorenzo Riano, Rohan Chitnis, Stuart Russell, and Pieter Abbeel. "Combined task and motion planning through an extensible planner-independent interface layer". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. 2014, pp. 639–646. doi: 10.1109/ICRA.2014.6906922.

[136] Joseph A. Starek, Javier V. Gomez, Edward Schmerling, Lucas Janson, Luis Moreno, and Marco Pavone. "An asymptotically-optimal sampling-based algorithm for Bi-directional motion planning". In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2015, pp. 2072–2078. doi: 10.1109/IROS.2015.7353652.

[137] Simon Stelter, Georg Bartels, and Michael Beetz. "An open-source motion planning framework for mobile manipulators using constraint-based task space control with linear MPC". In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2022, pp. 1671–1678. doi: 10.1109/IROS47612.2022.9982245.

[138] Marlin P Strub and Jonathan D Gammell. "Adaptively Informed Trees (AIT*) and Effort Informed Trees (EIT*): Asymmetric bidirectional sampling-based path planning". In: *The International Journal of Robotics Research* 41.4 (2022), pp. 390–417. doi: 10.1177/02783649211069572.

[139] Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. "The Open Motion Planning Library". In: *IEEE Robotics & Automation Magazine* 19.4 (Dec. 2012), pp. 72–82. doi: 10.1109/MRA.2012.2205651.

[140] Balakumar Sundaralingam, Siva Kumar Sastry Hari, Adam Fishman, Caelan Garrett, Karl Van Wyk, Valts Blukis, Alexander Millane, Helen Oleynikova, Ankur Handa, Fabio Ramos, Nathan Ratliff, and Dieter Fox. "CuRobo: Parallelized Collision-Free Robot Motion Generation". In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. 2023, pp. 8112–8119. doi: 10.1109/ICRA48891.2023.10160765.

[141] Stone Tao, Fanbo Xiang, Arth Shukla, Yuzhe Qin, Xander Hinrichsen, Xiaodi Yuan, Chen Bao, Xinsong Lin, Yulin Liu, Tse-kai Chan, Yuan Gao, Xuanlin Li, Tongzhou

Mu, Nan Xiao, Arnav Gurha, Zhiao Huang, Roberto Calandra, Rui Chen, Shan Luo, and Hao Su. *ManiSkill3: GPU Parallelized Robotics Simulation and Rendering for Generalizable Embodied AI*. 2024. doi: 10.48550/arXiv.2410.00425.

[142] Wil Thomason, Zachary Kingston, and Lydia E. Kavraki. "Motions in Microseconds via Vectorized Sampling-Based Planning". In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. 2024, pp. 8749–8756. doi: 10.1109/ICRA57147.2024.10611190.

[143] Wil Thomason, Marlin P. Strub, and Jonathan D. Gammell. "Task and Motion Informed Trees (TMIT*): Almost-Surely Asymptotically Optimal Integrated Task and Motion Planning". In: *IEEE Robotics and Automation Letters* 7.4 (2022), pp. 11370–11377. doi: 10.1109/LRA.2022.3199676.

[144] Marc Toussaint, Kelsey Allen, Kevin Smith, and Joshua Tenenbaum. "Differentiable Physics and Stable Modes for Tool-Use and Manipulation Planning". In: *Proceedings of Robotics: Science and Systems*. Pittsburgh, Pennsylvania, June 2018. doi: 10.15607/RSS.2018.XIV.044.

[145] Marc Toussaint, Joaquim Ortiz-Haro, Valentin N. Hartmann, Erez Karpas, and Wolfgang Hönig. "Effort Level Search in Infinite Completion Trees with Application to Task-and-Motion Planning". In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. 2024, pp. 14902–14908. doi: 10.1109/ICRA57147.2024.10611722.

[146] Cuong Trinh, Dimiter Zlatanov, Matteo Zoppi, and Rezia Molfino. "A Geometrical Approach to the Inverse Kinematics of 6R Serial Robots With Offset Wrists". In: vol. 5C: 39th Mechanisms and Robotics Conference. International Design Engineering Technical Conferences and Computers and Information in Engineering Conference. Aug. 2015, V05CT08A016. doi: 10.1115/DETC2015-47950.

[147] Nikolaus Vahrenkamp, Tamim Asfour, Giorgio Metta, Giulio Sandini, and Rüdiger Dillmann. "Manipulability analysis". In: *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*. 2012, pp. 568–573. doi: 10.1109/HUMANOIDS.2012.6651576.

[148] Christopher Waltham. "An Acoustical Comparison of East Asian and Western String Instruments". In: *Proceedings Intl. Symposium on Musical Acoustics (ISMA 2014), Le Mans, France*. 2014, pp. 375–380. url: http://www.conforg.fr/isma2014/cdrom/data/articles/000144.pdf.

[149] Yeping Wang, Pragathi Praveena, Daniel Rakita, and Michael Gleicher. "RangedIK: An Optimization-based Robot Motion Generation Method for Ranged-Goal Tasks". In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. 2023, pp. 9700–9706. doi: 10.1109/ICRA48891.2023.10161311.

[150] Ziyi Wang and Yin Cao. “An On-line Algorithm for Music-to-Score Alignment of Guzheng Performance”. In: *2018 IEEE 23rd International Conference on Digital Signal Processing (DSP)*. 2018, pp. 1–5. doi: 10.1109/ICDSP.2018.8631834.

[151] Gil Weinberg, Mason Brean, Guy Hoffman, and Scott Driscoll. *Robotic Musicianship: Embodied Artificial Creativity and Mechatronic Musical Expression*. Springer, 2020, p. 270. doi: 10.1007/978-3-030-38930-7.

[152] Nicholas Wettels, Jeremy A. Fishel, and Gerald E. Loeb. “Multimodal Tactile Sensor”. In: *The Human Hand as an Inspiration for Robot Hand Development*. Ed. by Ravi Balasubramanian and Veronica J. Santos. Cham: Springer International Publishing, 2014, pp. 405–429. isbn: 978-3-319-03017-3. doi: 10.1007/978-3-319-03017-3_19.

[153] Tyler S Wilson, Wil Thomason, Zachary Kingston, Lydia E Kavraki, and Jonathan D Gammell. “Nearest-neighbourless asymptotically optimal motion planning with Fully Connected Informed Trees (FCIT*)”. In: *2025 IEEE International Conference on Robotics and Automation (ICRA)*. To Appear. May 2025. doi: 10.48550/arXiv.2411.17902.

[154] William A. Wolovich and Howard Elliott. “A computational technique for inverse kinematics”. In: *The 23rd IEEE Conference on Decision and Control*. 1984, pp. 1359–1363. doi: 10.1109/CDC.1984.272258.

[155] Jimmy Wu, Rika Antonova, Adam Kan, Marion Lepert, Andy Zeng, Shuran Song, Jeannette Bohg, Szymon Rusinkiewicz, and Thomas Funkhouser. “TidyBot: personalized robot assistance with large language models”. In: *Autonomous Robots* 47.8 (Dec. 2023), pp. 1087–1102. issn: 1573-7527. doi: 10.1007/s10514-023-10139-z.

[156] Philipp Wu, Alejandro Escontrela, Danijar Hafner, Pieter Abbeel, and Ken Goldberg. “DayDreamer: World Models for Physical Robot Learning”. In: *Proceedings of The 6th Conference on Robot Learning*. Ed. by Karen Liu, Dana Kulic, and Jeff Ichnowski. Vol. 205. Proceedings of Machine Learning Research. PMLR, Dec. 2023, pp. 2226–2240. doi: 10.48550/arXiv.2206.14176.

[157] Philipp Wu, Yide Shentu, Zhongke Yi, Xingyu Lin, and Pieter Abbeel. “GELLO: A General, Low-Cost, and Intuitive Teleoperation Framework for Robot Manipulators”. In: *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2024, pp. 12156–12163. doi: 10.1109/IROS58592.2024.10801581.

[158] Qingyang Xi, Rachel M. Bittner, Johan Pauwels, Xuzhou Ye, and Juan P. Bello. “Guitarset: A dataset for guitar transcription”. In: *Proceedings of the 19th International Society for Music Information Retrieval Conference, ISMIR 2018*. 2018, pp. 453–460.

[159] Deng Xiaowei, Yu Zhengyue, Yao Weiping, and Chen Minjie. “Simulation Analysis of Vibro-Acoustic Characteristics of Traditional Guzheng”. In: *Journal of Shanghai Jiaotong University* 50.02, 300 (2016), p. 300. doi: 10.16183/j.cnki.jsjtu.2016.02.024.

[160] Huazhe Xu, Yuping Luo, Shaoxiong Wang, Trevor Darrell, and Roberto Calandra. "Towards Learning to Play Piano with Dexterous Hands and Touch". In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2022, pp. 10410–10416. DOI: 10.1109/IROS47612.2022.9981221.

[161] Kevin Zakka, Baruch Tabanpour, Qiayuan Liao, Mustafa Haiderbhai, Samuel Holt, Jing Yuan Luo, Arthur Allshire, Erik Frey, Koushil Sreenath, Lueder A. Kahrs, Carmelo Sferrazza, Yuval Tassa, and Pieter Abbeel. "MuJoCo Playground". In: *arXiv preprint* (2025). DOI: 10.48550/arXiv.2502.08844.

[162] Kevin Zakka, Philipp Wu, Laura Smith, Nimrod Gileadi, Taylor Howell, Xue Bin Peng, Sumeet Singh, Yuval Tassa, Pete Florence, Andy Zeng, and Pieter Abbeel. "RoboPianist: Dexterous Piano Playing with Deep Reinforcement Learning". In: *Proceedings of The 7th Conference on Robot Learning*. Ed. by Jie Tan, Marc Toussaint, and Kourosh Darvish. Vol. 229. Proceedings of Machine Learning Research. PMLR, Nov. 2023, pp. 2975–2994. DOI: arXiv:2304.04150.

[163] Andy Zeng, Shuran Song, Stefan Welker, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser. "Learning Synergies Between Pushing and Grasping with Self-Supervised Deep Reinforcement Learning". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2018, pp. 4238–4245. DOI: 10.1109/IROS.2018.8593986.

[164] Ada Zhang, Mark Malhotra, and Yoky Matsuoka. "Musical piano performance by the ACT Hand". In: *2011 IEEE International Conference on Robotics and Automation (ICRA)*. 2011, pp. 3536–3541. DOI: 10.1109/ICRA.2011.5980342.

[165] Enda Zhang, Gopal Gupta, Charles Greif, and Andrew Paplinski. "An Efficient Modal-based Approach Towards Guzheng Sound Synthesis". In: *arXiv preprint* (2019). DOI: 10.48550/arXiv.1910.05447.

[166] Tianhao Zhang, Zoe McCarthy, Owen Jow, Dennis Lee, Xi Chen, Ken Goldberg, and Pieter Abbeel. "Deep Imitation Learning for Complex Manipulation Tasks from Virtual Reality Teleoperation". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018, pp. 5628–5635. DOI: 10.1109/ICRA.2018.8461249.

[167] Ruohua Zhou and Josh D. Reiss. "Music Onset Detection". In: *Machine Audition: Principles, Algorithms and Systems*. Ed. by Wenwu Wang. Hershey, PA, USA: IGI Global, 2011, pp. 297–316. ISBN: 9781615209194. DOI: 10.4018/978-1-61520-919-4.ch012.

[168] Matt Zucker, Nathan Ratliff, Anca D. Dragan, Mihail Pivtoraiko, Matthew Klingensmith, Christopher M. Dellin, J. Andrew Bagnell, and Siddhartha S. Srinivasa. "CHOMP: Covariant Hamiltonian optimization for motion planning". In: *The International Journal of Robotics Research* 32.9-10 (2013), pp. 1164–1193. DOI: 10.1177/0278364913488805.

Online-References

[W1] *Actin: Software for Robotics Simulation and Control.*
URL: https://outgoing.energid.info/documentation/actin_7.1.x/ (visited on 02/20/2025).

[W2] Levi Armstrong and the Tesseract Development Team. *Tesseract - Motion Planning Environment.*
URL: https://github.com/tesseract-robotics/tesseract (visited on 08/13/2024).

[W3] *Benchmark: a microbenchmark support library.* 2022.
URL: https://github.com/google/benchmark (visited on 09/23/2022).

[W4] Sebastian Castro. *PyRoboPlan: Educational Python library for manipulator motion planning.*
URL: https://github.com/sea-bass/pyroboplan (visited on 09/27/2024).

[W5] Boston Cleek. *Deep Grasp Pose.*
URL: https://github.com/PickNikRobotics/deep_grasp_demo (visited on 12/04/2024).

[W6] *Intrinsic Flowstate.*
URL: https://www.intrinsic.ai/flowstate (visited on 02/20/2025).

[W7] *KUKA.SystemSoftware.*
URL: https://www.kuka.com/en-de/products/robot-systems/software/system-software/kuka_systemsoftware (visited on 03/03/2025).

[W8] *MIKADO: camera-guided robotics.*
URL: https://www.optonic.com/en/brands/mikado/ (visited on 02/20/2025).

[W9] MoveIt Maintainer Group. *The MoveIt planning framework.* 2024.
URL: http://moveit.ros.org (visited on 01/15/2024).

[W10] *MoveIt Pro: Robotics Developer Platform.*
URL: https://picknik.ai/pro/ (visited on 02/20/2025).

[W11] *moveit_ros_manipulation - Components of MoveIt used for manipulation.*
URL: https://github.com/moveit/moveit/tree/master/moveit_ros/manipulation (visited on 11/03/2024).

[W12] *MuJoCo - Advanced Physics Simulation.*
URL: http://www.mujoco.org (visited on 01/04/2025).

[W13] *Omniverse Platform for OpenUSD.*
URL: https://www.nvidia.com/en-us/omniverse/ (visited on 01/17/2025).

[W14] Scott Paulin. *Elbow joint self-collisions break path planning when full joint space is used on UR5.*
URL: https://github.com/ros-industrial/universal_robot/issues/265 (visited on 12/18/2024).

[W15] David Pivin. *Google Summer of Code - Creation of a Benchmark Suite.*
URL: https://summerofcode.withgoogle.com/archive/2021/projects/6662801057120256 (visited on 06/10/2024).

[W16] *Polyscope 5.*
URL: https://www.universal-robots.com/products/polyscope-5/ (visited on 03/03/2025).

[W17] *rviz - 3D visualization tool for ROS.*
URL: http://wiki.ros.org/rviz (visited on 11/03/2024).

[W18] *SDFormat: Simulation Description Format.*
URL: http://sdformat.org (visited on 01/04/2025).

[W19] Russ Tedrake and the Drake Development Team. *Drake: Model-based design and verification for robotics.*
URL: https://drake.mit.edu (visited on 01/15/2024).

[W20] *The Orocos Project.*
URL: https://orocos.org (visited on 09/23/2024).

[W21] *urdf.*
URL: http://wiki.ros.org/urdf (visited on 02/16/2025).

[W22] *YAML: Anchors and Aliases.*
URL: https://yaml.org/spec/1.2.2/#3222-anchors-and-aliases (visited on 12/18/2024).

[W23] *YAML: YAML Ain't Markup Language.*
URL: https://yaml.org (visited on 12/18/2024).

[W24] Kevin Zakka, Yuval Tassa, and MuJoCo Menagerie Contributors. *MuJoCo Menagerie: A collection of high-quality simulation models for MuJoCo.*
URL: http://github.com/google-deepmind/mujoco_menagerie (visited on 12/18/2024).

Appendix A

Prior Publications

This dissertation is partly based on prior publications, with each relevant publication cited in the individual chapters. For a complete overview, the following lists all my peer-reviewed publications, sorted chronologically, over the course of this dissertation.

Journal Papers

- Jianzhi Lyu, Alexander Maye, *Michael Görner*, Philipp Ruppel, Andreas K. Engel and Jianwei Zhang, Coordinating human-robot collaboration by EEG-based human intention prediction and vigilance control, Frontiers in Neurorobotics, 2022.

- Jianzhi Lyu, Philipp Ruppel, Norman Hendrich, Shuang Li, *Michael Görner*, and Jianwei Zhang, Efficient and Collision-Free Human-Robot Collaboration Based on Intention and Trajectory Prediction, IEEE Transactions on Cognitive and Developmental Systems, 2022.

- Lin Cong, Hongzhuo Liang, Philipp Ruppel, Yunlei Shi, *Michael Görner*, Norman Hendrich, and Jianwei Zhang, Reinforcement Learning with Vision-Proprioception Model for Robot Planar Pushing, Frontiers in Neurorobotics, Vol 16, 2022.

- Máximo A. Roa, Mehmet Dogar, Jordi Pages, Carlos Vivas, Antonio Morales, Nikolaus Correll, *Michael Görner*, Jan Rosell, Sergi Foix, Raphael Memmesheimer, and Francesco Ferro, Mobile Manipulation Hackathon: Moving Into Real-World Applications, IEEE Robotics & Automation Magazine, March 2021, p. 2–14.

- Focko L. Higgen, Philipp Ruppel, *Michael Görner*, Matthias Kerzel, Norman Hendrich, Jan Feldheim, Stefan Wermter, Jianwei Zhang, and Christian Gerloff, Crossmodal Pattern Discrimination in Humans and Robots: A Visuo-Tactile Case Study, Frontiers in Robotics and AI 2020, Volume 7.

Conference Publications

- Manuel Gomes, *Michael Görner*, Miguel Riem Oliveira, Jianwei Zhang, Sensor-agnostic Visuo-Tactile Robot Calibration Exploiting Assembly-Precision Model Geometries, International Conference on Intelligent Robots and Systems, IROS 2024, Abu Dhabi.
- *Michael Görner*, Norman Hendrich, Jianwei Zhang, Pluck and Play: Self-supervised Exploration of Chordophones for Robotic Playing, International Conference on Robotics and Automation, ICRA 2024, Yokohama, Japan. Reference [47]
- Björn Sygo, Shang-Ching Liu, Fabian Wieczorek, Mykhailo Koshil, *Michael Görner*, Norman Hendrich, Jianwei Zhang, Multi-Stage Book Perception and Bimanual Manipulation for Rearranging Book Shelves, The 18th International Conference on Intelligent Autonomous Systems, IAS-18 2023, Suwon, Korea.
- Niklas Fiedler, Jasper Güldenstein, Theresa Naß, *Michael Görner*, Norman Hendrich, Jianwei Zhang, A Multimodal Robotic Blackjack Dealer: Design, Implementation, and Reliability Analysis, The 18th International Conference on Intelligent Autonomous Systems, IAS-18 2023, Suwon, Korea.
- Philipp Ruppel, *Michael Görner*, Norman Hendrich, and Jianwei Zhang, Detection and Reconstruction of Transparent Objects with Infrared Projection-based RGB-D Cameras, International Conference on Cognitive Systems and Information Processing, ICCSIP 2020, Zhuhai, China.
- Lin Cong, *Michael Görner*, Philipp Ruppel, Hongzhuo Liang, Norman Hendrich, and Jianwei Zhang, Self-Adapting Recurrent Models for Object Pushing from Learning in Simulation, International Conference on Intelligent Robots and Systems, IROS 2020, Las Vegas, USA.
- Ronja Güldenring, *Michael Görner*, Norman Hendrich, Niels Jul Jacobsen, and Jianwei Zhang, Learning Local Planners for Human-aware Navigation in Indoor Environments, International Conference on Intelligent Robots and Systems, IROS 2020, Las Vegas, USA.
- *Michael Görner*[*], Robert Haschke[*], Helge Ritter, and Jianwei Zhang, MoveIt! Task Constructor for Task-Level Motion Planning, International Conference on Robotics and Automation, ICRA 2019, Montreal, Canada. Reference [46]
- Hongzhuo Liang[*], Xiaojian Ma[*], Shuang Li, *Michael Görner*, Song Tang, Bin Fang, Fuchun Sun, and Jianwei Zhang, PointNetGPD: Detecting Grasp Configurations from Point Sets, International Conference on Robotics and Automation, ICRA 2019, Montreal, Canada. Reference [87]

- Shuang Li[*], Xiaojian Ma[*], Hongzhuo Liang, *Michael Görner*, Philipp Ruppel, Bin Fang, Fuchun Sun, and Jianwei Zhang, Vision-based Teleoperation of Shadow Dexterous Hand using End-to-End Deep Neural Network, International Conference on Robotics and Automation, ICRA 2019, Montreal, Canada.

- Dennis Krupke, Frank Steinicke, Paul Lubos, Yannick Jonetzko, *Michael Görner* and Jianwei Zhang, Comparison of Multimodal Heading and Pointing Gestures for Co-Located Mixed Reality Human-Robot Interaction, International Conference on Intelligent Robots and Systems, IROS 2018, Madrid, Spain.

- Philipp Ruppel, Yannick Jonetzko, *Michael Görner*, Norman Hendrich and Jianwei Zhang, Simulation of the SynTouch BioTac Sensor, The 15th International Conference on Intelligent Autonomous Systems, IAS-15 2018, Baden-Baden, Germany.

Other Publications

- *Michael Görner*[*], David Pivin[*], Francois Michaud, and Jianwei Zhang, The MoveIt Benchmark Suite for Whole-Stack Planner Evaluation, Workshop on Evaluating Motion Planning Performance at International Conference on Intelligent Robots and Systems, IROS 2022, Kyoto, Japan. Reference [48]

- *Michael Görner*, What's Past the End? The Future of ROS One, ROSCon-2022, October 19–21, 2022, Kyoto, Japan.

- Focko Higgen, Philipp Ruppel, *Michael Görner*, Matthias Kerzel, Sven Magg, and Norman Hendrich, Crossmodal Pattern Discrimination in Humans and Robots: A Visuo-Tactile Case Study, Workshop on Crossmodal Learning for Intelligent Robotics at International Conference on Intelligent Robots and Systems, IROS 2018, Madrid, Spain.

- *Michael Görner*[*], Lars Henning Kayser[*], Matthias Kerzel, Stefan Wermter and Jianwei Zhang, Planning to Poke: Sampling-based Planning with Self-Explored Neural Forward Models, Workshop on Machine Learning in Robot Motion Planning at International Conference on Intelligent Robots and Systems, IROS 2018, Madrid, Spain.

- Hadi Beik-Mohammadi, Matthias Kerzel, *Michael Görner*, Mohammad Ali Zamani, Manfred Eppe, and Stefan Wermter, Neural End-to-End Learning of Reach for Grasp Ability with a 6-DoF Robot Arm, Workshop on Machine Learning in Robot Motion Planning at International Conference on Intelligent Robots and Systems, IROS 2018, Madrid, Spain.

- *Michael Görner*, Robert Haschke, MoveIt! Task Planning, ROSCon-2018, September 29–30, 2018, Madrid, Spain.

- Hongzhuo Liang, Shuang Li, *Michael Görner* and Jianwei Zhang, Generating Robust Grasps for Unknown Objects in Clutter Using Point Cloud Data, Shanghai International Symposium on Human-Centered Robotics (HCR), 2018, Shanghai.
- *Michael Görner*, Philipp Ruppel, Norman Hendrich. Upgrading MoveIt!, ROSCon-2017, September 21–22, 2017, Vancouver, Canada.
- Sven Albrecht, *Michael Görner*, Joachim Hertzberg, and Jianwei Zhang, Autonomous Transparent Object Reconstruction, DGR Days (Deutsche Gesellschaft für Robotik), 2016, Leipzig, Germany.

Appendix B

Task Specification Example

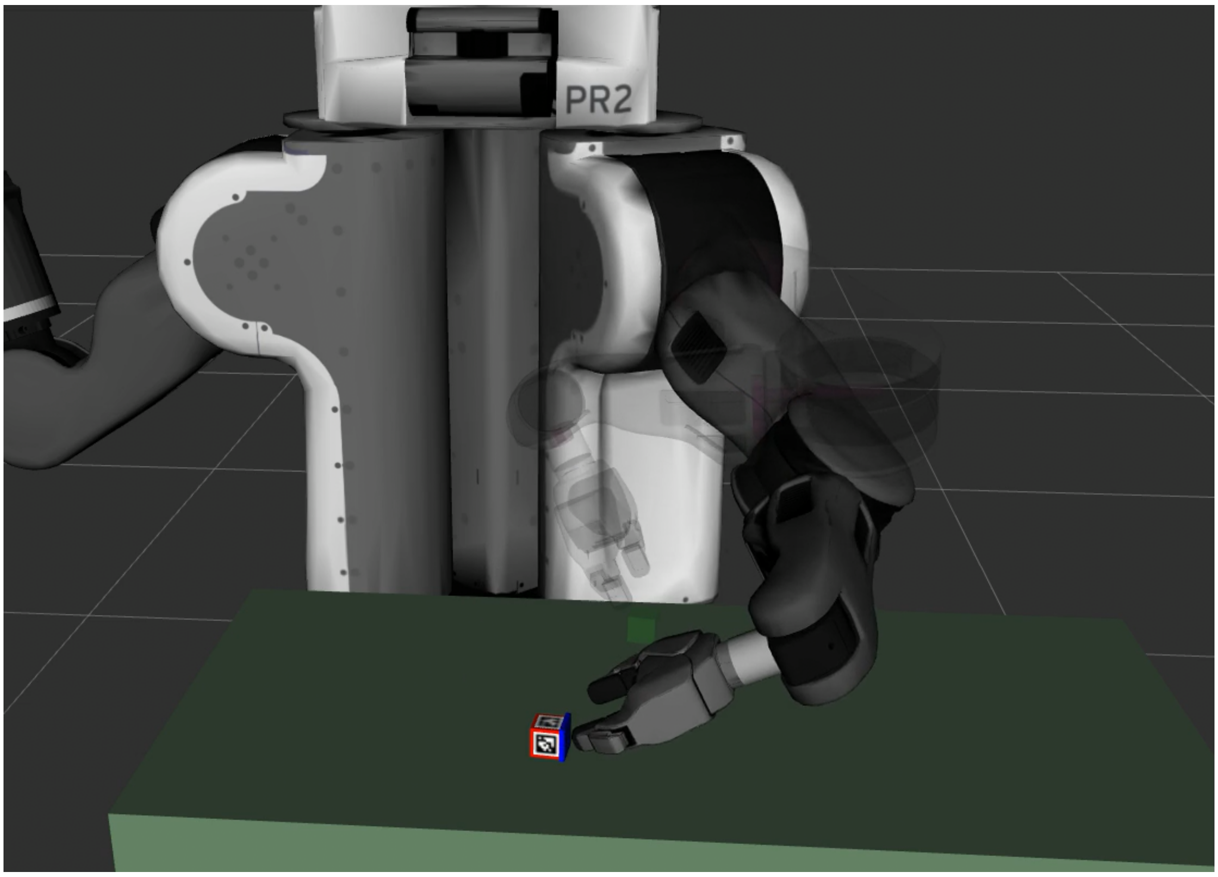


Figure B.1 – PR2 robot model turning a cube (red with apriltag markers) to turn a target face (blue) up, illustrating the Task modelled in the section.

This section lists a complete advanced example for a Task specification in YAML format, as described in section 3.7. The Task specifies a manipulation scenario where the PR2 robot has to turn a cube on the table in front of it over to a specified face, as illustrated in Figure B.1. The setting explicitly considers quasi-static prehensile actions and must carefully lift and place the cube during manipulation. In this setting, turning the cube to the right face can require up to two

pick-and-place operations — due to the small dimensions of the cube and kinematic reachability of grasping poses — or no motion at all if the correct face is already up.
The main Task structure utilizes a Fallback container to attempt potential solution sequences (no motion, turning the cube once, or turning it twice) in order of increased complexity. Domain-specific Stages in the scenario (referenced with the `mtc_turn_cube/` prefix) are used to verify whether the correct face is up already, generate adequate grasp poses to turn the cube to another face, and generate placement poses for the cube on the table.
The Task is parameterized in the target face of the cube (implemented as a detected frame id), the cube itself (implemented as the id of the object model), and the geometry of the table surface on which to place the cube. Notable, the definition of the `"turn twice"` Container simply reuses the `"turn once"` sequence of Stages. The same Task can be reparameterized for other robot systems, such as the PA10 robot described in subsection 3.9.3.

```yaml
name: pr2_turn_cube
timeout: 20.0
max_solutions: 25
properties:
  - target_face
  - target_cube
  - table_surface_geometry
stages:
- stage: CurrentState
  name: "current state"
- stage: Fallbacks
  name: "fallbacks"
  stages:
  - stage: mtc_turn_cube/TargetFaceUpPredicateFilter
    name: "solved without motion"
    target_face: <PARENT>
    wrapped:
      stage: mtc_turn_cube/Wait
      name: "no motion"
      duration: 0.0
  - stage: SerialContainer
    name: "turn once"
    grasp_face_to_turn: <PARENT:target_face>
    place_face: <PARENT:target_face>
    stages: &turn_once
    - stage: MoveTo
      name: "open gripper"
      group: left_gripper
      goal: open
      planner: JointInterpolation
    - stage: Connect
      name: "move to pre-grasp"
      group: left_arm
      planner: ompl:RRTConnect
    - stage: MoveRelative
      name: "approach grasp"
      group: left_arm
      direction:
        frame: l_gripper_tool_frame
        vector: [1.0, 0.0, 0.0]
      distance:
        min: 0.01
        max: 0.05
      planner: CartesianPath
    - stage: ComputeIK
      name: "grasp pose IK"
      group: left_arm
      ik_frame: l_gripper_tool_frame
      max_ik_solutions: 200
      wrapped:
        name: "grasp pose"
        stage: mtc_turn_cube/CubeGraspFace
        face: <PARENT:grasp_face_to_turn>
        cube: <PARENT:grasp_cube_to_turn>
        # canonical_only: true
        gripper_flip_axis: [1,0,0]
        canonical_grasp:
          position: [-0.015, 0.0, 0.0]
          orientation: [0.0, 1.1, 0.0]
        monitors: '../open gripper'
    - stage: mtc_turn_cube/AllowCollision
      name: "allow collision with gripper"
      link1: <PARENT:target_cube>
      group2: left_gripper
    - stage: MoveTo
      name: "close gripper"
      group: left_gripper
      goal: closed
      planner: JointInterpolation
    - stage: mtc_turn_cube/Attach
      name: "attach cube"
      object: <PARENT:target_cube>
      end_effector: left_gripper
    - stage: mtc_turn_cube/AllowCollision
      name: "allow collision with support surface"
      link1: <PARENT:target_cube>
      link2: <PARENT:table_surface_geometry>
    - stage: MoveRelative
      name: "lift cube"
      group: left_arm
      direction:
        frame: table_top
        vector: [0.0, 0.0, 1.0]
      distance:
        min: 0.01
        max: 0.05
        planner: CartesianPath
      - stage: mtc_turn_cube/ForbidCollision
        name: "forbid collision with support surface"
        link1: <PARENT:target_cube>
        link2: <PARENT:table_surface_geometry>
      - stage: Connect
        name: "turn"
        group: left_arm
        planner: ompl:RRTConnect
      - stage: mtc_turn_cube/AllowCollision
        name: "allow collision with support surface"
        link1: <PARENT:target_cube>
        link2: <PARENT:table_surface_geometry>
      - stage: MoveRelative
        name: "put down cube"
        group: left_arm
        direction:
          frame: table_top
          vector: [0.0, 0.0, -1.0]
        distance:
          min: 0.01
          max: 0.03
        planner: CartesianPath
      - stage: ComputeIK
        name: "place pose IK"
        group: left_arm
        ik_frame: <INTERFACE:target_face_up>
        max_ik_solutions: 3
        wrapped:
          name: "cube place pose"
          stage: mtc_turn_cube/CubeFaceUpPlacePose
          face_up: <PARENT:place_face>
          cube: <PARENT:place_cube>
          table_surface_geometry: <PARENT:table_surface_geometry>
          table_top_frame: table_top
          monitors: "../attach cube"
      - stage: MoveTo
        name: "open gripper"
        group: left_gripper
        goal: open
        planner: JointInterpolation
      - stage: mtc_turn_cube/Detach
        name: "detach cube"
        object: <PARENT:target_cube>
        end_effector: left_gripper
      - stage: mtc_turn_cube/ForbidCollision
        name: "forbid collision with support surface"
        link1: <PARENT:target_cube>
        link2: <PARENT:table_surface_geometry>
      - stage: mtc_turn_cube/ForbidCollision
        name: "forbid collision with gripper"
        link1: <PARENT:target_cube>
```

```yaml
        group2: left_gripper
      - stage: MoveRelative
        name: "lift gripper"
        group: left_arm
        direction:
          frame: table_top
          vector: [0.0, 0.0, 0.01]
        distance:
          min: 0.02
          max: 0.03
        planner: CartesianPath
    - stage: SerialContainer
      name: "turn twice"
      stages:
      - stage: SerialContainer
        name: "turn first time"
        grasp_cube_to_turn: <PARENT:target_cube>
        place_cube: <PARENT:target_cube>
        stages: *turn_once
      - stage: SerialContainer
        name: "turn second time"
        grasp_face_to_turn: <PARENT:target_face>
        place_face: <PARENT:target_face>
        stages: *turn_once
```

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Dissertationsschrift selbst verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Sofern im Zuge der Erstellung der vorliegenden Dissertationsschrift generative Künstliche Intelligenz (gKI) basierte elektronische Hilfsmittel verwendet wurden, versichere ich, dass meine eigene Leistung im Vordergrund stand und dass eine vollständige Dokumentation aller verwendeten Hilfsmittel gemäß der Guten wissenschaftlichen Praxis vorliegt. Ich trage die Verantwortung für eventuell durch die gKI generierte fehlerhafte oder verzerrte Inhalte, fehlerhafte Referenzen, Verstöße gegen das Datenschutz- und Urheberrecht oder Plagiate.

Hamburg, den 16. April 2025

Unterschrift