Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Bio IK: A Memetic Evolutionary Algorithm for Generic Multi-Objective Inverse Kinematics

Dissertation

submitted in partial fulfilment of
the requirements for the degree of
Doctor rerum naturalium
(Dr. rer. nat.)

Universität Hamburg
Faculty of Mathematics, Informatics and Natural Sciences
Department of Informatics

Sebastian Starke
Hamburg, 2020

Dissertation Committee

Advisor & Reviewer
Prof. Dr. Jianwei Zhang, Universität Hamburg, Germany

Reviewer
Dr. Kevin Wampler, Adobe Research, USA

Chair
Prof. Dr.-Ing. Timo Gerkmann, Universität Hamburg, Germany

Vice Chair
Prof. Dr.-Ing. H. Siegfried Stiehl, Universität Hamburg, Germany

Oral Defence Date

August 20th, 2020

Abstract

Inverse kinematics constitutes an essential task for control of motion, manipulation as well as interaction in robotics and animation. In this thesis, a novel efficient algorithm Bio IK is presented for solving complex kinematic body postures on generic and fully-constrained geometries with multiple joint chains and objectives. It is based on memetic evolution — combining biologically-inspired evolutionary and swarm optimisation with the gradient-based L-BFGS-B algorithm. This aims to combine the characteristic strengths of different optimisation methodologies. Accurate solutions both for position and orientation can be found in real-time while robustly avoiding suboptimal extrema as well as singularity issues, and scaling well even for greatly higher degree of freedom.

The algorithm provides high flexibility for the design of custom cost functions, which can be used for concrete specifications of desired body postures both in joint and Cartesian space. In particular, the ability to arbitrarily combine different objectives extends traditional inverse kinematics by handling multiple end effectors simultaneously while further allowing intermediate goals along the chains, such as an elbow position or wrist orientation while grasping. Additionally, task-specific objectives such as minimal displacement between solutions, prioritised joint values, functional joint dependencies, as well as real-time collision avoidance can directly be integrated into the optimisation. The algorithm represents a general method for bounded continuous optimisation, and only requires two parameters to be set for the population size and number of elite individuals. It adaptively handles varying dimensionality as well as dynamic exploitation and exploration.

Experiments were conducted on several industrial and humanoid robot models as well as virtual characters in order to demonstrate its applicability for different challenging tasks in robotics, human-robot interaction and character animation. Those include dexterous object manipulation with anthropomorphic robotic hands, full-body motion, modeling and teleoperation in virtual reality, collision-free trajectory generation, as well as animation post-processing for video games and films. The results show that the proposed algorithm can compete with popular state-of-the-art methods for inverse kinematics in terms of speed and robustness, and is further able to outperform them in flexibility and scalability. In particular, it has already gained popularity and active usage through dissemination of available implementations. The algorithm can contribute to solving more complex tasks and kinematic structures, which is of high interest for current research problems in robotics and animation.

Zusammenfassung

Inverse Kinematik stellt eine grundlegende Aufgabe zur Kontrolle von Bewegung, Manipulation sowie Interaktion in der Robotik und Animation dar. Diese Dissertation prsentiert einen neuartigen und effizienten Algorithmus Bio IK zum Lsen komplexer kinematischer Körperstellungen auf generischen und vollstndig eingeschränkten Geometrien mit mehreren Gelenkketten und Zielen. Der Algorithmus basiert auf memetischer Evolution, welche biologisch-inspirierte evolutionre und schwarmartige Optimierung mit dem L-BFGS-B Gradientenverfahren kombiniert. Dies zielt darauf ab die charakteristischen Stärken verschiedener Optimierungsmethodologien zu vereinen. Präzise Lösungen für sowohl für Position als auch Orientierung können in Echtzeit gefunden werden, wobei suboptimale Extrema sowie Singularitäten robust umgangen werden, und eine hohe Skalierbarkeit sogar für deutlich höhere Freiheitsgrade erreicht wird.

Der Algorithmus besitzt hohe Flexibilität zum Entwickeln beliebiger Kostenfunktionen, welche zur konkreten Spezifizierung gewünschter Körperstellungen sowohl im kartesischen als auch Gelenkraum benutzt werden können. Insbesondere erweitert die Mglichkeit einer beliebigen Kombination von Zielen traditionelle inverse Kinematik insofern, dass mehrere Endeffektoren und Zwischenziele entlang der Ketten berücksichtigt werden können, beispielsweise eine Ellenbogenposition oder eine Handgelenksorientierung während des Greifens. Weiterhin können Aufgaben-spezifische Ziele wie minimale Abweichung zwischen Lösungen, priorisierte Gelenkstellungen, funktionale Gelenkabhängigkeiten oder Echtzeitkollisionsvermeidung direkt in die Optimierung integriert werden. Der Algorithmus repräsentiert ein generelles Verfahren für beschränkte kontinuierliche Optimierung, und erfordert ledigich zwei Parameter für die Populationsgröße und die Anzahl der Eliten zu definieren. Variierende Dimensionalität sowie eine dynamische Exploration und Exploitation werden adaptiv gehandhabt.

Die Experimente wurden auf verschiedenen industriellen und humanoiden Robotermodellen sowie virutellen Charakteren durchgeführt um die Anwendbarkeit für verschiedene anspruchsvolle Aufgaben in der Robotik, Mensch-Roboter Interaktion sowie Charakteranimation zu demonstrieren. Diese umfassen geschickte Manipulation mit anthropomorphischen Roboterhänden, Ganzkörperbewegung in virtueller Realität, kollisionsfreie Trajektoriengenerierung, sowie Nachbearbeitung von Animationen für Videospiele und Filme. Die Ergebnisse zeigen dass der vorgestellte Algorithmus mit gängigen modernen Methoden für inverse Kinematik bezüglich Geschwindigkeit und Robustheit konkurrieren kann, und ist darüberhinaus in der Lage diese in Flexibilität und Skalierbarkeit zu übertreffen. Insbesondere hat dieser bereits Bekanntheit und aktive Anwendung durch Verbreitung der verfügbaren Implementationen erlangt. Der Algorithmus kann zum Lösen komplexerer Aufgaben und kinematischer Strukturen beitragen, was Relevanz für aktuelle Forschungsprobleme in der Robotik und Animation besitzt.

Acknowledgements

First and foremost, I want to greatly thank Prof. Dr. Jianwei Zhang and Prof. Dr. Leonie Dreschler-Fischer for supervising my thesis, and for their valueable inspirations, guidance and comments on my work. I appreciate all their time, ideas and suggestions that contributed to a productive and stimulating research experience.

I further want to give my sincere thanks to Dr. Norman Hendrich, not only for his always helpful advice and great collaboration during this research, but also for his continuous support throughout the entirety of my studies. I am thankful for all the motivational ideas and critical discussions happening on a daily basis between our office doors, which definitely contributed a lot to this research.

I also want to show my appreciation to Dr. Matthias Kerzel, Dr. Sven Magg, Dennis Krupke, Tobias Hinz, Ge Gao, Tayfun Alpay, as well as all other research members that immensely contributed to my personal and professional time in Hamburg, and whose office doors were always open for fruitful discussions or general conversations.

I want to thank my former fellow students and good friends Jonas, Pablo, Jenny, Michaela and Matthis for the amazing time both inside and outside the academic world, and who made those years in Hamburg as special as they were.

Along with them, I also want to extend my gratitude to all the lovely people and friends I have made at conferences and workshops, for making these scientific trips such a great experience and for making me feel home no matter where I was.

I also gratefully acknowledge the funding sources from the DFG / German Research Foundation as part of the CML (Crossmodal Learning, TRR 169) project that made this work possible.

Finally, many thanks go to my family for their everlasting backing and support that brought me through school, my studies, research, and my whole life in general.

"The more I study nature, the more I become impressed with ever-increasing force with the conclusion, that the contrivances and beautiful adaptations slowly acquired through each part occasionally varying in a slight degree in many ways, with the preservation of natural selection and those variations which are beneficial to the organism under complex and ever-varying conditions of life, transcend in an incomparable degree the contrivances and adaptations which the most fertile imagination of the most imaginative of man could suggest with unlimited time at his disposal."

– Charles Darwin

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

In recent years, there has been a significant increase of interest for research in artificial and computational intelligence. Biologically-inspired models and algorithms for learning and optimisation have gained larger popularity and acceptance due to their generic algorithmic functionality, which can be applied similarly for different types of problems. In particular, sometimes it can be difficult to find an explicit computational formulation for a problem, but for which it is still required to obtain a suitable solution. Previous methods have often been tailored specifically for the tasks they should solve and under well-defined settings, but showed less flexibility and scalability, and seemed limited in case of dynamically changing environments. However, with the rising interest in solving more challenging problems and understanding natural phenomena, accompanied by the steadily increasing amount of data and computational power, it is possible and required to deploy more general and adaptive methods to process high-dimensional and complex information.

The domain of robotics is one of the fields which increasingly relies on research in such directions. While it originates from the field of mechanical engineering, it has evolved into an interdisciplinary research area which combines further knowledge and experience from informatics and computer science, physics and mathematics, as well as psychology and biology. Although it still represents a relatively young discipline, it has quickly gained an essential role in various industrial sectors, such as automation and manufacturing, medical surgery, home assistance, military and security, as well as transport or space exploration. This also introduces a growing number of research topics in control of motion, perception of visual, auditory and tacile sensory information, or creating behaviour for intelligent interaction with other robots or humans. Therefore, research in artificial intelligence and providing generic methods which imitate and resemble the capabilities of nature for solving difficult tasks can be considered as a key challenge for further advances in robotics.

This thesis adresses the problem of inverse kinematics, which constitutes a fundamental task in the general field of motion and mechanics. It can be described by finding a suitable configuration of joint values to result in an appropriate posture for an articulated body by satisfying a given set of objectives. While inverse kinematics has originally evolved from the domain of robotics [92], it has also become a fundamental tool for animation in games and films [77]. It represents the opposite to forward kinematics, which describes deriving a particular posture from a given joint variable configuration. However, while this can be calculated straightforwardly through a unique sequence of coordinate transformations, a general method for inverse kinematics could not yet be found. In particular, analytic solutions are only available for simple kinematic structures, and larger focus has been paid on numerical methods to yield approximate solutions — on which this thesis also focuses on. Still, a manifold of equally suitable solutions can exist, and it is not generally clear which one to prefer. Furthermore, several issues such as joint limits, singularities and the *Curse of Dimensionality* for highly articulated geometries can turn inverse kinematics into a non-convex optimisation problem for which fast, robust and scalable methods are required.

Traditionally, the interest for inverse kinematics is in solving a single end effector position and orientation for a serial kinematic chain. Most popular approaches aim to iteratively minimise the error by computing the Jacobian [14, 103, 48]. Although such methods are very fast, accurate, and both applied in graphics and robotics, they largely suffer from suboptimal extrema as they purely rely on following the gradient. Specifically in animation, heuristics methods such as the CCD *(Cyclic Coordinate Descent)* [110] and FABRIK *(Forward and Backward Reaching Inverse Kinematics)* [4] have become more popularly applied due to their simplicity and even lower computational cost. However, they are more suited for position-only goals, and are likely to violate joint limits by operating in Cartesian instead of joint space. Further methods are based on SQP *(Sequential Quadratic Programming)* [7, 36] which could be demonstrated to perform a more stable, flexible and reliable optimisation. Nevertheless, adding constraints to those works best only for under-constrained systems, and local extrema configurations are still remaining an issue. Next, sampling-based methods such as GA *(Genetic Algorithms)* [78, 99, 101] and PSO *(Particle Swarm Optimisation)* [25, 83, 18] have also frequently been applied to solving inverse kinematics. Those provide flexible and scalable methodologies which support joint limits and further constraints, can robustly avoid suboptimal extrema, and also do not suffer from singularities since solutions are directly generated in joint space. However, the required time to converge to appropriate solutions often remains unfeasibly large for many time-critical applications. Finally, learning-based methods such as ANN *(Artificial Neural Networks)* [43, 2] or SVR *(Support Vector Regression)* [73] have been used to approximate a function between joint and Cartesian space. Nevertheless, a major issue comes with a proper choice and preparation of training samples, and the accuracy often remains too low.

In this work, a memetic evolutionary algorithm Bio IK was developed, which successfully combines the characteristic strengths of biologically-inspired and gradient-based optimisation for multi-objective inverse kinematics. It extends the research that was originally started in [93] and further published in [94, 97, 87, 95, 96, 55], and is presented and discussed in more detail throughout this thesis.

1.1 Motivation

Inverse kinematics is a challenging topic which has been investigated over decades, but still represents an active field of research with open issues both in robotics and animation. Although the traditional problem setup is well-researched, existing methods still seem rather limited in robustly handling more complex and highly articulated geometries, solving multiple kinematic chains and goals simultaneously, being sufficiently accurate and fast, integrating further constraints in joint or Cartesian space, or supporting collision avoidance while generating postures. While those benefits often seem mutually exclusive, combining such abilities is highly relevant and helpful for several tasks and current research problems. Those include grasping and dexterous object manipulation with anthropomorphic robotic hands, full-body motion of humanoid robots and virtual characters, motion reconstruction and teleoperation, simulation and modeling in virtual reality, animation editing and post-processing, as well as collision-free trajectory generation. The motivation for this research was initiated by observing exactly those limitations and challenges for inverse kinematics methods that are implemented in popular tools for robotics, graphics and simulation. Specifically in robotics, existing plugins for ROS *(Robot Operating System)* [34] — such as *KDL* [50] or *Trac-IK* [80] — do only support serial kinematic chains with single end effector pose goals. Available implementations in Unity3D [105], Unreal Engine [27] as well as Maya [61] are more flexible in handling higher articulated geometries, but showed difficulties in solving end effector orientations, and frequently produced unrealistic body postures by violating joint limits. Furthermore, in context of the CML (Crossmodal Learning, TRR 169) project [81], a major interest was in developing novel methods to improve robotic manipulation using the anthropomorphic Shadow Dexterous Hand [38], as well as in generating realistic motion for human-robot interaction. This requires defining intermediate goals along an arm while manipulating objects, functional joint dependencies such as for a real-human hand, as well as concurrently solving multiple chains and end effectors goals. Altogether, this demonstrates the demand for efficient methods which support specifying a variety of custom objectives and constraints to solve complex articulated postures.

Since evolutionary computation usually suffers from large computation times to converge, they can often not leap the hurdle into industrial applications. Recently, a larger focus in non-linear optimisation has been paid to memetic algorithms, which hybridise evolutionary computation with local search techniques. This aims to avoid the slow convergence of simple genetic algorithms while maintaining their robustness and scalability. This methodology can be efficiently applied to the inverse kinematics problem, and offers high flexibility for the design of custom cost functions. In particular, the developed Bio IK algorithm directly follows this idea of encoding joint variable populations, and has already proven itself a competitive tool both in research and industry through dissemination of this work. Fig. 1.1 initially highlights its capabilities and suitability if applied to different tasks in robotics and animation, which will be explained in more detail in chapter 5.

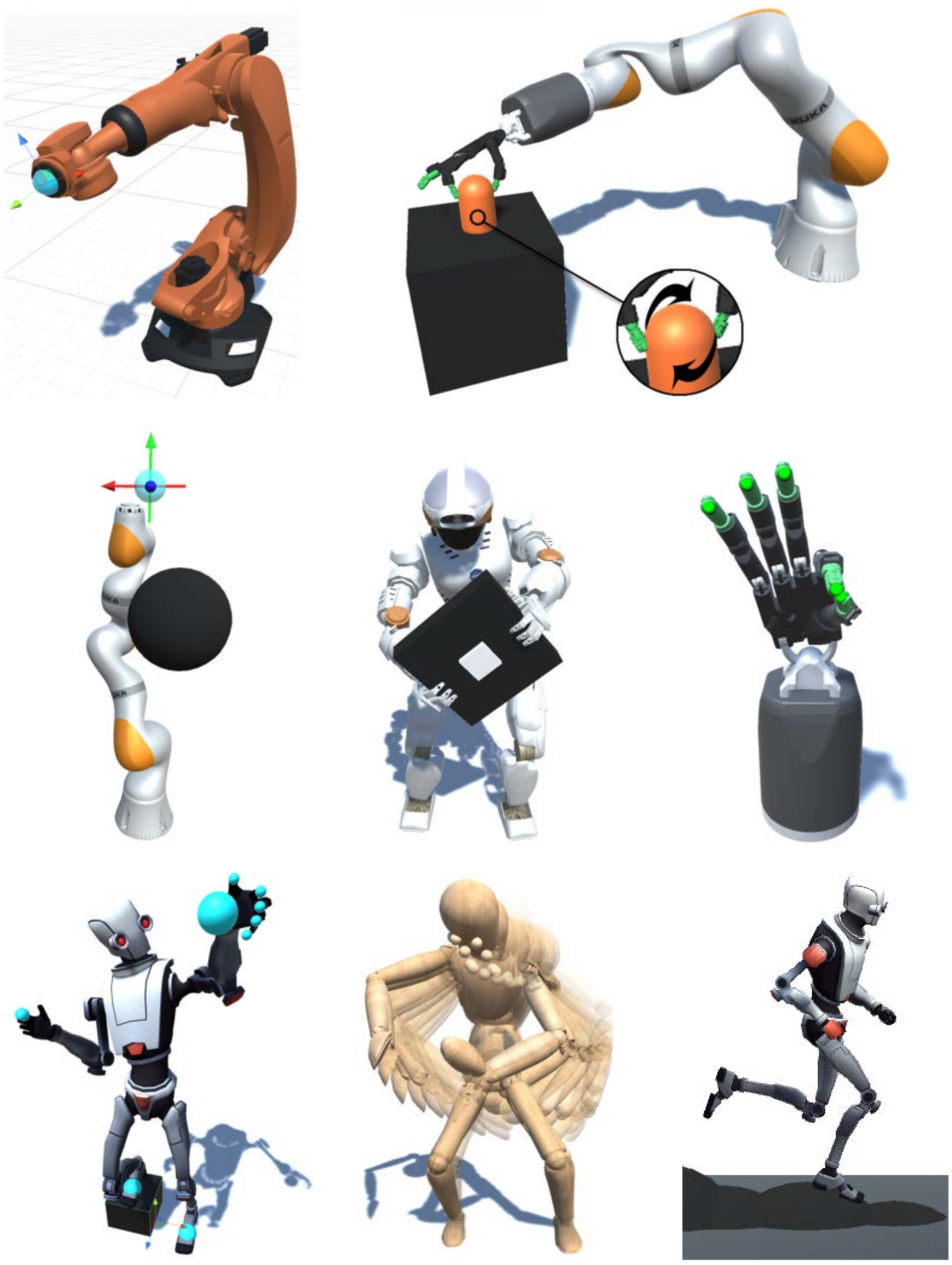


Figure 1.1: The proposed Bio IK algorithm applied to different types of application scenarios for inverse kinematics in robotics and character animation.

1.2 Research Objectives

During this work, the following main research objectives were considered:

- Design of a generic approach for solving inverse kinematics on arbitrary kinematic structures with multiple objectives and constraints.
- Combining the characteristic strengths of evolutionary and gradient-based optimisation to achieve speed, accuracy, robustness, scalability and flexibility.
- Supporting real-time collision avoidance for generating kinematic postures and trajectories on highly articulated geometries.
- Developing suitable objectives for serving different tasks, such as grasping and manipulation, intelligent interaction, and animation post-processing.
- Availability of an efficient algorithmic implementation which can be used in research and industry in robotics (ROS) and character animation (Unity3D).

1.3 Structural Outline

The outline of this thesis to present the condcuted research is structured as follows:

Chapter 2 provides the fundamental knowledge which is essential for a clear understanding of the following sections. This includes the mathematical and technical components of kinematis, as well as a description of different articulated geometries which were used for this research. Then, the relevant optimisation methodologies which are combined in the proposed hybrid algorithm are discussed.

Chapter 3 continues with a more detailed presentation of related work, with a particular focus on numerical and approximate methods for inverse kinematics. Those are summarised with a critical discussion regarding their benefits and limitations.

Followingly, chapter 4 presents the developed Bio IK algorithm, and discusses all relevant components, such as the encoding scheme, the objective function design, the chosen and developed genetic operators, as well as the single evolutionary phases. In addition, a pseudocode and references to existing implementations are provided.

Chapter 5 then shows the experimental results on a variety of application scenarios using different robot models and virtual characters, and evaluates the algorithmic performance compared to popular related methods. In addition, the impact of this research will be demonstrated by a selection of work examples that were created through dissemination of the proposed method.

Finally, chapter 6 concludes with a summary, a list of contributions, a discussion of limitations as well as some inspirations for future work.

Chapter 2

Fundamentals

Kinematics is a field which is strongly related to advanced knowledge in linear algebra, computational models, design and optimisation. Since this thesis addresses the inverse kinematics problem both for robotics and computer animation, the relevant knowledge for both domains will be provided.

This chapter primarily aims introducing several fundamental components and terms for kinematics, starting with the mathematical equations for coordinate transformations (2.1.1) which are essentially required for this topic. It then describes the composition and complexity of motion for articulated geometries (2.1.2, 2.1.3), and also explaining the related singularity issues (2.1.4). Then, a mathematical formulation of forward kinematics (2.1.5) as well as a generalisation for inverse kinematics with multiple targets (2.1.6) will be given.

It then introduces the kinematic design of different geometries for robots and characters which were used throughout this research. Those include serial (2.2.1) manipulators and robotic platforms (2.2.2), anthropomorphic robots (2.2.3), as well as humanoid (2.2.4) and quadruped (2.2.5) structures. In combination, the particular challenges for inverse kinematics which are related to these structures are described.

Lastly, the fundamental concepts and recent advances of the relevant algorithmic methodologies for the proposed algorithm are presented. Those focus on gradient-based optimisation (2.3.2), as well as biologically-inspired evolutionary computation 2.3.3) and swarm intelligence (2.3.4).

2.1 Kinematics

Kinematics outlines one of the most fundamental aspects in the general field of motion, covering the transformation of objects with respect to position and orientation, as well as velocity and acceleration, but regardless of mass, force and torque. It has particular relevance for various applications in robotics and computer graphics, including the design and control of robots, visualisation and simulation, as well as for creating and post-processing animations of virtual characters [92, 77].

The fundamental equations to determine the pose $\mathcal{X} = (x_P, y_P, z_P, \phi_R, \theta_R, \psi_R)$ of an object in three-dimensional Cartesian space are given by (2.1) and (2.2, where $\mathbf{v}$ is the velocity and $\mathbf{a}$ the acceleration, and $\mathbf{s}$ is a vector of translation or rotation with respect to the time t. Note that $\mathcal{X}$ consists of two separate tuples describing the position $P = (x, y, z)$ and $R = (\phi, \theta, \psi)$, where the latter is defined by a set of Euler angles which will be described in more detail in section 2.1.1.

$$\mathbf{v}(t) = \dot{\mathbf{s}}(t) = \frac{d\mathbf{s}(t)}{dt} \tag{2.1}$$

$$\mathbf{a}(t) = \ddot{\mathbf{s}}(t) = \frac{d\mathbf{v}(t)}{dt} \tag{2.2}$$

Those equations can be used for generating smooth trajectories when designing a motion controller, i.e. to determine the required amount of acceleration and decceleration such that velocity is increased and stopped soon enough in order to reach and not overshoot the target. This feature is available for the implementation of the proposed Bio IK algorithm in [10], and aims to simulate naturally-looking movements and transitions between kinematic postures.

Furthermore, the geometry of an articulated kinematic system can be described by a set of kinematic chains, where each of them is defined by a consecutive set of linked segments and joints from the root to the end effectors. An end effector denotes the last segment which is located at the end of a kinematic chain. Each joint along the particular kinematic chains is given a specific axis of motion which defines the either translational or rotational movement for the connected segment, which is performed relative to a default zero-configuration. In context of this research, it is important to highlight that the inverse kinematics problem is not considered exclusively for single kinematic chains, but rather for a skeleton of multiple kinematic chains defining a hierarchical tree structure. An example is shown in Fig. 2.1 using the NASA Valkyrie humanoid robot mode. It visualises the hierarchy of the single segments which are connected by joints, and also shows the particular motion limits for each joint axis. While the legs represent two independent chains starting from the root, the chains along the arms and to the head are mutually affected by the configuration of the upper body. When further aiming for full-body postures, it is required to also incorporate adjustments for the root pose. Thus, the whole kinematic structure with all chains must be solved together, for which advanced methods are required that can handle multiple objectives simultaneously.

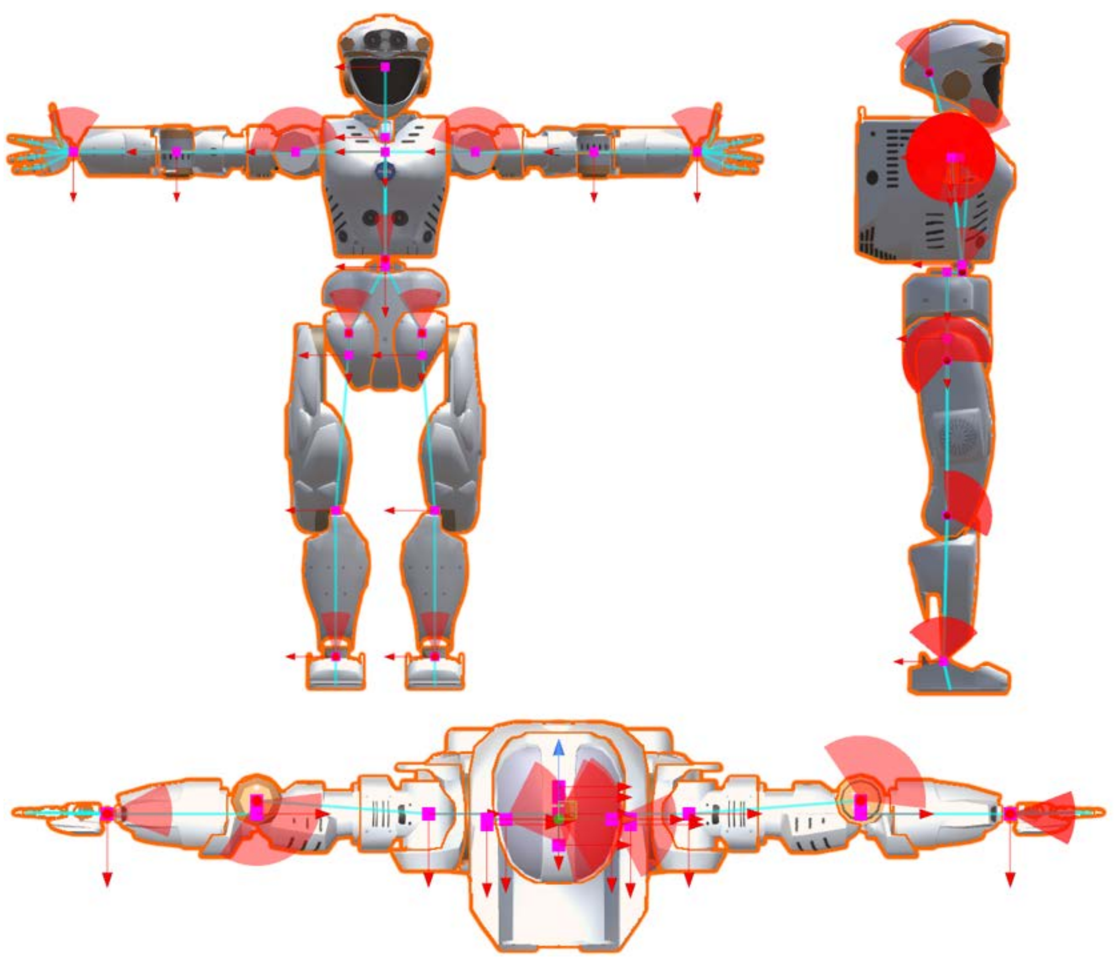


Figure 2.1: Composition of kinematic chains on the NASA Valkyrie humanoid robot model. The skeleton of segments (cyan) is connected by the single joints (magenta), where each motion axis (red arrows) has particular lower and upper limits (red circle arcs) to constrain the motion relative to the default posture.

To obtain the poses of the single segments and end effectors with respect to a particular joint variable configuration, the transformations along the hierarchy must be calculated recursively starting from the root. This process is called forward kinematics, and represents the opposite to inverse kinematics, as visualised in Fig. 2.2. More specifically, let θ denote a n-dimensional joint variable configuration which dimensionality is given by the number of motion axes, then forward kinematics yields the desired segment poses $\mathcal{X}_1, ..., \mathcal{X}_k$ and vice versa for inverse kinematics. However, this mapping is generally not bijective or uniquely invertible — while calculating forward kinematics always produces a unique solution for the segment poses, inverse kinematics can accept multiple solutions of joint variable configurations, which will be discussed in more detail in sections 2.1.5 and 2.1.6. In addition, it is important to note that $\mathcal{X}$ must not only be an end effector, but can also represent an intermediate segment within the kinematic tree. This interpretation and notation will also be used throughout this thesis.

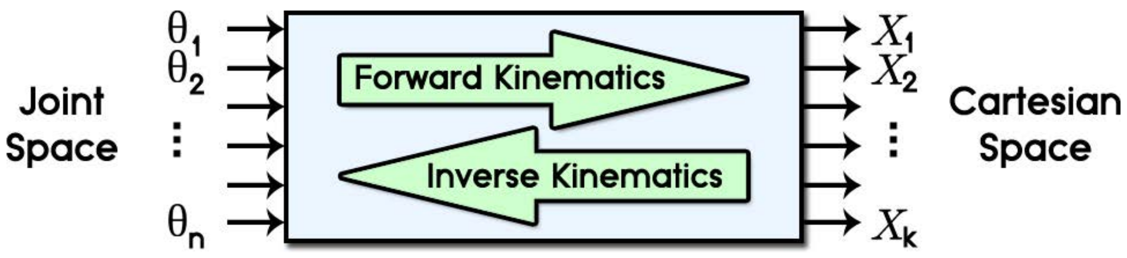


Figure 2.2: Mapping between joint and Cartesian space by forward and inverse kinematics using the joint variables $\theta_1, ..., \theta_n$ and segment poses $X_1, ..., X_k$.

2.1.1 Coordinate Transformations

Geometric coordinate transformations between two domains A, B in Cartesian space can be defined through a bijective mapping $f : A \mapsto B \wedge f^{-1} : B \mapsto A$ where $ff^{-1} = I$ and $f : \mathbb{R}^n \mapsto \mathbb{R}^n$ holds. In kinematics, only translation and rotation need to be considered when having a constant uniform scale between the connected joints. This section will discuss two methods for implementing these transformations, given by homogeneous matrices and quaternion-vector representations, with the latter being used for this research. [59, 24]

Homogeneous Transformations

Coordinate transformations can be specified using homogeneous transformation matrices as shown in (2.3) where $t, R \in \mathbb{R}^3$ denote a three-dimensional translation vector t and rotation matrix R. This composition enables combining both algebraic operations of translation and rotation within a fixed and consistent four-dimensional matrix representation.

$$T_t = \begin{bmatrix} 1 & 0 & 0 & \\ 0 & 1 & 0 & t \\ 0 & 0 & 1 & \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad T_R = \begin{bmatrix} & & & 0 \\ & R & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

The rotation matrices $R_x, R_y, R_z \in \mathbb{R}^3$ then define a rotation about a particular Cartesian axis by an angle α, and can be formulated as (2.4) considering the right-hand rule, where S and C denote the sine and cosine functions.

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & C\alpha & -S\alpha \\ 0 & S\alpha & C\alpha \end{bmatrix} R_y(\alpha) = \begin{bmatrix} C\alpha & 0 & S\alpha \\ 0 & 1 & 0 \\ -S\alpha & 0 & C\alpha \end{bmatrix} R_z(\alpha) = \begin{bmatrix} C\alpha & -S\alpha & 0 \\ S\alpha & C\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

Applying a successive multiplication of transformations $T_1, ..., T_n$ again yields a transformation matrix of the final resulting coordinate system, where its first three columns define the axis alignments and fourth the position with respect to the root transformation T_0. Considering that matrix multiplications are generally not commutative, it can be shown that left-hand side multiplication causes global transformations while right-hand side multiplication results in local transformations for the resulting frame. Note that this also affects the order of rotation and translation to compose a coordinate frame. Given a set of Euler angles (ϕ, θ, ψ) where ϕ denotes a *Roll* about the X-axis, θ a *Pitch* about the Y-axis and ψ a *Yaw* about the Z-axis, those can be used to formulate a single rotation matrix. This composition usually follows the Z-X-Y order, along with a further preceding translation, and finally results in the transformation sequence $T = T_t \cdot T_{R_\psi} \cdot T_{R_\phi} \cdot T_{R_\theta}$ 2.5 that needs to be consistent for the kinematic calculations.

$$T = \begin{bmatrix} C\psi\ C\theta - S\psi\ S\phi\ S\theta & -C\phi\ S\psi & C\psi\ S\theta + C\theta\ S\psi\ S\phi & t_x \\ C\theta\ S\psi + C\psi\ S\phi\ S\theta & C\psi\ C\phi & S\psi\ S\theta - C\psi\ C\theta\ S\phi & t_y \\ -C\phi\ S\theta & S\phi & C\phi\ C\theta & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

Quaternion-Vector Transformations

A more computationally efficient way to perform coordinate transformations is to use vector addition for translation and quaternion multiplication for rotation. Although the consistent representation is lost, one can show that the required amount of arithmetic operations for each frame transformation can be significantly reduced compared to using homogeneous matrices. In addition, using quaternions can avoid the related gimbal lock issues, in which case the rotation matrix becomes rank deficient. A quaternion is given by a four-dimensional vector $q = (x, y, z, w)$ which can be constructed from a rotation by an angle α about an arbitrary three-dimensional normalised axis $a = (x, y, z)$ as denoted by (2.6). Thus, any sequence of homogeneous rotation matrices can also be expressed through a single quaternion.

$$q_a(\alpha) = (q_x, q_y, q_z, q_w) = (a_x S\frac{\alpha}{2}, a_y S\frac{\alpha}{2}, a_z S\frac{\alpha}{2}, C\frac{\alpha}{2}) \tag{2.6}$$

Regarding the order of multiplication sequences, the same principle as for matrix concatenation holds true. Using the same Z-X-Y order for kinematic transformations, a quaternion rotation can be constructed from Euler angles through 2.7.

$$q = q_z(\psi) \cdot q_x(\phi) \cdot q_y(\theta) \tag{2.7}$$

If required, a quaternion can also be transformed into a rotation matrix (2.8).

$$R(q) = \begin{bmatrix} 1 - 2q_y^2 - 2q_z^2 & 2q_xq_y - 2q_zq_w & 2q_xq_z + 2q_yq_w \\ 2q_xq_y + 2q_zq_w & 1 - 2q_x^2 - 2q_z^2 & 2q_yq_z - 2q_xq_w \\ 2q_xq_z - 2q_yq_w & 2q_yq_z + 2q_xq_w & 1 - 2q_x^2 - 2q_y^2 \end{bmatrix} \tag{2.8}$$

Finally, the transformation T_C which concatenates two particular frames T_A and T_B following a left-handside multiplication can be obtained according to 2.9, where (t_A, q_A) and (t_B, q_B) are the translation and rotation of T_A and T_B respectively.

$$T_C = T_A \cdot T_B = \begin{cases} t_C = t_A + q_A \cdot t_B \\ q_C = q_A \cdot q_B \end{cases} \tag{2.9}$$

2.1.2 Degree of Freedom

The DoF (Degree of Freedom) of an articulated body is typically related to its overall kinematic complexity. In particular, it can be derived from the existing motion axes of the single joints along the chains, as shown in Fig. 2.3. Usually, a higher DoF also increases the amount of redundancy for available joint variable configurations to reaching identical end effector poses. However, this introduces further mathematical issues, such as singularities and suboptimal extrema, as will be discussed more in sections 2.1.4 and 2.1.6. In order to reach full 3D-poses of position and orientation, 6 DoF are usually sufficient. Systems with lower or higher DoF can be described as under- or overarticulated. Note that this only holds if all pairs of consecutive motion axes are not coaligned to each other, which would violate the laws of Euler rotations — resulting in an immediate loss of the available DoF. Thus, the DoF of articulated bodies can grow arbitrarily high, and a suggestion for different methods to derive the DoF is discussed in [79, 62].

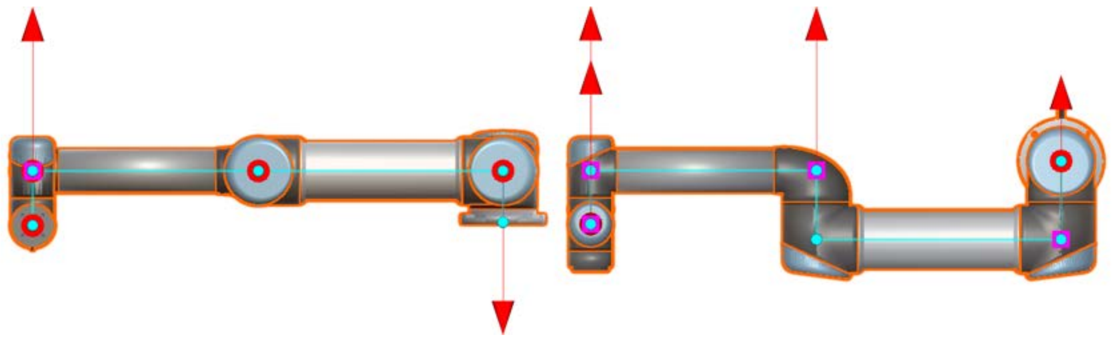


Figure 2.3: 6 DoF geometry of the articulated UR5 manipulator.

Furthermore, each DoF can also be represented by the translational (surge, sway, heave) and rotational (roll, pitch, yaw) dimensions in which a particular object can move or operate. Hence, these dimensions give an upper six-dimensional bound in Cartesian space, where surge/roll is related to the X-axis, sway/pitch to the Y -axis and heave/yaw to the Z-axis with respect to the coordinate system of the object. This interpretation is often used in engineering or aeronautics in order to describe the workspace or the space of motion of an object.

In this work, the DoF is calculated as the sum over all motion axes that are influencing the desired segment poses of the articulated body, and which then give rise to the search space dimensionality of the proposed evolutionary algorithm.

2.1.3 Joint Types

There are several types of joints available to define the either translational or rotational motion of the single segments. However, the classification and understanding of joints slightly differs between robotics and graphics. In particular, robotic joints are typically understood as connections between rigid objects which are controlled by a joint value and a specified axis to calculate the relative offset between segments. This is important for mechanical applications since it directly allows integrating joint limits to constrain the motion by lower and upper bounds. In animation, joints can also be represented as raw transformations between those segments — and thus can be controlled purely based on position and orientation information in Cartesian space. However, this makes it more difficult to avoid violation of joint limits, which is particulary important to create natural motion. Therefore, this research uses the former representation by solving postures in joint space rather than in Cartesian space.

An overview of different joint types is given by Tbl. 2.1, and which are all supported by the Bio IK algorithm. The most relevant types are given by revolute and prismatic joints, which can be combined to create 2 or 3 DoF rotational or translational motion. Note that every joint motion of can be limited by an upper or lower bound, which formulates box-constraints for the specific joint space dimensions. This makes it easier to sample from realistic postures in joint space. However, there are also robots which have joints that allow unconstrained continuous motion, i.e. for the gripper for an robotic arm. Such robotic design can be helpful for manipulating objects, and to avoid running into joint limits when generating a trajectory to transfer from one posture into another. Unconstrained translational joints can be used to solve a required position of a robot base for manipulation, or to update a character pelvis while running over uneven terrain.

Joint	**DoF**	**Description**
Rigid	0	This type does not allow any motion, but can be used to defineg fixed connections between segments.
Revolute	1	This joint allows rotational motion about a particular axis.
Prismatic	1	This joint allows translational motion along a particular axis.
Cylindric	2	This joint allows one rotational and one translational motions with respect to their axes.
Planar	3	This joint allows two translational and one rotational motions within a plane that is perpendicular to a particular axis.
Spheric	3	This joint allows rotational motion about all three axes.

Table 2.1: Overview of different joint types.

2.1.4 Singularities

Singularities are a major issue in the kinematic design of robots. A singularity can be observed as an instantaneous loss in the DoF when solving motion postures. In particular, the first derivative of the kinematics equations 2.1 yields the velocity and allows calculation of the Jacobian matrix, which is popularly used for inverse kinematics as will be desribed in section 3.2.1. When a singularity occurs, this matrix becomes rank deficient and numerically unstable, and causes infinite velocities to occur for the joint variables. The problem then becomes that physical systems like robots can not move infinitely fast. In practise, this means that tasks which require precise control of motion, such as industrial manufacturing, might become more difficult to handle. In addition, there exists an infinite number of joint variable configurations which all lead to an identical pose for the end effector. Although the amount of singular configurations increases with a growing DoF, the probability to end up in a singularity can be decreased due to the larger range of solutions and transitions between postures. For 6 DoF robots, there are only a few types of singularities that can occur, and the goal typically is to detect or avoid such configurations in order not to break the mechanical system. Some intuitive methods to achieve this are to strictly define a maximum velocity for each joint, or to design the robot geometry with small angles between connected segments so they are not lined up straight. It is also possible to add small offsets around near-singular configurations, typically when the joint approaches a zero value. Finally, an efficient but not always possible technique is to transform the task into a workspace region for which is known that no singular configurations exist. [23, 56, 91]

While gradient-based approaches often suffer from singularities, the proposed method in this research avoids these issues by sampling configurations directly from joint space, and not solely requiring a gradient in order to obtain solutions.

2.1.5 Forward Kinematics

The computation of forward kinematics (2.10) relates to the mapping from joint to Cartesian space. Using the joint variables $\theta_{1,\ldots,n}$, the resulting set of segment transformations $\mathcal{X}_{1,\ldots,k}$ can be obtained by a recursive calculation of coordinate transformations along the hierarchical kinematic geometry. This mapping is straightforward and always return unique solutions.

$$f(\theta) = \mathcal{X}_{1,\ldots,k} \tag{2.10}$$

Considering a single kinematic chain, the transformation from the root to the end effector segment is given by a series of successive transformations (2.11), where each ${}^{i}T_{i+1}$ describes a transformation between a pair of consecutive segments.

$${}^{root}T_{ee} ={}^{root} T_1 \; {}^{1}T_2 \; ... \; {}^{ee-1}T_{ee} \tag{2.11}$$

In this thesis, every transformation ${}^{i}T_{i+1}$ is understood as a local transformation, whereas a transformation ${}^{W}T_{i}$ denotes the global transformation of a segment relative to the world coordinate system T_W.

Since forward kinematics can be computed analytically and for any geometry, it is often used to find an approximate solution for inverse kinematics. The key concept is to calculate relative offsets for the segments with respect to the joint variable changes, and to obtain error measurements to optimise an objective function, such as for gradient-based or evolutionary and swarm optimisation. Those methods will be discussed in section 2.3.

2.1.6 Inverse Kinematics

Inverse kinematics 2.12 describes the opposite of forward kinematics, and hence refers to the mapping from Cartesian to joint space. It formulates the problem of finding a suitable n-dimensional joint variable configuration θ of joint values $\theta_1, ..., \theta_n$ for a given set of segment transformations $\mathcal{X}_{1,...,k}$.

$$\theta = f^{-1}(\mathcal{X}_{1,...,k}) \tag{2.12}$$

However, this problem is non-trivial, and zero up to infinite solutions can exist to satisfy an identical Cartesian target by different joint variable configurations. Usually, the target is defined by a full pose, but can also be given only by position or orientation depending on the task. Although algebraic solutions can be derived for simple geometries, and which are also able to return all existing solutions, those become unavailable as soon as the geometry grows a little more complex. Therefore, many approaches rely on numerical approximation, but which again brings other problems into play, such as suboptimal extrema, joint constraints, as well as higher computational cost. Suboptimal extrema can particularly occur when running into joint limits. For example, although a Cartesian target might be closer to reach when exceeding a joint limit, a large turn into the other direction can often be required in order not to violate the kinematic constraints. The further presence of singularities also complicates solving inverse kinematics, as described previously in section 2.1.4. Particular methods can be used to avoid such problems, but often are either too slow or not accurate enough for many applications. Some popular related methods addressing these issues will be discussed in chapter 3.

This research extends the general understanding of inverse kinematics by the ability to specify an arbitrary number of objectives to be considered for solving body postures. In particular, those do not only need to be in Cartesian space, but can also be designed to support joint space objectives, such as minimal displacement between configurations or prioritisation for particular joint values. More generally, any objective which defines a continuous loss term can be used for the optimisation, which is considered as a minimisation problem. This methodology for the proposed algorithm with concrete formulations for a list of objectives will be presented in detail in section 4.4.

2.2 Articulated Geometries

There exists a variety of different kinematic geometries which are specifically designed for their applications in robotics, games and films. This section introduces some of them, and which were all used for the experiments in chapter 5 in order to demonstrate the flexibility and scalability of the proposed method.

2.2.1 Serial Manipulators

Serial manipulators are most common in industrial robotic applications, and are described by a single serial kinematic chain. They typically consist of 6 or 7 DoF like the robot arms shown in Fig. 2.4. Manipulators with higher DoF can be described as hyperredundant, mostly providing a continuous range of joint configurations to reach identical poses. The reason why 6 DoF are often chosen for the kinematic design is influenced by Jacobian methods, which are traditionally used for inverse kinematics. When solving a single end effector pose, the dimensionality in joint and Cartesian space matches becomes coinciding. This is particularly helpful as these methods depend on matrix inversion, as will be described more in section 3.2.1. Since the manipulators are specifically designed for manufacturing tasks, a high accuracy of 10^{-5}m/rad is usually desired.

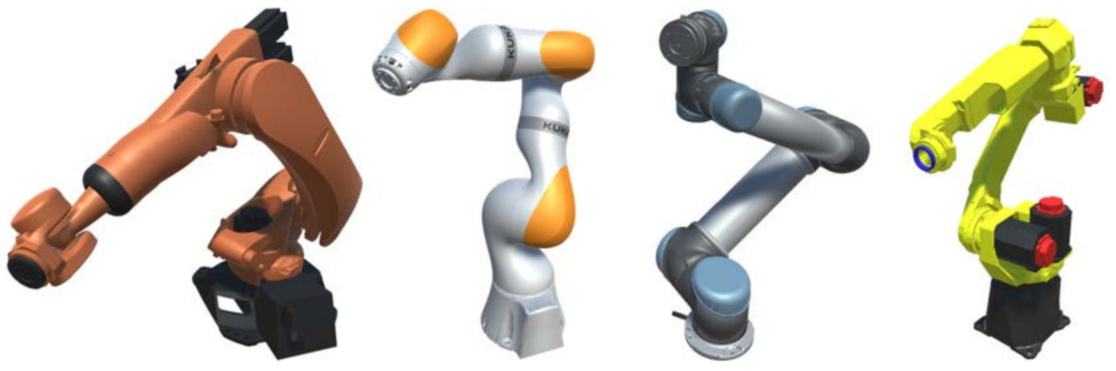


Figure 2.4: Examples of serial manipulators, including the KuKA KR120 (left), KuKA LBR iiwa (middle left), UR5 (middle right) and FANUC M-10iA (right).

2.2.2 Robotic Platforms

Two robots for more complex applications are shown in Fig. 2.5. Those typically consist of multiple arms and a higher articulated body, as well as a multi-sensory platform. Although the arms are often used as serial independent chains, using the available body joints can be used to ease manipulation. Both gripper poses are then affected in case of the liftable torso of the PR2 robot. As this joint has a considerably lower velocity than the arm joints, a weighting to prefer joint value changes among those would be preferred. The mounted head and arm cameras further allow using distance information that can be used for collision avoidance.

Figure 2.5: Example of robotic platforms using the Baxter and PR2 robot models.

2.2.3 Anthropomorphic Robots

Anthropomorphic geometries represent a similar kinematic structure and visual appearance as humans. Those cause a more difficult problem for inverse kinematics, as they typically involve a much higher DoF as well as dependent kinematic chains. As shown in Fig. 2.6, all fingers of the hand are affected by the configuration of the wrist, and thus must be handled together when solving grasps for dexterous object manipulation. In addition, encoding functional relations between joints as for a real-human hand is required to create natural configurations.

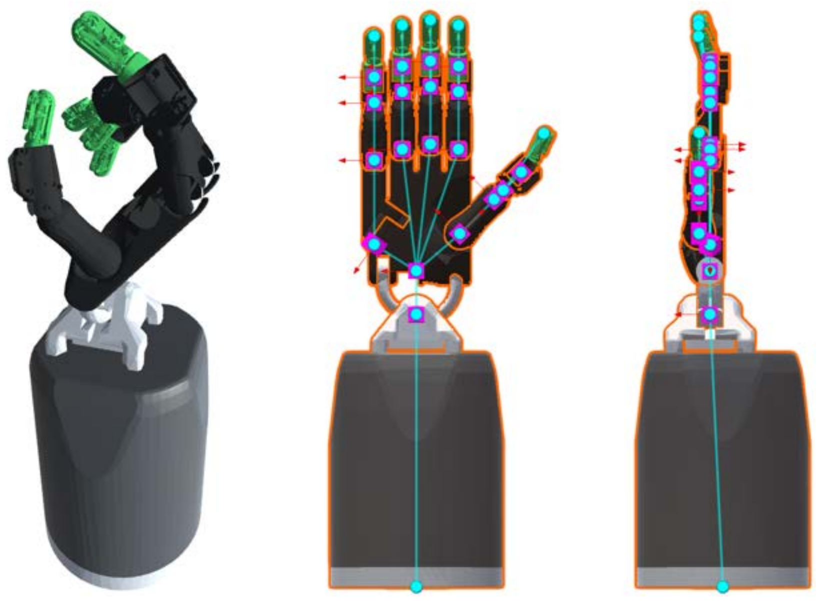


Figure 2.6: Example of an anthropomorphic robot by the Shadow Dexterous Hand.

2.2.4 Humanoids

Humanoid geometries as shown in Fig. 2.7 outline a challenging task for inverse kinematics both in robotics and animation. In addition to their very high DoF and multiple dependent joint chains, the root pose of the pelvis needs to be incorporated. This is neccessary in order to create realistic postures, such as for kneeling, walking or running. Thus, a whole joint hierarchy must be solved simultaneously.

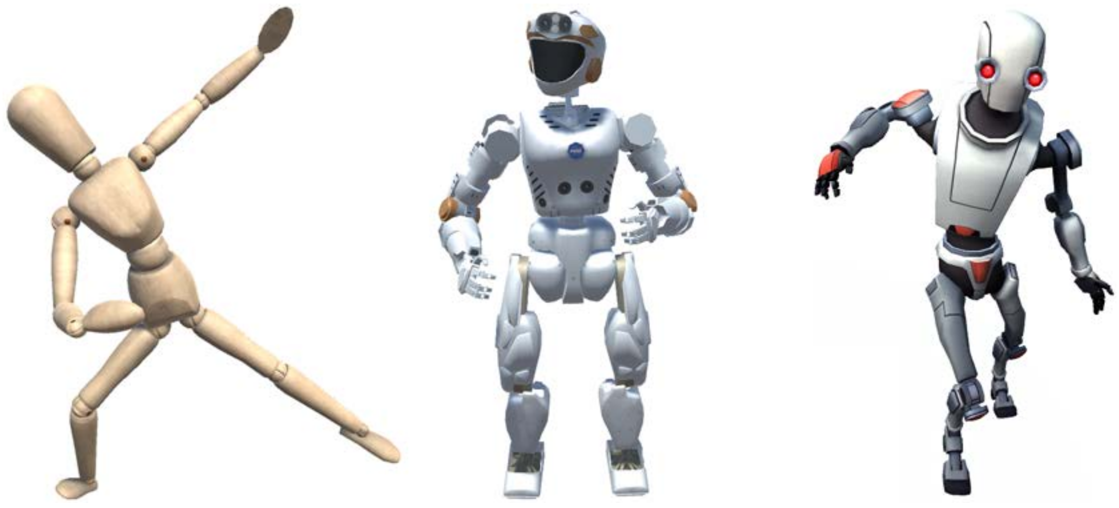


Figure 2.7: Example of humanoid geometries using a mannequin doll, the NASA Valkyrie robot model, and the Kyle game character.

2.2.5 Quadrupeds

Quadruped geometries, like the one shown in Fig. 2.8, also combine the different challenges for inverse kinematics similarly to humanoids. However, since they have four legs, it is usually desired to ensure multiple contact points. Additionally, known joint couplings between the fore- and hindlegs can be used to ease the creation of animations, and for generating suitable motion sequences.

Figure 2.8: Example of a quadruped geometry using a wolf game character.

2.3 Algorithmic Methodologies

Solving inverse kinematics can be done either by analytical or numerical methods. Although analytical would be preferred, they are often not feasible to construct in opposit to numerical techniques. In general, numerical optimisation is a mathematical technique that has consistently performed well on complex problems [58, 51]. Since this research also focuses on the latter methodology, this section aims the introduce the fundamental concepts to obtain approximate solutions for inverse kinematics, and which are used for the design of the proposed memetic algorithm.

2.3.1 Multi-Objective and Multi-Modality

When formulating a task as an optimisation problem, there is usually more than one variable which influences the quality of a solution. Thus, a larger number of concurrent variables must be considered, which increases the complexity and formulates a multi-objective optimisation problem. The range of solutions which optimise the quality with respect to the single quantities is called Pareto front, and from which a final solution must be selected. However, it is not generally clear how to chose this solution, and requires to define an individual tradeoff between the involved criterions. For inverse kinematics, this can be understood as assigning more relevance to particular segment poses — i.e. for the finger tips rather than the additionally specified elbow position when solving a grasp. Multi-modality then refers to the problem of having multiple locally-optimal solutions under the same cost function. This gives rise to a non-convex optimisation problem, and for which it becomes desirable to search for multiple solutions simultaneously. However, only few methods are able to do so, and the challenge becomes to avoid suboptimal extrema and to find the global optimum. Though, finding the global solution is never guaranteed for any optimiser in non-convex optimisation. This multi-modality is strongly present for inverse kinematics, given by the range of solutions in joint space that map to the same segment poses for the end effectors. The core concepts of these two terms are visualised in Fig. 2.9.

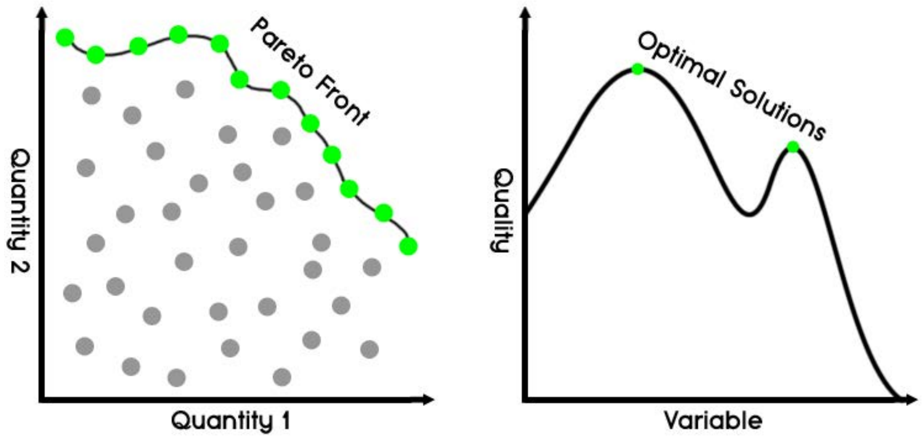


Figure 2.9: Visualisation of multi-objective and multi-modal optimisation.

2.3.2 Gradient-Based Optimisation

Methods that rely on computing a gradient to obtain a solution are among the most popular and powerful tools in numerical and non-linear optimisation. The gradient vector provides information about the direct neighbourhood around one particular point of the function to be optimised, and is used to derive a direction for an optimal update step. Therefore, gradient-based optimisation can be considered as a class of iterative and local optimisation techniques, which are fast and accurate if a proper solution is found. In particular, they quickly converge to local optima that are closest to the current location, but what often ends up running into suboptimal solutions if the optimisation was started from an unfortunate location, as shown in Fig. 2.10. For inverse kinematics, this can be understood as requiring a larger variation of particular joints due to joint limits or objectives which constrain the search space. Generally, chosing the starting location is not straightforward, and usually makes them less robust and scalable than related sampling-based techniques, which will be described in sections 2.3.3 and 2.3.4. A simple technique to avoid this issue is to start multiple times with different initialisations. However, the number of required restarts is not given, and it also remains difficult to decide when to perform a reinitialisation. Additionally, since gradient-based methods are uni-modal techniques, which means they can only track one solution at a time, the progress over last iterations gets lost. In this context, section 4.5.8 of the proposed algorithm shows how evolution can easily be used to reintegrate such information.

Besides the traditional gradient and coordinate descent methods, which always follow the direction of the steepest descent either for the whole vector or each dimension individually, there are further techniques that can be applied to larger-scale problem of higher dimensionality as well as non-linearity. Newton and Quasi-Newton methods make use of the Hessian matrix, which provides a more smooth and robust convergence, but with higher computational cost per iteration. Since convergence is only guaranteed for convex or concave problems, a popular approach is to formulate the objective function using single quadratic terms. [76, 88]

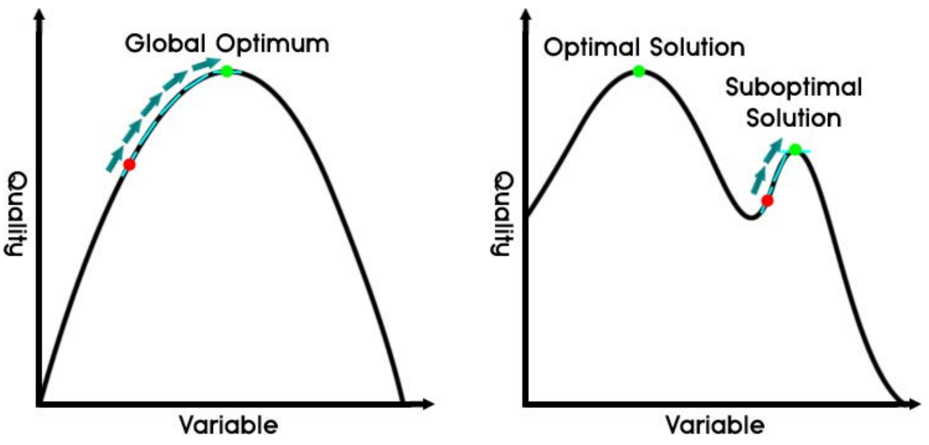


Figure 2.10: Visualisation of gradient-based optimisation.

2.3.3 Evolutionary Computation

Evolutionary computation is an optimisation technique in biologically-inspired artificial intelligence that is driven by the theory of natural evolution of C. Darwin and G. Mendel. Those algorithms simulate a selection, recombination and mutation of chromosomes which follows the principle of *Survival of the Fittest* with respect to those individuals that are most responsive to change. Intuitively, if there is a complex computational problem which is difficult to formulate, but of which it is easily possible to generate samples and to measure their quality, then evolution can be used to solve such problems without requiring particular assumptions over the search space function. The chromosomes of the individuals are the variables of the function to be optimised, and are randomly combined and modified until a satisfactory solution is found. The quality of a solution is described by the fitness, which measures the success of an individual within the population under the objective function. In particular, the objective function can be designed arbitrarily, which is why evolutionary algorithms have proven theirselves especially well-suited for multi-objective and bounded optimisation. In addition, they do not neccesarily rely on computing a gradient, which makes them applicable also for non-differentiable as well as discrete combinatorial problems. However, since they act as global optimisers, they are typically considerably slower than gradient-based local search techniques, as introduced previously in section 2.3.2. More specifically, the fitness evaluation often causes the bottleneck for such algorithms, which makes them difficult to use for time-critical applications. Nevertheless, even simple variants perform comparatively well on all types of problems, so they tend to remain independent from the specific problem complexity. Finally, the concept of having multiple particles for search space exploitation and exploration allows to perform a robust and computationally scalable multi-modal optimisation, as demonstrated in Fig. 2.11. In comparison to Fig. 2.10, evolution performs less efficient if the objective function convex or concave, but becomes more reliable and potentially successful if there are multiple local extrema, and where escaping from suboptimal points is required in order to find better or even global solutions. [26, 31]

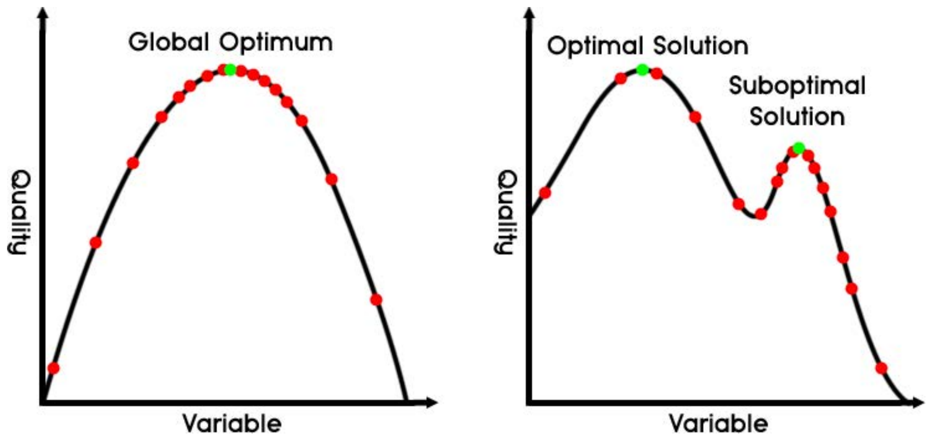


Figure 2.11: Visualisation of evolutionary optimisation.

GA (Genetic Algorithms) are the most popular subfield of evolutionary systems. Those were introduced by J. H. Holland [40], although first motivations were already given earlier in [104, 107, 82, 33]. However, since they represent the most basic version of evolutionary computation as shown in Fig. 2.12, they are often outperformed by many related and more advanced evolutionary methods.

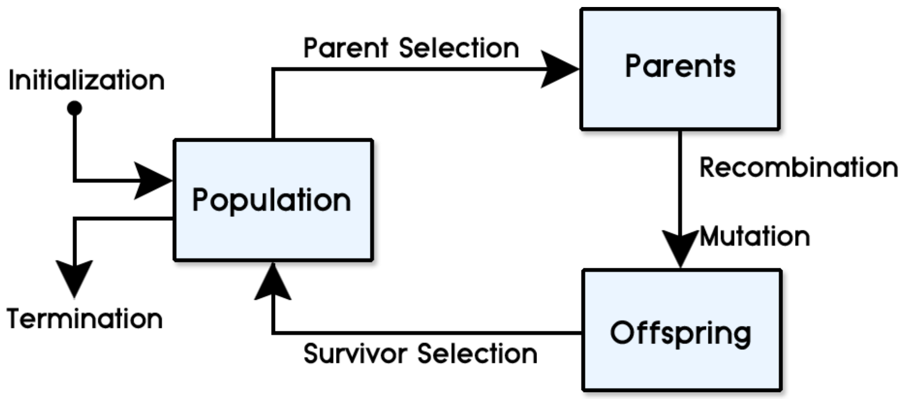


Figure 2.12: Optimisation cycle of GA.

One of them is given by DE (Differential Evolution) [100] which has become a well-acknowledged approach for global continuous optimisation problems. Furthermore, memetic algorithms that were initially inspired by [19] have also gained larger popularity. Those combine evolution with suitable local optimisation methods with the aim to speed up convergence and to increase the accuracy. Lastly, the No-Free-Lunch theorem [113] states that there can be no single approach that performs well under all problems. As visualised in Fig. 2.13, this can be considered as a particular strength of evolutionary algorithms in comparison to specific problem tailored methods. Since they do not require any assumptions on the fitness landscape, they can be observed to perform well over all different problems.

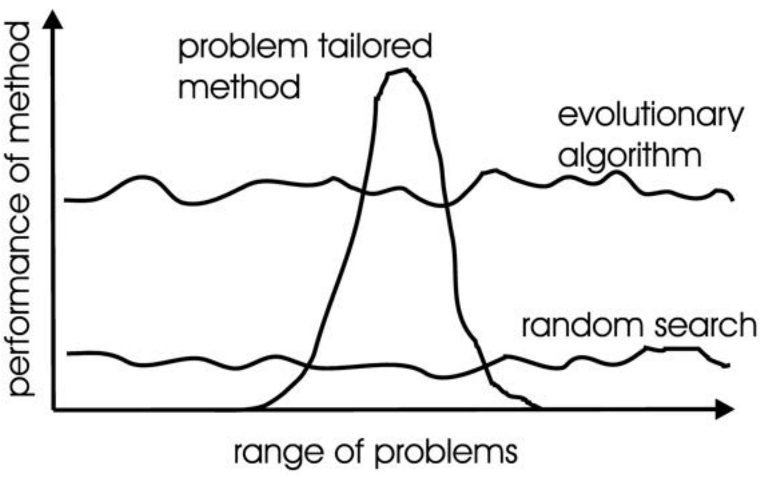


Figure 2.13: No free lunch theorem for evolutionary algorithms. [26]

2.3.4 Swarm Intelligence

Swarm optimisation is a biologically-inspired methodology which aims to imitate the behaviour of natural collective systems in which a group of rather simple organisms can collectively solve complex problems. Intuitively, this can be understood as having a flock of birds or schools of fish that collectively find food sources or avoid predators by adopting behaviour of other individuals. This methodology shares many similarities with evolution in terms of search space exploration, multi-objective and multi-modal optimisation, as well as robustness and scalability, as discussed in section 2.3.3. They also do not require a gradient to be computable. However, instead of iteratively recombining and mutating new particles from existing ones, the particles are flying through the fitness landscape through updating their velocity and direction with respect to their surrounding particles. This is expected to quickly converge towards optimal solutions. In more detail, they are predominantly suited for global continuous optimisation problems, but usually with a higher rate of exploitation than exploration. As visualised in Fig. 2.14, the particles tend to be more attracted by local extrema, and can often be observed to circulate around them. While this usually achieves a higher quality in less time when already being close to good local optima, it also becomes more difficult and unlikely to escape from suboptimal solutions. [26, 31]

This optimisation methodology was initially introduced by J. Kennedy and R. Eberhart [46] with PSO (Particle Swarm Optimisation) as a basic variant. In this, the direction and velocity of each particle is updated according to the success of itself, its locally neighbouring particles as well as the globally best particle. This yields three directions where each of them is individually and randomly weighted. It was later extended in [47] using an additional inertia weight to improve the algorithm. Since then, it has gained large popularity and further variants were introduced, including multi-swarm approaches, neighbourhood topologies, adaptive parameter adjustments, as well as combinatorial versions for discrete optimisation problems. A survey and review over these techniques can be found in [114, 11].

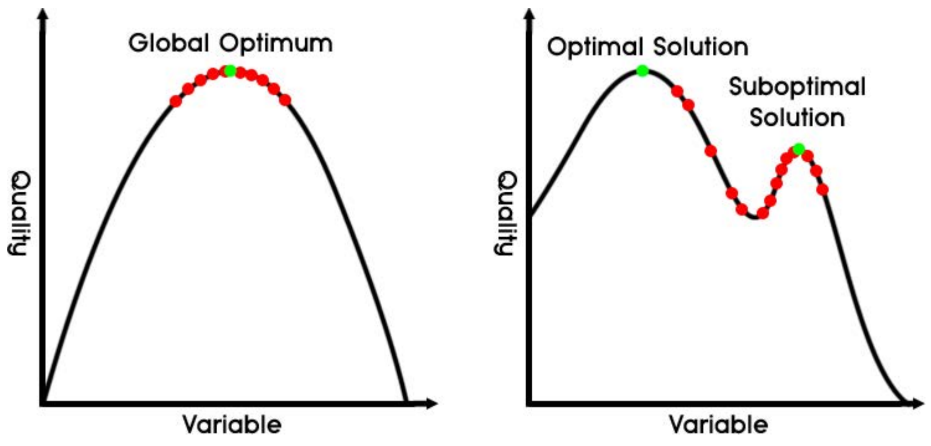


Figure 2.14: Visualisation of swarm optimisation.

Chapter 3

Related Work

This chapter provides a literature review of existing state-of-the-art methods for inverse kinematics. This specifically includes algorithms both from robotics and animation in order to discuss and compare the particular strengths and limitations. Those can be classified as shown in Fig. 3.1. First, section 3.1 mentions some analytical methods to construct the kinematic equations for kinematic chains. Section 3.2 then discusses some traditional and popularly applied gradient-based methods for numerical inverse kinematics in robotics, including Jacobian and Newton methods. A review of heuristic algorithms which are particularly designed for character animation, such as CCD and FABRIK, will be given in section 3.3. Section 3.4 then compares some recent research for inverse kinematics using biologically-inspired sampling-based approaches based on GA and PSO. Lastly, section 3.5 reviews the results that could be obtained by learning-based methods using deep learning or statistical regression. The approaches are then discussed in section 3.6 in order to motivate the chosen design of the Bio IK algorithm in chapter 4.

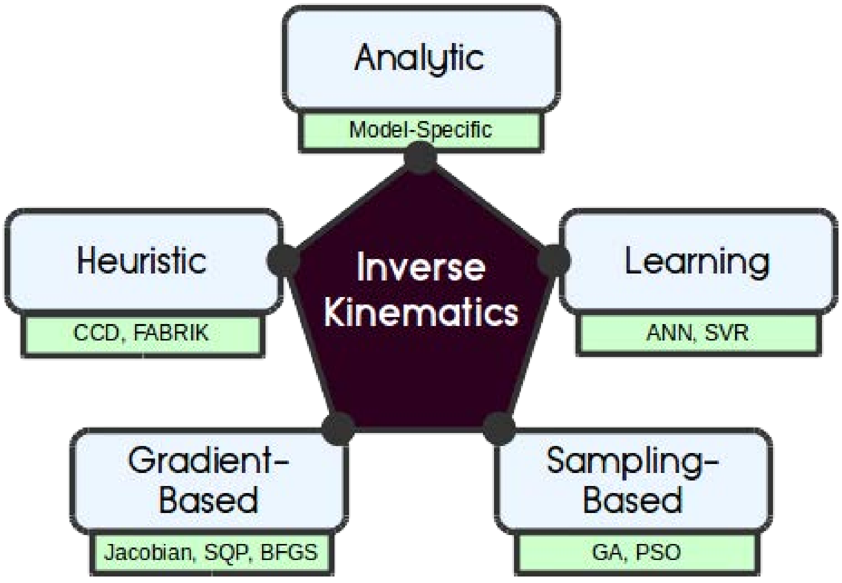


Figure 3.1: Classification of different methods for inverse kinematics.

3.1 Analytic

Given a particular kinematic geometry, analytic methods can be constructed to provide closed-form expressions. The obtained solutions are exact, and can also return all existing joint configurations for a particular reachable Cartesian pose. Although these methods are the fastest among the related methods, they are typically only available for rather simple geometries of lower dimensionality as the complexity rapidly increases with each additional DoF. However, they are sometimes used in combination with other techniques, and to support the optimisation.

In [54], a method was presented which utilises closed-form solutions for parts of the kinematic chains using a non-linear equation solver. Those partial subchains are then combined after solving each of them analytically, which could be demonstrated to improve solving manipulators and robots with high redundancy. In particular, solutions on the Justin robot could be obtained in less than 1 ms. However, the inverse kinematics problem was solved for position only, and it was not clear how to integrate weights for multiple solution criteria.

Furthermore, [108] presented a combination of sampling-based techniques with a precomputation the reachability space. Therefore, it was able to handle arbitrary kinematic chains, and also to generate collision-free solutions of grasp configurations in cluttered environments. For the reachability analysis, a discretised data structure was used to efficiently query the validity of a sampled configuration. The experiments demonstrated promising results in solving grasps on bimanual setups with up to 20 DoF, and for which computation times of less than 1 s were required depending on the number of obstacles.

The IKFast [21] method provides an implementation [22] which automatically generates the algebraic equations. However, it performs well only for kinematic chains with up to 6 DoF, as the equations grow too complex otherwise. Hence, it is rather suited only for serial industrial manipulators, and impractical for solving multiple and highly articulated chains simultaneously.

In [44], a full-body approach for analytical inverse kinematics in animation was presented. The solutions for typical reaching tasks were obtained by interpolating between pre-defined postures which are organised as a functional sequence to the goal. In particular, this method was also able to efficiently avoid joint limits as well as collisions with the environment.

A complete analytical solution for the humanoid NAO robot was published in [52]. They also provided a implementation of the kinematic equations which can obtain solutions for the kinematic chains within only a few microseconds. Nevertheless, the solutions are also only considering position-only inverse kinematics for the single end effectors targets.

3.2 Gradient-Based

Gradient-based methods are among the most widely applied techniques for solving inverse kinematics in robotics. This is reasoned by their ability to achieve high accuracy within a proper short amount of time, and since they can also be applied to different geometries. As they require approximating first- or second-order derivatives, they are more computationally expensive than related analytic and heuristic methods in sections 3.1 and 3.3. However, they can directly operate in joint space which eases the use of joint limits, and seem to be more flexible to include additional objectives, such as orientation goals for solving full pose targets.

3.2.1 Jacobian Methods

Most traditionally used methods for robotic inverse kinematics are based on computing the Jacobian (3.1), which is a matrix of first-order partial derivatives of each joint variable, and yields a linear approximation of the resulting end effector velocities in Cartesian space. [14]

$$J(\theta)_{ij} = \left(\frac{\delta \mathcal{X}_i}{\delta \theta_j}\right) \tag{3.1}$$

The task then is to find an appropriate update of the joint variable configuration $\theta' = \theta + \Delta\theta$ such that the error vector $\vec{e}$ in direction from the segment to the given target is minimised as smoothly and quickly as possible. The most simple and computationally cheap version is using the transpose method (3.2), where α defines a small factor to move in direction of the gradient [6, 112]. This solution generates smooth motion, but usually suffers from a slower convergence.

$$\Delta\theta = \alpha J^T \vec{e} \tag{3.2}$$

A quicker optimisation can be obtained by matrix inversion. However, calculating the inverse is not always possible as it requires the DoF to be similar to the dimensionality of the error vector, which is why many manipulators are intentionally designed with exactly 6 DoF. Therefore, a common method to handle geometries with different DoF is using the Moore-Penrose pseudoinverse (3.3) method [111, 63]. Nevertheless, it can be observed to become instable in near-singular configurations, which can cause jittery motion and no solutions to be found at all.

$$\Delta\theta = J^T (J J^T)^{-1} \vec{e} \tag{3.3}$$

A way to avoid such issues is by using the DLS (Damped Least Squares) method (3.4), which is also known as the Levenberg-Marquardt technique. It was firstly applied to inverse kinematics in [109, 37], and further extended with SVD (Singular Value Decomposition) in [13]. However, chosing the damping constant λ is not trivial, and can considerably slow down the optimisation if chosen too high.

$$\Delta\theta = J^T (J J^T + \lambda^2 I)^{-1} \vec{e} \tag{3.4}$$

In Fig. 3.2, the different approaches are compared regarding their computational performance and accuracy. The pseudoinverse method is not contained due to the discussed singularity issues. It can be observed that the DLS and SVD-DLS methods are able to outperform the transpose technique when a higher accuracy is desired, or a larger number of iterations is available. However, while such methods do usually not scale well for higher articulated geometries, an extended version of the Jacobian was proposed in [103] for solving inverse kinematics on humanoid robots. In [48], the Jacobian method was also successfully used for real-time character animation tasks, achieving good results on different articulated geometries.

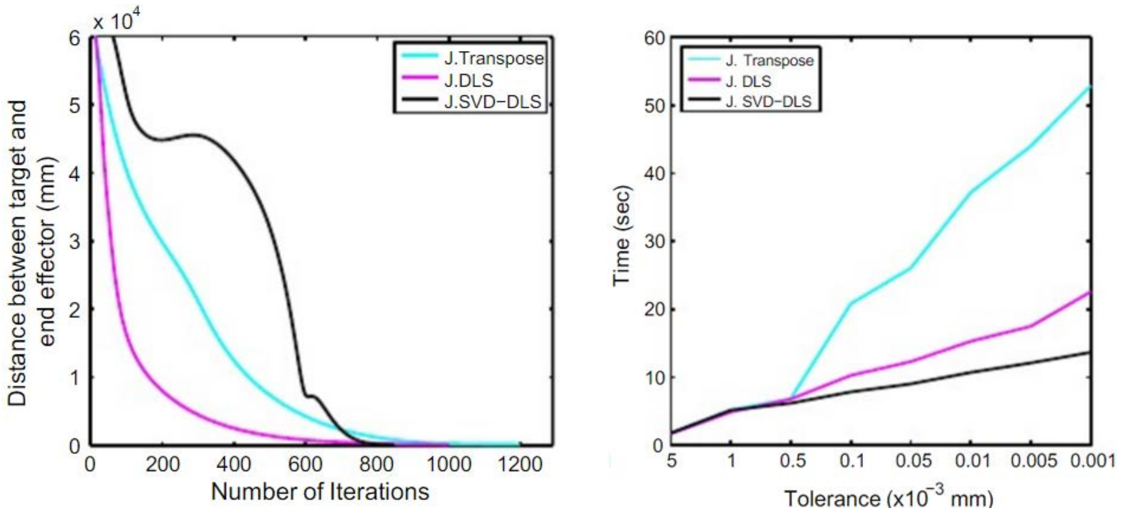


Figure 3.2: Jacobian Transpose, DLS and SVD-DLS comparison on 10 DoF. [4]

A popular implementation which is commonly being used with robotics software such as ROS is available by the Orocos KDL framework [50]. A major drawback of the Jacobian method is that it largely suffer from multiple local extrema, which can cause no acceptable solution to be found at all. This can be particularly observed for the results on different serial manipulators listed in Tbl. 3.1, which were obtained from 10000 randomly-sampled and reachable queries with a Cartesian pose error of 10^{-5}. Although the DoF is relatively low, using an optimisation timeout of 5 ms does not seem to achieve satisfactory results in terms of reliability.

Manipulator	**DoF**	**Time**	**Success**
PR2 Arm	7	1.37 ms	83.14%
Baxter Arm	7	2.21 ms	61.07%
KuKA LBR iiwa 14 R820	7	3.37 ms	37.71%
UR5	6	3.30 ms	35.88%
NASA Valkyrie Arm	7	3.01 ms	45.18%

Table 3.1: Orocos KDL performance on serial manipulators. [80]

3.2.2 Newton Methods

More promising results could be achieved using Newton-based optimisation for inverse kinematics. Although this technique is more costly than the Jacobian methods in section 3.2.1 as it utilises computation of second-order derivatives, a more reliable optimisation with higher robustness for non-smooth functions could be observed. In order to lower the computational cost per iteration, Quasi-Newton methods were designed to only estimate the inverse of the Hessian matrix using a second-order Taylor approximation instead of fully calculating it [60, 76, 88]. One of the most popularly applied and acknowledged methods is given by the BFGS (Broyden-Fletcher-Goldfarb-Shannon) algorithm [12, 30, 35, 90]. The algorithm was originally designed for unconstrained optimisation, and required an expensive calculation of the Hessian matrix. It was then further improved to the L-BFGS-B (Limited-Memory BFGS with Bound Constraints) method, which has improved computational performance in approximation of the Quasi-Newton method [75], and also allows incorporation of box constraints for each variable [15]. Intuitively, its particular strength is that it robustly allows optimisation on arbitrary twice-differentiable objective functions Ω 3.5, and performs notably well and efficient even with a very high number of variables.

$$\Omega(x+\sigma) \approx \Omega(x) + [\nabla\Omega(x)]^T\sigma + \frac{1}{2}\sigma^T H_\Omega(x)\sigma \tag{3.5}$$

The method was successfully applied for complex inverse kinematics on highly articulated geometries in animation and motion capture [115, 36]. Smooth joint variable updates with a fast convergence could be observed, and which also did not suffer from singular configurations. Therefore, good results could also be achieved in robotics. The *TRAC-IK* plugin for ROS [80] utilises a combination of the BFGS algorithm with SQP (Sequential Quadratic Programming) [7]. Further heuristic restarts from random initial seeds were integrated with the aim to detect and avoid suboptimal configurations. The results in Tbl. 3.2 show better results on serial kinematic chains, and can outperform the previous Jacobian results in Tbl. 3.1 in terms of faster as well as more reliable optimisation using the same experimental setup. Still, it remains unclear how the algorithm performs in solving multiple segment poses, which is currently not supported by the implementation.

Manipulator	**DoF**	**Time**	**Success**
PR2 Arm	7	0.59 ms	99.84%
Baxter Arm	7	0.60 ms	99.17%
KuKA LBR iiwa 14 R820	7	0.56 ms	99.63%
UR5	6	0.42 ms	99.55%
NASA Valkyrie Arm	7	0.61 ms	99.63%

Table 3.2: *TRAC-IK* performance on serial manipulators. [80]

3.3 Heuristic

Heuristic methods for inverse kinematics iteratively estimate the required joint updates using geometric calculations. They are more popularly applied in graphics and animation since they require less time per iteration, and thus are often preferred in order to maintain high frame rates. Most prominent algorithms are given by CCD and FABRIK, and for which different variants are implemented in [74] and [57], as well as used in modern graphics engines such as Unity3D [105], Unreal [27] and Maya [61]. However, the algorithms are mainly optimising joint positions rather than joint values. This is often sufficient for applications such as motion capture and animation editing as well as post-processing, but makes it more difficult to have a direct relation between Cartesian and joint space in order not to violate joint limits. In addition, it generally becomes more difficult to integrate further constraints and objectives, and the methods seem more suited for solving position-only goals.

3.3.1 CCD

The CCD (Cyclic Coordinate Descent) [110, 17] is a traditional and one of the most popular heuristic algorithms. It is simple to implement and only requires very little computational cost per iteration, which makes it applicable for processing multiple characters simultaneously. Each joint along the kinematic chain is updated indepedently, iterating from the last to the first link. The calculations can be done purely by using dot and cross product operations between the positions of the end effector p_{ee} and the current joint and p_i with respect to the target position p_t. The required angular change $\Delta\theta_i$ of the current joint about the derived normal vector $\vec{n}$ can then be calculated as denoted in (3.6) and (3.7).

$$\Delta\theta_i = cos^{-1}(\frac{p_{ee} - p_i}{||p_{ee} - p_i||} \cdot \frac{p_t - p_i}{||p_t - p_i||}) \tag{3.6}$$

$$\vec{n} = \frac{p_{ee} - p_i}{||p_{ee} - p_i||} \times \frac{p_t - p_i}{||p_t - p_i||} \tag{3.7}$$

In partiular, the rotation normal is obtained at each calculation step instead of allowing to use specified rotation axes about which the motion is constrained. This makes it less predictable for the optimisation since joint limits can not be specified directly for the single axes. As a result, the method can often be observed to produce unrealistic postures, and tends to overestimate particular joints along the kinematic chain. Therefore, the amount by which each joint is updated within every optimisation step is usually controlled by an additional custom weight. Nevertheless, since CCD operates locally on each joint, it is rather difficult to efficiently combine multiple kinematic chains. More specifically, the equations shown in (3.6) and (3.7) break if there are multiple end effectors or segments that shall be solved simultaneously. Also note that only the position of the end effector is optimised, and the method becomes more complex if solving an orientation goal is also desired, in which case the relative rotation also needs to be incorporated.

3.3.2 FABRIK

FABRIK (Forward and Backward Reaching Inverse Kinematics) [4, 3, 5] is a more recent algorithm which has quickly gained strong popularity and relevance in character animation. Similarly to the introduced CCD method in previous section 3.3.1, it optimises joint positions in Cartesian instead of operating in joint space. However, instead of traversing a single direction along the chain, it consists of two consecutive phases in an iterative and recursive forward and backward reaching mode. Let $\lambda_i = \frac{d_i}{r_i}$ be defined by the initial and new distances d_i and r_i between two joint positions p_i and p_{i+1} with $i = 1, ..., n-1$ and p_n being the target position of the end effector, then the forward (3.8) and backward (3.9) calculations iteratively optimise the required joint locations $p_{1,...,n}$ to reach a desired position for the end effector.

$$p_i = (1 - \lambda_i)p_{i+1} + \lambda_i p_i \tag{3.8}$$

$$p_{i+1} = (1 - \lambda_i)p_i + \lambda_i p_{i+1} \tag{3.9}$$

Intuitively, FABRIK solves articulated postures through finding points on lines recursively instead of calculating rotational joint updates with respect to the end effector for each joint individually. This technique is fast, scalable, and provides several advantages over CCD. In example, multiple end effectors can be solved simultaneously and it does also not tend to overextend particular joints. However, further inclusion of orientational joint limit constraints is rather diffcult, and the method remains unapplicable for applications in robotics where solutions in joint space are required.

In Fig. 3.3, CCD and FABRIK methods are compared regarding their required amount of iterations and computation time to reach a desired accuracy for an end effector position, and for which FABRIK produces considerably better results.

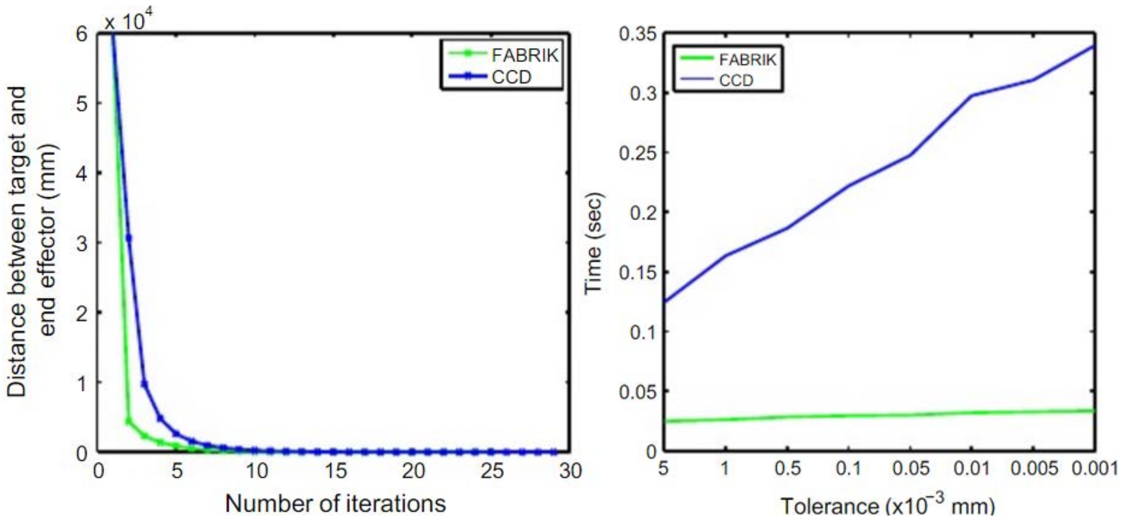


Figure 3.3: CCD and FABRIK comparison on 10 DoF. [4]

3.4 Sampling-Based

Sampling-based methods based on biologically-inspired artificial intelligence provide a generic and robust technique for bounded non-linear optimisation. They offer high flexibility to design custom multi-objective functions, and perform well even under high-dimensional and non-convex search spaces. In particular, they are more robust in avoiding suboptimal extrema, but typically remain considerable slower than previous introduced gradient-based and heuristic methods in sections 3.2 and 3.3. They do not suffer from singularities and can directly operate in joint space through encoding joint variable configurations as particles. In addition, while integrating joint limits is often difficult for other methods, it turns beneficial for sampling-based techniques as it allows to restrict the search space.

3.4.1 Genetic Algorithms

Solving inverse kinematics by using GA is not novel, and has already been presented initially in [78]. In this, the flexibile applicability of these methods on generic and redundant geometries for solving single end effector configurations was demonstrated. Later, genetic niching methods have been used in [101, 102] to find multiple solutions of joint configurations for a given target pose on industrial manipulators, and it was possible to obtain smaller error rates both for position and orientation. Since GA are often costly and require expensive fitness evaluations, [1, 49] demonstrated the higher computational efficiency when using parallel implementations. In particular, methods based on DE and memetic evolution seem to have recently gained a larger popularity. A hybridisation of evolution and Jacobian optimisation was proposed in [28]. Multiple parallel local searches were implemented to achieve a higher accuracy in less time and fewer iterations using robotic geometries with up to 7 DoF. Another memetic variant combined with DE was also proposed in [106]. In this, the method was not only applied to inverse kinematics, but also to path generation problems on serial as well as more complex anthropomorphic geometries. Nevertheless, although the algortihm performed better than the original DE, the accuracy was only about 10^{-3}m/rad Cartesian error, and computation times of a few hundred milliseconds for inverse kinematics or up to a few seconds for path generation were required. Some results are shown in Fig. 3.4, which visualises the success rate and computational cost for their method on serial redundant manipulators. Considering the success rate, it was demonstrated that solutions can be robustly found by evolution, and for which the required number of generations was much lower using their proposed δDE (Memetic DE) compared to DE. An improvement could also be observed regarding the computation time. Nevertheless, it still remains too high for real-time applications. Finally, [99] presented an evolutionary method to generate task-specific motion trajectories with multiple constraints on the iCub humanoid robot. Their inverse kinematics solution was able to robustly handle further objectives for inverse kinematics, such as the viewing direction of the head for inspecting objects from different angles, or for bimanual manipulation tasks while providing collision-free solutions.

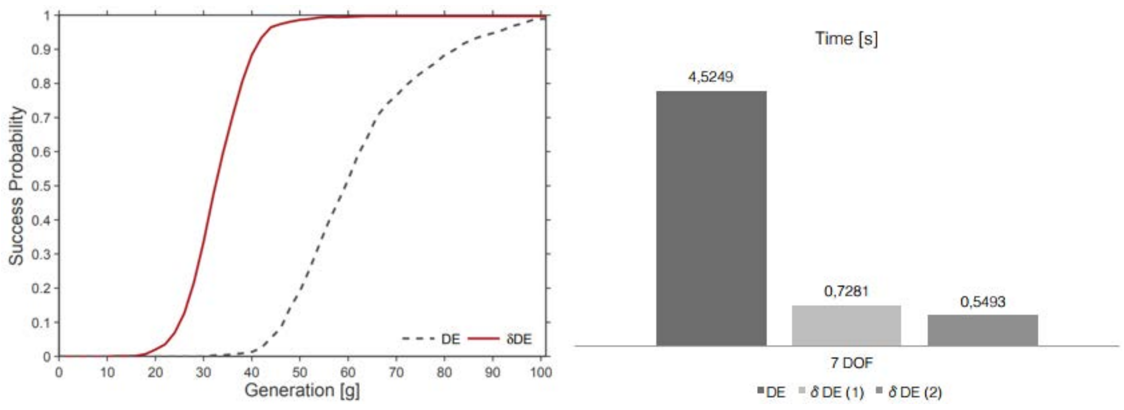


Figure 3.4: Success rate and time for DE and δDE. [106]

3.4.2 Particle Swarm Optimisation

PSO represents a more recent optimisation technique than GA in previous section 3.4.1, and which could also be demonstrated to be suitable for inverse kinematics problems. Throughout the literature, similar benefits as for GA are reported due to their generic optimisation methodology for which no specific conditions or assumption are required. In [25, 41], the method was applied to solve end effector configurations for serial and redundant robotic manipulators. However, inverse kinematics was only solved for raw position goals. More challenging scenarios were considered in [83], where an extended variant of PSO was applied for providing collision-free solutions in cluttered environments. Furthermore, full pose targets were successfully solved in [16] with sufficiently high accuracy. A statistical analysis and applicability for biped gait generation was done in [85, 84]. Lastly, [18] demonstrated the scalability of PSO on hyper-redundant serial manipulators with up to 180 DoF. Their results for inverse kinematics with single pose goals are listed in Tbl. 3.3. Although the required computation times are relatively high, solutions are found for which Jacobian methods would likely run into singularities. In addition, all optimised solutions were free from self-collisions and also did not result in collisions with obstacles in the environment. The method was also applied for generating collision-free paths in environments with many obstacles with cube shapes. The experiments aimed to demonstrate the scalability of their method in environments with up to 150 obstacles. However, the collision-checking was found to be rather expensive, and obtaining solutions typically required a minute in total.

DoF	30	60	90	120	150	180
Time	1.57 s	3.36 s	7.46 s	15.47 s	22.11 s	37.03 s

Table 3.3: PSO performance on hyperredundant serial manipulators.[18]

3.5 Learning

Another perspective for solving inverse kinematics comes with learning methods. Since it is not generally available to obtain a closed-form analytic expression for generating solutions, the aim of using learning approaches is to find an approximate of this functional mapping between joint and Cartesian space. Once this function is learned, solutions can be generated very quickly and repeatable, which offers an advantage over numerical optimisation methods. However, since inverse kinematics typically includes multiple solutions for Cartesian targets, the issue becomes that learning can start interpolating between multiple training samples. Therefore, the choice of training samples remains difficult, and the error can eventually remain too large for many practical applications if the whole workspace shall be learned.

3.5.1 Artificial Neural Networks

ANN (Artificial Neural Networks) are biologically-inspired models which are suitable for learning non-linear functions. In [43], those were applied to solving inverse kinematics poses on the SCARA serial manipulator of 4 DoF using a standard MLP (Multi-Layer-Perceptron) architecture. The solutions were found to be accurate enough, but were only conducted for a relatively simple and low-dimensional geometry. A combination with GA was proposed in [53] using the 6 DoF Stanford manipulator and achieved micrometer accuracy, but only considerd position-only targets. An ELM (Extreme Learning Machine) with a single hidden layer MLP was used for the 7 DoF PUMA 560 robot arm in [29]. This method was able to considerably lower the required training time, and also to improve the accuracy when solving pose goals. However, only very small training sets with a maximum of 1000 training samples were used, and it remains unclear how much of the actual workspace of the robot has been learned. Recently, [2] proposed a solution which additionally uses the current joint variable configuration as input besides the desired end effector pose for a 6 DoF robot. They reported considerably better results in terms of motion control, and demonstrated smoothly generated paths and end effector trajectories. Nevertheless, there seems no literature available which applies these models for more complex and highly-articulated geometries.

3.5.2 Support Vector Regression

Similar results as for ANN in section 3.5.1 could also be achieved using SVR (Support Vector Regression). In [73], an additional spatial decomposition method and an analytic method for the orientation were integrated for solving poses of a 7 DoF manipulator. Although queries could be computed very quickly, this technique formulates a very specific non-generic setup, and position errors of 10^{-3} m remained. A comparison of SVR and ANN has been conducted in [89] using the 7 DoF PA-10 model. It was found that SVR requires less training data, and does not get stuck in suboptimal extrema configurations. However, there also seems no research conducted for geometries with higher DoF or multiple concurrent targets.

3.6 Discussion

Among the discussed methods in this chapter, it could be observed that there exists a huge discrepancy between the different methods regarding accuracy and computational cost, robustness and scalability, as well as flexibility to be applied for different applications and kinematic geometries. Even more, their particular strengths and limitations seem mutually exclusive. However, analysing their aspects was essentially relevant for the chosen design of the Bio IK algorithm that will be presented in the following chapter 4.

This research has the aim to provide a generic method for multi-objective inverse kinematics. Hence, analytic solutions are not an option since they need to be constructed individually for each geometry. Also, they are typically not available for kinematic structures with higher DoF, such as humanoid robots or virtual game characters. For those, it is rather impracticable to find explicit or a combination of closed-form solutions in order to solve postures for full-body motion.

The results of gradient-based methods demonstrated a fast and accurate convergence, and which could be applied to generic geometries. In particular, SQP as well as Quasi-Newton methods like the BFGS algorithm could be observed to provide a more robust and flexible optimisation in terms of constraint design than the popularly applied Jacobian approaches. However, the suboptimal extrema issue remains, and collision avoidance was neglected for such methods. Finaly, no research explicitly considered solving multiple kinematic chains and objectives simultaneously, so it is difficult to tell how they scale for higher kinematic complexity.

The heuristic CCD and FABRIK methods are well-suited for character animation tasks, but usually not applicable for real robots. This is because joint limits can be violated, solving orientation is not easily available, and motion axes can not be specified but are derived as required by the geometric calculations.

In contrast to gradient-based and heuristic techniques, a review of related GA and PSO methods showed much slower convergence and lower accuracy. Nevertheless, they excelled in terms of robustness and scalability, and also covered the only research that considered significantly higher DoF, whilst other methods were only applied to 6 or 7 DoF geometries. In addition, they could also be demonstrated to be applicable for generating collision-free solutions and trajectories.

Furthermore, ANN and SVR which aim to learn the inverse kinematics function have been demonstrated to be a possible alternative to analytic solutions for relatively simple geometries. Nevertheless, they must be trained individually for each geometry, and seem to be unapplicable for more complex setups.

In conclusion, gradient- and sampling-based methods suggest to be combined, which together could serve the different performance aspects for inverse kinematics.

Chapter 4

Bio IK Algorithm

This chapter presents the developed Bio IK algorithm. The chosen algorithmic design is mainly motivated by the literature review in previous chapter 3, and can be used for solving complex and highly articulated postures in robotics and animation. The following sections are based on the research that was published in [94, 97, 87, 95, 96, 55].

This chapter starts with a problem statement in section 4.1 which motivates the performance requirements and design challenges for the proposed algorithm.

The general algorithmic scheme will be shortly introduced in section 4.2. This includes a visualisation and higher-level explanation of the memetic optimisation framework, as well as an overview of mathematical symbols that are used during the following sections.

Section 4.3 then describes the genetic encoding scheme of joint variable configurations as well as the formulation of constraints for the evolutionary optimisation.

Followingly, the objective function design is presented in section 4.4. This particulary covers a detailed explanation and discussion of several objectives which can be directly used for creating custom cost functions.

The single evolutionary phases are then discussed in section 4.5. A mathematical description is provided for each of them, along with some visualisations for the main phases of the algorithm.

Section 4.6 further discusses some computational improvements which are optional for the algorithm. Those include a data structure to efficiently calculate the forward kinematics equations, as well as a possible integration of multi-core threading.

Finally, section 4.7 provides a pseudocode for the complete algorithm, as well as a reference to existing implementations in Unity3D (C#) and ROS (C++) that are ready to be used for robotics and animation.

4.1 Problem Statement

The aim of this research is to provide an efficient and generic method for inverse kinematics. The proposed algorithm seeks to extend the traditional understanding of inverse kinematics by the ability to handle multiple kinematic chains simultaneously, defining custom objectives both in joint and Cartesian space, and to be robustly applicable for solving postures on fully-constrained and highly articulated structures. The intention is to provide a solution that can be used both for robotics and animation, and can be individually extended for custom needs and requirements. The different methods that were discussed in the previous chapter 4 all have their particular benefits, but are either suffering from local extrema, high computation time, low scalability, or being difficult to extend for multiple or different objectives. Therefore, the aim of the proposed algorithm aims to serve five different performance aspects for solving inverse kinematics, which can be formulated as follows:

- **Success** – An existing solution for a given accuracy under the defined objectives and constraints can also be found within a specified amount of time.
- **Accuracy** – The solution shall be as precise as required, typically with an error below 10^{-3}m/rad.
- **Time** – The solution shall be found as fast as possible, preferably within a very few milliseconds, but never more than 10 ms.
- **Continuity** – The distance between solutions in joint as well as Cartesian space shall be as minimal as possible.
- **Adaptivity** – Robustness, scalability and fast convergence can be maintained for varying kinematic geometries.
- **Flexibility** – The algorithmic methodology allows adding further objectives and constraints, and can be extended to fulfil task-specific requirements.

In order to meet these challenges, biologically-inspired optimisation algorithms can provide robustness and scalability, while gradient-based methods can offer a fast and accurate convergence — which combination results in a memetic evolutionary optimisation strategy. This methodology could be demonstrated to perform well, but the challenge remains in finding a suitable combination of global and local optimisation methods. In particular, not every local search technique can be flexibly used for custom objective functions and concurrently be applied to constrained optimisation problems, but which is important in order to create a generic memetic algorithm. Surprisingly, although the L-BFGS-B method can serve these requirements, it seems that this hybridisation has not yet been applied to the inverse kinematics problem. The research question then becomes how well this combination can perform for the different challenging tasks and applications in robotics and animation, which will be extensively investigated in chapter 5.

4.2 Algorithmic Scheme

The general scheme of the proposed Bio IK algorithm is visualised in Fig. 4.1, and represents a memetic evolutionary framework for bounded continuous optimisation problems. It is based on a hybridisation of GA and PSO, which is implemented by combination of the recombination and the additional adoption phase. These phases together introduce an evolutionary momentum for the individuals, which is gradually updated and propagated over generations. Intuitively, this heuristic aims to follow directions which achieved improvements during previous iterations, and lets offspring keep particular characteristics of successful individuals. For the mutation phase, an extinction operator was designed which adaptively controls exploration and exploitation, reduces the required number of parameters, and is suitable for varying problem dimensionalities. This is especially relevant when solving very different kinematic geometries of varying DoF. The niching phase aims to concurrently explore different regions in the search space in order to avoid premature convergence in suboptimal regions. The elitism exploitation then implements the memetic combination, which specifically speeks to enhance potentially good individuals among the population. Those are given by a dynamically selected set of elites, and which control the number of concurrently tracked solutions. In addition, both the global and local search can utilise the same objective function through an additional gradient approximation on the fitness landscape. Finally, configurations might occur in which all niches are stuck in suboptimal regions. Therefore, the wipe out criterion was designed to heuristically detect such situations, and in which case a partial reinitialisation is performed.

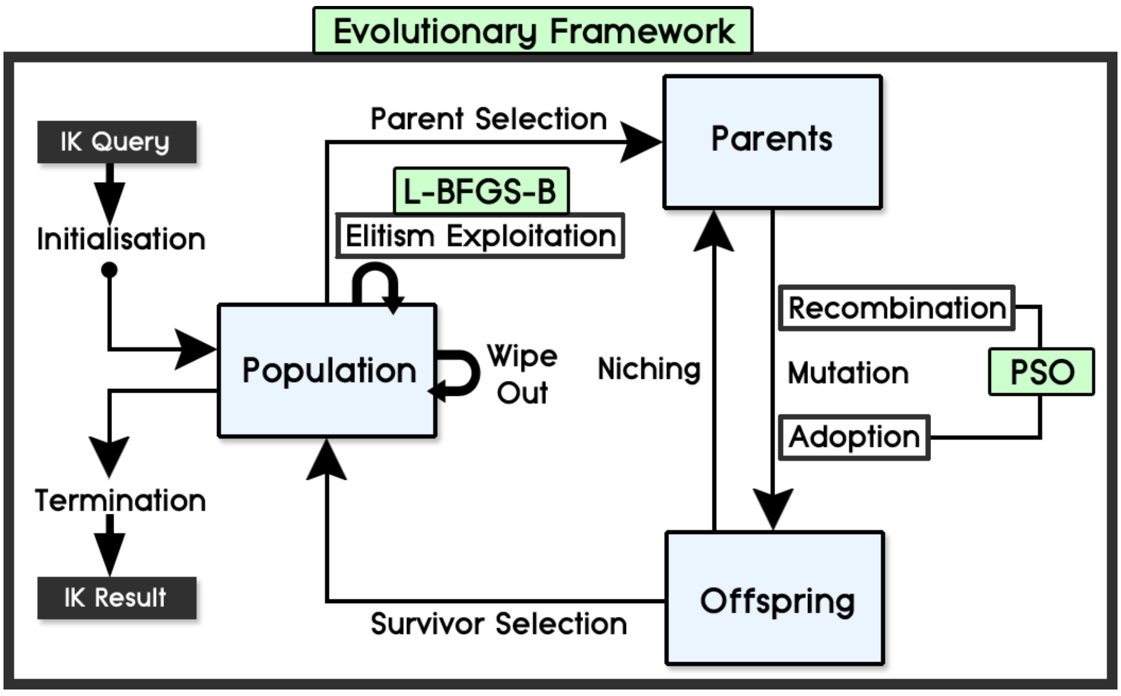


Figure 4.1: Algorithmic model of the Bio IK algorithm.

Fig. 4.2 visualises the optimisation strategy of Bio IK, which combines the previously discussed benefits of gradient-based, evolutionary and swarm optimisation in terms of a fast, robust and multi-modal optimisation in Fig. 2.10, 2.11 and 2.14. The extinction operator and niching phase aim to maintain a reasonable genetic diversity by covering information from different regions in the search space, regardless of how much progress has been done so far by the population. At the same time, recombination and adoption aims to produce a higher density of individuals around potential solutions. The memetic exploitation then performs a gradient-based search on some potentially good solutions, and the mutation helps to get around non-smooth regions in the fitness landscape. Lastly, Tbl. 4.1 lists the mathematical symbols which are used consistently used over the next sections.

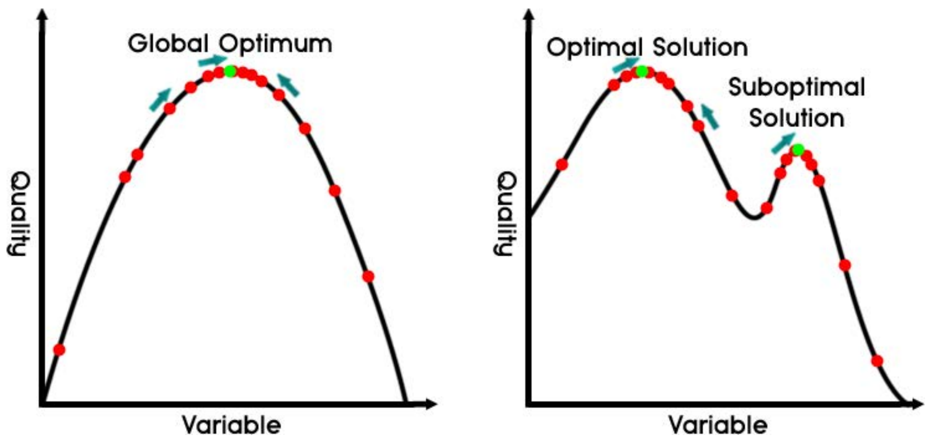


Figure 4.2: Visualisation of memetic evolutionary optimisation of Bio IK.

θ	Joint variable configuration
n	Problem dimensionality
x	Genotype
g	Momentum
ξ	Extinction
ϕ	Fitness
Ω	Objective function
$\mathcal{L}$	Objective loss term
Ψ	Population of size φ with $\Psi_{1,\ldots,\varphi}$
$\mathcal{E}$	Elites of size κ with $\mathcal{E}_{1,\ldots,\kappa}$
Γ	Mating pool
S, R, M, A, E, W	Evolutionary operators for [S]election, [R]ecombination, [M]utation, [A]doption, [E]xploitation and [W]ipe out
$U_{[a,b]}$	Uniformly-distributed random value between a and b
$I_c(a,b)$	Linear interpolation between a and b weighted by c

Table 4.1: Mathematical symbols for the algorithmic formulations.

4.3 Gene Encoding

Intuitively, the basic concept of solving inverse kinematics through evolution is to encode body postures as individuals, which are then evolved until a satisfactory solution is found. This can be done either through using quaternion-vector or joint value representations. The former requires seven dimensions for each local transformation between segments, and makes it more difficult to consider joint limits. The latter representation requires as many dimensions as given by the DoF in total, and can directly integrate joint limits. Therefore, the latter method is used for which a joint variable configuration θ is encoded as the genotype x of an individual. This can be denoted as (4.1), which uses a real-valued gene encoding for each related joint, and gives rise to a n-dimensional search space.

$$\theta \mathrel{\hat{=}} x = \Big(x_1 \mid x_2 \mid x_3 \mid ... \mid x_{n-1} \mid x_n\Big) \tag{4.1}$$

Integrating joint limits is an essential task when operating on real robots, or to achieve realistic postures on virtual characters in games and films. Thus, each gene is constrained (4.2) by the lower and upper limits of its particular joint, and which is simply clipped in case of exceeding its bounded search space domain. If a joint has no limits, no constraints are assigned to the gene and it can evolve freely.

$$\theta_{i_{min}} \leqslant x_i \leqslant \theta_{i_{max}} \quad \forall i = 1, ..., n \tag{4.2}$$

Note that the set of all existing joints in the segment hierarchy is encoded instead of each kinematic chain for itself. Thus, each evolved individual directly represents a valid joint variable configuration for the whole body posture, which is an advantage over related methods that often require costly post-processing in order not to violate joint limits. The resulting transformations for the body posture can then be obtained by evaluating the genotype x through forward kinematics (2.10).

4.4 Objective Function

The design of the objective function usually outlines one of the most crucial challenges in evolutionary optimisation. In particular, especially if multiple objectives shall be considered concurrently, the formulation of the function is not straightforward. This is because the single loss terms must be formulated to be in a sensible relation to each other, so that their minimisation occurs equally. Furthermore, the combination of those should not cause strong fluctuations in the resulting fitness landscape, as this would increase the chance to get stuck in suboptimal regions. Instead, smooth transitions between regions are preferred, and all objectives should independently yield the same value for the global optimum.

With this in mind, the fitness ϕ of an individual under the objective function Ω (4.3) that shall be minimsed is evaluated using the RMSE (Root-Mean-Square-Error) over all single objectives whose loss terms are calculated by $\mathcal{L}$. In addition, each objective can be further weighted by w to integate customisable posture specifications. This supports task-specific needs, such as preferring higher accuracy in position over orientation while forcing a joint along to arm to keep a particular joint value. Another example is desiring a smaller error for the finger tips than for the viewing direction of the head or the position of the elbow. This allows to easily define target priorities for manipulation or interaction tasks.

$$\phi = \Omega(x) = \sqrt{\frac{1}{k}\sum_{j=1}^{k} w_j \mathcal{L}_j^2(x)} \tag{4.3}$$

By using a squared error metric to combine the different objectives, the aim is to transform the initial linear problem into a quadratic which introduces smoothness, and eventually leads to a near-convex solution for the optimisation. Finally, the objective function formulates a single-objective evolutionary optimisation problem, but in which multiple objectives for inverse kinematics are combined. A list of objectives which are currently available for the algorithm are presented during the following sections. All objectives are formulated in a way such that they yield zero error for an optimal solution. Note that further objectives can be easily added to the algorithm, and only need to describe a twice-differentiable continuous function that becomes minimised to zero if a solution is found.

4.4.1 Position

A position objective aims to minimise the translational error between a segment and a given target it shall reach. It represents the most elementary goal for inverse kinematics besides the orientation objective, which will be described in the next section 4.4.2. Both together then define the full pose of a segment. Although both position and orientation could be combined in a single pose objective, the design decision was to define single objectives for both since solving them together may not always be required. However, in contrast to orientation errors which always fall into the range $[0, \pi]$, position errors can grow arbitrarily large. Intuitively, specifying a target at the end of the room yields a considerably higher position error than if right in front at a table, but the orientation error can still remain the same and never grow larger than π. Therefore, a normalisation of the position error needs to be done so that the evolution equally minimises both the errors in position and orientation. This normalisation mainly depends on the size of the kinematic geometry. Therefore, the normalisation is performed according to (4.4), where $d = ||\mathcal{Y}^{pos} - \mathcal{X}^{pos}||$ denotes the Euclidean distance between the position of the Cartesian target $\mathcal{Y}$ and the resulting segment transformation $\mathcal{X}$, L is the constant length of the kinematic chain from the root to the segment, and λ is the variable distance from the root to $\mathcal{X}$ under the evaluated joint variable configuration, as visualised in Fig. 4.3. Multiplying by π finally yields a position error between $[0, \pi]$ to be used for optimising the position loss term.

$$\mathcal{L}_{Position} = \frac{\pi d}{\sqrt{(L + d)(\lambda + d)}} \tag{4.4}$$

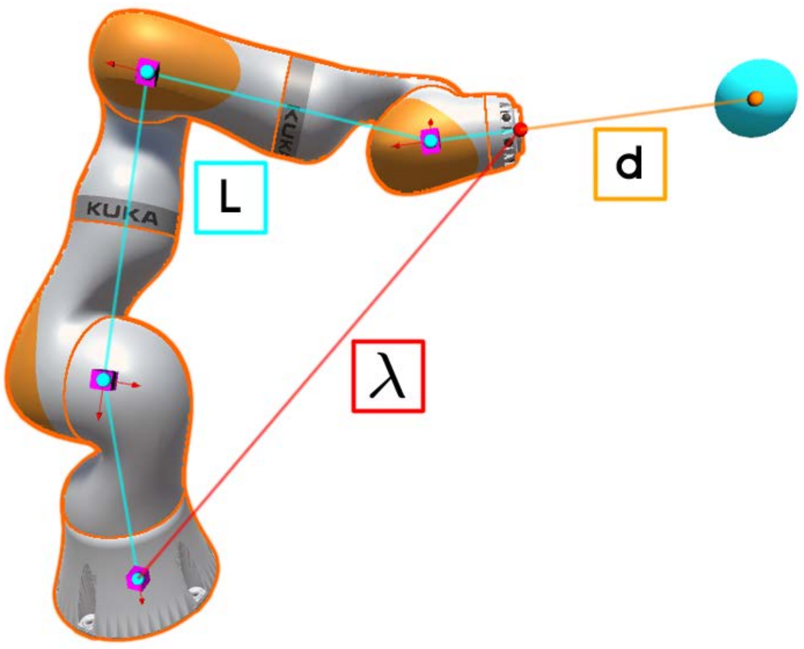


Figure 4.3: Visualisation of the position objective.

4.4.2 Orientation

In addition to the position objective in section 4.4.1, solving an orientation objective as shown in Fig. 4.4 is usually also desired, and neccessary in order to solve full pose goals for the segments. The loss for the orientation error can be defined through calculating the quaternion dot product between a particular segment $\mathcal{X}$ and its Cartesian target $\mathcal{Y}$, as denoted by (4.5). This yields an error between $[0, \pi]$ which becomes zero if all Cartesian axes of the segment and the target are aligned.

$$\mathcal{L}_{Orientation} = 2\cos^{-1}(\mathcal{X}^{quat} \cdot \mathcal{Y}^{quat}) \tag{4.5}$$

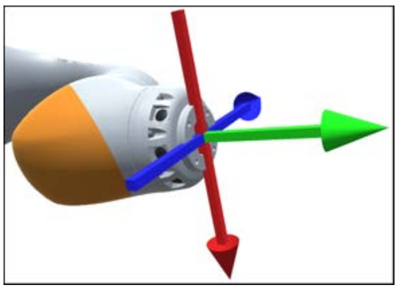


Figure 4.4: Visualisation of the orientation objective.

4.4.3 Direction

Sometimes the user wants to have the head or eyes of a character looking at a particular point while moving the rest of the body. In robotics, it can be required to have the camera which is located at a robot arm to track an object while manipulating it. Existing implementations for inverse kinematics do usually not consider this ability, and it can be rather difficult to be integrated for some methods. However, such tasks can be easily solved simultaneously by the Bio IK algorithm through adding a further objective to minimise the angular error between the direction from the segment to the target as well as an arbitrary viewing direction V, as visualised in Fig. 4.5.

$$\mathcal{L}_{Direction} = \cos^{-1}(V \cdot (\mathcal{Y}^{pos} - \mathcal{X}^{pos})) \tag{4.6}$$

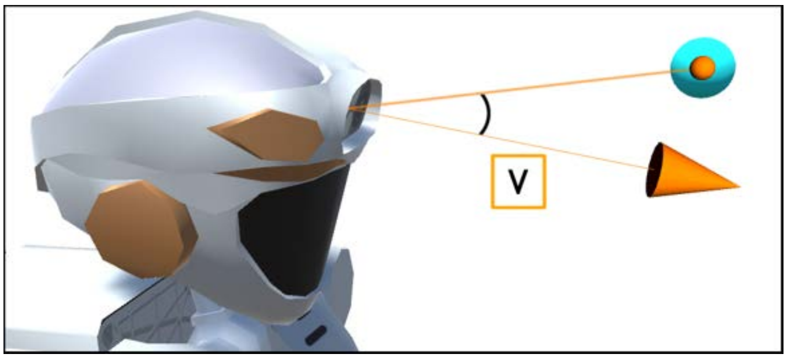


Figure 4.5: Visualisation of the direction objective.

4.4.4 Distance

Collision avoidance is a challenging issue which is often neglected by generic inverse kinematics solvers. Mostly, computationally expensive collision checks are required after a solution was found, and often that solution has to be discarded again. Therefore, efficiently integrating collision avoidance for generic inverse kinematics is not yet considered to be solved. However, using the proposed method, self-collision as well as collisions with objects in the environment can be avoided through defining an objective which forces the optimisation to punish small distances between points. More specifically, the distance d between two particular points can be forced not to fall under a specific radial distance threshold r using a shifted hyperbolic function, as denoted by (4.7). As visualised in Fig. 4.6, this technique prevents spherical collisions as a hard-constraint by returning an infinite error if d becomes lower than r, but acts as a soft-constraint otherwise. Adjusting the objective weight lastly controls the responsiveness in maintaining lower or larger distances between points.

$$\mathcal{L}_{Distance} = \begin{cases} \infty & if\ (d-r) \leqslant 0 \\ \frac{1}{d-r} & else \end{cases} \tag{4.7}$$

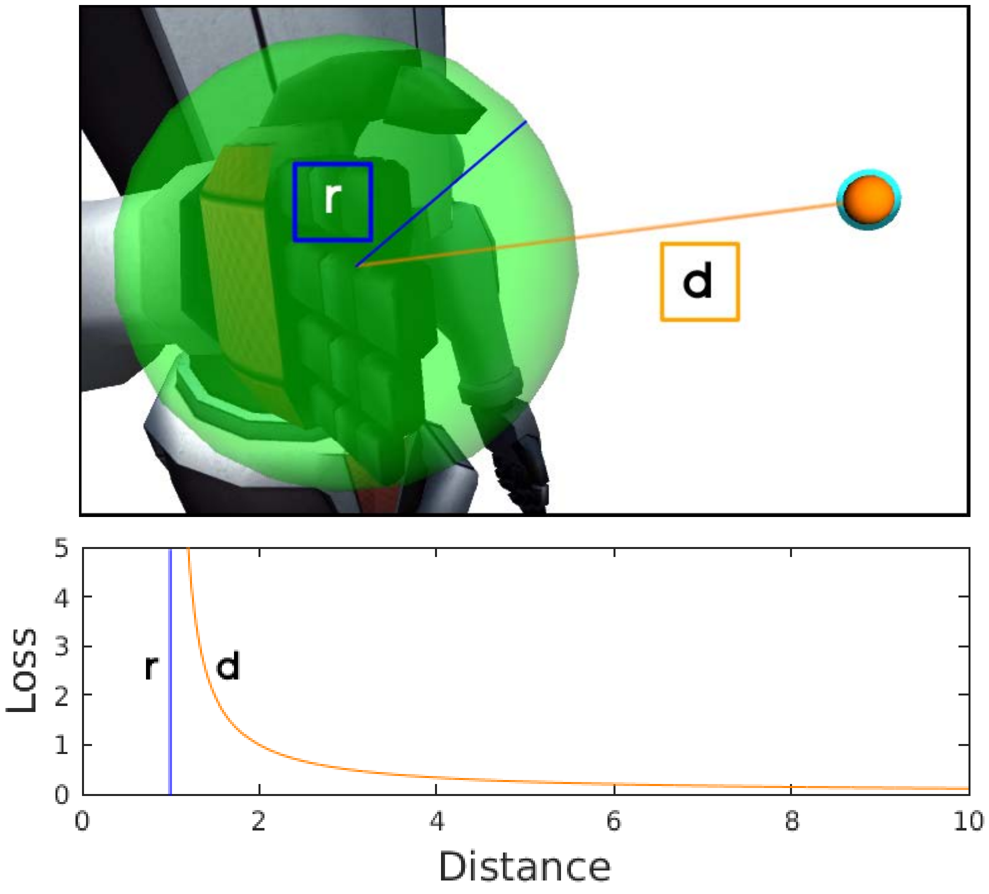


Figure 4.6: Visualisation of the distance objective.

4.4.5 Displacement

In some cases, it might be desired to perform larger updates only for particular joints, and to move those preferably more than others. The displacement objective is specifically designed for such goals, and punishes configurations with larger distances in joint space. As formulated by (4.8), each joint is assigned an individual weight ϵ for which the objective seeks to keep the average variation between x and the current solution x^* to be minimal. This can be helpful if some joints have slow acceleration or maximum velocity, or to create more realistic motion for virtual characters. An example on the slowly lifting torso of the PR2 robot is visualised in Fig. 4.7. Also, pointing and looking at a target should preferably cause more motion for the neck, arm and fingers instead for the pelvis and upper body. In addition, this objective passively causes smoother motion between solutions, but is also likely to cause a stronger exploitation than exploration for the optimisation.

$$\mathcal{L}_{Displacement} = \frac{1}{n}\sum_{i=1}^{n}\epsilon_i|x_i^* - x_i| \tag{4.8}$$

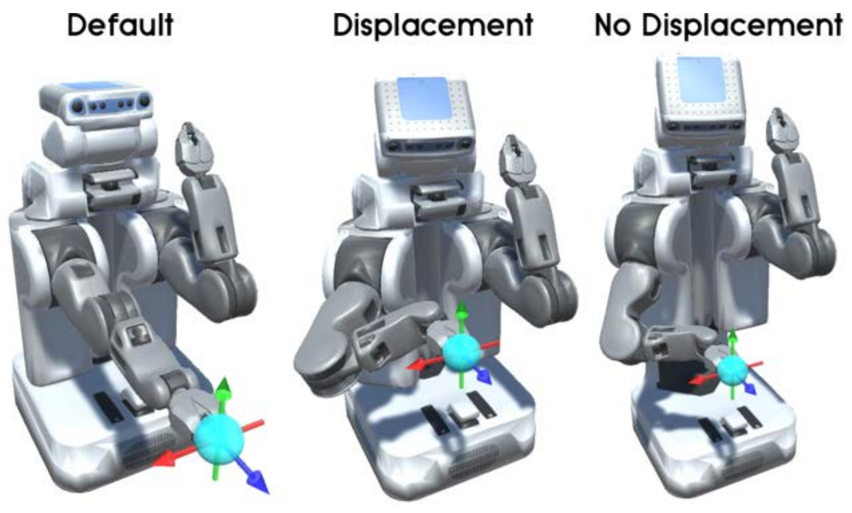


Figure 4.7: Visualisation of the displacement objective.

4.4.6 Joint Value

A similar technique as for the displacement objective in previous section 4.4.5 can also be applied to single genes. For those, it is possible to try keeping a prioritised value Θ for the gene x_i of a particular joint, as denoted by (4.9). This forces the optimisation to transfer the required motion to the remaining joints in a way that the error for maintaining Θ is compensated. The introduced amount of stiffness for the joint is then controlled by the objective weight, for which a lower weight allows larger motion and a higher weight strongly forces the evolution to keep that particular value. This can be helpful for manipulation tasks, in which an elbow or wrist shall approximately stick to a desired angle while moving an object.

$$\mathcal{L}_{JointValue} = |\Theta - x_i| \tag{4.9}$$

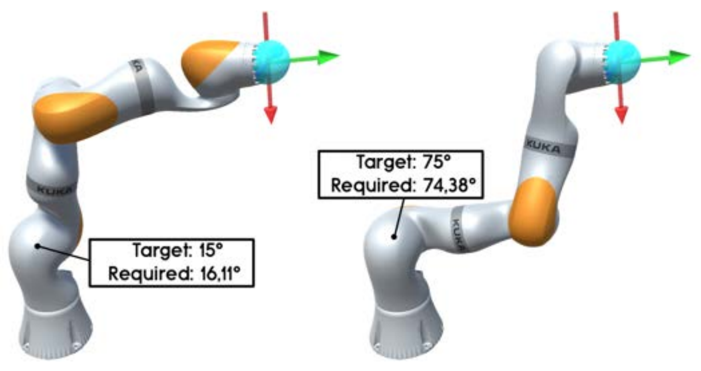


Figure 4.8: Visualisation of the joint value objective.

4.4.7 Functional Relation

Using the proposed algorithm, it is also possible to specify functional relations between joints. Intuitively, this can be imagined as coupling joint values to be depending to each other, such as the fingers of a real-human hand for which the little finger also causes joint variations for the ring, middle and index fingers. Integrating this functionality is relevant for creating more realistic movement for virtual characters, dexterous manipulation with anthropomorphic hands, as well as for more plausible motion of robots when interacting with humans. Let $h_i(x)$ denote a function over the encoded joint values x in relation to a particular gene x_i, then formulating this relation as $x_i = h_i(x)$ and converting it into an optimisation problem $0 = x_i - h(x)$ gives rise to the resulting loss function to be minimsed as (4.10). Note that this method is not restricted to linear functions, but can also serve non-linear relations between joints that usually exist for living organisms.

$$\mathcal{L}_{FunctionalRelation} = |x_i - h(x)| \tag{4.10}$$

4.4.8 Projection

It is also possible to construct more complex objectives by formulating them as a combination of others. An example is given by an objective which aims to project a point of a segment on top of the surface of an object such that the normals of both become aligned. This can be particularly helpful for animation post-processing tasks like interaction with objects or automatic foot placement as visualised in Fig 4.9. The corresponding loss term (4.11) can be formulated as a combination of the position and orientation objectives. In case a projection ray in direction V from the segment point results in a hit H, the inputs for the single objectives are given by the hit position H^{pos} as well as the rotation from V to the negative hit normal $-H^{norm}$ multiplied with the segment rotation.

$$\mathcal{L}_{Projection} = \frac{\mathcal{L}_{Position}(H^{pos}) + \mathcal{L}_{Orientation}(\Delta(V, -H^{norm}) \cdot \mathcal{X}^{rot})}{2} \tag{4.11}$$

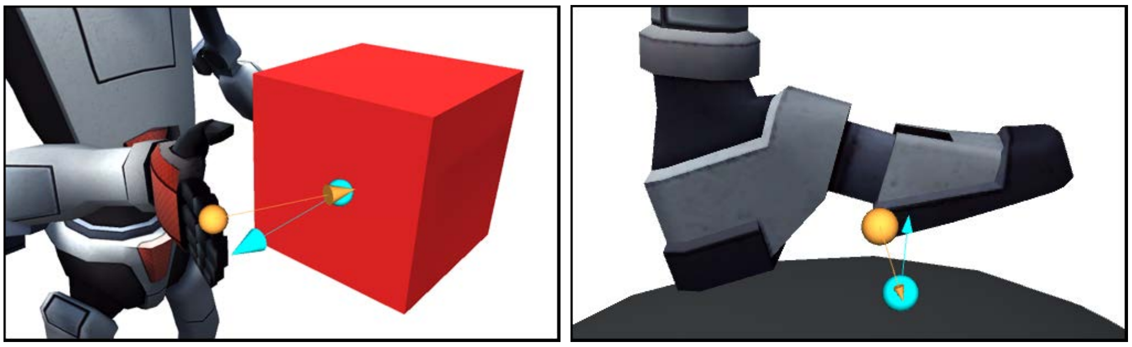


Figure 4.9: Visualisation of the projection objective.

4.5 Evolutionary Phases

This section discusses the design of the evolutionary phases for the Bio IK algorithm. After describing the initialisation in section 4.5.1, the operator for the parent and survivor selection is formulised in section 4.5.2. Followingly, the designed recombination, mutation and adoption operators are presented in sections 4.5.3, 4.5.4 and 4.5.5, which jointly produce an offspring from selected individuals of the previous generation. Section 4.5.6 then describes the niching scheme, which interferes the reproduction in a way that multiple potential paths are explored. The elitism exploitation is discussed in section 4.5.7, and which represents a very relevant part of the algorithm. Finally, the wipe out criterion and termination conditions are given in sections 4.5.8 and 4.5.9.

4.5.1 Initialisation

The initialisation of the algorithm is important to directly support a fast convergence, and can be done straightforwardly through randomly sampling in the search space. For each inverse kinematics query, the algorithm is initialised using $\varphi - 1$ randomly generated individuals where φ denotes the population size, and $U_{[a,b]}$ describes a uniformly-distributed random value between a and b. This can be denoted as 4.12). Thus, all genes are randomly sampled between the particular lower and upper joint limits, and one further individual is created from a seed state, which is given by the currently assigned joint variable configuration θ.

$$x^1 = \theta \qquad x_i^{2,...,\varphi} = U_{[\theta_{i_{min}},\theta_{i_{max}}]} \;\; \forall i = 1, ..., n \tag{4.12}$$

This technique aims to support finding a solution that is near to the current solution of the kinematic body, but also to quickly find solution if the new target has a considerably larger distance to the current configuration.

In addition, for each individual a momentum variable g is encoded, which is required for the integration of swarm dynamics into the evolution. This value is gradually updated with the modifications of the gene vector, and is initialised as zero for all individuals according to (4.13).

$$g_i^{1,...,\psi} = 0 \;\; \forall i = 1, ..., n \tag{4.13}$$

4.5.2 Selection

This section describes both the parent and survivor selection strategy. Considering the former, the parents for creating new offspring are chosen using a rank-based selection scheme. This has particular benefits over raw fitness-based selections for which the fittest individual might be selected too frequently. This woud typically lead to a decreased diversity and premature convergence, usually getting stuck in suboptimal extrema. For the rank-based selection, the intrinsic selection probability tables for selecting a parent $\mathcal{P}$ can be efficiently precalculated depending on the size γ of the mating pool Γ. Accordingly, the parent selection operater $\mathcal{S}_{\mathcal{P}}$ can be formulated as (4.14), for which the probabilities are normalised within the range $[0,1]$ and sum up to a total probability of 1. Note that this selection operator is also used to select the prototype individual for the adoption phase in section 4.5.5, as well as for selecting a dynamic subset of individuals for the elitism exploitation in section 4.5.7. As soon as all individuals have evolved, the survivor selection phase straightforwardly combines all evolved offspring Λ of size λ with the exploited elite individuals $\mathcal{E}$ of size κ into the next generation Ψ' of size ψ. Hence, the survivor selection operator $\mathcal{S}_{\mathcal{S}}$ (4.15) always maintains the same size of individuals within the population where $\psi = \lambda + \kappa$ holds.

$$\mathcal{S}_{\mathcal{P}} : \mathcal{P} \leftarrow p(\Gamma_i) = \frac{\gamma - i + 1}{\sum_{i=1}^{\gamma} i} \tag{4.14}$$

$$\mathcal{S}_{\mathcal{S}} : \Psi' \leftarrow \mathcal{E} \cup \Lambda \tag{4.15}$$

4.5.3 Recombination

The recombination phase creates $\lambda = \psi - \kappa$ offspring where each of them is generated from two parent individuals $\mathcal{P}_1$ and $\mathcal{P}_2$, as visualised in Fig. 4.10. Each parents are selected based on their rank as described in previous section 4.5.2. Every gene is recombined through a randomly-weighted average of both parent genes, where $I_c(a,b)$ is a linear interpolation between a and b weighted by c and $U_{[0,1]}$ is a uniformly-distributed random value within $[0,1]$. In addition, a small amount of the momentum g of both parents is added, which aims to let offspring immediately dive a little deeper into the direction which likely caused improvement for their parents. This aims to integrate the particle velocity procreation of PSO into the evolutionary optimisation. Similarly, the initial momentum for the new offspring is created by a decayed random combination of both parent momentums.

$$R : \begin{cases} x_i = I_{U_{[0,1]}}(x_i^{\mathcal{P}_1}, x_i^{\mathcal{P}_2}) + g_i \\ g_i = U_{[0,1]}\ g_i^{\mathcal{P}_1} + U_{[0,1]}\ g_i^{\mathcal{P}_2} \end{cases} \tag{4.16}$$

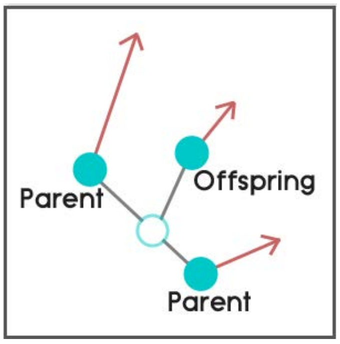


Figure 4.10: Particle visualisation for the recombination phase.

4.5.4 Mutation

Mutation aims to maintain a genetic diversity among individuals, and to discover potential solutions in unexplored regions. This is particularly relevant in order to escape from suboptimal extrema, and to increase the robustness for non-smooth fitness landscapes. In terms of generic inverse kinematics, an operator is required which remains adaptive to varying kinematic geometries whose DoF define the search space dimensionalities. Thus, higher DoF requires smaller while a lower DoF needs higher probabilities for the mutation rate in order to achieve effective changes for the genes. In addition, the mutation strength should be designed adaptively such that the evolution remains sensitive to local extrema but also explorative at the same time. A simple method would be using an inverse-proportional probability for the number of genes, but which is not adaptive to the current optimisation progress. Instead, an extinction operator was designed which uses a factor ξ to adaptively control mutation rate and strength for offspring depending on the fitness ϕ of their parents, and can be calculated as (4.17). This factor measures the relative success of an individual within the whole population regarding its rank and fitness, and finally yields a normalised value between $[0, 1]$. Note that ϕ_{min} and ϕ_{max} define the minimum and maximum fitness values within the population. It is then possible to define the mutation operator as (4.18), where p defines the mutation rate between $[\frac{1}{n}, 1]$ and the mutation strength is calculated by a random offset using the average extinction of both parents and the domain size given by the joint limits. Finally, the performed mutation is also added to the momentum of the individual. This operator is visualied in Fig. 4.11 and achieves small variations for offspring with good parents, but can still allow any configuration in the search space to be sampled.

$$\xi = \frac{\phi + \phi_{min}(\frac{i-1}{\varphi-1} - 1)}{\phi_{max}} \tag{4.17}$$

$$M : \begin{cases} p = \frac{\xi^{\mathcal{P}_\varnothing}(n-1)+1}{n} \\ x_i' = x_i + U_{[-1,1]}\frac{\xi^{\mathcal{P}_1}+\xi^{\mathcal{P}_2}}{2}(\theta_{i_{max}} - \theta_{i_{min}}) \\ g_i' = g_i + (x_i' - x_i) \end{cases} \tag{4.18}$$

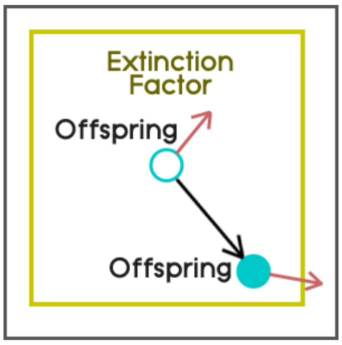


Figure 4.11: Particle visualisation for the mutation phase.

4.5.5 Adoption

Adoption is an additionally designed phase for the Bio IK algorithm which further integrates the swarm dynamics of PSO in combination with the recombination phase in section 4.5.3. This phase aims simulate the velocity update step of PSO where each particle adjusts its motion based on its neighbours and the globally best performing particle. In an evolutionary algorithm, this can be achieved through considering the neighbours as the parents and the globally best particle as the fittest individual in the population. Therefore, the adoption operator updates the genes of an individual by adding an interpolated direction to the randomly-weighted average genotype of its parents $P_{\varnothing}$ as well as to a prototype individual P_{*} which is similarly selected by its rank. This operator can be formulated as 4.19), and is visualised in Fig. 4.1. Intuitively, this operator aims to adopt promising characteristics of successful individuals within the population.

$$A : \begin{cases} x_i' = x_i + I_{U_{[0,1]}}(U_{[0,1]}(x_i^{\mathcal{P}_{\varnothing}} - x_i), U_{[0,1]}(x_i^{\mathcal{P}_*} - x_i)) \\ g_i' = g_i + (x_i' - x_i) \end{cases} \tag{4.19}$$

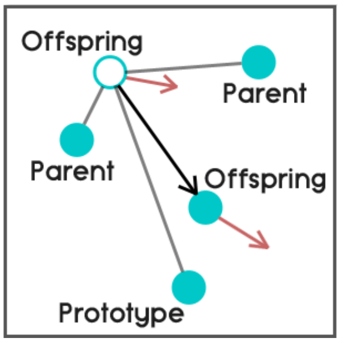


Figure 4.12: Particle visualisation for the adoption phase.

4.5.6 Niching

In evolutionary computation, niching is a common technique which aims to avoid premature convergence which typically lead to suboptimal solutions. The goal is to let the population discover multiple solutions simultaneously by forming smaller subgroups of individuals. The intention for inverse kinematics is to concurrently track multiple suitable postures and thus to increase the robustness of the algorithm. In more detail, a pre-selection scheme is used which immediately removes a parent from the mating pool Γ whose offspring scored a better fitness value. This can be formulated as (4.20), and encourages the population to explore different paths at the same time. Note that in case the mating pool becomes empty, a random offspring is created.

$$N : \begin{cases} \Gamma \backslash \mathcal{P}_1 & if\ \Omega(x) < \Omega(x^{\mathcal{P}_1}) \\ \Gamma \backslash \mathcal{P}_2 & if\ \Omega(x) < \Omega(x^{\mathcal{P}_2}) \end{cases} \tag{4.20}$$

4.5.7 Elitism Exploitation

The exploitation phase specifically seeks to enhance potentially good solutions among the population. This formulates the concept of memetic evolution, which is assumed to improve the speed and continuity of convergence. Intuitively, the algorithm continually samples configurations from the search space using a global search which is based on recombination, mutation and adoption, and then performs a more expensive local search only on a small subset of individuals. The phase is visualised in Fig. 4.13, and performs a gradient-based search on a set of selected elite individuals. In total, there are κ elites which are given by the fittest individual Ψ_1 as well as $\kappa - 1$ other randomly selected elite individuals. For those, a rank-based selection strategy without duplicates is used, which means that no individual is chosen as elite twice and $\kappa \leqslant \psi$ must hold. The gradient-based search utilises the L-BFGS-B algorithm, which was described previously in section 2.3.2. While there is a larger number of possible methods for the exploitation, not all of them satisfy the algorithmic requirements. The L-BFGS-B is particularly suited for bounded, high-dimensional and non-smooth optimisation problems, which can be the case for inverse kinematics on highly articulated geometries. However, most important is its ability that it can optimise arbitrary twice-differentiable continuous functions, and thus allows the same objective function Ω to be used by the global evolutionary and local gradient-based search. It only requires to additionally approximate the gradient $\nabla\Omega$ on the fitness landscape at point x. This can be done by calculating the partial derivatives on the genotype of an individual, as denoted by (4.21). For this, all genes are iteratively modified by a very small value to approximate the gradient on the objective function. The operator can then be formulated as (4.22), which causes elite individuals to not only survive but also to be improved, to guide the momentum, as well as to search for multiple solutions simultaneously.

$$\nabla\Omega_i = \left(\frac{\delta\Omega}{\delta x_i}\right) \tag{4.21}$$

$$E : \begin{cases} x' = L\text{-}BFGS\text{-}B(x, \Omega, \nabla\Omega) \\ g' = U_{[0,1]} g + (x' - x) \end{cases} \tag{4.22}$$

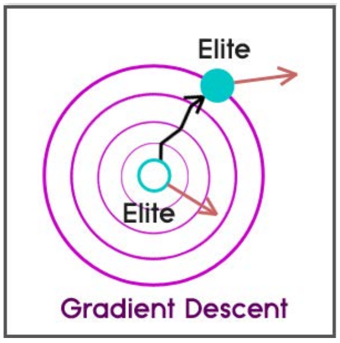


Figure 4.13: Particle visualisation for the exploitation phase.

4.5.8 Wipe Out

Although the evolutionary phases are designed to robustly avoid suboptimal solutions, it can still be possible that all niches get stuck and achieve no further improvements. In such cases, performing a reinitialisation can often be assumed to achieve an overall faster convergence instead propagating new individuals until a successful exploration occurs. The condition to heuristically initiate a resampling can be denoted as (4.23), and is met if the current solution x^* could not be replaced by the fittest genotype x^1, and if no further fitness improvement could be achieved for any elite individual x^ε within the set of elites $\mathcal{E}$.

$$W : \phi^1 \geqslant \phi^* \;\; \wedge \;\; \phi^{\varepsilon\prime} \geqslant \phi^\varepsilon \;\; \forall\, \varepsilon \in \mathcal{E} \tag{4.23}$$

The reinitialisation is then performed as described in (4.12). However, the current solution x^* is used as the new seed and is reintegrated into the new population with $x^1 = x^*$. This achieves a partial reinitialisation which does not lose information about the best solution that could be found so far. More specifically, this can be considered as an advantage of multi-modal sampling-based methods over unimodal gradient-based techniques for optimisation, where the latter would basically be started from scratch after being reinitalised.

4.5.9 Termination

The algorithm finally terminates if either all objectives are satisfied or a specified time limit was exceeded. Convergence is measured by a condition for each objective. In particular, using a fitness threshold to decide for convergence is not appropriate as it does not give rise to the individual errors, and can not guarantee that all of them are satisfied. Therefore, individual termination conditions for each objective are defined. For the position and orientation objective in sections 4.4.1 and 4.4.2, an error threshold is used to specify the desired accuracy for each segment pose. This can be calculated similarly to their loss functions by computing the Euclidean and angular distance between the segment and the target. Same holds for the direction objective in section 4.4.3 which is also given an error threshold for the angular error between two vectors. For the distance objective in section 4.4.4, it is required that the Euclidean distance between two points remains larger than the specified minimum distance. Nevertheless, this should always be the case anyways as approaching this threshold causes significantly increasing loss values going to infinity. The projection objective in section 4.4.8 is given both an error threshold for the position and orientation error, but is always considered as converged if no hit occured. For the remaining joint value, displacement and functional relation objectives in sections 4.4.6, 4.4.5 and 4.4.7, the convergence conditions are optional. Those are not specficially required since they are primarily designed for guiding the evolution. If all defined termination conditions over the objectives are met, the algorithm stops optimisation and returns the evolved solution. However, note that even if the algorithm did not converge, the best approximate solution that could be found so far can be returned.

4.6 Computational Improvements

This section describes two computational improvements that were developed for the evolutionary algorithm to achieve a faster optimisation. The first is given by a data structure which can be used to efficiently compute the forward kinematics equations for fitness evaluations, and will be discussed in section 4.6.1. The second is given section 4.6.2, and describes how multi-core threading can be efficiently used to evolve generations. Both extensions are optional, and do not influence any performance of the algorithm apart from its computational speed in which solutions can be obtained.

4.6.1 Optimised Forward Kinematics Tree

While evolutionary computation outlines a very general method for arbitrary optimisation problems as it requires no particular knowledge or assumptions of the fitness landscape, there is usual no direct link between algorithmic and problem complexity. Instead, the typical issue for evolutionary algorithms is that the complexity is transferred to the amount of fitness evaluations. This can quickly accumulate to a larger amount of computation time, and for which improvements in approximating or improving evaluation can achieve considerable speedups.

When solving inverse kinematics, multiple end effector systems such as the finger tips of an anthropomorphic arm, humanoid robots or highly articulated game characters contain many shared joints along their kinematic chains. For those, the forward kinematics equations then become partially equivalent, and many of the transformations can become redundant. This is especially the case when only small joint variable changes are applied by the evolutionary operators for the encoded genotypes of the individuals. Same holds for the gradient approximation, where only one gene is changed at a time. In this context, this section presents the OFKT (Optimised Forward Kinematics Tree) which is designed to efficiently process multiple forward kinematics queries by caching transformations and settings from preceding calculations, and assuming that only a few joint values are changed between successive queries. Repeated evaluations can then be processed more quickly rather than naively calculating the full recursive set of forward kinematics equations along the kinematic tree. Although improvements could be implemented manually, the data structure can efficiently handles different types of coordinate transformations among the segments.

As described previously in section 2.1.5, the transformations for the forward kinematics function (2.10) from the root to each single end effector segment can be computed via (2.11). As denoted by (4.24), the transformations between the single relevant segments can be grouped into *reference* and *local* transformations, given as R_i and L_i respectively. While a local transformation is understood as the segment frame itself, a reference transformation represents the transformation from the root to the segment. However, not every segment must have a joint attached, and therefore S_i denotes the *static* transformation transformation between a segment's preceding non-static segment to a segment's local transformation with $\theta_j = 0$ as the default configuration. In particular, also note that S_i only needs to be computed once, and is then stored to avoid recalculating non-changing transformations.

$$R_i = R_{i-1}\, L_i \qquad L_i = S_i\, T(\theta_j) \tag{4.24}$$

Construction

The OFKT itself basically represents a kinematic tree given by a linked list of segments, one for each moveable part of the kinematic structure. Within each node, R_i and L_i are individually computed and stored, together with the currently assigned joint variable θ_j. While the former depends on the preceding reference and the current local transformation, the latter is calculated using the segment's static transformation S_i modified by θ_j. Alg. 1 summarises building the OFKT data structure which can then be used for efficiently processing multiple successive forward kinematics queries.

```
  Input  : Geometry, Root, End Effectors
1 OFKT = CreateLinkedList(Root, End Effectors);
2 Chains = GetChains(Root, End Effectors);
3 foreach Segment in Chains do
4     if Segment.HasJoint() then
5         Node = OFKT.Insert(Segment);
6         Node.ComputeAndStoreStaticTransformation();
7         Node.StoreJointVariable();
8         Node.ComputeAndStoreLocalTransformation();
9         Node.ComputeAndStoreReferenceTransformation();
```

Algorithm 1: Building the OFKT

Querying

The input for an forward kinematics query is given by a joint variable configuration which needs to be processed by the OFKT. It then uses the stored variables within each node for the current transformation and the joint value under which the local transformation was evaluated. As described in Alg. (2), the procedure for updating forward kinematics transformations is started at the root node of the linked list, and is recursively called for all childs. In addition, a boolean parameter is recursively passed which initially assumes that no update of reference transformations would be required. The flag is set in case of joint variable changes, which requires recalculating the local transformation and thus also the reference transformation. As soon as one local update was performed, a relative update is also neccessary for all subsequent nodes. After the tree traversal, the resulting segment transformations W_i can be returned in world space using (4.25), where the additional ${}^{world}T_{root}$ transformation is prepended. Intuitively, the OFKT computes all transformations in reference to the root of the kinematic model and only requires updating a smaller set of non-static transformations.

$$W_i =^{world} T_{root} \; R_i \tag{4.25}$$

```
Input  : Joint Variable Configuration
Output: End Effector Transformations
Function UpdateFK(Node, RequireUpdate):
    if HasJointVariableChanged() then
        Node.StoreJointVariable();
        Node.ComputeAndStoreLocalTransformation();
        RequireUpdate = true
    if RequireUpdate then
        Node.ComputeAndStoreReferenceTransformation();
    foreach Child of Node do
        UpdateFK(Child, RequireUpdate);
    return;
UpdateFK(OFKT.Root, false);
foreach End Effector Node do
    return Node.ComputeWorldTransformation();
```

Algorithm 2: Querying the OFKT

Complexity

This section conducts a theoretical evaluation for the OFKT data structure regarding the total required transformations in four application scenarios, which are listed in Tbl. 22. Given a n-dimensional serial joint variable configuration, one forward kinematics pass requires calculating n local transformations which are then concatenated by n transformations, and one further from the world to the root transformation. This calculation will be used as baseline for performance comparison with the OFKT data structure.

1. *Forward kinematics computation by updating a (random) number of values along a serial kinematic chain:* This is the typical query after evolving the genes of an individual. In general, $2n + 1$ calculations would be needed for independent forward kinematics queries. Using the OFKT, previous results can be reused and the required amount of local transformation updates becomes equivalent to the number of changed joint values $j < n$. Traversing the single segments then results in $n - i$ instead of n reference transformations, where i is the first modified index along the serial kinematic chain.

Scenario	Standard	OFKT
Random Modifications	$2n + 1$	$n - i + j + 1$
End Effector Computation	$2n + 1$	5
Single Iterative Modifications	$2n^2 + n$	$\frac{n^2}{2} + \frac{5n}{2}$
Multiple End Effectors	$k(2n + 1)$	$k(2n + 1) - 2(k - 1)s$

Table 4.2: Overview of the computational complexity for the OFKT.

2. *Predicting the end effector world transformation by modifying exactly one joint value:* This is helpful for determining or estimating the error gradient. Only one local $L_i^{'}$ as well as three further transformations $R_{i-1}\ L_i^{'}\ R_i^{-1}\ R_{ee}$ are neccessary for calculating the end effector transformation, followed by one additional world transformation. In particular, the required transformations of the single segments are already available, and enable to directly obtain the relative end effector change.

3. *Iteratively updating exactly one joint value while maintaining information about all segment transformations:* This is particularly important for enabling efficient further computation of relative transformations within the kinematic tree. n queries are performed iteratively, requiring $n(2n + 1)$ calculations. Using the OFKT, each of the n queries automatically avoids recalculating unchanged transformations, resulting in n local updated segments and a total of $\frac{n(n+1)}{2}$ calculations for the affected reference transformations.

4. *Forward kinematics calculations on different chains with multiple end effectors of an anthropomorphic arm:* This is relevant in terms of scalability for complex geometries. A 27 DoF anthropomorphic geometry is considered, starting with a 7 DoF arm and splitting up into a hand with five 4 DoF fingers — giving rise to $k = 5$ chains with 11 DoF each. Hence, calculating all end effectors individually would require $k(2n + 1)$ transformations, while the OFKT automatically avoids recalculating the shared $s = 7$ arm joints.

4.6.2 Multi-Threading

The algorithm allows to be parallelised on the CPU to speedup the evolution. For each elite individual, an additional thread is created to perform the L-BFGS-B exploitation. This is because the exploitation is rather costly compared to the other tasks, but can be run independently from the offspring creation. In particular, the remaining individuals are evolved on the main thread, which must be done sequentially because of the niching phase that was introduced in section 4.5.6, and which introduces mutual dependency between parents and offspring. Concurrently, the elitism exploitation on the allocated threads continues iterating until the main thread has finished evolving offspring so that no computation time remains unused. Thus, the population size implicitly controls the number of exploitation steps that can be performed by the L-BFGS-B modules within each generation.

4.7 Implementation

The algorithm is availably implemented for research and development in robotics and animation. The implementations are given as an asset for Unity3D (C#) and a plugin for ROS (C++), which both are popular tools in their particular fields.

4.7.1 Pseudocode

The pseudocode of the complete Bio IK algorithm is shown in Alg. 3. It covers the algorithmic details for initialisation and the phases for one evolutionary cycle.

```
Input   : Population Size, Number of Elites, Seed
Assign Seed as Solution;
Initialize Population;
⟫ Incorporate Solution;
⟫ Create $PopulationSize - 1$ Random Individuals;
⟫ Evaluate and Sort Individuals by Fitness;
⟫ Calculate Extinction Factors;
⟫ Try Update Solution;
while Not Terminated do
    Assign whole Population to the Mating Pool;
    for All Selected Elite Individuals do
        Perform Exploitation using L-BFGS-B;
    for All Non-Elite Individuals do
        if Mating Pool is not empty then
            Select Parents and Prototype from Mating Pool and Create
              new Individual by Recombination, Mutation and
              Adoption;
            Constrain Genes to Dimension Bounds;
            Evaluate Fitness;
            Remove worse Parents from Mating Pool;
        else
            Create Random Individual;
            Evaluate Fitness;
        Add Individual to Offspring;
    Select Elites and Offspring as the new Population;
    Sort Individuals by Fitness;
    Calculate Extinction Factors;
    Try Update Solution;
    if Wipe Out Criterion Fulfilled then
        Reinitialise Population;
Return Solution;
```

Algorithm 3: Bio IK Algorithm

4.7.2 Unity3D Asset (C#)

The Unity3D implementation [10] consists of a single component which can be added to the start of a transform hierarchy from where inverse kinematics shall be solved. The asset automatically detects the existing kinematic geometry of segments, and provides user-friendly custom inspectors as well as parameter visualisations for the solver, joints and objectives. It further includes an API to control the algorithm, although it can also be entirely used without any programming. The visualisation of the models is shown in Fig. 4.14. The segments are connected by teal lines, where each is marked by a small sphere. Each joint is represented by a magenta cube which rotation is given by the orientation of joint axes. When a segment is selected, it becomes highlighted by a circle in which the particular motion limits are drawn in case a joint is attached to it. The custom inspector for the script component, as well as the segments and joints is shown in Fig. 4.15. All variables are serialised in the editor, and implement an *Undo* and *Redo* functionality. The first variable defines whether threading shall be used or not. This is because is it not always supported for particular target platforms such as WebGL when compiled, in which case it needs to be deactivated. The next parameters are used to let the user specify the desired number of generations per frame, as well as the population size and number of elites. The smoothing factor

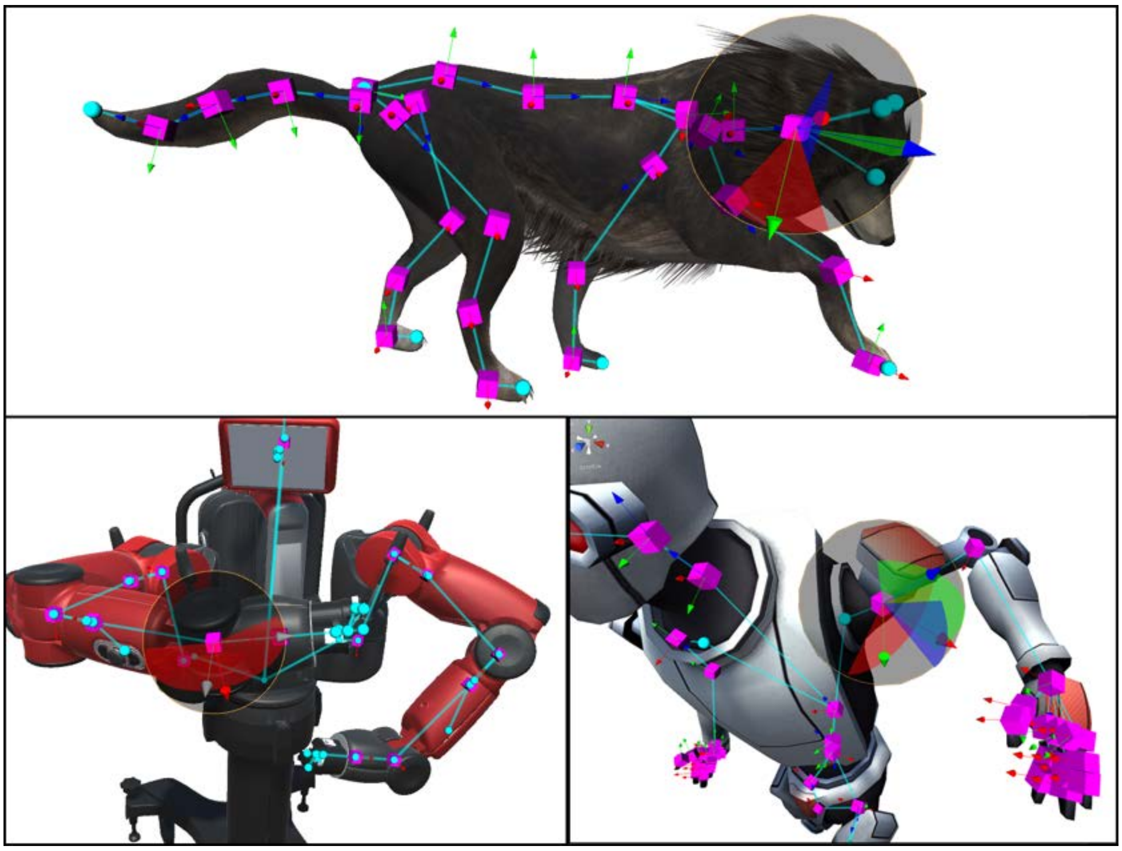
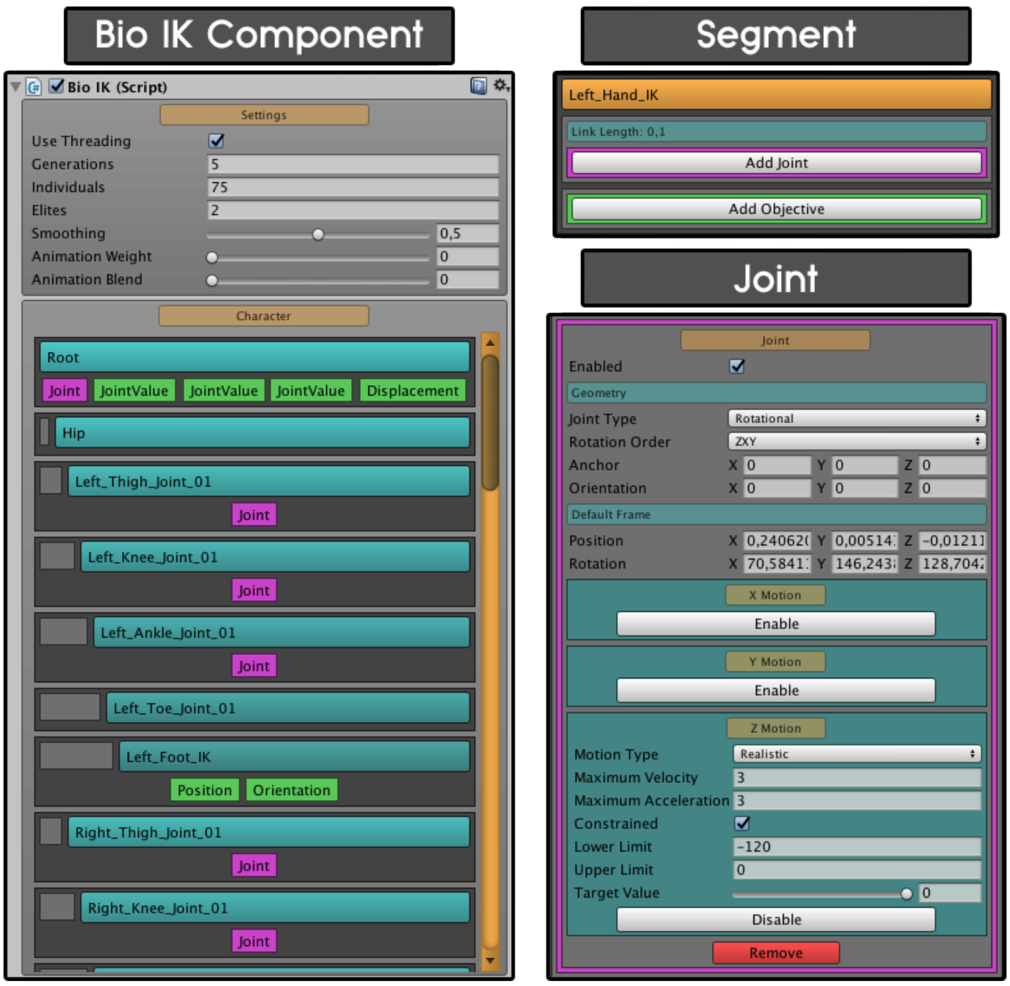


Figure 4.14: Geometry visualisation for the Bio IK asset in Unity3D.

is a value between [0, 1] which interpolates the Cartesian transformations of segments given the last and current frame, and which smooths out the motion a bit in case a larger configuration change occured. If the character is animated, it can be weighted into the optimisation as a seed to force the evolution to stick as close as possible to the original animation. Similarly, it is possible to blend between the inverse kinematics solution and the animation. Furthermore, while each segment can have at most one joint, it is possible to add multiple objectives to it. The joint motion can then either be rotatioal or translational, and allows setting a rotation order for the Euler angles to construct the quaternion rotations. All motion is calculated relative to the default frame of the segment for which an additional anchor or orientation offset can also be defined if needed. The motion axes for each joint can then be controlled independently and specify a 0 to 3 DoF joint. The joint motion can either be instantaneous or realistic, where the latter requires to define a maximum velocity and maximum acceleration. Both unconstrained as well as constrained motion is supported, which allows floating motion for the pelvis as well as setting angular joint limits for the bones to ensure realistic postures. Next,


Bio IK Component

Bio IK (Script)

Settings

Use Threading	✓
Generations	5
Individuals	75
Elites	2
Smoothing	0,5
Animation Weight	0
Animation Blend	0

Character

Root: Joint, JointValue, JointValue, JointValue, Displacement
Hip
Left_Thigh_Joint_01: Joint
Left_Knee_Joint_01: Joint
Left_Ankle_Joint_01: Joint
Left_Toe_Joint_01
Left_Foot_IK: Position, Orientation
Right_Thigh_Joint_01: Joint
Right_Knee_Joint_01: Joint

Segment

Left_Hand_IK
Link Length: 0,1
Add Joint
Add Objective

Joint

Joint

Enabled	✓					
Joint Type	Rotational					
Rotation Order	ZXY					
Anchor	X	0	Y	0	Z	0
Orientation	X	0	Y	0	Z	0
Position	X	0,24062(	Y	0,00514:	Z	-0,01211
Rotation	X	70,5841:	Y	146,243:	Z	128,704:

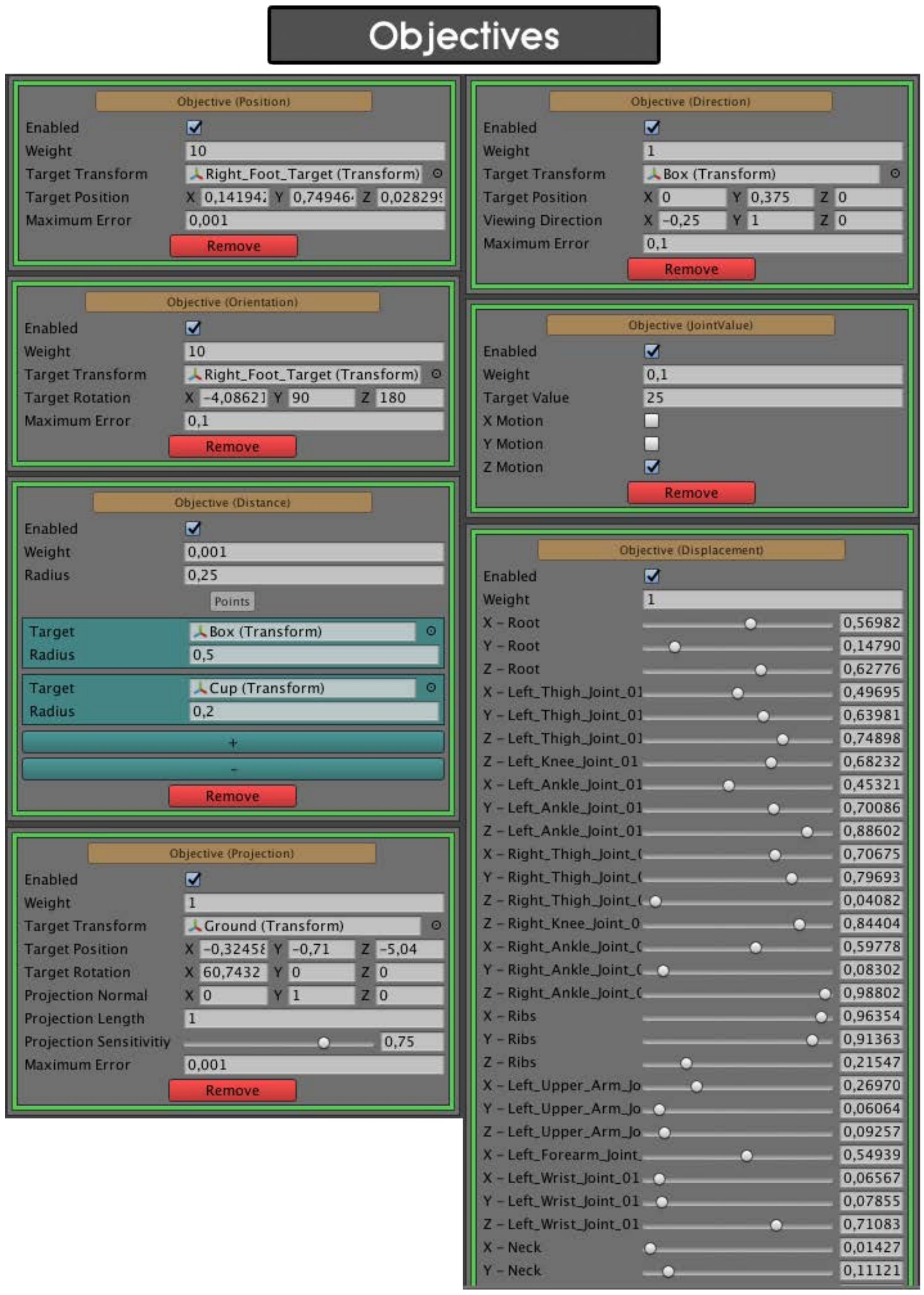
X Motion: Enable
Y Motion: Enable
Z Motion

Motion Type	Realistic
Maximum Velocity	3
Maximum Acceleration	3
Constrained	✓
Lower Limit	-120
Upper Limit	0
Target Value	0

Disable
Remove

Figure 4.15: Main component and inspectors for the Bio IK asset in Unity3D.

the custom inspectors for the single objectives are visualsed in Fig. 4.16. Cartesian targets can be set via drag-and-drop by object transforms, but also manually via position or angular rotation values. Note that it is also possible to apply changes to the parameter values of the solver, joints or objectives during runtime, as well as to enable, disable, add or remove them for dynamic interactive tasks.



Objective (Position)

Enabled	✓
Weight	10
Target Transform	Right_Foot_Target (Transform)
Target Position	X 0,14194 Y 0,74946 Z 0,02829
Maximum Error	0,001

Remove

Objective (Orientation)

Enabled	✓
Weight	10
Target Transform	Right_Foot_Target (Transform)
Target Rotation	X -4,08621 Y 90 Z 180
Maximum Error	0,1

Remove

Objective (Distance)

Enabled	✓
Weight	0,001
Radius	0,25

Points

Target	Box (Transform)
Radius	0,5
Target	Cup (Transform)
Radius	0,2

+

-

Remove

Objective (Projection)

Enabled	✓
Weight	1
Target Transform	Ground (Transform)
Target Position	X -0,32458 Y -0,71 Z -5,04
Target Rotation	X 60,7432 Y 0 Z 0
Projection Normal	X 0 Y 1 Z 0
Projection Length	1
Projection Sensitivitiy	0,75
Maximum Error	0,001

Remove

Objective (Direction)

Enabled	✓
Weight	1
Target Transform	Box (Transform)
Target Position	X 0 Y 0,375 Z 0
Viewing Direction	X -0,25 Y 1 Z 0
Maximum Error	0,1

Remove

Objective (JointValue)

Enabled	✓
Weight	0,1
Target Value	25
X Motion	☐
Y Motion	☐
Z Motion	✓

Remove

Objective (Displacement)

Enabled	✓
Weight	1
X - Root	0,56982
Y - Root	0,14790
Z - Root	0,62776
X - Left_Thigh_Joint_01	0,49695
Y - Left_Thigh_Joint_01	0,63981
Z - Left_Thigh_Joint_01	0,74898
Z - Left_Knee_Joint_01	0,68232
X - Left_Ankle_Joint_01	0,45321
Y - Left_Ankle_Joint_01	0,70086
Z - Left_Ankle_Joint_01	0,88602
X - Right_Thigh_Joint_(	0,70675
Y - Right_Thigh_Joint_(	0,79693
Z - Right_Thigh_Joint_(	0,04082
Z - Right_Knee_Joint_0	0,84404
X - Right_Ankle_Joint_(	0,59778
Y - Right_Ankle_Joint_(	0,08302
Z - Right_Ankle_Joint_(	0,98802
X - Ribs	0,96354
Y - Ribs	0,91363
Z - Ribs	0,21547
X - Left_Upper_Arm_Jo	0,26970
Y - Left_Upper_Arm_Jo	0,06064
Z - Left_Upper_Arm_Jo	0,09257
X - Left_Forearm_Joint_	0,54939
X - Left_Wrist_Joint_01	0,06567
Y - Left_Wrist_Joint_01	0,07855
Z - Left_Wrist_Joint_01	0,71083
X - Neck	0,01427
Y - Neck	0,11121

Figure 4.16: Objective inspectors for the Bio IK asset in Unity3D.

4.7.3 ROS Plugin (C++)

The ROS implementation [9] is written as an inverse kinematics plugin for MoveIt!. In contrast to *KDL* and *TRAC-IK*, the Bio IK plugin can solve multiple goals at once. All goal classes derive from a *Goal* base class, and implement an *evaluate* method as well as a *describe* method. Information is exchanged via a *GoalContext* object, allowing the interface to be extended at a later time without breaking the API. The *evaluate* method is called after each mutation and returns a fitness measure for the current joint values and segment poses. Which joints and segments a goal depends on is queried by the inverse kinematics solver during initialisation by calling the *describe* method. Bio IK can be extended by additional goal classes that do not have to be implemented within the package itself, but can be implemented within another package that calls the Bio IK solver. A detailed description of the implementation and the further available goal types can be found in [87].

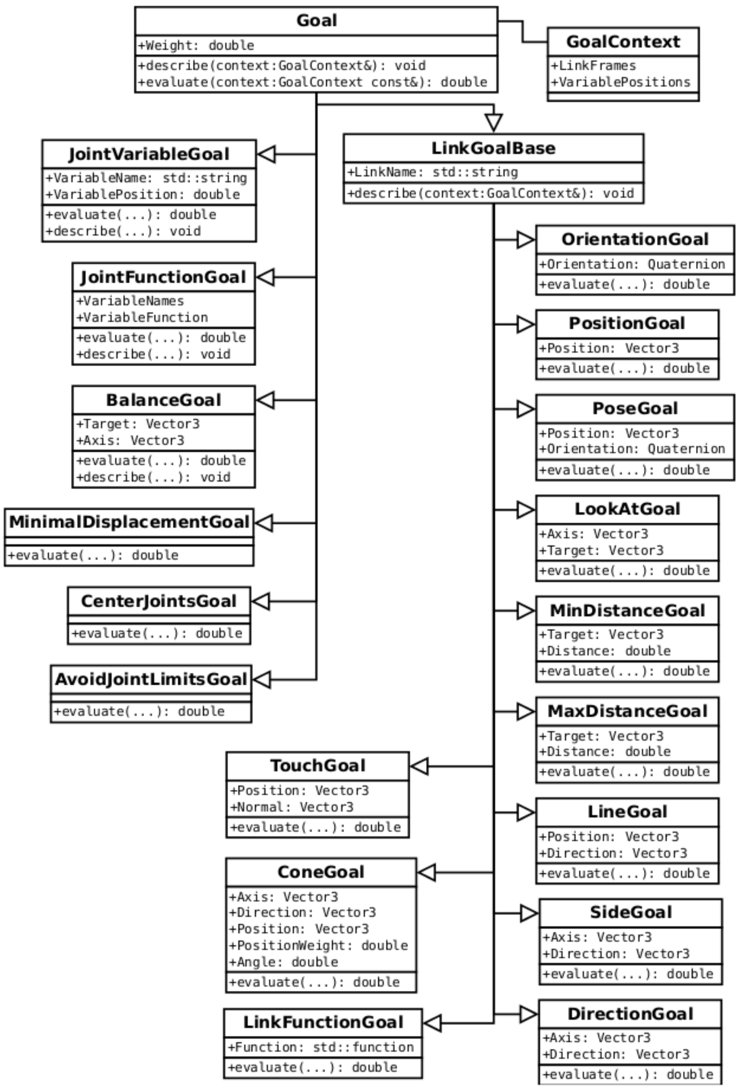


Figure 4.17: Software design for the Bio IK plugin in ROS. [87]

Chapter 5

Experiments and Results

This chapter presents the experimental results of the proposed Bio IK algorithm, which aim to demonstrate its performance and flexibility for different applications in robotics and animation. All experiments were conducted based on the experimental setup which is described in section 5.1. While the first sections are mainly investigating the general algorithmic aspects and improvements, the remaining sections are entirely focusing on concrete tasks or geometries for inverse kinematics. Initially, section 5.2 gives a parameter evaluation in order to provide an intuition for a suitable choice in population size and number of elites. Section 5.3 then follows with an evaluation of the algorithmic and computational improvements, which were designed to increase the speed and robustness of the algorithm. The performance for traditional inverse kinematics in solving pose goals on serial kinematic chains of robotic manipulators is then demonstrated in section 5.4. In addition, the algorithm is compared to related popular methods for inverse kinematics regarding its success rate and computation time. In section 5.5, the algorithm is applied to the more challenging task of solving full-body postures and motion with multiple goals using the NASA Valkyrie robot model. Followingly, the results for dexterous manipulation experiments with the anthropomorphic Shadow Dexterous Hand are given in section 5.6. Section 5.7 then demonstrates how collision avoidance can be achieved by the algorithm, and which usually outlines a difficult task for generic inverse kinematics. The algorithm was then applied for full-body motion reconstruction and teleoperation in virtual reality in section 5.8, as well as for collision-free trajectory generation in section 5.9. In section 5.10, the algorithm is demonstrated for animation editing and post-processing using highly-articulated game characters. In order to provide a visual understanding and explanation of the results, section 5.11 shows some evolutionary landscapes, posture distributions and particle propagations for optimising inverse kinematics queries. Finally, section 5.12 presents some work that has been created by other users through dissemination of the available implementation for Unity3D.

5.1 Experimental Setup

The experiments were conducted using an ASUS ROG G-751 notebook with 2.6 GHz processor cores. The Unity3D engine was chosen as the main research environment using the available Bio IK implementation [10]. Further experiments to compare the algorithm with existing implementations for robotic inverse kinematics and simulation were also done using the available Bio IK version [9] for ROS. All statistical results were evaluated over 10000 independent, uniformly-distributed and reachable inverse kinematics configurations that were optimised from random initial seeds. This number was found to produce unsignificantly small deviations between the results. Both computational improvements using multi-core threading and the OFKT data structre in sections 4.6.2 and 4.6.1 were used, leading to a faster convergence due to a lower computation time per generation. In Unity3D, the robot models were imported using a URDF (Unified Robot Description Format) models which has been implemented as part of this research.

Model	**Source**	**Description**
UR5	[70]	Serial manipulator with 6 DoF.
Fanuc M-10iA	[42]	Serial manipulator with 6 DoF.
KuKA LBR iiwa R820	[68]	Serial manipulator with 7 DoF.
KuKA KR120R2500pro	[66]	Serial manipulator with 6 DoF.
PR2	[69]	Mobile robotic platform with two 7 DoF arms, a 1 DoF translational torso, and an unconstrained rotational wrist motion.
Baxter	[64]	Robotic research platform with two 7 DoF arms.
NASA Valkyrie	[71]	Humanoid robot with a 35 DoF body and two further 13 DoF hands.
Shadow Dexterous Hand	[39]	Anthropomorphic robotic hand with 19 DoF by a 2 DoF wrist, two 4 DoF and three 3 DoF fingers.
Human Mannequin	[65]	Human mannequin with 42 DoF, having a 3 DoF pelvis, 6 DoF spine, two 7 DoF legs and arms, as well as a 5 DoF head.
Kyle	[67]	Humanoid game character with up to 78 DoF, having a 3 DoF pelvis, spine, neck and head, 7 DoF legs and arms, and 19 DoF for the fingers of each hand.
Wolf	[72]	Quadruped game character with up to 58 DoF, having a 3 DoF pelvis, neck and head, 9 DoF spine, 8 DoF front legs and 6 DoF back legs, as well as a 12 DoF tail.

Table 5.1: List of kinematic geometries for the experiments.

5.2 Parameter Evaluation

The selection of parameters has a major impact on the performance of the algorithm. However, chosing them is not straightforward, and is largely influenced by the DoF as well as the amount of objectives that shall be minimised. Typically, a larger DoF formulates a more complex optimisation problem, but can also introduce a larger solution manifold. In opposite, a higher number of objectives causes a higher constrained and non-smooth fitness landscape for the evolution. The only required parameters for the algorithm are given by the population size and the number of elites. Using a 15 DoF serial kinematic chain, Fig. 5.1 shows the normalised distribution of efficiency which is measured by the required time and achieved fitness. The efficiency E for each parameter selection is averaged over 10000 queries and uses a maximum timeout of 10 ms if not converged. It is calculated according to 5.1, where ψ and κ denote the population size and number of elites, m is the number of samples, ϕ is the achieved fitness and t is the required time for convergence.

$$E_{\psi,\kappa} = \frac{1}{m}\sum^{m}\frac{1}{\phi \cdot t} \tag{5.1}$$

The values are finally normalised to the range $[0, 1]$ for visualisation purposes. It can be observed that there is a larger range of possible combinations for ψ and κ for which the evolution performs similarly well.

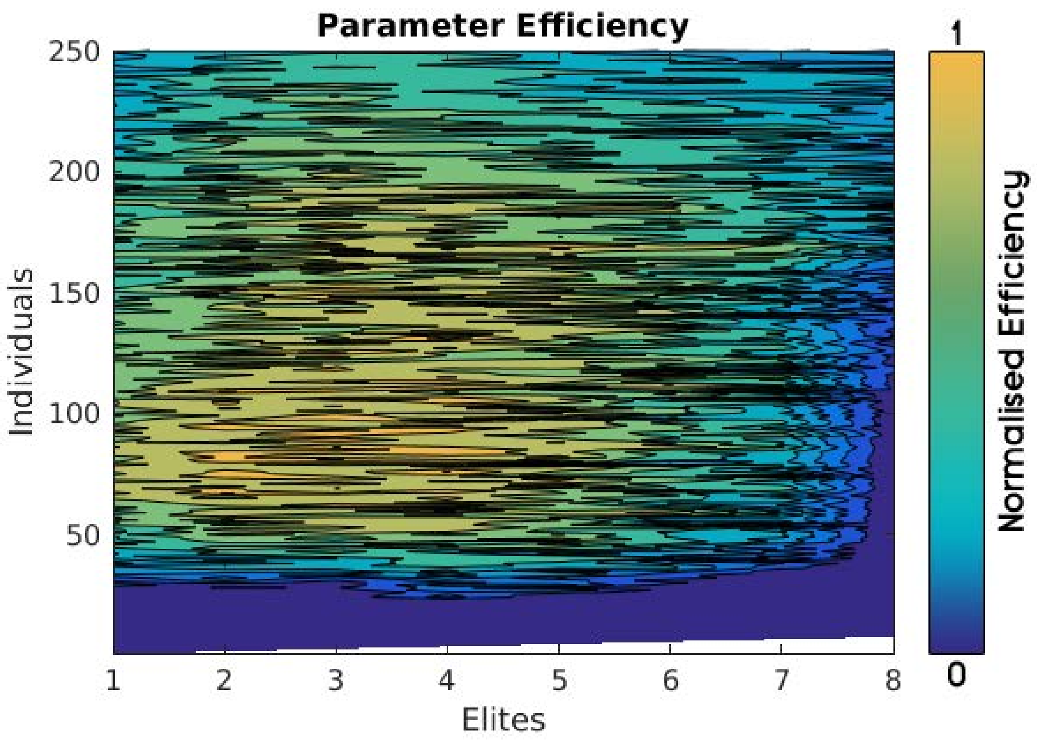


Figure 5.1: Parameter efficiency on a 15 DoF serial kinematic chain.

Similar efficiency distributions could also be observed for varying DoF on serial manipulators. As visualised in Fig. 5.2, choosing at least two elite individuals should be preferred regardless from the DoF. The results further suggest that increasing the population size has larger impact for smaller DoF, and seems to become stable for increasing DoF. In general, parameter selections between $[50, 150]$ individuals with $[2, 5]$ elites were observed to performed well in the experiments.

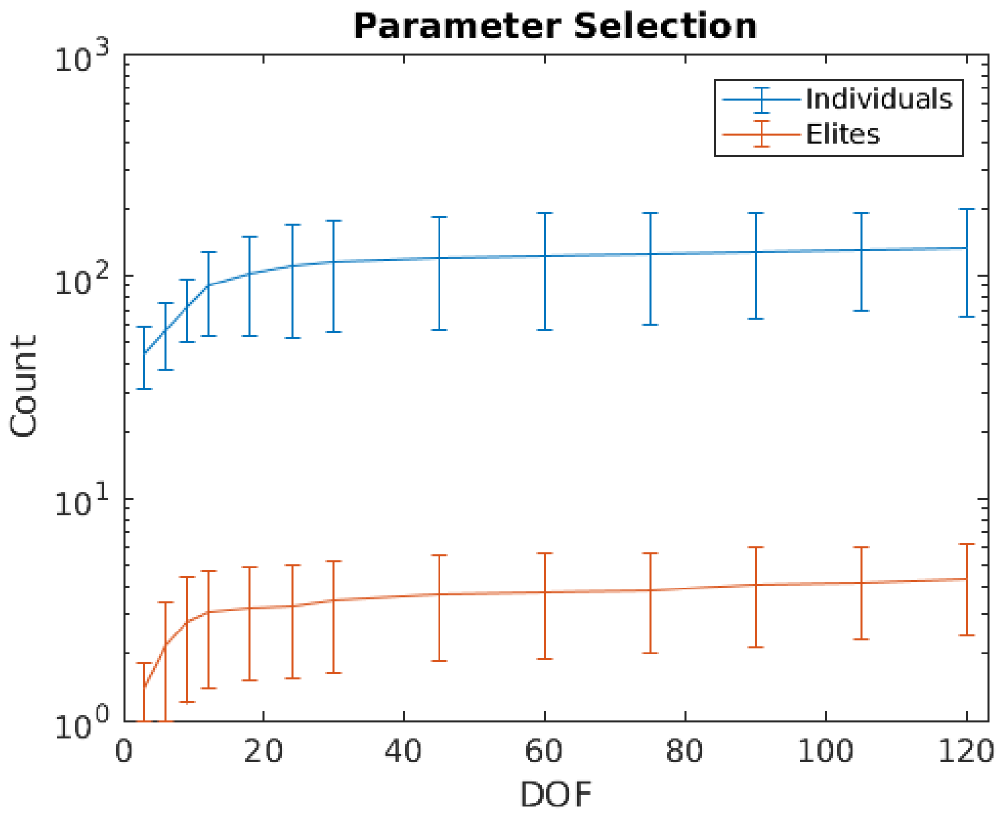


Figure 5.2: Suitable parameter selections for increasing DoF.

5.3 Selective Improvements

The algorithm was further investigated regarding the designed evolutionary and computational improvements. Fig. 5.3 visualises the obtained relative improvements on the Shadow Dexterous Hand and the NASA Valkyrie robot by the adoption, exploitation and wipe out operators that were presented in sections 4.5.5, 4.5.7 and 4.5.8. In addition, it was observed that a randomly-weighted interpolation of weights for the objectives could achieve a more robust optimisation. Although this strategy might give rise to doubts about its success at first, one can imagine that the best solution that can ever be found is exactly the one who performs best regardless of the chosen weights. Thus, potential individuals among will be eliminated if they do not perform successful repeatedly when facing different criteria.

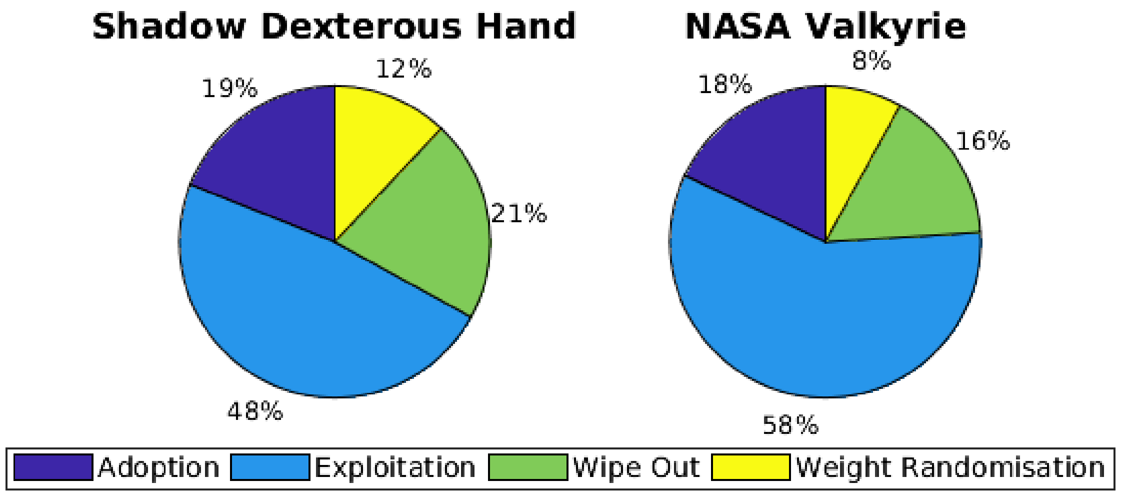


Figure 5.3: Relative contribution to the overall improvement of the Bio IK algorithm over GA regading the main algorithmic extensions and modifications.

The values are measured by the relative loss for the overall convergence when selectively removing one of them. Clearly, the largest improvement is achieved by the memetic exploitation phase. Furthermore, Fig. 5.4 (left) shows the computational improvement for fitness evaluations by measuring the required time per generation for increasing DoF when using the OFKT data structure prestend in section 4.6.1. It can be observed that the improvement considerably scales for more complex geometries, reaching a cost reduction per generation by a factor of ≈8 for 30 DoF. It also visualises the improvement that could be achieved by multi-core threading, which was designed for the elitism exploitation as described in section 4.6.2. The results are obtained from a 35 DoF setup on the NASA Valkyrie robot with a fixed population size of $\psi = 75$. When using threads, the additional computational cost for each elite can nearly be eliminated compared to a single-core implementation.

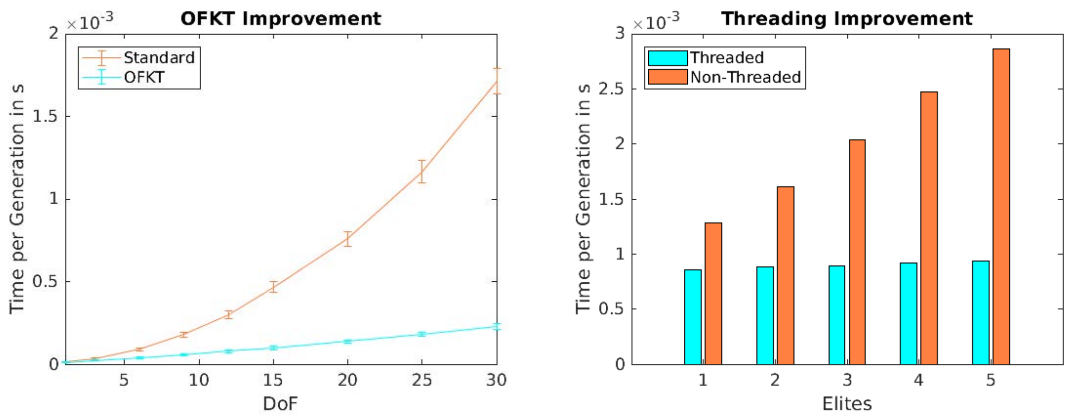


Figure 5.4: Computational cost per generation for increasing DoF using standard calculation and the OFKT data structure for forward kinematics (left [96]), as well as when using threads for every elite subject to the L-BFGS-B exploitation (right).

5.4 Serial Inverse Kinematics

In order to measure the general performance of the algorithm in comparison to related methods, initial experiments were made on common robotic serial manipulator arms shown in Fig. 5.5. Those were selected with respect to solving varying geometric structures, segment lengths and joint limits, which lastly formulate different search spaces of similar dimensionality. The teal target spheres visualise the pose objectives to be optimised, and which were generated from random valid joint configurations using the forward kinematics equations 2.10. The algorithm was then used to find a solution which minimises the translational and rotational error to an accuracy of 10^{-4} m and 10^{-3} rad. In Fig. 5.6, the average success rate that could be obtained after a certain number of generations is shown. The graph demonstrates that all 10000 goal configurations for the tested robot models could

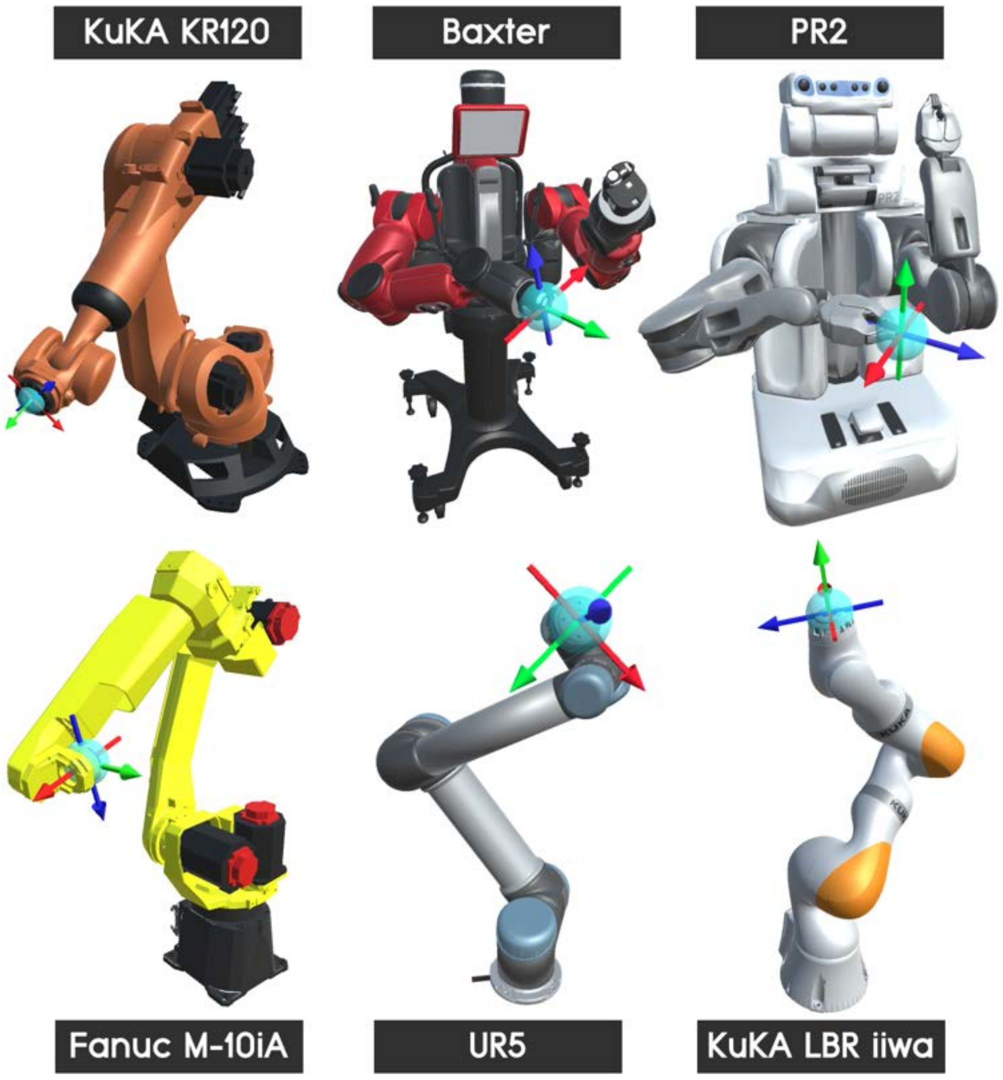


Figure 5.5: Robot models for the experimental results listed in Tbl. 5.2. The teal spheres visualise the Cartesian pose targets for solving inverse kinematics.

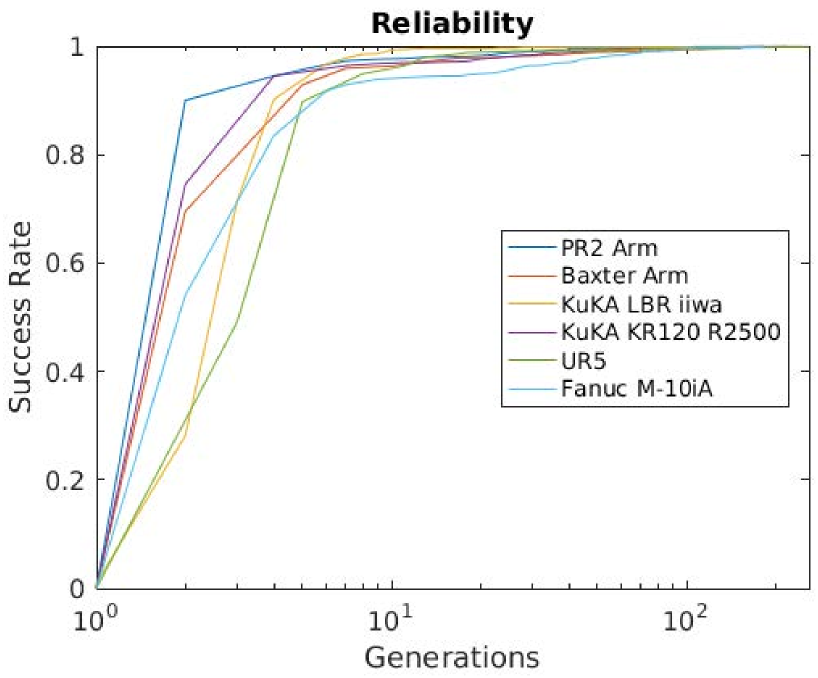


Figure 5.6: Success rate for serial inverse kinematics with respect to the number of evolved generations using the robot models in Fig. 5.5.

be successfully evolved. While about 10^2 generations were required to find all solutions, it can be expected that 10^1 generations in average are required to converge. In some cases, only a very small number of generations is sufficient, which can be reasoned by the elitism exploitation which performs multiple L-BFGS-B iterations in parallel within one generation. Thus, when starting from a seed configuration that is near to the solution, the evolution converges immediately. Tab. 5.2 compares the algorithm to existing methods for inverse kinematics. The results show the required computation times and success rates for the serial manipulator arms in Fig. 5.5. For the Jacobian (Transpose / DLS) and L-BFGS-B methods, the optimisation was terminated if the algorithm started violating joint limits, and in which case no valid solution would be found. For the GA and DE methods, a time limit of 1 s was specified by which a solution had to be found. While the gradient-based approaches converge much faster than the evolutionary methods, their reliability in finding a solution significantly varies for the different robot geometries. However, although this seems to be more stable for the evolutionary methods, they also require considerably more time to converge. Finally, Bio IK requires similar computation time like gradient-based methods of approximately 1 ms in average, but outperforms them with success rates of over 99% when using a maximum time limit of 10 ms. Furthermore, Fig. 5.7 shows the average required time to optimise inverse kinematics solutions on hyperredundant manipulators with up to 120 DoF. For those, the algorithm achieves a slightly sublinear relation between time and DoF. Note that solving such structures is often difficult for related Jacobian methods due to the typical singularity issues.

Manipulator (Full-Pose)	**DoF**	**Jacobian-T (Joint Limit Termination)**	**Jacobian-DLS (Joint Limit Termination)**	**L-BFGS-B (Joint Limit Termination)**	**GA (1s Timeout)**	**DE (1s Timeout)**	**Bio IK (1s Timeout)**	**Bio IK (10ms Timeout)**
PR2 Arm	7	4.06 ms (25.27%)	0.34 ms (28.61%)	0.36 ms (37.68%)	334.53 ms (36.24%)	197.65 ms (66.89%)	1.02 ms (100%)	0.91 ms (99.47%)
Baxter Arm	7	8.73 ms (36.92%)	0.59 ms (9.78%)	0.44 ms (43.19%)	362.18 ms (39.22%)	171.39 ms (56.84%)	1.34 ms (100%)	1.12 ms (99.16%)
KuKA LBR iiwa	7	5.27 ms (88.71%)	0.49 ms (68.64%)	0.41 ms (74.33%)	297.87 ms (61.14%)	129.42 ms (87.39%)	0.85 ms (100%)	0.74 ms (99.93%)
KuKA KR120 R2500	6	3.05 ms (58.77%)	0.36 ms (59.62%)	0.39 ms (67.74%)	398.49 ms (52.21%)	231.87 ms (77.17%)	1.33 ms (100%)	1.09 ms (99.21%)
UR5	6	7.39 ms (88.14%)	0.79 ms (77.81%)	0.36 ms (86.97%)	323.84 ms (68.53%)	159.11 ms (89.75%)	0.94 ms (100%)	0.91 ms (99.38%)
Fanuc M-10iA	6	6.92 ms (45.78%)	1.34 ms (55.83%)	0.35 ms (49.66%)	416.81 ms (42.67%)	203.64 ms (76.91%)	1.26 ms (100%)	1.07 ms (98.96%)

Table 5.2: Performance of the algorithm compared to related gradient-based and evolutionary approaches for solving inverse kinematics on serial manipulators. The results show the average computation times and success rates over 10.000 queries.

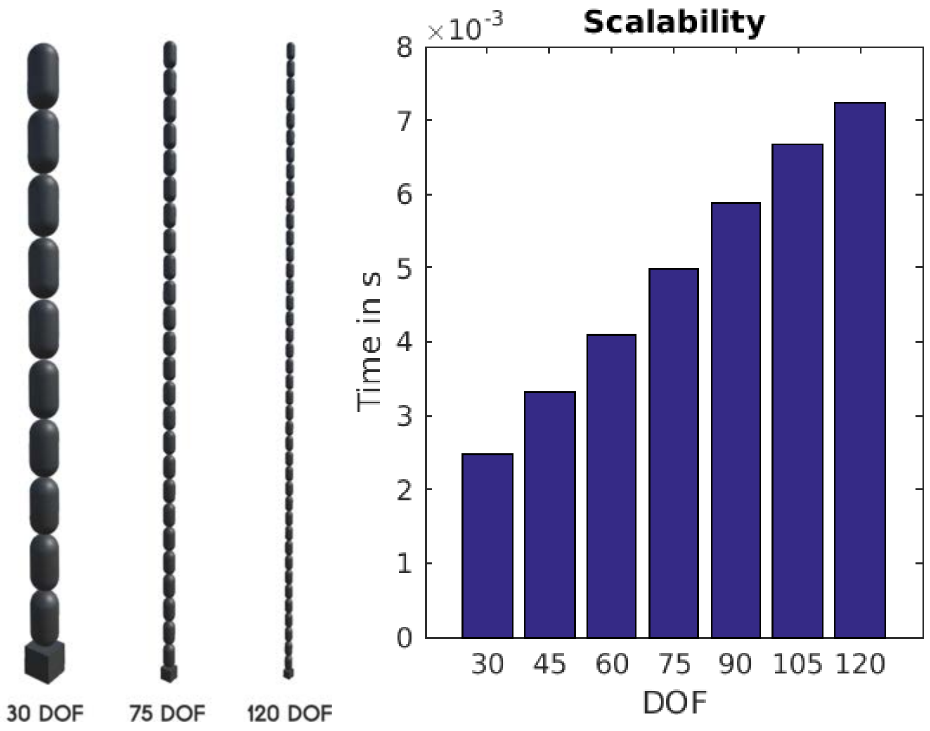


Figure 5.7: Inverse kinematics on hyperredundant serial manipulators.

The algorithm was also compared to existing inverse kinematics plugins for ROS using the implemented solver [9] that was developed and published in [86, 87]. The written C++ code is highly optimised, and can achieve even lower computation times than the implementation in [10]. The results in Fig. 5.3 compare the performance in speed and reliability using KDL [50], TRAC-IK [80], HGSA [95] and the proposed Bio IK algorithm. KDL is a plugin which is based on Jacobian op-

Solver	**PR2**	**UR5**	**Valkyrie Arm**	**Valkyrie Leg**	**LBR iiwa**
KDL	4.68 ms (53.10%)	4.59 ms (41.70%)	3.50 ms (41.13%)	0.91 ms (91.68%)	3.17 ms (51.62%)
TRAC-IK	0.77 ms (99.91%)	0.52 ms (99.29%)	0.73 ms (99.64%)	0.18 ms (99.98%)	0.41 ms (99.88%)
HGSA	3.93 ms (75.69%)	4.12 ms (51.20%)	4.67 ms (29.13%)	3.14 ms (70.14%)	3.49 ms (66.84%)
Bio IK	0.45 ms (100.00%)	0.50 ms (99.93%)	0.51 ms (99.93%)	0.28 ms (100.00%)	0,47 ms (99.93%)

Table 5.3: Inverse kinematics benchmark regarding computation time and success rate. The results compare different ROS plugins over 10000 valid joint configurations with a maximum timeout of 5 ms and target accuracy of 10^{-5}m/rad. [87]

timisation. TRAC-IK uses SQP and the BFGS method with random heuristic restarts. HGSA is a previous version of Bio IK which only combines GA, PSO and a rather simple local search heuristic. It can be observed that Bio IK achieves a both slightly better computation time and success rate in average than TRAC-IK, which seem to be the only competitors. This is also demonstrated in Fig. 5.8, which visualises successful (green) and failed (red) inverse kinematics queries for box-shaped and randomly-sampled pose goals.

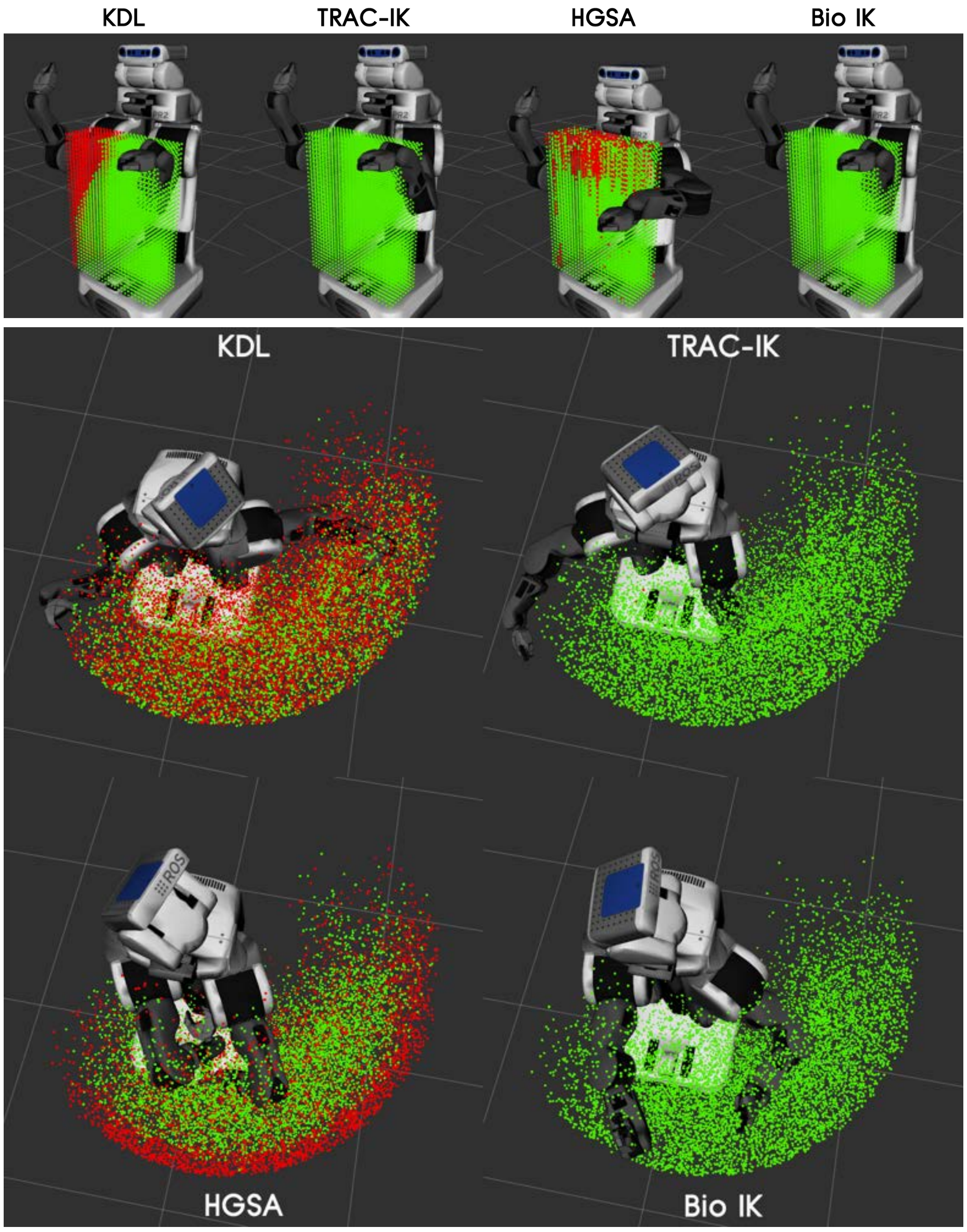


Figure 5.8: Inverse kinematics distributions on the PR2 robot arm. [87]

5.5 Full-Body Inverse Kinematics

A more challenging inverse kinematics problem is given by solving postures on fully-articulated humanoid robots. Those have multiple kinematic chains whose joints from the pelvis to hands, head and feet partially influence the segment and end effector poses of each other. Therefore, a kinematic tree with multiple kinematic chains needs to be optimised at once, rather than each chain independently. In Fig 5.9, the geometry of the NASA Valkyrie humanoid robot is shown. Both arms are affected by the pelvis and torso configuration, and which further also affect the finger poses of the anthropomorphic hands. Finding a posture which globally satisfies the different goals often causes analytic and gradient-based methods in sections 3.1, 3.2.1 and 3.2.2 to be not applicable anymore as the number of local extrema and dimensionality increases. Therefore, not much literature seems to be available for such problems, and most work focuses on traditional serial inverse kinematics. In Tab. 5.4, the required computation times and success rates for different inverse kinematics setups of Cartesian goal combinations is listed. This demonstrates the scalability of the Bio IK algorithm, which maintains a fast and robust convergence even for multiple objectives and high DoF. When adding further objectives, it was observed that using a RMSE loss instead of linearly averaging the single errors in (4.3) achieves a huge improvement for the optimisation — making the evolution significantly faster and applicable for robustly solving complex full-body postures.

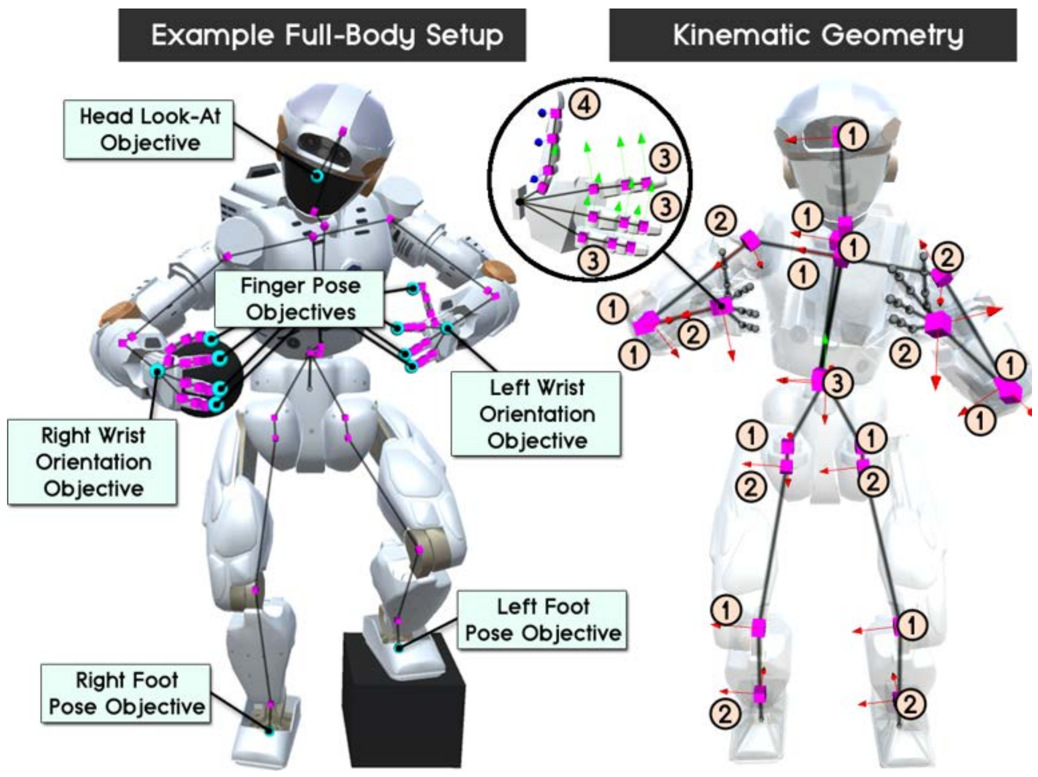


Figure 5.9: Kinematic geometry of the NASA Valkyrie humanoid robot with 61 DoF in total. The model has a 35 DoF body and two 13 DoF hands.

NASA Valkyrie Setup	DoF	Objectives	Parameters (Individuals / Elites)	Time and Success for ME Loss (10ms timeout)	Time and Success for RMSE Loss (10ms timeout)
Torso to Right Wrist (Position & Orientation)	10	2	90 / 2	1.79 ms (90.23%)	0.87 ms (99.96%)
Right Leg (Position & Orientation)	6	2	75 / 2	1.48 ms (93.36%)	0.63 ms (99.98%)
Torso to Left and Right Wrist (Position & Orientation)	17	4	100 / 3	5.97 ms (24.72%)	2.07 ms (98.31%)
Torso to Right Wrist (Position & Orientation) with Right Elbow (Position) and to Head (Look At)	13	4	110 / 3	5.31 ms (18.11%)	2.23 ms (98.82%)
Torso to Left and Right Wrist (Position & Orientation) and to Head (Look At)	20	5	130 / 3	6.53 ms (7.94%)	2.67 ms (97.98%)
Torso to Right Wrist (Orientation) and to Thumb, Index, Middle and Ring Fingers (Position)	23	5	150 / 3	6.22 ms (11.61%)	3.24 ms (97.69%)
Torso to Left and Right Wrist (Position & Orientation) and to both Feet (Position & Orientation)	32	8	160 / 4	8.43 ms (1.27%)	5.64 ms (95.29%)
Torso to Left and Right Wrist (Position & Orientation) and to both Feet (Position & Orientation) and to Head (Look At)	35	9	180 / 4	9.13 ms (0.48%)	6.22 ms (94.56%)

Table 5.4: Performance of the algorithm in solving full-body postures with multiple Cartesian objectives on the NASA Valkyrie robot. The results show the average computation times and success rates over 10.000 queries on valid kinematic postures.

Next, Fig. 5.10 visualises some keyframes of a full-body motion for lifting and rotating an object. The motion could be evolved in real-time, and was free from self-collisions using additional distance objectives between segments. Position and orientation objectives for the wrist and fingers were defined to grasp the surface of the object. Moving the object finally generated the required motion of the robot body. Similarly, an objective for the viewing direction of the head was defined to look at the object centre. Two further position and orientation objectives were defined to stick to the ground while optimising the pelvis pose. The average time to compute one posture using the previous as the seed was about 0.82 ms.

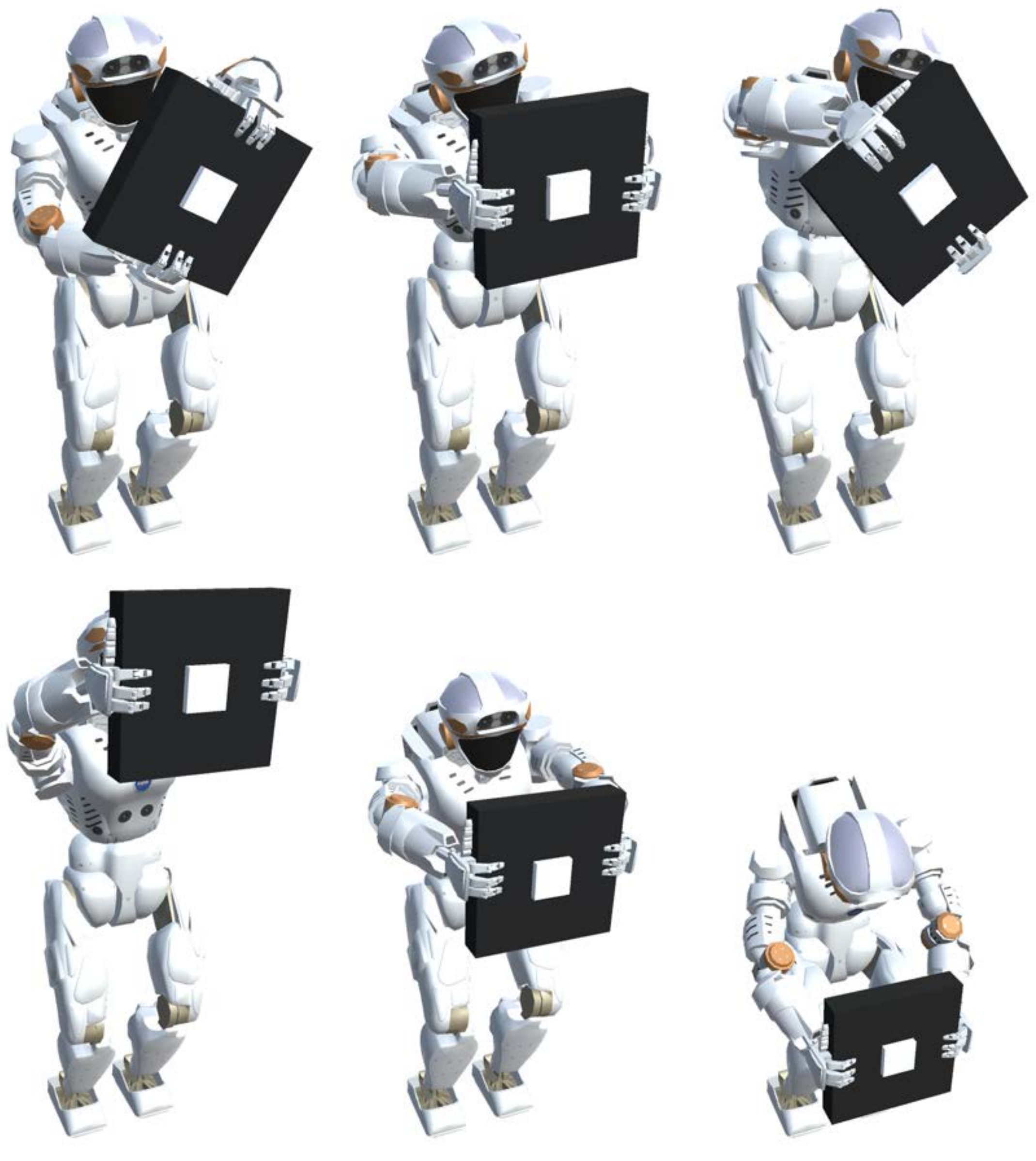


Figure 5.10: Full-body motion on the NASA Valkyrie robot while lifting/rotating an object. All postures were obtained in real-time and free from self-collisions.

5.6 Grasping and Dexterous Manipulation

The algorithm was further investigated on anthropomorphic robotic hands for solving grasps and dexterous manipulation scenarios. Such tasks are often difficult since all fingers are again directly controlled by the preceding joints of the wrist and the arm of a robot. Also, manipulation with real human hands typically involves rolling the fingers slightly on top of the surface instead of keeping a particular pose for the individual finger tips. From a computational perspective, this requires handling particular objectives as soft-constraints, such as having an appropriately lower weight for the orientation than for the position objective. The Shadow Dexterous Hand geometry is shown in Fig. 5.11, and consists of a 2 DoF wrist, 4 DoF thumb and pinky fingers and 3 DoF index, middle and ring fingers.

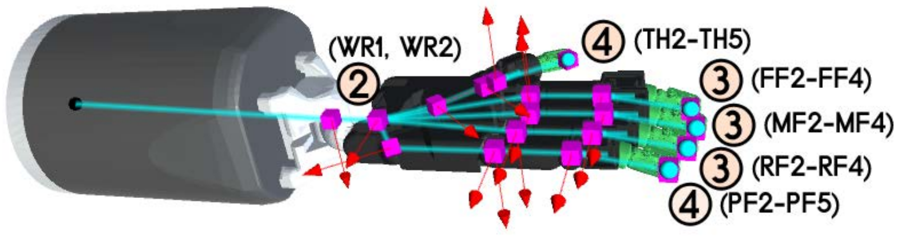


Figure 5.11: Kinematic geometry of the 19 DoF Shadow Dexterous Hand.

Fig. 5.12 demonstrates the proposed algorithm applied to different exemplary grasping types for the hand. Those were recorded from data of 444 real-human

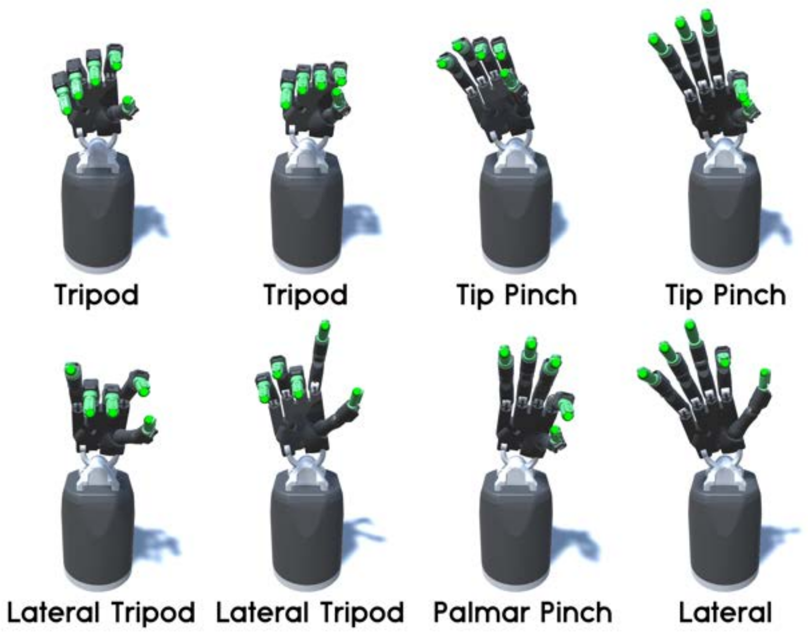


Figure 5.12: Solving inverse kinematics for different grasp types on the Shadow Dexterous Hand using captured real-human hand data.

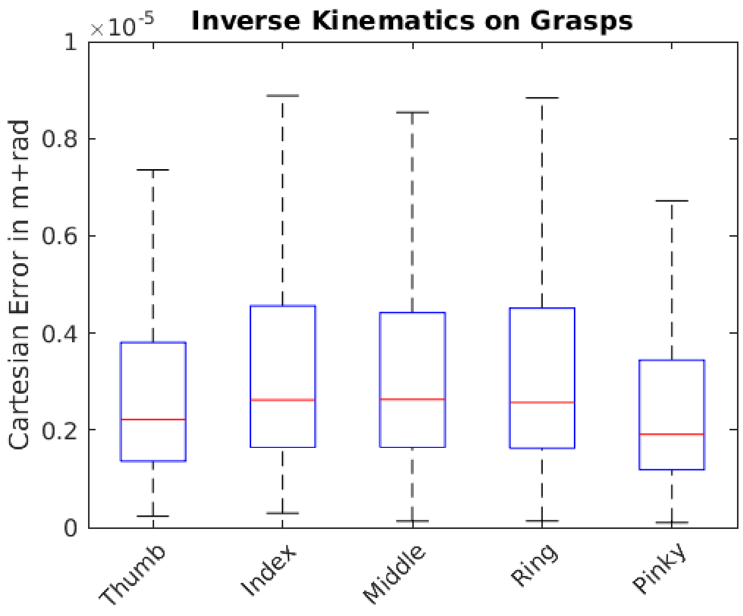


Figure 5.13: Finger tip accuracies for reconstructing 444 real-human grasp configurations on the Shadow Dexterous Hand as shown in Fig. 5.12.

grasps [8], and the algorithm was then used to find a solution which resembles the original finger poses of the hand, despite of varying link lengths and joint limits between the humanoid and robotic hand. All different tested grasp types could be successfully evolved within 2.1 ms in average. Using a fixed optimisation time of 10 ms, Fig. 5.13 shows the achieved Cartesian accuracy and error deviation of the single fingers over reconstructing all 444 captured human grasp configurations. The algorithm was able to obtain solutions with a pose accuracy between 10^{-5} - 10^{-4} m and rad. In particular, slightly better accuracies could be achieved for the thumb and pinky fingers — which can be reasoned by a higher kinematic flexibility due to one more DoF, and thus a larger solution manifold in the fitness landscape. Next, the Shadow Dexterous Hand was attached to the right arm of the PR2 robot as well as to the KuKA LBR iiwa manipulator. In the first experiment in Fig. 5.14, a wooden cube was rotated through evolving the motion of the robot from the arm to the thumb, index and middle fingers — giving rise to 17 DoF in total. A grasping pose on the surface of the cube was defined for each finger while using a lower weight for orientation with 0.1 than for position with 10.0. Two weighted soft-constraints for the wrist orientation with 0.5 and elbow position with 0.25 were specified to constrain the solution manifold to suitable grasping postures. Note that both intermediate goals are lower weighted in order to provide some flexibility for the arm configuration while solving grasps. The visualisation shows the motion of the single joints for two full object manipulation cycles. All joints of the robot arm can be observed to be modified over time, while also accounting for joints running into their lower or upper limits. In such cases, the evolution causes slightly stronger updates for the remaining joints in order to compensate the errors and to keep satisfying the objectives. For the second experiment in Fig. 5.15, a capsule was rotated around its vertical axis by defining target poses on the surface for the thumb and index finger — having 16 DoF in total. Two additional soft-constraints were similarly defined for the elbow position and wrist orientation to obtain reasonable grasp postures. The object was moved in real-time with 100 Hz optimisation time (0.01 s timeout) and low error below 1 mm, as

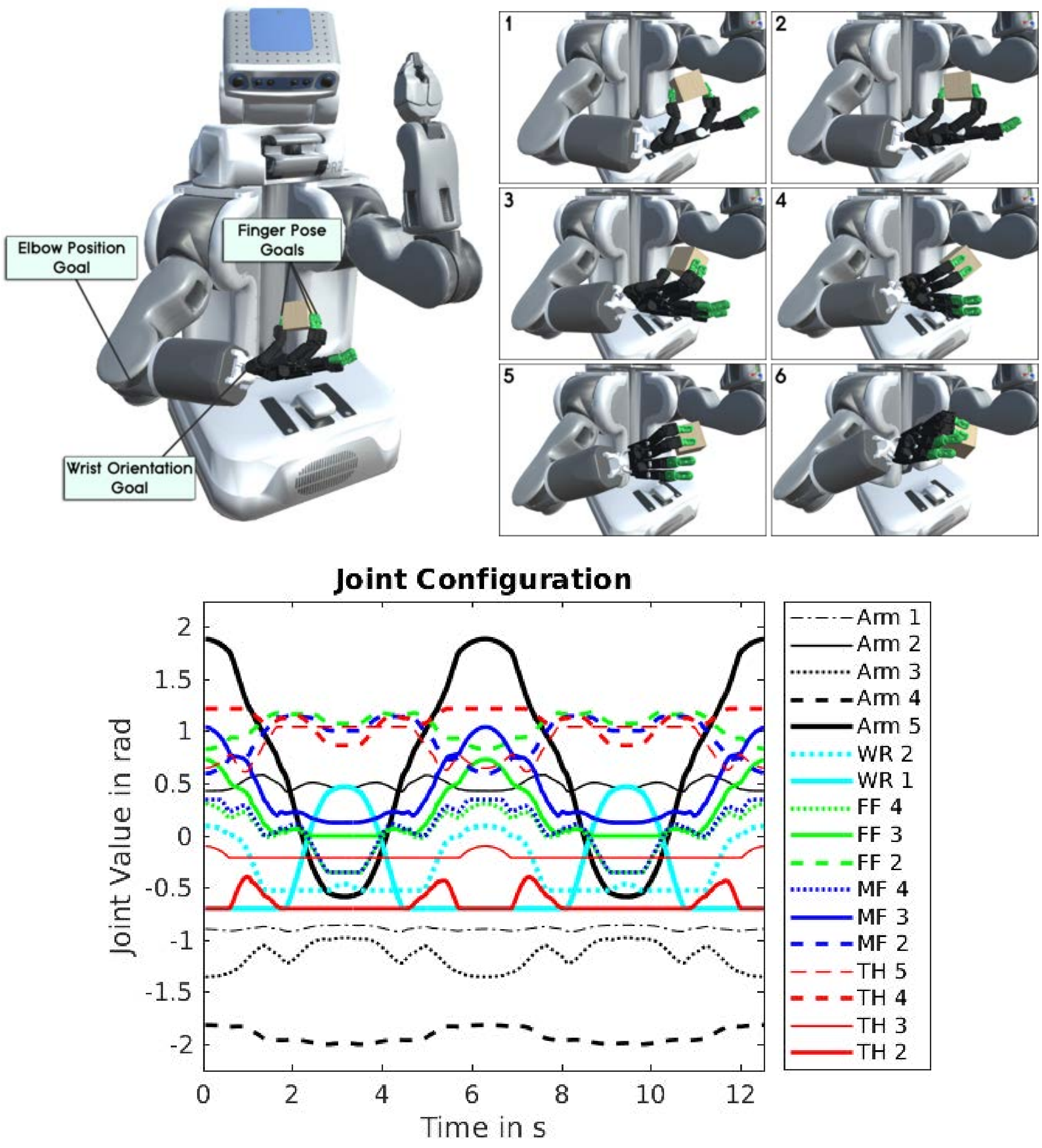


Figure 5.14: Dexterous object manipulation using the PR2 with the Shadow Dexterous Hand. Multiple objectives are used to update the arm, wrist and finger joints simultaneously to rotate the cube around all three Cartesian axes. Notation: WR – wrist, FF – first finger, MF – middle finger, TH – thumb.

shown in the manipulation and accuracy graphs. In general, the motion in both experiments could be observed to be more human-like since not solving each finger individually, but also providing proper joint updates for the wrist and arm.

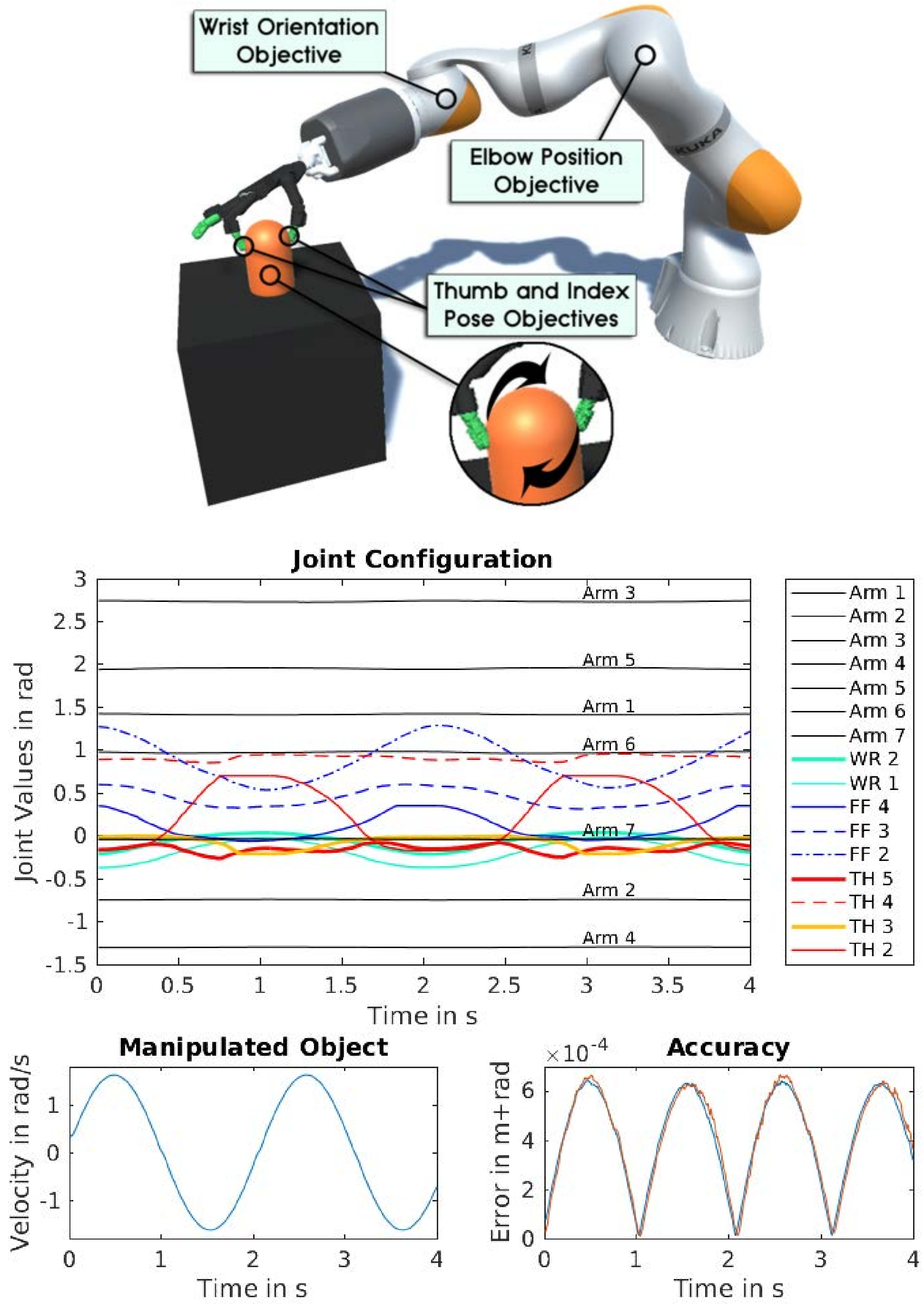


Figure 5.15: Dexterous object manipulation using the KuKA LBR iiwa with the Shadow Dexterous Hand. Multiple objectives are used to update the arm, wrist and finger joints simultaneously to rotate the capsule around the vertical axis. Notation: WR – wrist, FF – first finger, TH – thumb. [94]

5.7 Collision Avoidance

A difficult task both in robotics and animation is given by collision avoidance, but which is not generally straightforward to achieve in a computationally efficient way. However, while robotics requires collisions to be avoided with higher priority on reliability, the interest in animation is rather to ensure real-time capability to maintain high interactive frame rates. In particular, a method is desired which can be used to avoid collisions with the geometry itself and also with obstacles in the environment in order to provide valid kinematic solutions in Cartesian space. Although this is usually neglected for generic inverse kinematics solvers, the proposed Bio IK algorithm can efficiently handle this task using the distance objectives which were described in section 4.4.4. As shown in Fig. 5.16, the geometry of the KuKA LBR iiwa robot is approximated by a set of spherical distance objectives. The distance threshold for each pair of points between the robot segments and the black sphere obstacle is calculated by the radius sum of both bounding spheres. In the experiment, the robot arm smoothly avoids collisions by maintaining a variable safety distance while reaching for the teal target sphere. Note that this distance is controlled by the objective weights, which were set to 10^{-4} in the experiments. Solutions were possible to find in real-time using a maximum timeout of 5 ms.

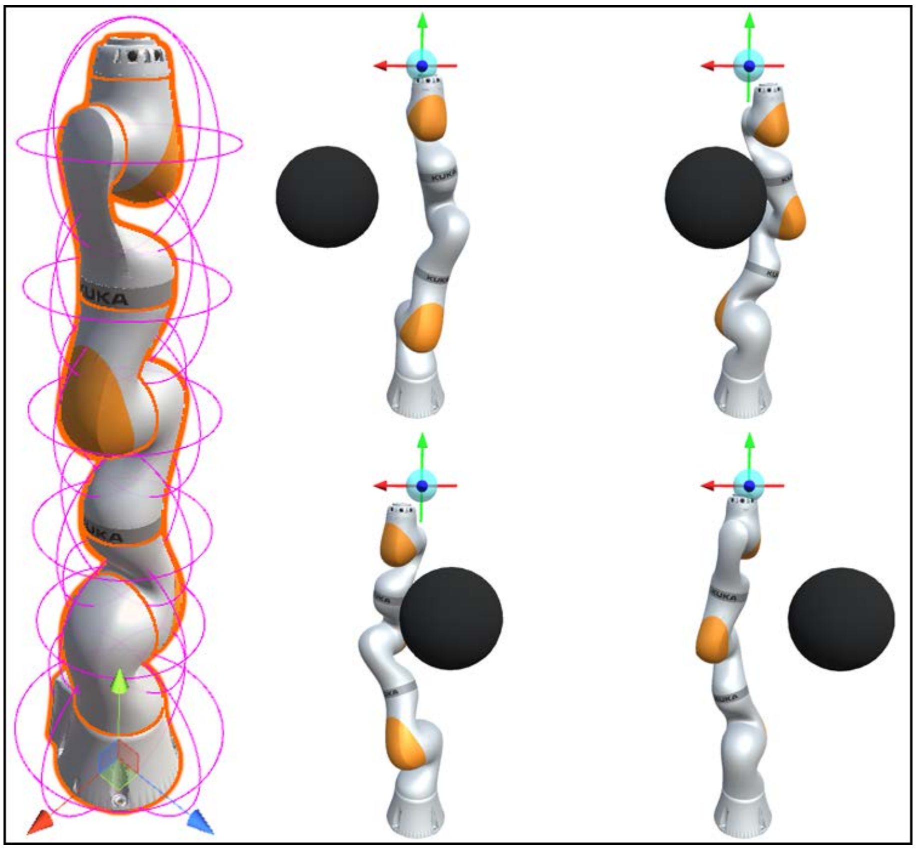


Figure 5.16: Real-time collision avoidance on the KuKA LBR iiwa manipulator through approximating its geometry by a set of distance objectives.

A more complex task was performed using the PR2 robot when moving the gripper to a pose goal that was specified within an open box. The experiment is visualised in Fig. 5.17. Similarly, the collision geometries for the arm and box were approximated by a set of spheres through multiple distance objectives. In more detail, each of the 6 distance objectives of the arm integrates the cost to all 32 distance points of the box, resulting in 192 distance calculations per fitness evaluation. When placing the target within a solid part of the box, the algorithm still tries to reach the target as close as possible without resulting in a collision. Finally, when placing the target within free space inside the box, a solution can be found with safety distances to the nearest collision points. This technique can be applied to achieve real-time collision avoidance for generic inverse kinematics, but also comes with two challenges. First, the collision geometry needs to be accurately specified which is not always straightforward. Second, a proper weighting of distance objectives with respect to the other objectives must be found such that the gripper gets close enough to the target.

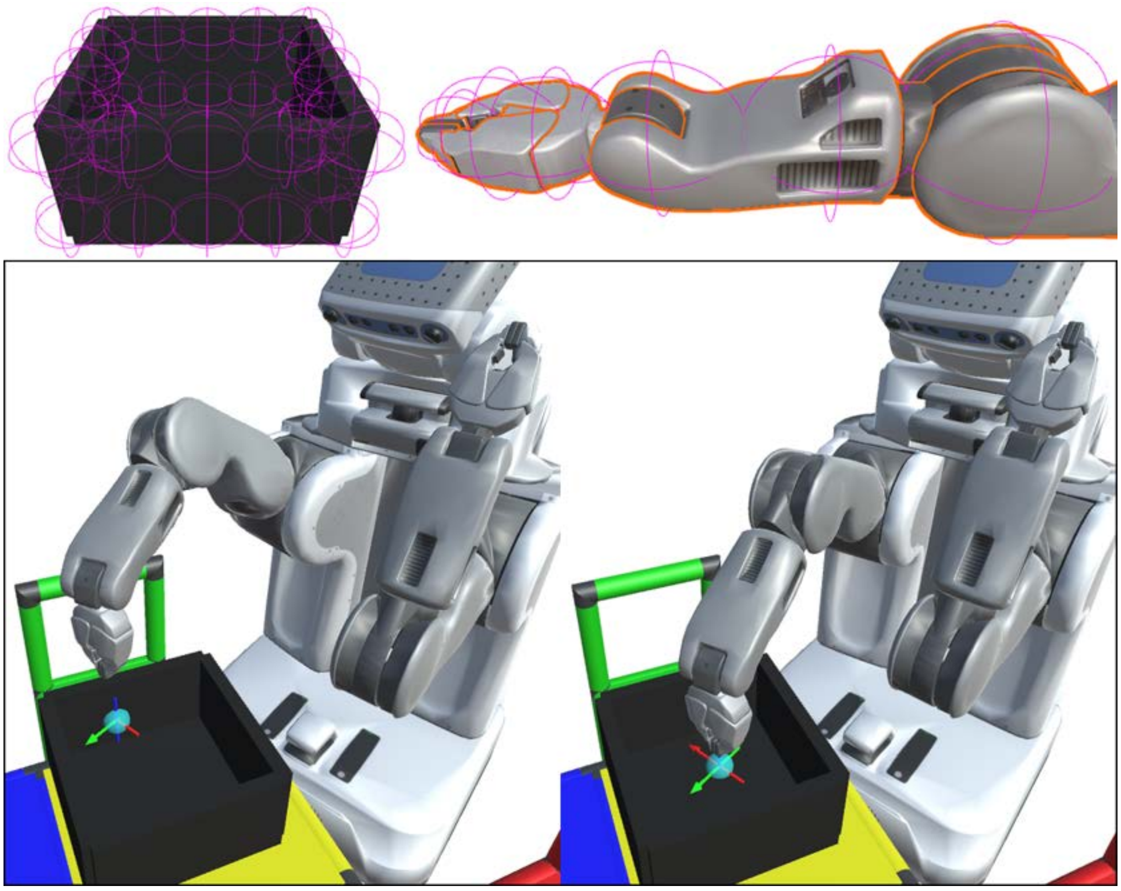


Figure 5.17: Real-time collision avoidance on the PR2 mobile robot platform while grasping into a box. The robot arm and box geometries were approximated by a set of distance objectives, using 6 points for the arm and 32 for the box. As visualised, the robot successfully rejects configurations which would result in collisions for approaching the teal target sphere (left), but converges otherwise (right).

5.8 Motion Reconstruction and Teleoperation

Another challenging application for inverse kinematics is given by remotely controlling a robot in virtual reality through teleoperation. Similarly, it can be used for interactively transferring motion of a real-human on a game character. Such tasks require a fast and responsive real-time computation, and also the ability to provide realistic postures for highly articulated models. Otherwise, difficulties in the human-robot or human-computer interaction can occur by perceiving visual feedback which largely differs from the expectation of the user. Thus, a robust handling of different link lengths and joint limits between the kinematic structure and the real-human operator must be ensured. In Fig. 5.19, an example of reconstructing human motion on the NASA Valkyrie robot model in virtual reality is shown. The robot geometry was defined as described in Fig. 5.18. Two position and orientation objectives were specified for the left and right wrist of the robot, along with a further orientation objective for the head. The targets were updated by the tracking information obtained from the HTC Vive controllers and headset. The pose for the feet was set to be fixed on the ground while solving the pelvis pose. Hence, only a few goals were used to reconstruct a full-body motion on a complex humanoid robot. As can be observed, the algorithm successfully evolves similar looking robot postures as those of the operator. The algorithm further achieved smooth motion between posture transitions of standing, turning, stretching, waving or kneeling, and was run in real-time with a timeout of 5 ms.

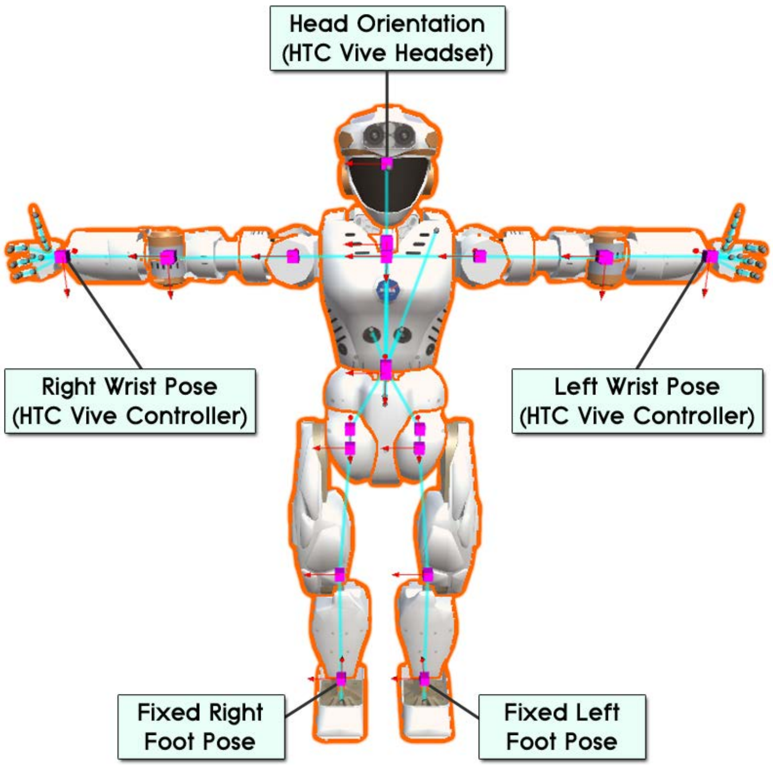


Figure 5.18: Kinematic setup for the experiment in Fig. 5.19.

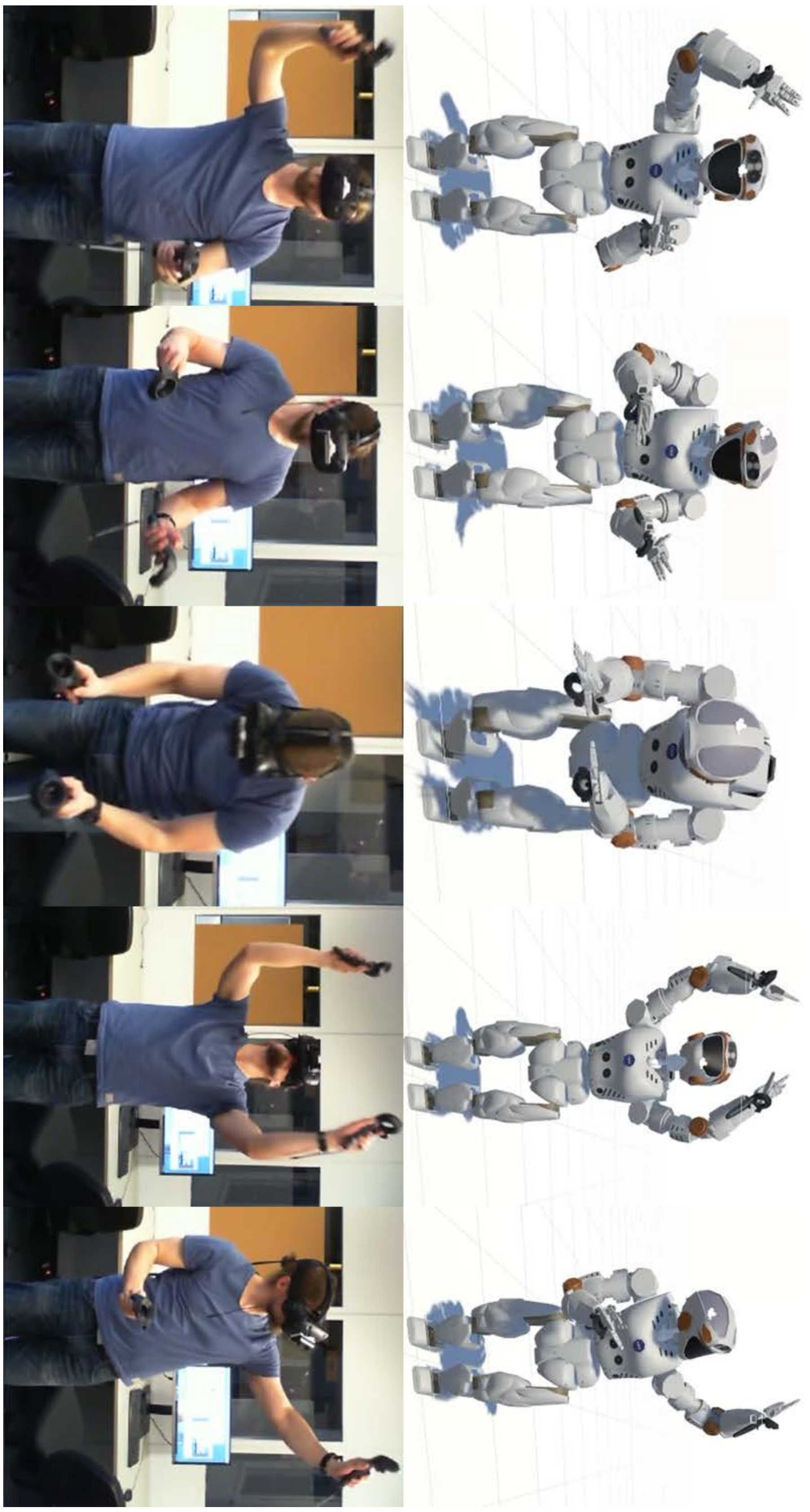


Figure 5.19: Motion reconstruction for teleoperation in virtual reality on the NASA Valkyrie robot using the HTC Vive controllers and headset.

5.9 Trajectory Generation

When generating motion on robots or virtual characters, the interest is often not only in solving final postures, but also in providing trajectories of multiple neighbouring postures between the start and goal configurations. In particular, interactive applications in human-robot interaction and procedural character animation typically require solving targets which are continually updated by the user, which results in a dynamic optimisation problem with the objectives changing over time. Thus, solutions are desired with minimal displacement to the given seed configuration in order to avoid noisy and fluctuating movements. Solving such tasks is not straightforward since an adaptive switching between exploitation and exploration for the optimisation is required. More specifically, the goal of this is to follow the direction of the error gradient as long as it leads to a suitable solution, but to concurrently allow finding solutions with a larger distance in joint and Cartesian space and to avoid getting stuck in joint limits.

Fig. 5.20 demonstrates the developed algorithm to be applicable for generating smooth motion trajectories demonstrated on the 42 DoF human body mannequin. Pose targets were specified for each end effector point on the feet and hands, and were interactively moved by user input. Within each frame, only few generations were required for posture refinements. In particular, constant update rates of at least 250 Hz could be maintained for solving neighbouring solutions, making the algorihtm fast enough to be used for multiple characters concurrently. However, good looking motion could also be achieved for much lower optimisation times in order to provide higher update rates. While pure genetic algorithms caused frequent changes in the assigned postures due to the random jumps, it was observed that these motion artifacts could be largely eliminated by the integrated swarm adoption and elitism exploitation. Note that it was also possible to specify unreachable goals, and the evolution could find an optimal joint variable configuration.

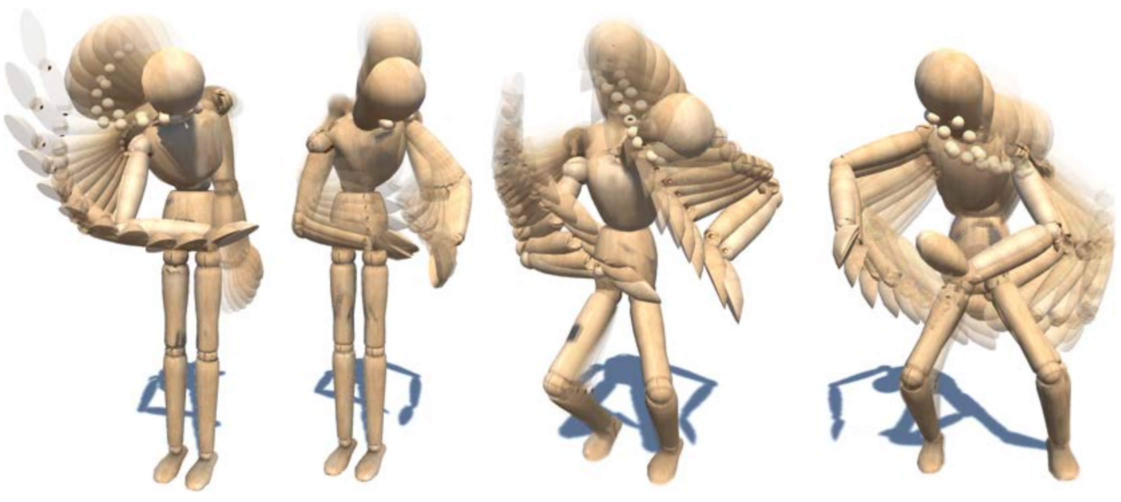


Figure 5.20: Real-time motion trajectories on the human body mannequin. [97]

A further challenge that is particularly relevant in robotics is given by collision-free trajectory generation. As visualised in Fig. 5.21, a linear interpolation of trajectory poses from the start to the end pose of the right hand was initially generated. Each pose was then solved and corrected with the Bio IK algorithm while using additional distance objectives along the body in order to avoid collisions with the obstacle sphere. Concurrently, the head was given a direction objective to keep looking at the right hand. In total, the motion from the pelvis to the head and right hand was considered while using a timeout of 10 ms. It can be observed that a smooth and collision-free trajectory around the obstacle could be produced.

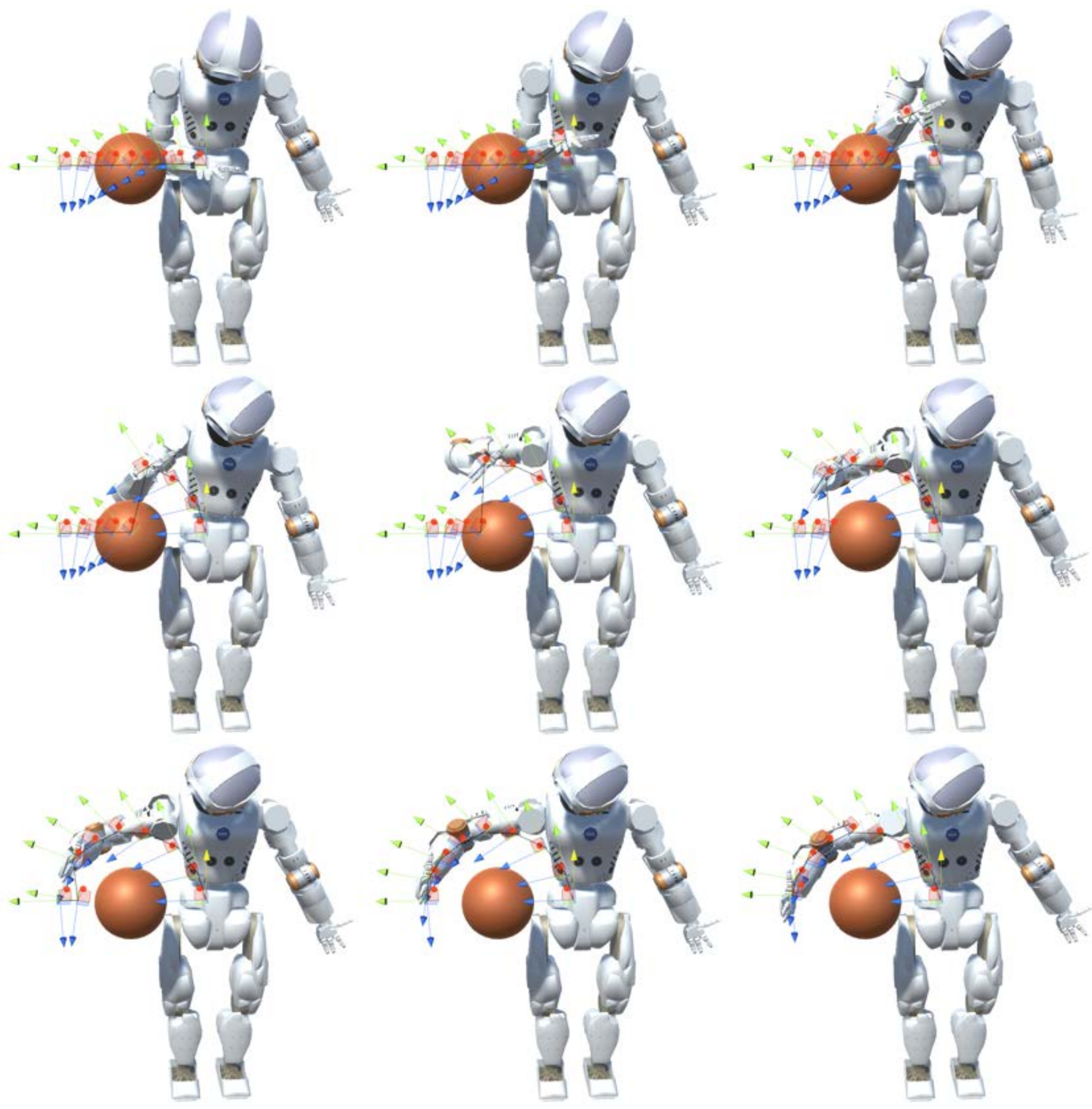


Figure 5.21: Real-time motion trajectories with collision avoidance on the NASA Valkyrie humanoid robot model. Using position, orientation and distance objectives, the given trajectory from the starting to the end pose through the sphere obstacle is corrected such that the hand is smoothly moved around it.

5.10 Animation Editing and Post-Processing

Character animation is a challenging field for which inverse kinematics has always played an essential role. It is most commonly used for solving character postures for animation keyframe creation. Two example postures for kicking and running on the Kyle robot are visualised in Fig. 5.22. The full-body postures were generated using four pose targets for the hand and feet. Through interactively updating the targets by user input, a set of keyframe postures can be created which finally produce the animations by blending between postures. Once an animation has been created, inverse kinematics can be used to create different styles of motion by animation editing. The developed algorithm can be used for such tasks as demonstrated in Fig. 5.23 using a quadruped Wolf character. Based on a default walking animation, the motion was separately modified for two different applications. First, it was edited into a sneaking animation by resembling the full-body motion through pose targets for the leg end effectors. The position and orientation values were obtained by the transformations of the default animation. Furthermore, the body was lowered by two position objectives for the pelvis and shoulder, and a higher position goal was given for the tail. This animation could then again be separated into keyframes for a new animation. Second, the motion was manipulated to keep tracking an object while walking using a direction objective at the head. The motion of the joints for the upper body from the pelvis along the spine to the forelegs and to the head was optimised by the evolutionary algorithm.

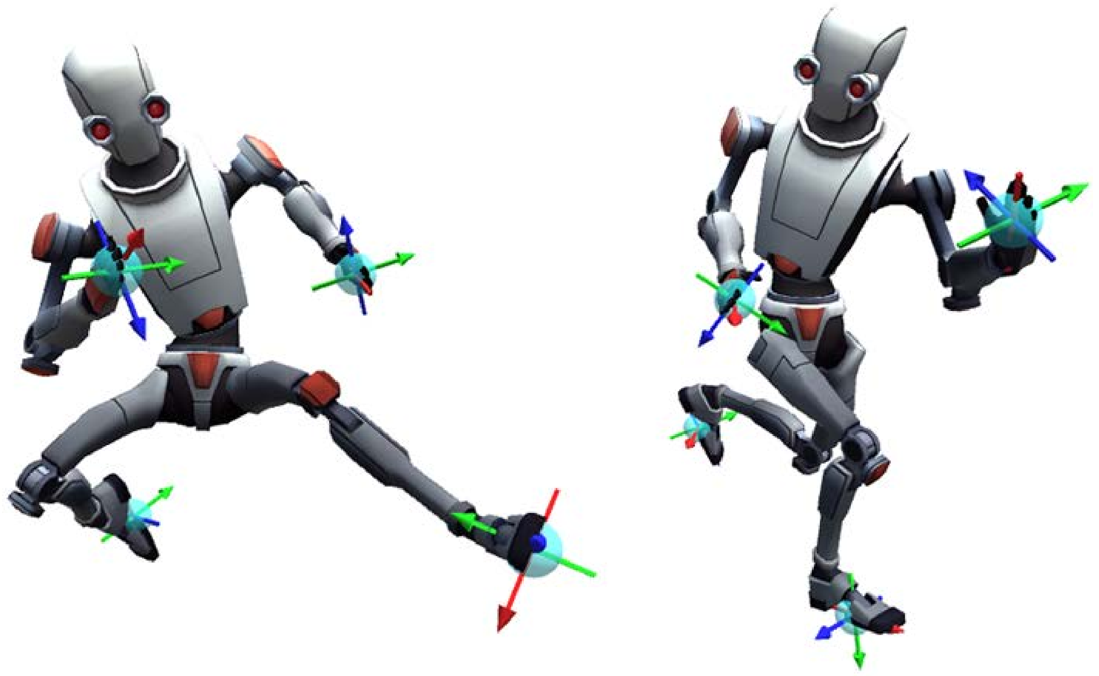


Figure 5.22: Example postures for generating keyframes for kicking (left) and running (right) animations on the Kyle humanoid. Four position and orientation objectives are specified for the desired hand and feet poses.

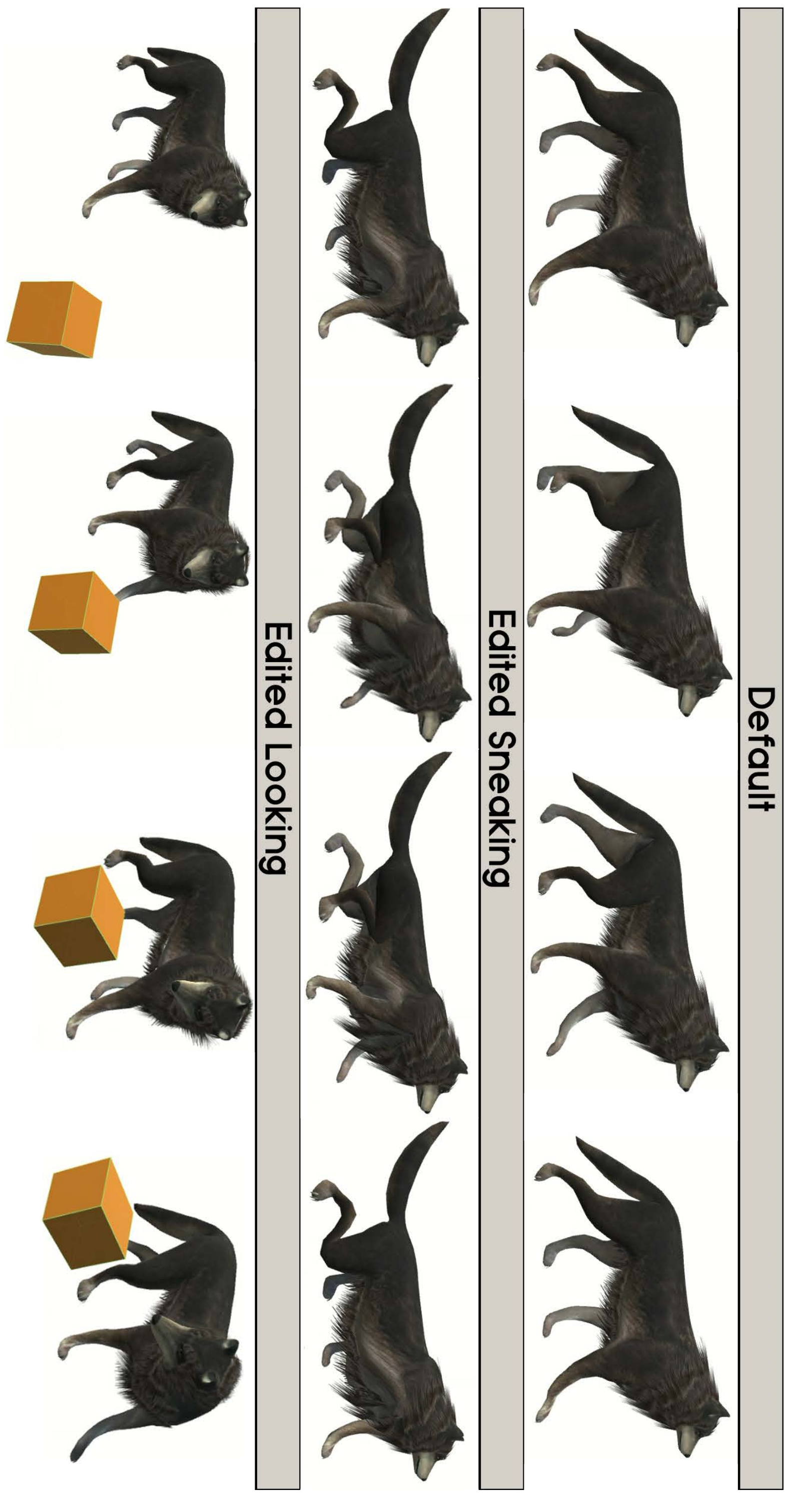


Figure 5.23: Real-time animation editing on the quadruped wolf character using the proposed algorithm. The top row shows an existing animation for a walking motion. The middle row shows the edited animation into a sneaking motion by defining two position objectives for the pelvis and shoulder at a lower height, as well as a further position goal at higher height for the tail. The bottom row shows the walking animation being manipulated to keep looking at the orange box target.

Furthermore, the algorithm was used for runtime animation post-processing on the Kyle humanoid. A common task for this is given by automatic foot placement in games, which aims to correct the poses of the feet in a way that they do not disappear in the ground when contacting the surface, but become properly aligned while traversing the terrain. As visualised in Fig. 5.24, the joints from the pelvis along both legs to the contact points located at the underside of the feet are post-processed using projection objectives. For each objective, a normal is pointing

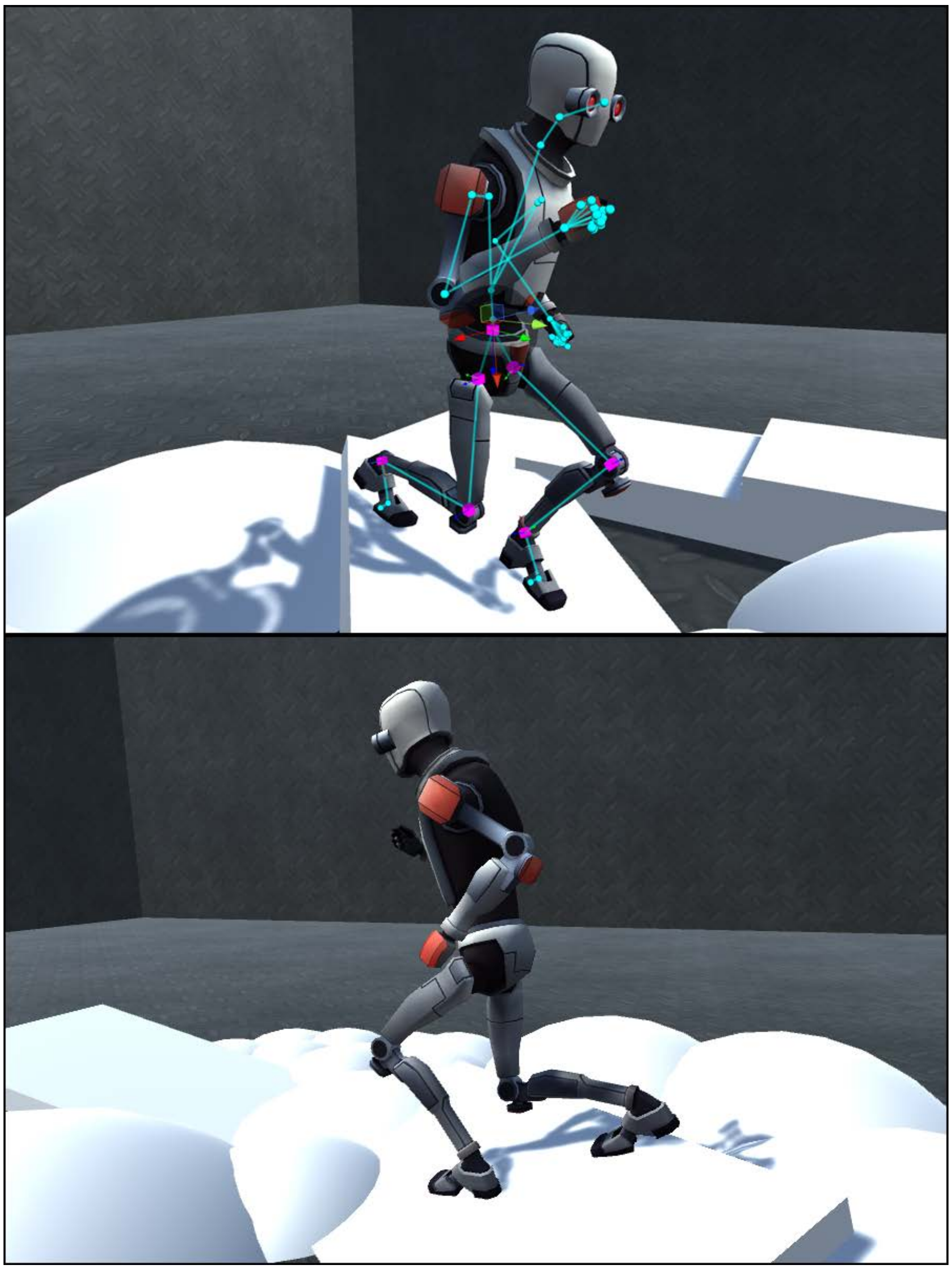


Figure 5.24: Setup for terrain-adaptive foot placement through animation post-processing as demonstrated in Fig. 5.25. The geometry of the Kyle humanoid from the pelvis to both feet contact points is post-processed by the algorithm.

downwards through each foot. Each raycast starts from a 0.5 m offset above the foot and ends exactly at the specified contact point. This aims to find a projected target pose in case the feet collide or disappear in the ground. If no surface intersection occured, the position and orientation values from the default running animation are used. Fig. 5.25 demonstrates that the algorithm can successfully be applied to solving such tasks while using a timeout of 1 ms per frame.

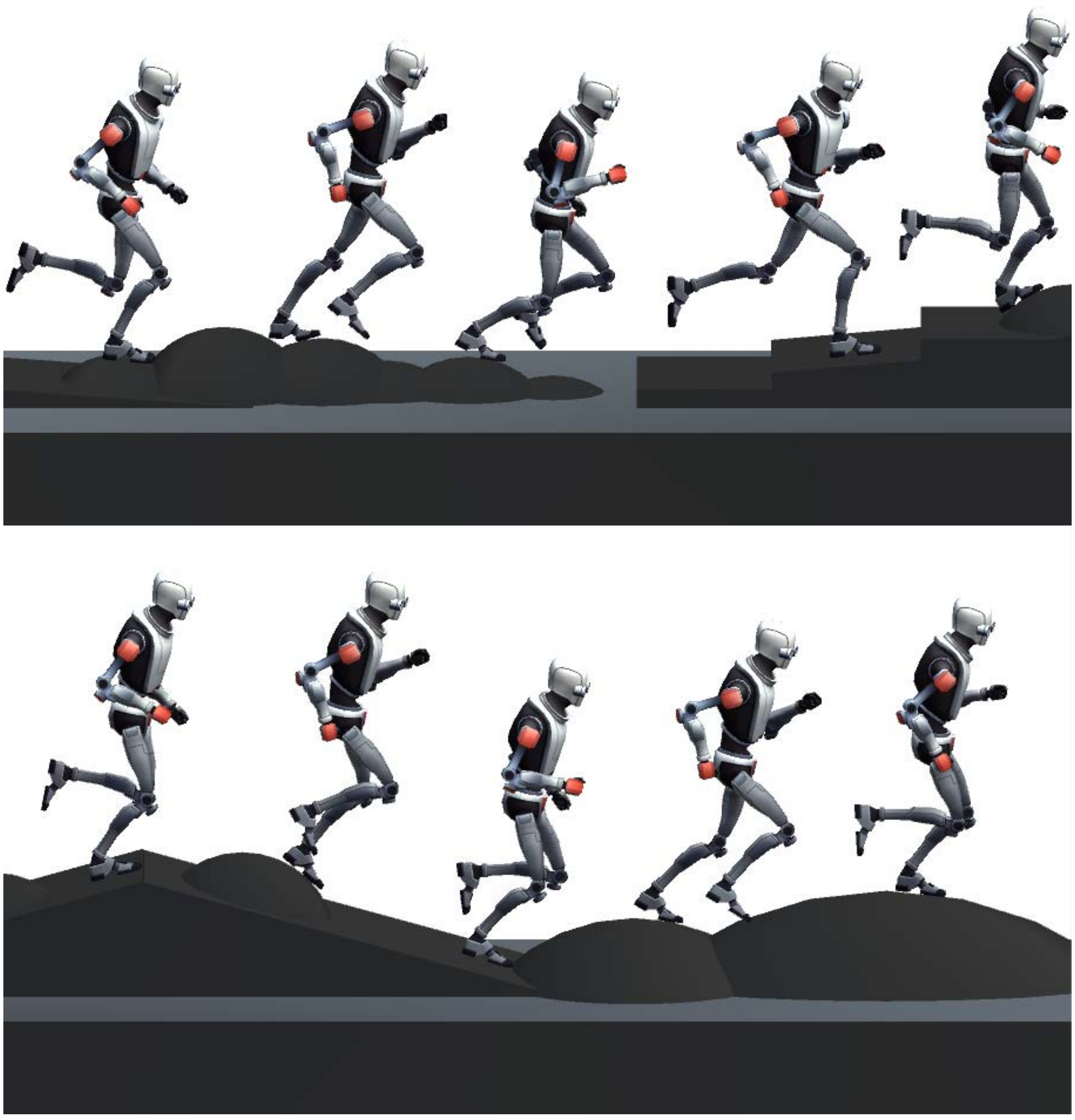


Figure 5.25: Runtime animation post-processing for terrain-adaptive foot placement. Given the running animation, the feet of the Kyle humanoid are automatically corrected by projection objectives to be appropriately placed on the ground in case they would disappear in the environment geometry.

5.11 Evolutionary Landscapes

The following experiments aim to validate the intended behaviour in search space exploration and exploitation by visualising the evolutionary optimisation. First, Fig. 5.26 shows the distribution of the fitness landscape after optimising a full-body posture on the NASA Valkyrie humanoid with pose targets for the hand and feet, as well as a viewing target for the head. The corresponding distribution of evolved postures with their fitness values over all individuals is visualised in Fig. 5.27 and Fig. 5.28. The population size was set to $\varphi = 50$ and $\kappa = 3$ elites were used. It can be observed that a high quality could be achieved especially for the elites, while a reasonable amount of diversity was maintained for the remaining population. Thus, the evolution did not entirely run into a single solution, but still covered a larger area in the search space after convergence. Furthermore, Fig. 5.30, Fig. 5.31 and Fig. 5.32 visualise the particle propagation of individuals on the human mannequin, Kyle, Shadow Dexterous Hand and NASA Valkyrie geometries. The plots demonstrate that multiple solutions can be tracked and found simultaneously, which is mainly controlled by the number of elites as shown in Fig. 5.29. More specificaly, using $\kappa = 1$ elite individuals makes the algorithm to behave very similar to a local search and less likely to escape from suboptimal extrema. This aligns with the parameter efficiency previously shown in Fig. 5.1. Using more elite individuals with $\kappa = 3$ or $\kappa = 6$ demonstrates a more multi-modal optimisation, and seems to provide a larger amount of individuals with better fitness values. In the fitness bars, the color of the particles represents their quality from low (red) to high (green). In general, it is demonstrated that the algorithm can robustly find solutions even for high-dimensional problems.

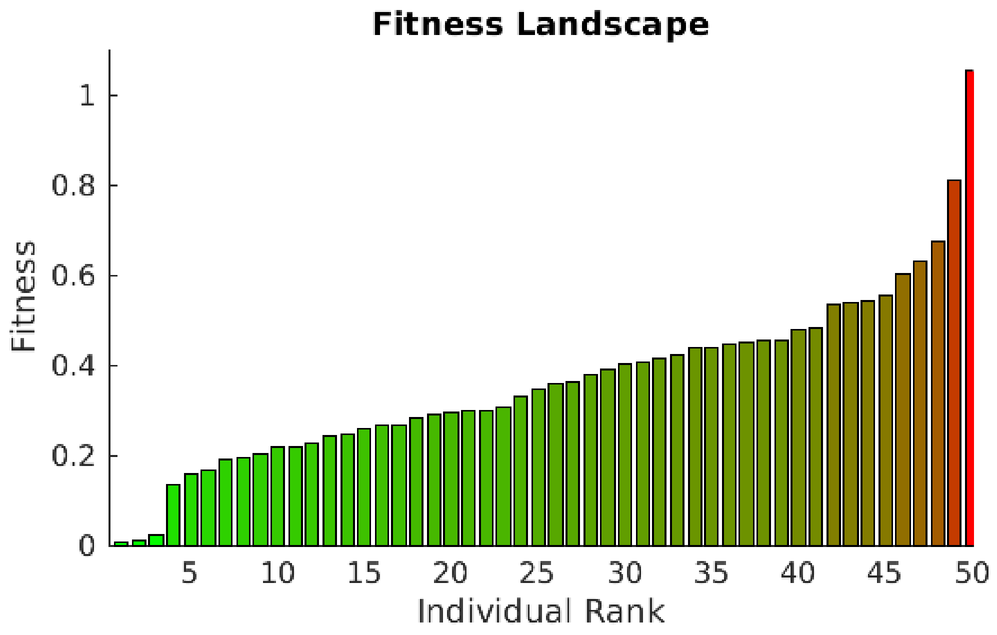


Figure 5.26: Fitness landscape for the experiment in Fig. 5.27 and Fig. 5.28.

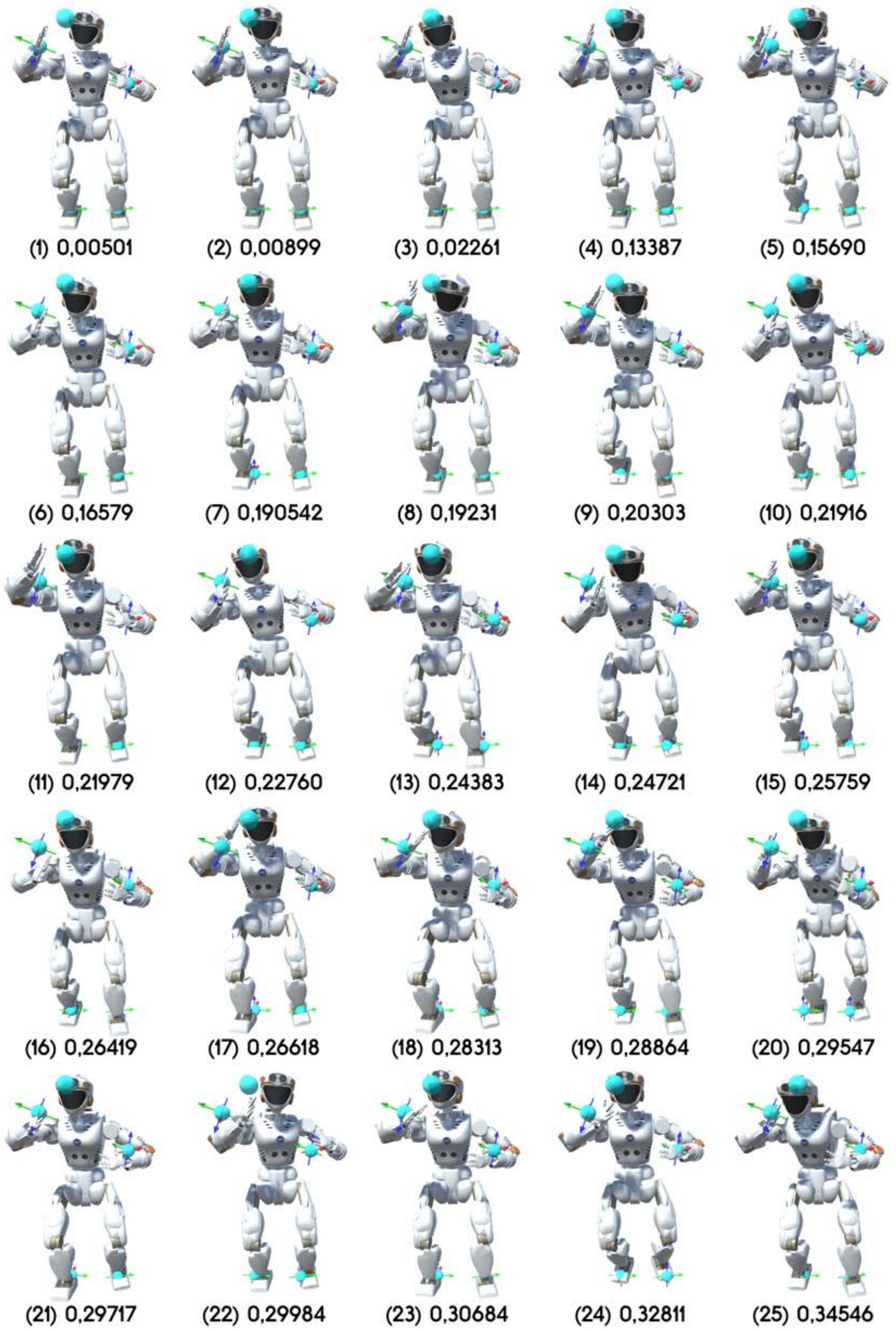


Figure 5.27: Evolved postures on the NASA Valkyrie of individuals $\Psi_{1,\ldots,25}$.

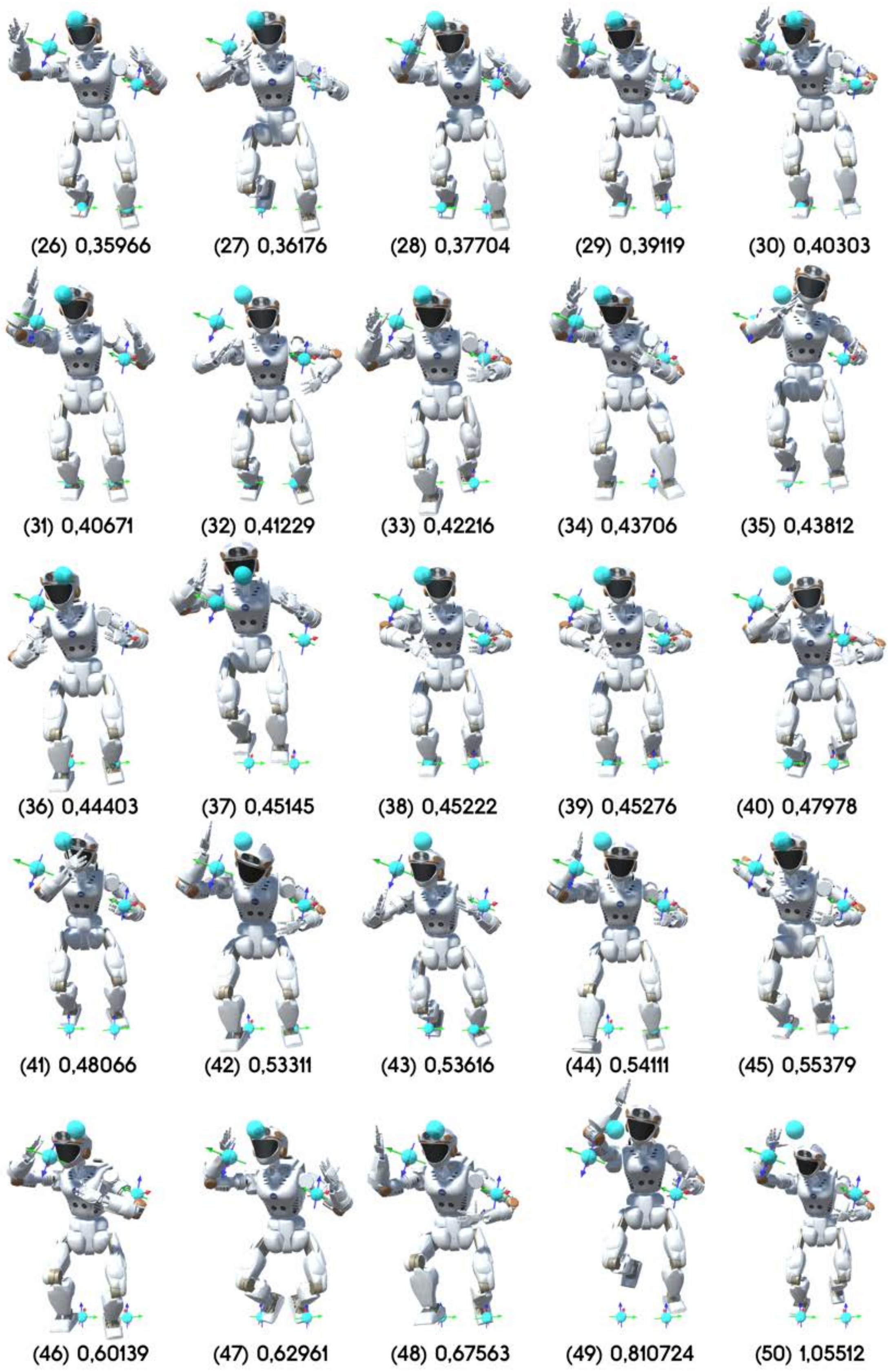


Figure 5.28: Evolved postures on the NASA Valkyrie of individuals $\Psi_{26,\ldots,50}$.

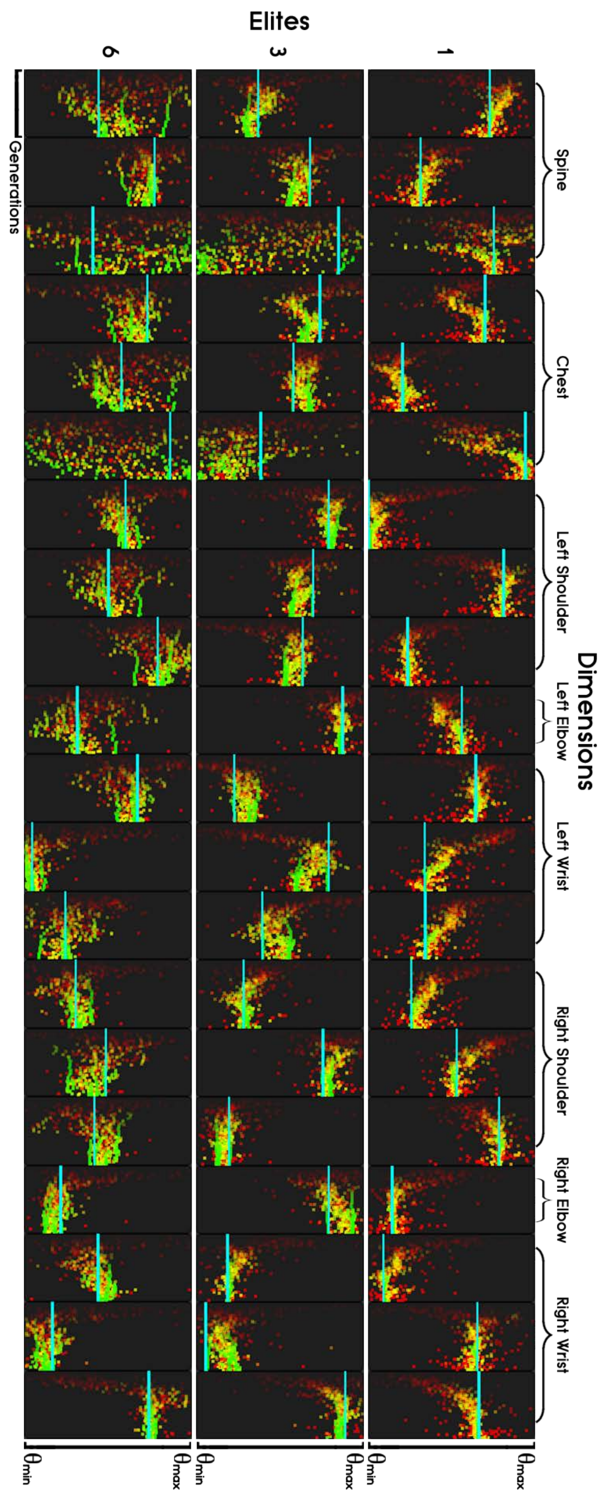


Figure 5.29: Evolutionary landscape for inverse kinematics on the upper body of the human mannequin. Each column represents one joint dimension with the full optimisation history until convergence from left to right. Each row shows the fitness distribution over all individuals with the color indicating the fitness. [97]

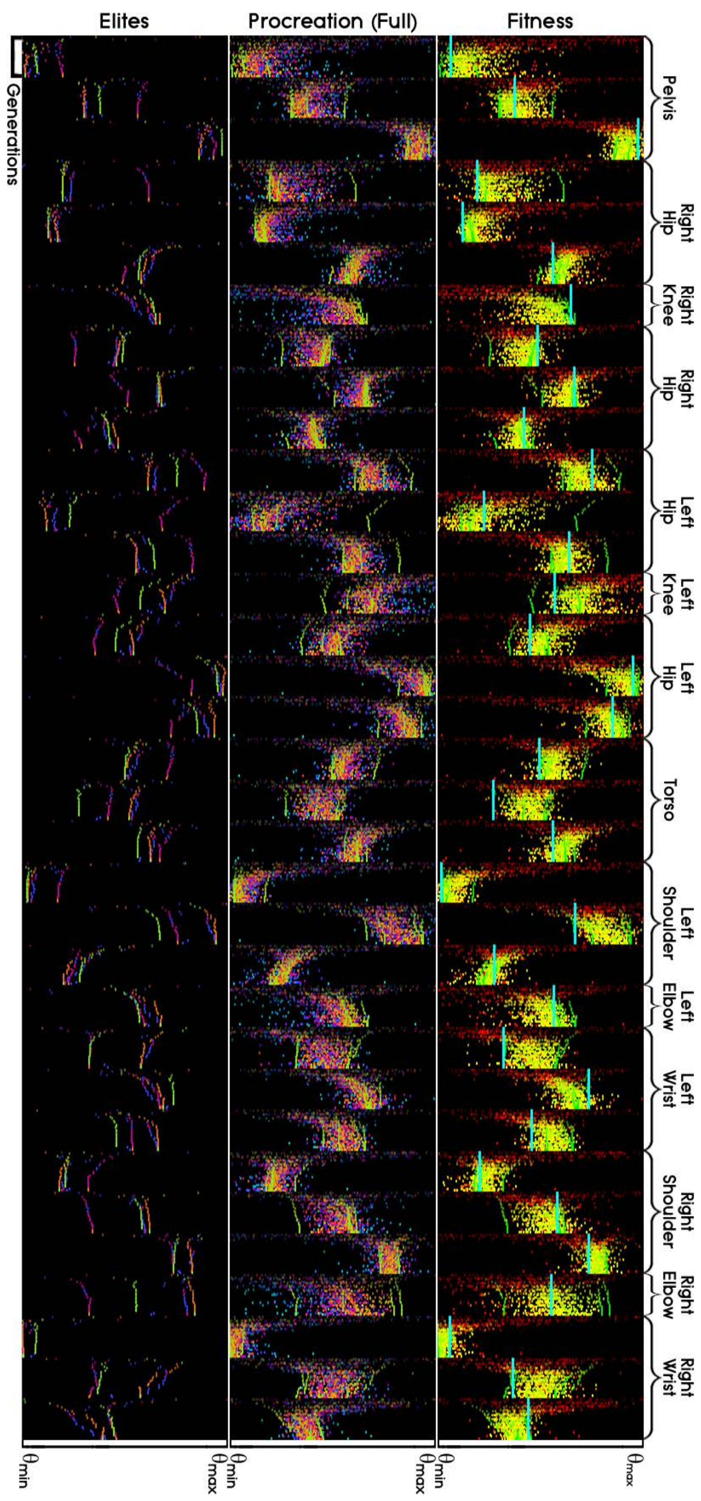


Figure 5.30: Evolutionary landscape for full-body inverse kinematics on the Kyle humanoid. Each column represents one joint dimension with the full optimisation history until convergence from left to right. The top row shows the fitness distribution over all individuals with the color indicating the fitness. The middle row shows the procreation from parent to offspring individuals within each generation. The bottom row exclusively visualises the history of elite individuals.

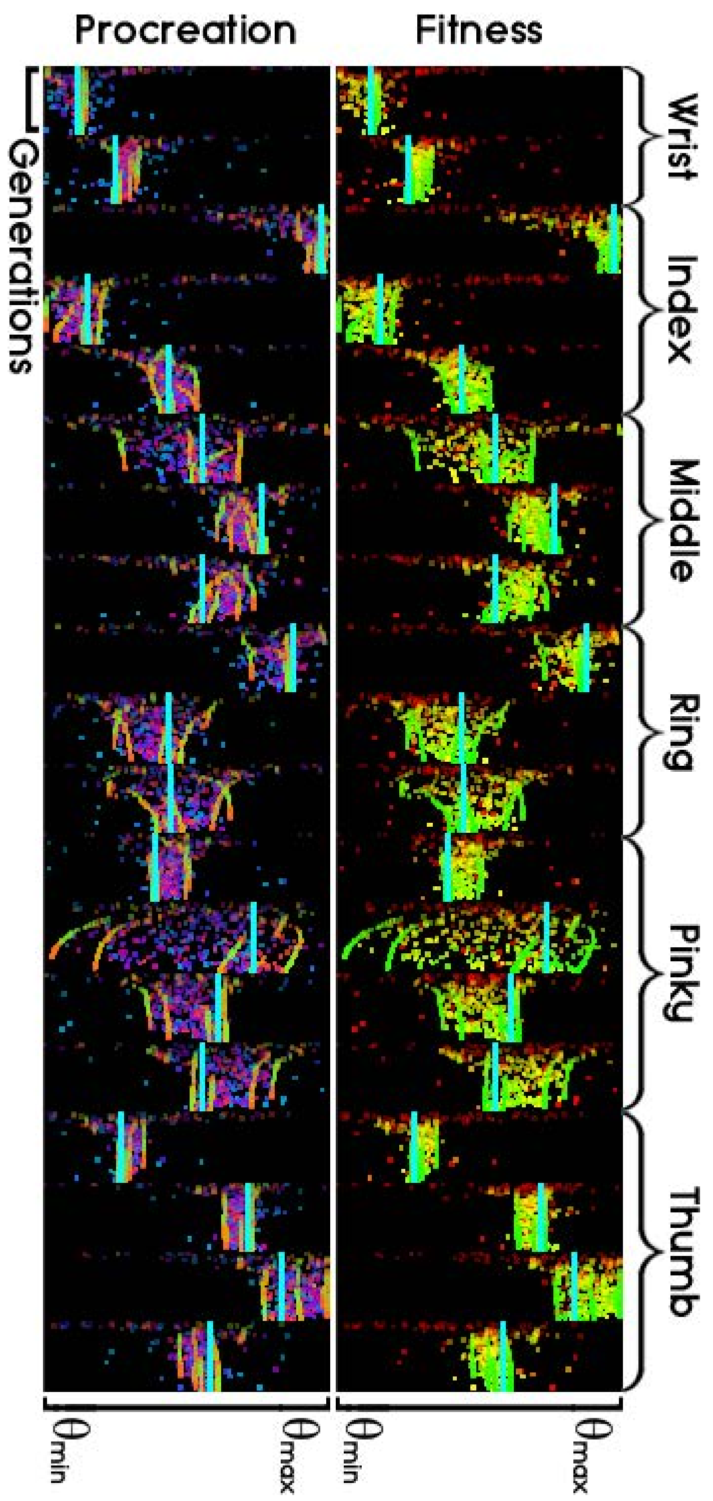


Figure 5.31: Evolutionary landscape for solving humanoid grasps on the anthropomorphic Shadow Dexterous Hand. Each column represents one joint dimension with the full optimisation history until convergence from left to right. The upper row shows the fitness distribution over all individuals with the color indicating the fitness. The lower row shows the procreation from parent to offspring individuals within each generation. [94]

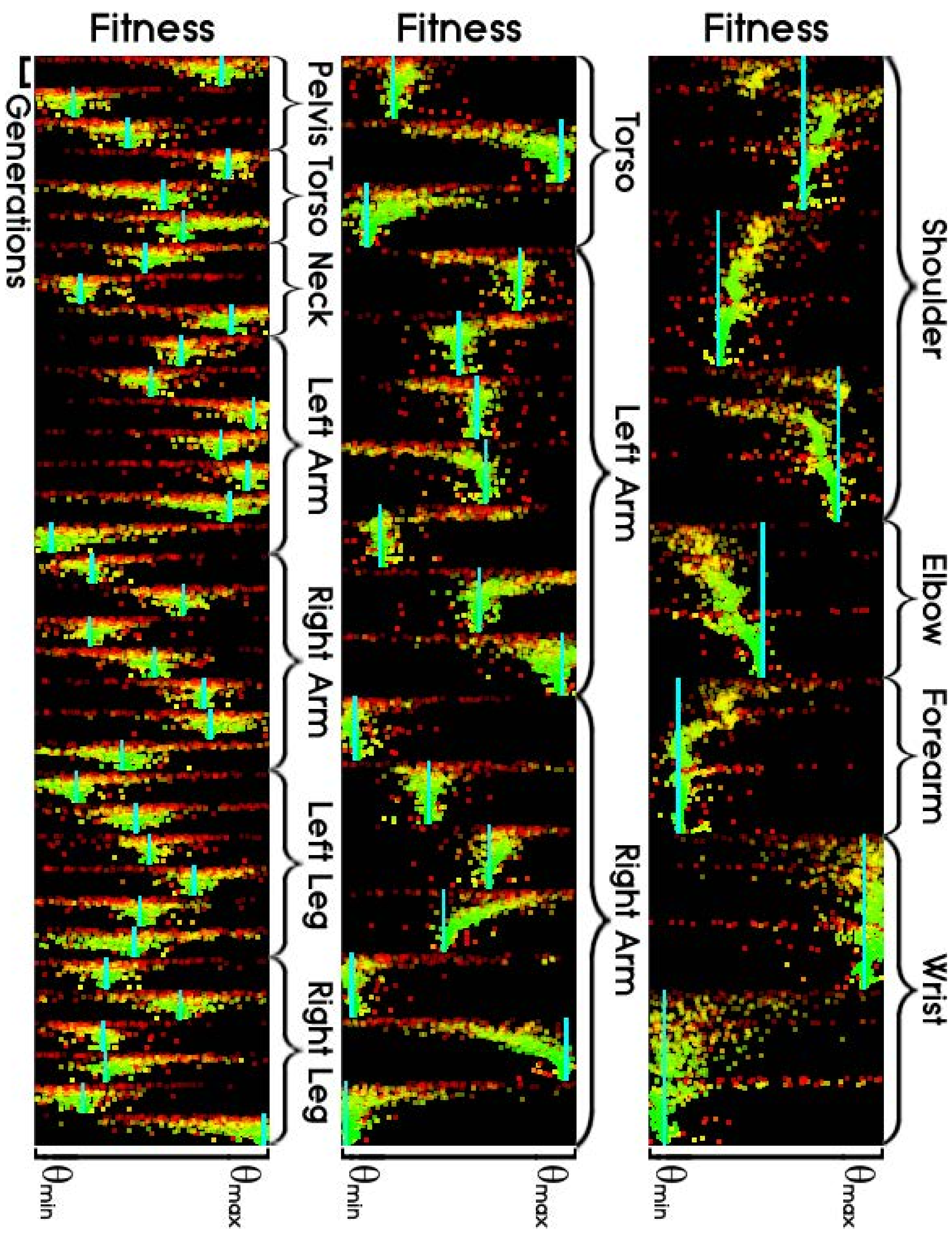


Figure 5.32: Evolutionary landscape for inverse kinematics on the NASA Valkyrie arm. Each column represents one joint dimension with the full optimisation history until convergence from left to right. Each row shows the fitness distribution over all individuals with the color indicating the fitness. [94]

5.12 Work Examples and Dissemination

This final experiment section includes some work that has been created through dissemination of the developed Bio IK algorithm in Unity3D. In particular, the examples suggest that it is especially well-suited for interactive control of highly-articulated geometries in virtual reality applications.

5.12.1 Holographer

The Holographer tool was developed for artful photography of posed characters in virtual reality [32]. The skeleton of a character is adjusted through full-body inverse kinematics by interactively moving its bones using the HTC Vive controllers. In particular, pose targets are not only specified for the end effector, but also dynamically activated for intermediate segments along the kinematic chains. In this application, is was important being able to create arbitrary inverse kinematics setups for different character geometries, and to allow joint limit specifications in order to obtain realistic postures at any time. First, the models are placed in the scene and are located inside the camera port (1). The bones of each character are then manipulated by user control until a desired posture is obtained (2). After some final adjustments of the character positions in the camera port (3), the scene is rendered into a photo (4). This workflow in Fig. 5.33 was demonstrated to be an elegant method for artists to easily pose their characters in virtual reality. As Bio IK can handle generic geometries, it could directly be used for any desired setup. Furthermore, it can similarly be applied to animation keyframe generation.

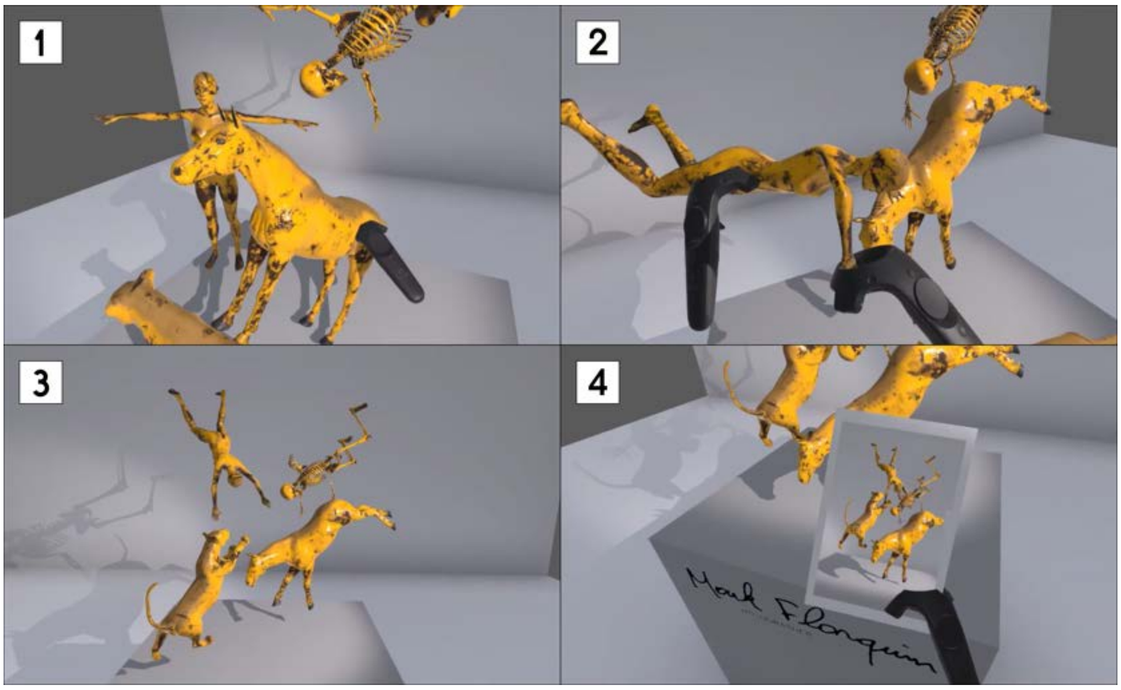


Figure 5.33: Virtual photography by posing characters in Holographer. [32]

5.12.2 Robot Simulator

The Bio IK asset was further used for the development of a virtual reality robot simulator [98]. In this, the user can interactively use different industrial robots, apply joint configuration adjustments, and generate trajectories and motion sequences for manipulation tasks. The control input is given by the HTC Vive controllers. In Fig. 5.34, the workflow for an interactive trajectory generation is visualised using the Fanuc M-10iA serial manipulator. A welder is attached to the robot end effector. The robot is first moved into a suitable start configuration by the user (1), defining the first trajectory pose. The user then interactively generates a sequence of poses on the surface of the object to be manipulated (2). Those are then solved by the inverse kinematics algorithm to move the welder along the defined trajectory lines by the target poses (3,4). In addition to the previous experiment in Fig. 5.19, this simulation tool demonstrates the applicability of the algorithm for tasks in human-robot interaction. Furthermore, it was also successfully applied to solving inverse kinematics on parallel robots as visualised in (5,6,7).

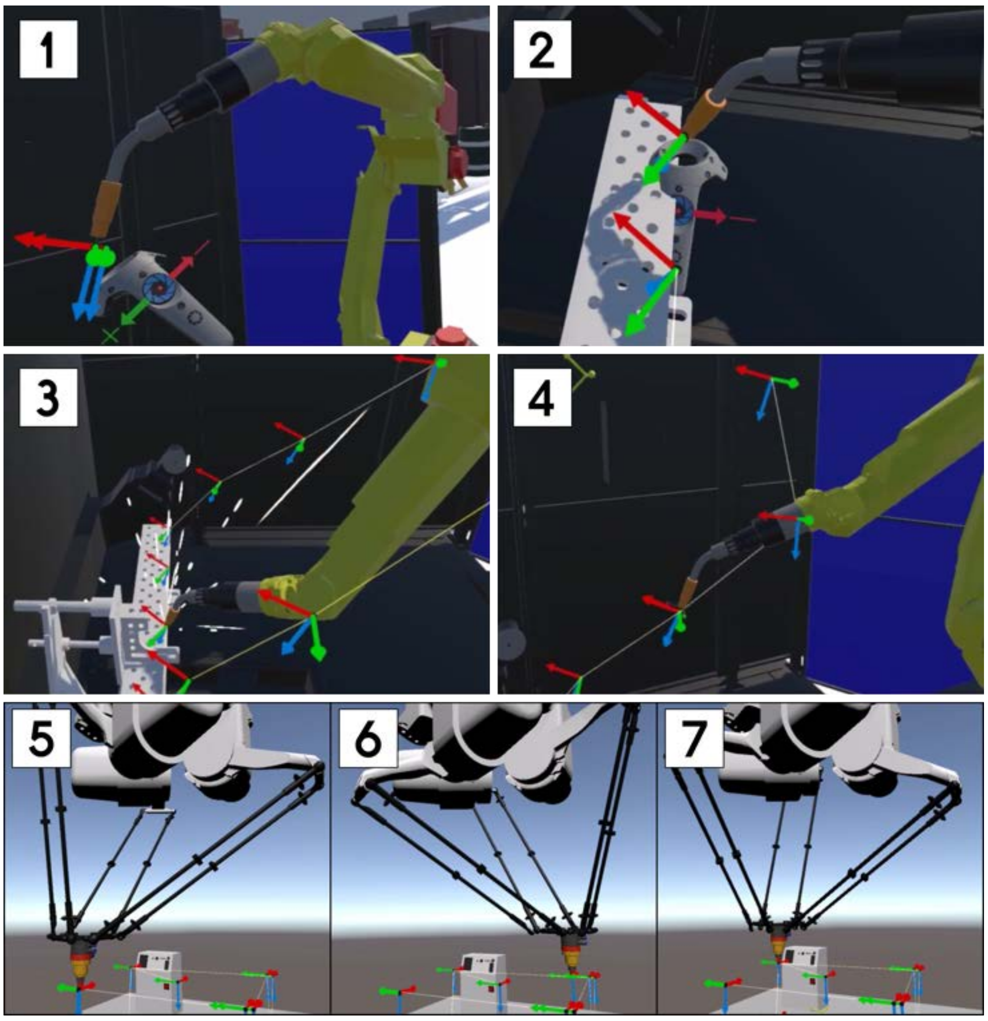


Figure 5.34: Interactive robot simulator in virtual reality. [98]

5.12.3 Guitar Teacher

Another software that was built using Bio IK is an interactive tool for teaching guitar playing [20]. The finger grasps for the chords can be viewed from different camera perspectives by the user. Every finger is assigned a particular pose target on the guitar fret. However, as this offers a variety of grasp solutions for inverse kinematics, additional position objectives were used for intermediate finger segments in order to shape optimal and naturally-looking grasps. In particular, some fingers should rather be bended while others are desired to be stretched depending on the grasp types. Concurrently, slight rolling for the finger tips was preferred while adjusting them, which could be achieved by a proper weighting between position and orientation objectives. This is a particularly challenging task for such applications, and for which related methods were found to be rather limited.

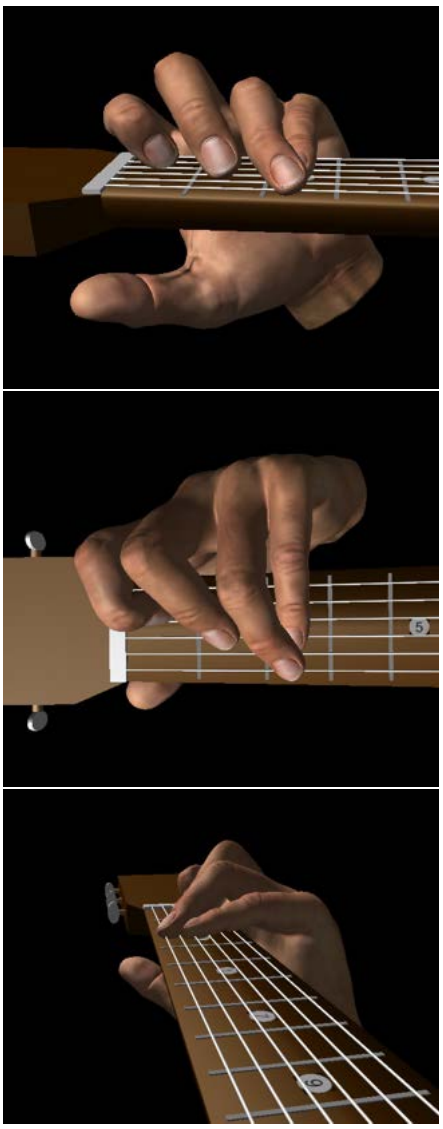


Figure 5.35: Guitar teaching with finger grasp visualisations. [20]

5.12.4 Posture Training in Augmented Reality

The algorithm was further used for scientific research in human-computer interaction [45]. The aim was to evaluate the human perception of visuo-haptic feedback in augmented reality by pushing the arms of a user. The experimental setup is visualised in Fig. 5.36, and which uses the Stylo-Handifact spatial user interface to generate the psychophysical stimuli. Stylo is a haptic device which is attached to the forearm, and Handifact refers to the visualisation of the virtual hand in augmented reality. In particular, the Handifact is given by a rigged and textured model of the human hand. The Bio IK asset was then used for solving postures of the hand for visualising different pressure levels of pushing, as shown in the top row. The middle row shows the generated view of the user in augmented reality when wearing a head-mounted display. The bottom row visualises the method used for an interactive Tai-Chi training system to correct the motion of the practitioner. For these tasks, realistic configurations based on the joint limits had to be generated robustly in real-time. This was important to provide an immersive perception for the user when getting pushed by the virtual hand.

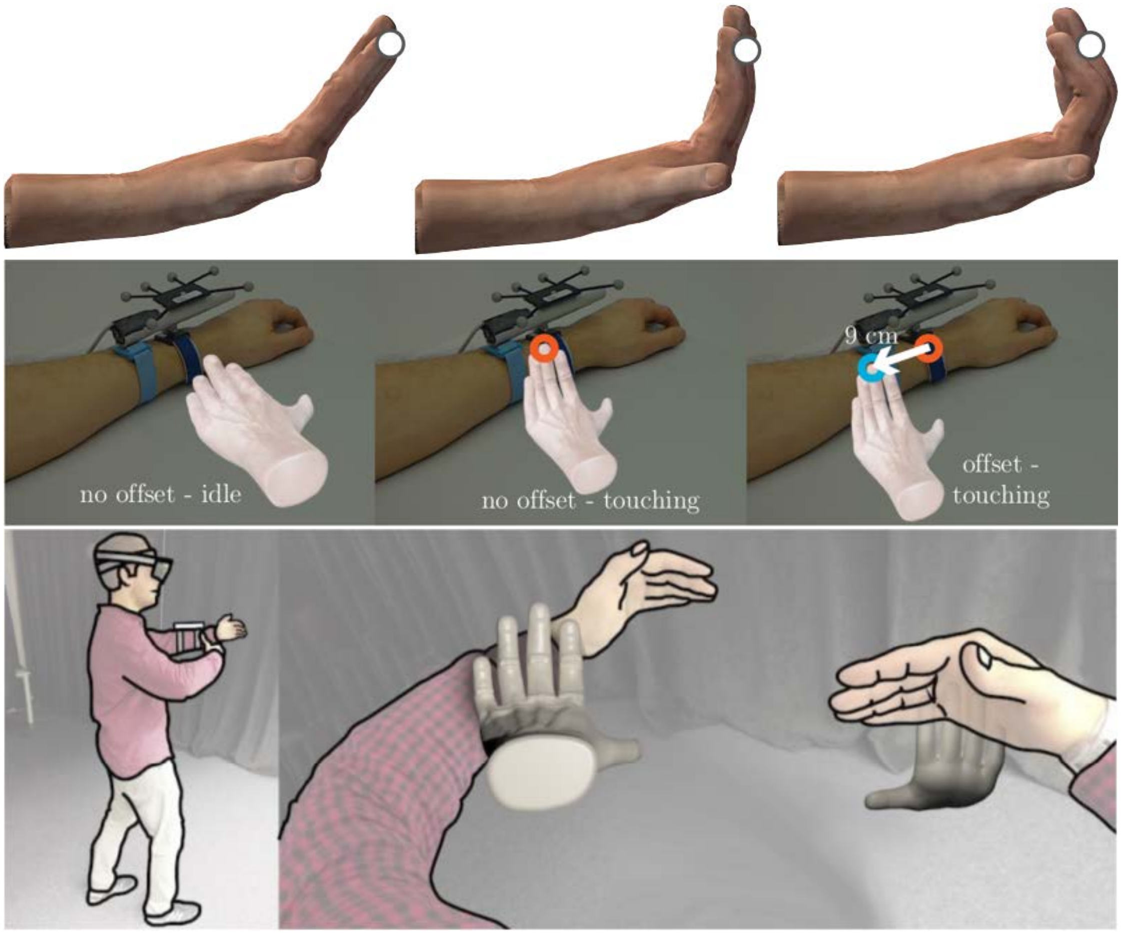


Figure 5.36: Interactive posture training system in augmented reality using the Stylo-Handifact spatial user interface for visuo-haptic feedback. [45]

Chapter 6

Conclusion

6.1 Summary

This thesis proposed Bio IK as a novel memetic evolutionary algorithm for solving inverse kinematics with multiple objectives on fully-constrained generic geometries. Given a hierarchical kinematic structure, Cartesian position and orientation targets can be solved concurrently for multiple segments along the kinematic chains. It further extends traditional inverse kinematics by the ability to specify custom cost functions by additional task-specific objectives. Those can be used for solving directional goals to look at particular objects, real-time collision avoidance with the own geometry or the environment, functional joint dependencies for anthropomorphic grasps, or to prioritise particular posture configurations not only in Cartesian but also in joint space. Further objectives can also be created for individual needs of the user given a twice-differentiable formulation. The research was motivated by the issue that methods for such tasks did neither seem readily available nor extensively studied, but are essentially required for various applications in robotics and animation. In particular, the literature review of related popular methods of both domains revealed that their benefits and limitations in solving inverse kinematics seem to behave mutually exclusive. Therefore, the aim was to combine the characteristic strengths of different optimisation methodologies — hybridising biologically-inspired evolutionary and swarm optimisation with the gradient-based L-BFGS-B method. This allows the developed algorithm to perform a multi-modal global and local search at the same time while robustly avoiding suboptimal extrema and singularity issues, as well as providing fast convergence and scalability for high DoF geometries. In addition, the algorithm was designed to perform an adaptive optimisation, and only requires two parameters for the population size and number of elites to be defined. The experiments on several industrial manipulators as well as on humanoid and anthropomorphic robots and virtual characters demonstrated the algorithm to be suitable for different tasks in robotics and animation. When solving a larger set of randomly-sampled reachable pose configurations, similar speed of convergence as for popular gradient-based methods could be achieved, but with significantly higher success rates on all tested geometries. The algorithm was further successfully applied to solving full-body postures and kinematic motion on the NASA Valkyrie robot, collision-free trajectory generation, as well as for reconstructing real-human grasps and dexterous manipulation with the Shadow Dexterous Hand. Especially the abilities to integrate collision avoidance for generic inverse kinematics as well as to specify intermediate and individually weighted objectives along the kinematic chains for controlling manipulation tasks were found to be particular strengths of the proposed optimisation framework. The method was also demonstrated to create realistic postures for teleoperation in human-robot interaction by reconstructing the motion of a real-human operator. Extending the field of application to character animation, the algorithm could be applied to full-body animation editing of a highly-articulated quadruped character, as well as to animation post-processing for terrain-adaptive foot placement of a humanoid character. All experiments were done in real-time without requiring any offline precomputations, and no model-specific kinematic equations were hard-coded to support the optimisation. The visualisations of evolutionary landscapes then validated the intended behaviour of the algorithmic design in performing a multi-modal, robust and adaptive optimisation. The algorithm was made available as implementations for Unity3D (C#) and ROS (C++). It has so far gained popularity and acknowledgement in both domains by contributing to research in robotics, human-computer interaction, animation and game development. Finally, this thesis demonstrated and aims to motivate memetic evolution as a competitive, efficient and flexible methodology for solving complex inverse kinematics postures.

6.2 Novelty and Contributions

The following most relevant aspects of the proposed research were found to be novel and contributing to the scientific community:

- A novel memetic algorithm for generic multi-objective inverse kinematics that can be used for solving postures on highly-articulated kinematic geometries.
- Applicability for challenging applications in robotics, such as grasping with dexterous anthropomorphic hands and full-body motion of humanoid robots, as well as for character animation editing and post-processing.
- Competitive against popular state-of-the-art methods, flexible and extendable design of cost functions and objectives, and integration of real-time collision avoidance for generic inverse kinematics.
- Available implementations for Unity3D and ROS in order to support research and development in robotics and animation.
- Algorithmic hybridisation of evolutionary, swarm and gradient-based optimisation using the L-BFGS-B method, which acts as a general optimisation methodology that is not restricted to inverse kinematics, but can be applied to bounded twice-differentiable continuous optimisation problems.

6.3 Limitations and Challenges

Generally, the algorithm was found being flexibly applicable for different tasks by using suitable combinations of objectives. Even if some functionalities were missing to fulfil a task, new objectives could be defined quite straightforwardly. However, main difficulties were observed in finding proper weightings to combine the different objectives, and which is not generally clear how to solve. In particular, setting higher weights for collision avoidance or intermediate position or orientation goals along the kinematic chains can result in not accurately solving end effector targets anymore, and which are usually of higher importance. Although this behaviour is correct in terms of multi-objective optimisation, it might not always yield the desired results for the user. Furthermore, due to the probabilistic optimisation, it is possible that sudden changes between solutions might occur. Although this was only observed in very rare cases or for near-singular configurations, and could be eliminated using a displacement objective to penalise large changes, this issue should be considered when applied to real-world robotics applications. It was also observed that the algorithm should be given 1–2 ms optimisation time for animation editing or post-processing tasks whereas 16 ms (60 Hz) represents the usual time limit in games in order to maintain real-time frame rates. This suggests that the algorithm is predominantly suited for fine control of one or some few main characters rather than for controlling multiple but less important characters at the same time, and for which computationally faster but less powerful methods are typically sufficient. Nevertheless, there should be very few cases in which solving fine control of motion on multiple characters concurrently is actually required.

6.4 Future Work

So far, the algorithm has been predominantly applied to pure kinematic purposes, but not yet considering the dynamics of the system. Although first experiments for robot balancing have been successfully conducted in [87] by designing center-of-gravity objectives, extending the method to inverse dynamics outlines a very promising goal for future work. Another research goal would be to design a functionality or several heuristics for automatically chosing appropriate weights to combine the objectives. Further investigations might include solving more challenging tasks in dexterous manipulation, as well as deploying the method for procedurally-generated animation and close-character interactions. The method can also be utilised for data augmentation for deep learning by generating manifolds of different motions, trajectories or animations. Finally, the evolutionary optimisation might be combined with a pretrained neural network controller which generates an initial solution for inverse kinematics. This solution can then be used as a bias individual for the evolution, which might not only achieve a faster convergence, but perhaps also offer a more predictive solution manifold for a given input.

Bibliography

[1] Omar Alejandro Aguilar and Joel Carlos Huegel. Inverse kinematics solution for robotic manipulators using a cuda-based parallel genetic algorithm. *Springer, Lecture Notes in Computer Science*, 7094:490–503, 2011.

[2] A. Almusawi, L. Dülger, and S. Kapucu. *A New Artificial Neural Network Approach in Solving Inverse Kinematics of Robotic Arm.* Computational Intelligence and Neuroscience, 2016.

[3] Andreas Aristidou and Joan Lasenby. *Inverse Kinematics: a review of existing techniques and introduction of a new fast iterative solver.* University of Cambridge, 2009.

[4] Andreas Aristidou and Joan Lasenby. Fabrik: A fast, iterative solver for the inverse kinematics problem. *Graphical Models*, 73:243–260, 2011.

[5] Andreas Aristidou and Joan Lasenby. Real-time marker prediction and cor estimation in optical motion capture. *The Visual Computer*, 29:7–26, 2013.

[6] A. Balestrino, G. De Maria, and L. Sciavicco. Robust control of robotic manipulators. *IFAC*, 17:2435–2440, 1984.

[7] Patrick Beeson and Barrett Ames. TRAC-IK: An open-source library for improved solving of generic inverse kinematics. In *Proceedings of the IEEE RAS International Conference on Humanoid Robotics*, Seoul, Korea, November 2015.

[8] Alexandre Bernardino, Marco Henriques, Norman Hendrich, and Jianwei Zhang. Precision grasp synergies for dexterous robotic hands. In *Proceedings of the IEEE International Conference on Robotics and Biomimetics*, pages 62–67, 2013.

[9] Bio IK for ROS. `https://github.com/TAMS-Group/bio_ik`. Accessed: 28-05-2018.

[10] Bio IK for Unity3D. `https://www.assetstore.unity3d.com/en/#!/content/67819`. Accessed: 28-05-2018.

[11] M. R. Bonyadi and Z. Michalewicz. Particle swarm optimization for single objective continuous space problems: a review. *Evolutionary Computation*, 25:1–54, 2017.

[12] Charles G. Broyden. The convergence of a class of double-rank minimization algorithms. *Journal of the Institute of Mathematics and Its Applications*, 6:76–90, 1970.

[13] S. R. Buss and J. S. Kim. Selectively damped least squares for inverse kinematics. *Journal of Graphics Tools*, 10:37–49, 2005.

[14] Samuel R. Buss. *Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Squares methods.* Survey, University of California, 2004.

[15] Richard Byrd, Peihuang Lu, and Jorge Nocedal. A limited memory algorithm for bound constrained optimization. *SIAM Scientific and Statistical Computing*, 16:1190–1208, 1995.

[16] Ze Fan Cai, Dao Ping Huang, and Yi Qi Liu. Inverse kinematics of multi-joint robot based on particle swarm optimization algorithm. *Journal of Automation and Control Engineering*, 4:305–308, 2016.

[17] Adrian Canutescu and Roland Dunbrack. Cyclic coordinate descent: A robotics algorithm for protein loop closure. *Protein Science*, 12:963–972, 2003.

[18] Thomas Collins and Wei-Min Shen. *PASO: An Integrated, Scalable PSO-based Optimization Framework for Hyper-Redundant Manipulator Path Planning and Inverse Kinematics.* Information Sciences Institute Technical Report, 2016.

[19] Richard Dawkins. *The Selfish Gene.* Oxford University Press, 1989.

[20] Bryan Dempsey. Guitar teaching tool, 2017.

[21] Rosen Diankov. *Automated Construction of Robotic Manipulation Programs.* Doctoral Thesis, The Robotics Institute, Carnegie Mellon University, 2010.

[22] Rosen Diankov, Kenji Sato, Hiroaki Yaguchi, Kei Okada, and Masayuki Inaba. *Manipulation Planning for the JSK Kitchen Assistant Robot Using OpenRAVE.* University of Tokyo, 2011.

[23] P. S. Donelan. Kinematic singularities of robot manipulators. *Advances in Robot Manipulator, In-Tech*, pages 401–416, 2010.

[24] Fletcher Dunn and Ian Parberry. *3D Math Primer For Graphics And Game Development.* Jones & Bartlett Learning, 2002.

[25] B. Durmus, H. Temurtas, and A. Gün. *An Inverse Kinematics Solution using Particle Swarm Optimization.* 6th International Advanced Technologies Symposium, 2011.

[26] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing.* Springer, Natural Computing Series, 2003.

[27] Unreal Engine. `https://www.unrealengine.com`. Accessed: 28-05-2018.

[28] Siavash Farzan and G. N. DeSouza. A parallel evolutionary solution for the inverse kinematics of generic robotic manipulators. In *Proceedings of the IEEE Congress on Evolutionary Computation*, 2014.

[29] Yin Feng, Wang Yao-nan, and Yang Yi-min. Inverse kinematics solution for robot manipulator based on neural network under joint subspace. *International Journal of Computers Communications and Control*, 7:459–472, 2012.

[30] Roger Fletcher. A new approach to variable metric algorithms. *Computer Journal*, 13:317–322, 1970.

[31] Dario Floreano and Claudio Mattiussi. *Bio-Inspired Artificial Intelligence – Theories, Methods, and Technologies.* MIT Press, 2008.

[32] Mark Florquin. Holographer. `https://www.markflorquin.be`, 2017. Accessed: 28-05-2018.

[33] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution.* Wiley, 1966.

[34] ROS Framework. `http://www.ros.org`. Accessed: 28-05-2018.

[35] Donald Goldfarb. A family of variable metric updates derived by variational means. *Mathematics of Computation*, 24:23–26, 1970.

[36] Keith Grochow, Steven L. Martin, Aaron Hertzmann, and Zoran Popovi. Style-based inverse kinematics. *ACM Transactions on Graphics*, 23:522–531, 2004.

[37] H. Hanafusa and Y. Nakamura. Inverse kinematics solutions with singularity robustness for robot manipulator control. *Dynamic Systems, Measurement, and Control*, 108:163–171, 1986.

[38] Shadow Dexterous Hand. `https://www.shadowrobot.com/products/dexterous-hand/`. Accessed: 28-05-2018.

[39] Shadow Dexterous Hand. `https://github.com/shadow-robot/sr_common`. Accessed: 28-05-2018.

[40] John H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, 1975.

[41] Hsu-Chih Huang, Chien-Po Chen, and Pei-Ru Wang. Particle swarm optimization for solving the inverse kinematics of 7-dof robotic manipulators. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, 2012.

[42] Fanuc M10 iA Model. `https://github.com/ros-industrial/fanuc/tree/indigo-devel/fanuc_m10ia_support`. Accessed: 28-05-2018.

[43] Panchanand Jha and BB Biswal. A neural network approach for inverse kinematic of a scara manipulator. *International Journal of Robotics and Automation*, 3:52–61, 2014.

[44] M. Kallmann. Analytical inverse kinematics with body posture control. *Computer Animation and Virtual Worlds*, 19:79–91, 2008.

[45] Nicholas Katzakis, Jonathan Tong, Oscar Ariza, Gudrun Klinker, Brigitte Roeder, and Frank Steinicke. Stylo and handifact: Modulating haptic perception through visualizations for posture training in augmented reality. In *Proceedings of the ACM Symposium on Spatial User Interaction*, 2017.

[46] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948, 1995.

[47] J. Kennedy and R. Eberhart. A modified particle swarm optimizer. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 69–73, 1998.

[48] Ben Kenwright. Real-time character inverse kinematics using the gauss-seidel iterative approximation method. In *Proceedings of the International Conference on Creative Content Technologies*, pages 63–68, 2012.

[49] Ben Kenwright. Inverse kinematic solutions for articulated characters using massively parallel architectures and differential evolutionary algorithms. In *Proceedings of the Workshop on Virtual Reality Interaction and Physical Simulation*, 2017.

[50] Orocos Kinematics and Dynamics. `http://www.orocos.org`. Accessed: 28-05-2018.

[51] Donald E. Knuth. *The Art of Computer Programming, Volumes 1-4, 3rd Edition.* Addison Wesley, 2011.

[52] Nikos Kofinas, Emmanouil Orfanoudakis, and Michail G. Lagoudakis. Complete analytical inverse kinematics for nao. In *Proceedings of the International Conference on Autonomous Robot Systems*, 2013.

[53] Rasit Köker. A genetic algorithm approach to a neural-network-based inverse kinematics solution of robotic manipulators based on error minimization. *Information Sciences, New Trends in Ambient Intelligence and Bio-inspired Systems*, 222:528–543, 2013.

[54] R. Konietschke and G. Hirzinger. *Inverse Kinematics with Closed Form Solutions for Highly Redundant Robotic Systems.* IEEE International Conference on Robotics and Automation, 2009.

[55] Dennis Krupke, Sebastian Starke, Lasse Einig, Frank Steinicke, and Jianwei Zhang. Prototyping of immersive hri scenarios. In *Proceedings of the International Conference on Climbing and Walking Robots*, 2017.

[56] Serdar Kucuk and Zafer Bingul. *Industrial Robotics: Theory, Modelling and Control.* pro literatur, 2007.

[57] Alastair Lansley. Caliko: An inverse kinematics software library implementation of the fabrik algorithm. *Journal of Open Research Software*, 4, 2016.

[58] Jeffery J. Leader. *Numerical Analysis and Scientific Computation, 1st Edition.* Pearson, 2004.

[59] Eric Lengyel. *Mathematics for 3D Game Programming and Computer Graphics, 3rd Edition.* Cengage Learning PTR, 2011.

[60] Adrian Lewis and Michael Overton. Nonsmooth optimization via quasi-newton methods. *Mathematical Programming*, 141:135–163, 2013.

[61] Autodesk Maya. `https://knowledge.autodesk.com/support/maya`. Accessed: 28-05-2018.

[62] J. M. McCarthy and G. S. Soh. *Geometric Design of Linkages, 2nd Edition.* Springer, 2010.

[63] Michael Meredith and Steve Maddock. *Real-Time Inverse Kinematics: The Return of the Jacobian.* University of Sheffield, 2004.

[64] Baxter Model. `https://github.com/RethinkRobotics/baxter_common`. Accessed: 28-05-2018.

[65] Human Mannequin Model. `https://www.assetstore.unity3d.com/en/#!/content/9489`. Accessed: 28-05-2018.

[66] KR120 R2500 PRO Model. `https://github.com/ros-industrial/kuka_experimental/tree/indigo-devel/kuka_kr120_support`. Accessed: 28-05-2018.

[67] Kyle Model. `https://www.assetstore.unity3d.com/en/#!/content/4696`. Accessed: 28-05-2018.

[68] LBR IIWA 14 R820 Model. `https://github.com/ros-industrial/kuka_experimental/tree/indigo-devel/kuka_lbr_iiwa_support`. Accessed: 28-05-2018.

[69] PR2 Model. `http://wiki.ros.org/pr2_description`. Accessed: 28-05-2018.

[70] UR5 Model. `http://wiki.ros.org/ur5_description`. Accessed: 28-05-2018.

[71] Valkyrie Model. `https://github.com/openhumanoids/val_description`. Accessed: 28-05-2018.

[72] Wolf Model. `http://wp.me/P3dmoi-1uc`. Accessed: 28-05-2018.

[73] Antonio Morell, Mahmoud Tarokh, and Leopoldo Acosta. Inverse kinematics solutions for serial robots using support vector regression. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2013.

[74] Root Motion. `http://root-motion.com`. Accessed: 28-05-2018.

[75] Jorge Nocedal. Updating quasi-newton matrices with limited storage. *Mathematics of Computation*, 35:773–782, 1980.

[76] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, 2006.

[77] Rick Parent. *Computer Animation, Third Edition: Algorithms and Techniques*. Morgan Kaufmann, 2012.

[78] J. K. Parker, A. R. Khoogar, and D. E. Goldberg. Inverse kinematics of redundant robots using genetic algorithms. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 1, pages 271–276, 1989.

[79] E. Pennestri, M. Cavacece, and L. Vita. On the computation of degrees-of-freedom: A didactic perspective. In *Proceedings of the International Design Engineering Technical Conference*, 2005.

[80] TRAC-IK Plugin. `https://bitbucket.org/traclabs/trac_ik.git`. Accessed: 28-05-2018.

[81] Crossmodal Learning Project. `https://www.crossmodal-learning.org/`. Accessed: 28-05-2018.

[82] I. Rechenberg. *Cybernetic Solution Path of an Experimental Problem. Royal Aircraft Establishment*. Royal Aircraft Establishment, Ministry of Aviation, Farnborough Hants, UK, 1965.

[83] Claudio Melchiorri Riccardo Falconi, Raffaele Grandi. Inverse kinematics of serial manipulators in cluttered environments using a new paradigm of particle swarm optimization. *The International Federation of Automatic Control*, 19:8475–8480, 2014.

[84] Nizar Rokbani and Adel. M Alimi. Ik-pso: Pso inverse kinematics solver with application to biped gait generation. *International Journal of Computer Applications*, 58:33–39, 2012.

[85] Nizar Rokbani and Adel. M Alimi. Inverse kinematics using particle swarm optimization, a statistical analysis. In *Proceedings of the International Conference on Design and Manufacturing*, volume 64, pages 1602–1611, 2013.

[86] Philipp Ruppel. *Performance optimization and implementation of evolutionary inverse kinematics in ROS*. Master Thesis, University of Hamburg, 2017.

[87] Philipp Ruppel, Norman Hendrich, Sebastian Starke, and Jianwei Zhang. Cost functions to specify full-body motion and multi-goal manipulation tasks. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2018.

[88] Andrzej Ruszczynski. *Nonlinear Optimization*. Princeton, 2006.

[89] Emre Sariyildiz, Kemal Ucak, Gulay Oke, Hakan Temeltas, and Kouhei Ohnishi. Support vector regression based inverse kinematic modeling for a 7-dof redundant robot arm. *International Symposium on Innovations in Intelligent Systems and Applications*, 2012.

[90] David F. Shanno. Conditioning of quasi-newton methods for function minimization. *Mathematics of Computation*, 24:76–90, 1970.

[91] Bruno Siciliano. Kinematic control of redundant robot manipulators: A tutorial. *Journal of Intelligent and Robotic Systems*, 3:201–212, 1990.

[92] Bruno Siciliano and Oussama Kathib. *Handbook of Robotics*. Springer, 2008.

[93] Sebastian Starke. *A Hybrid Genetic Swarm Algorithm for Interactive Inverse Kinematics*. Master Thesis, University of Hamburg, 2016.

[94] Sebastian Starke, Norman Hendrich, Dennis Krupke, and Jianwei Zhang. Evolutionary multi-objective inverse kinematics on highly articulated and humanoid robots. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2017.

[95] Sebastian Starke, Norman Hendrich, Sven Magg, and Jianwei Zhang. An efficient hybridization of genetic algorithms and particle swarm optimization for inverse kinematics. In *Proceedings of the IEEE International Conference on Robotics and Biomimetics*, 2016.

[96] Sebastian Starke, Norman Hendrich, and Jianwei Zhang. *A Forward Kinematics Data Structure for Efficient Evolutionary Inverse Kinematics*, pages 560–568. Springer, Mechanisms and Machine Science, 2017.

[97] Sebastian Starke, Norman Hendrich, and Jianwei Zhang. A memetic evolutionary algorithm for real-time articulated kinematic motion. In *Proceedings of the IEEE Congress on Evolutionary Computation*, 2017.

[98] Richard Stokes. Robot simulator. `http://vrrobotsim.com`, 2017. Accessed: 28-05-2018.

[99] Marijn Stollenga, Leo Pape, Mikhail Frank, Jurgen Leitner, Alexander Forster, and Jurgen Schmidhuber. Task-relevant roadmaps: A framework for humanoid motion planning. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013.

[100] Rainer Storn and Kenneth Price. Differential evolution: A simple and efficient adaptive scheme for global optimization over continuous spaces. *Journal of Global Optimization*, 11:341–359, 1997.

[101] Saleh Tabandeh, Christopher M. Clark, and William W. Melek. A genetic algorithm approach to solve for multiple solutions of inverse kinematics using adaptive niching and clustering. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 1815–1822, 2006.

[102] Saleh Tabandeh, Christopher M. Clark, and William W. Melek. An adaptive niching genetic algorithm approach for generating multiple solutions of serial manipulator inverse kinematics with applications to modular robots. *Robotica*, 28:493–507, 2010.

[103] Gaurav Tevatia and Stefan Schaal. Inverse kinematics for humanoid robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2000.

[104] A. M. Turing. Computing machinery and intelligence. *Mind*, 59:433–460, 1950.

[105] Unity3D. `http://www.unity3d.com`. Accessed: 28-05-2018.

[106] Carla Elena González Uzcátegui. *A Memetic Approach to the Inverse Kinematics Problem for Robotic Applications.* Doctoral Thesis, Carlos III University of Madrid, 2014.

[107] John v. Neumann. The general and logical theory of automata. *Design of Computers, Theory of Automata and Numerical Analysis*, 5:288–328, 1951.

[108] Nikolaus Vahrenkamp, Tamin Asfour, and Rüdiger Dillmann. Efficient inverse kinematics computation based on reachability analysis. *International Journal of Humanoid Robotics*, 9:2, 2012.

[109] C. W. Wampler. Manipulator inverse kinematic solutions based on vector formulations and damped least squares methods. *IEEE Transactions on Systems, Man, and Cybernetics*, 16:93–101, 1986.

[110] L. Wang and C. Chen. A combined optimization method for solving the inverse kinematics problems of mechanical manipulators. *IEEE Transactions on Robotics and Automation*, 7:489–499, 1991.

[111] D. E. Whitney. Resolved motion rate control of manipulators and human prostheses. *IEEE Transactions on Man-Machine Systems*, 10:47–53, 1969.

[112] W. A. Wolovich and H. Ellio. A computational technique for inverse kinematics. In *Proceedings of the IEEE Conference on Decision and Control*, pages 1359–1363, 1984.

[113] David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1:67–82, 1997.

[114] Y. Zhang. A comprehensive survey on particle swarm optimization algorithm and its applications. *Mathematical Problems in Engineering*, 1:1–38, 2015.

[115] Jianmin Zhao and Norman Badler. Inverse kinematics positioning using nonlinear programming for highly articulated figures. *ACM Transactions on Graphics*, 13:313–336, 1994.

Erklärung der Urheberschaft

Ich versichere an Eides statt, dass ich die vorliegende Dissertation im Bereich der Informatik selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Insbesondere wurden dabei keine nicht im Quellenverzeichnis benannten Internet-Quellen verwendet. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Ort, Datum Unterschrift

Erklärung zur Veröffentlichung

Ich erkläre hiermit mein Einverständnis zur Einstellung dieser Dissertation in den Bestand der Bibliothek.

Ort, Datum Unterschrift