



Fakultät für Maschinenwesen  
Lehrstuhl für Angewandte Mechanik

# Efficient Algorithms for Biped Robots

Simulation, Collision Avoidance and Angular Momentum Tracking

**Dipl.-Ing. Univ. Markus Schwienbacher**

Vollständiger Abdruck der von der Fakultät für Maschinenwesen der  
Technischen Universität München zur Erlangung des akademischen Grades eines

**Doktors-Ingenieurs (Dr.-Ing.)**

genehmigten Dissertation.

**Vorsitzender:** Univ.-Prof. dr. ir. Daniel Rixen

**Prüfer der Dissertation:**

1. Univ.-Prof. Dr.-Ing. habil. Heinz Ulbrich (i. R.)
2. Univ.-Prof. Dr.-Ing. Carlo L. Bottasso

Die Dissertation wurde am 24. 10. 2013 bei der Technischen Universität München  
eingereicht und durch die Fakultät für Maschinenwesen am 22. 5. 2014 angenommen.



## Abstract

This PhD thesis covers efficient algorithms for biped robots. A multibody simulation extends the  $\mathcal{O}(n)$ -algorithm in order to cope with small kinematic loops inherent in the modeling of mechatronic systems. Self-collision avoidance is based on the efficient computation of distances between segments of the robot. Angular momentum is typically not considered in the control of biped robots. A method for collision avoidance and angular momentum tracking is presented which improves fast walking. Simulations and experiments are shown with the biped robot Lola.

*Keywords:* Humanoid, Biped, Walking, Efficient Algorithms, Multibody Simulation, Self-Collision Avoidance, Angular Momentum Tracking

## Zusammenfassung

Thema der Arbeit sind effiziente Algorithmen für zweibeinige Roboter. Eine Mehrkörpersimulation erweitert das  $\mathcal{O}(n)$ -Verfahren zur Behandlung kleiner kinematischer Schleifen, die bei der Modellierung mechatronischer Systeme auftreten. Ein Verfahren zur Vermeidung von Selbstkollisionen basiert auf der effizienten Berechnung der Distanzen zwischen Robotersegmenten. Es wird ein Verfahren vorgestellt, das die Kollisionsvermeidung mit einer Drallregelung integriert und so insbesondere schnelle Laufbewegungen verbessert. Simulationen und Experimente mit dem zweibeinigen Roboter Lola werden gezeigt.

*Stichworte:* Humanoide, Zweibeiner, effiziente Algorithmen, Mehrkörpersimulation, Kollisionsvermeidung, Drallregelung



## Acknowledgments

This thesis summarizes a large part of my research carried out at the Institute of Applied Mechanics, Technische Universität München. Many people have supported me during the past six years and made this work possible.

First, I would like to thank my supervisor Professor Heinz Ulbrich for giving me the opportunity to work on this research topic and providing such an excellent research environment. The given freedom combined with his guidance and support made this work possible. I would also like to acknowledge Professor Carlo Bottasso and Professor Daniel Rixen for serving on my thesis defense committee. After becoming the new head of the institute, Professor Rixen gave me the opportunity to finish my research for which I am thankful.

I am deeply grateful for having had the chance to work with a number of very talented and highly motivated people. I am especially thankful to the research group working on the robot Lola. While still being a student I learned a lot on mechanical design from Dr. Sebastian Lohmeier, who was responsible for Lola's mechatronic system architecture. He later encouraged me to join the team as a researcher and we continued discussing mechanical designs. I would like to thank Dr. Thomas Buschmann, who was responsible for the simulation and control of Lola. I learned so much about program design and walking control from him. His knowledge in robotics and guidance during his time as a group leader was invaluable. I warmly thank Valerio Favot, who worked on the decentralized controllers and communication system. His skills as an electronics wizard were greatly appreciated. Throughout the project we enjoyed good times but also endured some hardship. I would like to thank the other robotics team members Alexander Ewald, Robert Wittmann, Arne Hildebrand, Jörg Baur and Christoph Schütz for the inspiring discussions on robotics research topics.

Quality hardware is crucial for experimental work. I would like to thank the institute's electrical and mechanical workshops. I owe special thanks to Georg "Schorsch" Mayr. His long experience with electronics on research projects enabled both smooth experimental work with Lola and further developments on Lola. I would like to thank Wilhelm Miller, Walter Wöß, Simon Gerer, Philip Schneider and Tobias Schmidt for their terrific work in manufacturing parts for Lola. I am grateful to PD Dr. Thomas Thümmel for managing project resources and his support throughout this thesis.

I would also like to thank my ex-colleagues Dr. Thomas Buschmann, Dr. Sebastian Lohmeier and Robert Wittmann for proofreading this thesis and giving helpful comments.

Finally, I would like to thank my family, my girlfriend Karina and my good friend Dr. Reinhard Tschiesner for their continuous support and encouragement.

Munich, July 2014

Markus Schwienbacher



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Problem Statement . . . . .	1
1.2	Background and Related Work . . . . .	4
1.3	Overview of this Thesis . . . . .	6
<b>2</b>	<b>An <math>\mathcal{O}(n)</math>-formalism for the Simulation of MBS with Small Kinematic Loops</b>	<b>9</b>
2.1	Introduction . . . . .	9
2.1.1	Basic Dynamics Equation . . . . .	10
2.1.2	Related Work . . . . .	11
2.2	Rigid Body Kinematics . . . . .	13
2.3	Rigid Body Dynamics . . . . .	15
2.4	Relative Kinematics of Rigid Multibody Systems . . . . .	17
2.4.1	Topology of Multibody Systems . . . . .	17
2.4.2	Motion Constraints and Minimal Coordinates . . . . .	19
2.4.3	Recursive Kinematics Calculation . . . . .	20
2.4.4	Recursive Kinematics using Spatial Vector Notation . . . . .	21
2.5	Dynamics of Rigid Multibody Systems . . . . .	25
2.6	Sub-Systems . . . . .	26
2.7	Detailed Derivation of the $\mathcal{O}(n)$ -Algorithm with Sub-Systems . . . . .	28
2.8	Resulting Formalism . . . . .	36
2.9	Automatic Sub-System Generation . . . . .	39
2.10	Results . . . . .	42
2.10.1	Run-Time Comparisons for the Dill Example . . . . .	42
2.10.2	Run-Time Comparison for the Lola Model . . . . .	45
2.11	Discussion . . . . .	51
2.12	Chapter Summary . . . . .	52
<b>3</b>	<b>Kinematics</b>	<b>55</b>
3.1	Harmonic Drive Gears . . . . .	55
3.2	Knee Joint Drive Kinematics . . . . .	55
3.3	Ankle Joint Drive Kinematics . . . . .	58
3.4	Camera Vergence Kinematics . . . . .	60
3.5	Kinematics Calibration . . . . .	62
3.6	Chapter Summary . . . . .	67
<b>4</b>	<b>Inverse Kinematics</b>	<b>69</b>
4.1	Problem Statement . . . . .	69
4.2	Position Based Inverse Kinematics . . . . .	70

4.3	Differential Inverse Kinematics . . . . .	72
4.3.1	Resolved Motion Rate Control and Redundancy Resolution . . . . .	72
4.3.2	Jacobian Transpose . . . . .	75
4.3.3	Resolved Acceleration Rate Control . . . . .	76
4.3.4	Hierarchical Approaches . . . . .	76
4.4	Singularities and Manipulability . . . . .	77
4.5	Task Description of Lola . . . . .	78
4.6	Chapter Summary . . . . .	78
<b>5</b>	<b>Self-Collision Avoidance</b>	<b>81</b>
5.1	Background and Related Work . . . . .	81
5.2	Self-Collision Avoidance . . . . .	82
5.3	Chapter Summary . . . . .	86
<b>6</b>	<b>Real-Time Distance Computation using Swept-Sphere-Volumes</b>	<b>89</b>
6.1	Background and Related Work . . . . .	89
6.2	Formal Aspects of Distance Computation . . . . .	91
6.3	SSV Primitives . . . . .	92
6.3.1	Point-Swept-Sphere Volume . . . . .	93
6.3.2	Line-Swept-Sphere Volume . . . . .	93
6.3.3	Triangle-Swept-Sphere Volume . . . . .	94
6.4	Distance Calculation between SSV Primitives . . . . .	94
6.4.1	Point to Point Distance Computation . . . . .	96
6.4.2	Point to Line-Segment Distance Computation . . . . .	97
6.4.3	Point to Triangle Distance Computation . . . . .	98
6.4.4	Line-Segment to Line-Segment Distance Computation . . . . .	99
6.4.5	Line-Segment to Triangle Distance Computation . . . . .	105
6.4.6	Triangle to Triangle Distance Computation . . . . .	110
6.4.7	General Framework of Minimization Using Inequality Constraints . . . . .	110
6.5	Implementation Details and Run-Time Performance . . . . .	113
6.6	Modeling of the Robot Segments . . . . .	114
6.6.1	Compounds of SSVs as Robot Segments . . . . .	114
6.6.2	A Versatile Modeling Tool . . . . .	114
6.7	Integration of Bounding Boxes . . . . .	117
6.8	System Overview . . . . .	118
6.9	Chapter Summary . . . . .	121
<b>7</b>	<b>Use of Angular Momentum in Walking Control</b>	<b>123</b>
7.1	Introduction . . . . .	123
7.2	Angular Momentum . . . . .	123
7.3	Angular Momentum Compensation via Null-Space Motion . . . . .	124
7.3.1	Reference Method . . . . .	124
7.3.2	Proposed Method . . . . .	125
7.3.3	Simulations . . . . .	126
7.3.4	Experiments . . . . .	126



7.4	Angular Momentum Trajectory . . . . .	127
7.5	Angular Momentum Minimization . . . . .	129
7.6	Chapter Summary . . . . .	131
<b>8</b>	<b>Conclusion and Outlook</b>	<b>133</b>
8.1	Summary . . . . .	133
8.2	Recommendations for Future Work . . . . .	135
<b>A</b>	<b>Mathematical Toolbox</b>	<b>137</b>
A.1	Notation and Operators . . . . .	137
A.2	Coordinate Transformations . . . . .	138
A.3	Partial Derivatives . . . . .	139
<b>B</b>	<b>Pseudo-Code for Distance Calculation Algorithms</b>	<b>141</b>
B.1	Line-Segment to Line-Segment . . . . .	141
<b>C</b>	<b>Inverse Kinematics: Orientation Error in Task Space</b>	<b>145</b>
	<b>List of Abbreviations</b>	<b>149</b>
	<b>Bibliography</b>	<b>151</b>



# 1 Introduction

The human-like structure and size makes biped robots ideal for operating in almost every environment also accessible by humans. Cluttered environments such as uneven ground, stairs, ladders, small spaces etc. are encountered in urban life which complicates locomotion. On the other hand, wheel-based systems typically are less complex, enable faster locomotion and are more stable. However, most of the mentioned scenarios are only viable for biped systems. This is one of the main motivations for the research on biped walking technology.

It is only due to significant advancements in enabling technologies which facilitated the realization of biped walking machines with fully actuated joints. These technologies include computer systems, control theory, drives, etc. Although, the first systems were already built about four decades ago, the research on humanoid robots is still an active field and must be considered rather young. For this reason, we are far from commercially available systems with capabilities comparable to humans. Moreover, proposed solutions to some of the more complex scenarios are only of theoretical significance.

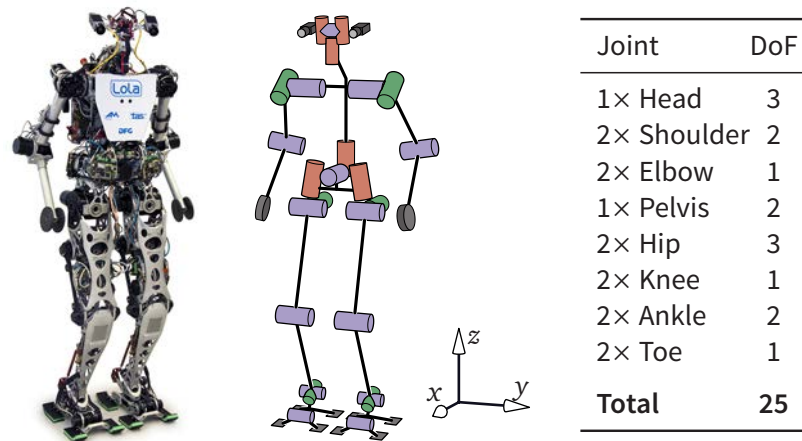
Potential application fields for humanoids reach from entertainment to service robotics. Thereby, the requirements expand in many directions including (among others), perception, computer vision, high-level cognition, speech synthesis, speech recognition, manipulation, whole body motion and biped locomotion etc. Most of them are active research fields where further progress is required.

Also triggered by the FUKUSHIMA DAI-ICHI nuclear disaster in Japan early 2011, robots are spotted to be ideal when deployed in harsh environments. In this vision, robots could one day save lives by taking over extremely dangerous and high-stakes human tasks like shutting off deep-water oil spills, or to put out large fires. They can be of help in search, rescue and cleaning missions after devastating environmental events, such as earthquakes, tsunamis and hurricanes.

Finally, findings from research on biped walking is also of great importance to other fields like medical engineering, prosthetics, rehabilitation and therapy. All this is expected to give robotic research higher importance for mankind in the future.

## 1.1 Problem Statement

This thesis covers the efficient handling of some important aspects for biped robots: simulation, self-collision avoidance and angular momentum in walking control. The main research platform for this work is the robot Lola, developed at the Institute of



**Figure 1.1:** The biped humanoid robot Lola with 25 DoFs: Photograph and kinematic structure of the system (illustration from [125]).

Applied Mechanics,<sup>1</sup> Technische Universität München. Figure 1.1 shows a photograph of Lola as well as the basic kinematic structure with 25 actuated degrees of freedom (DoFs). The robot is 180 cm tall and weighs approximately 60 kg. The distinguishing characteristics of Lola are the redundant kinematic structure with 7-DoF legs, an extremely lightweight construction, and a modular joint design with high power density based on brushless motors. A comprehensive overview of Lola’s hardware is given in [85, 86, 88].

The methods described in this work, are not confined to Lola but apply to other biped robots too. A necessary condition, however, is that the robot’s DoFs are redundant with respect to the chosen task space.

## Walking Control

The problem of walking control is challenging. A variety of constraints have to be taken into account in order to obtain *feasible* walking patterns. During walking, balance is maintained (almost) only by the *unilateral contact* of the robot with its environment. Thereby, *feasible contact forces* must be generated while other body parts are moving. Since contact forces are influenced by this motion, multibody dynamics must be considered in the process. Unintended *collisions* between body parts and the environment can destabilize the robot and/or damage the robot hardware (or its environment). Constraints for minimal distances must be integrated to avoid

<sup>1</sup> In prior works, SEBASTIAN LOHMEIER developed the mechatronic design concept and did the mechanical design of the robot (cf. [85]). Moreover, THOMAS BUSCHMANN developed a versatile software infrastructure for dynamics simulation and control along with a walking controller for Lola (cf. [28]). In ongoing work, VALERIO FAVOT is developing methods for lower-level control and the communication infrastructure of Lola (cf. [50]).

interpenetration. Furthermore, the joints of the robot have capability limits for both velocity and acceleration as well as joint range boundaries. Additional constraints might arise from specific tasks like manipulation etc. All those constraints—some of which are nonlinear—have to be taken into account in order to obtain *global stability* and robustness. Finally, the resulting optimization problem is high-dimensional and nonlinear.

On the other hand, the robot has to be able to react to unforeseen changes of a *dynamic environment* which possibly invalidates a feasible solution obtained beforehand. Therefore, walking patterns must be computed in real-time. It is quite obvious, however, that current computer systems are unable to solve the full optimization problem in real-time. Furthermore, the iterative nature combined with uncertain convergence rates of nonlinear optimization algorithms disagrees with real-time computation.

A practical approach to facilitate real-time execution is to neglect less important constraints and plan stable trajectories for a simplified dynamics model. Because the resulting solution only approximates the full system dynamics, planned trajectories are perturbed during execution in order to regain stability. Thereby, perturbations are based on sensor feedback and/or state estimation. This approach, combining global optimization for a simplified model with local optimization of further constraints, is the basis for all state-of-art real-time walking (and running) pattern generators [28, 102, 135–137]. Furthermore, in order to facilitate real-time execution efficient algorithms have to be applied in all stages.

## Simulation

Simulation is an important engineering tool in different stages of the development process. The *efficient simulation* of a system enables faster and tighter development loops, thereby, optimizing the system in an iterative manner. Efficiency is gained on several levels of the simulation process: from modeling, over element-model evaluation and time-integration to post-processing of results. Each of these levels opens up potential for optimization. Ultimately, the degree of modeling depth depends on the desired accuracy of the effects relevant to the system behavior.

The modeling of mechatronic systems often results in kinematic couplings leading to loops in the structure. Especially the rigid modeling of drive mechanisms, like gears, generates kinematic loops. This mechanical modeling both reduces the number of degrees of freedom and the effort for time-integration at the same time. The latter is mostly due to different time scales induced by high frequencies from low motor shaft inertia combined with large drive mechanism/gear stiffness. As a consequence, the step-size for time integration must be prohibitively small and degrades performance.

Such multi-domain models can also be integrated efficiently using *mixed-mode integration* and *multi-rate integration*. By studying the eigenvalues of the system matrix, both methods divide the model into a *fast* and a *slow part*. An explicit integrator is applied for the slow part, while the fast part is either integrated using an implicit

integrator or a lower step-size for the mixed-mode and multi-rate method, respectively. Both methods are found in commercially available general purpose simulation software such as DYMOLA<sup>2</sup> [124].

## 1.2 Background and Related Work

The word “robot” is derived from the Czech word for forced labor and has its origin in KAREL ČAPEK’s play “R.U.R.: Rossum’s Universal Robots” (published 1920). Therein, human-shaped machines start to dominate the human race. Also influenced by other critical writings about technology, a significant part of the general public perceive humanoid robots as a threat. Science fiction films which include robots often provoke unrealistic expectations about the abilities of such machines. Moreover, it seems obvious that the military has interests in robotics technology even beyond defense applications. However, this research often is not publicly available. All this generates a human fear to possibly lose control over autonomously acting machines. The perception of the society is an important factor for the acceptance of such machines along with realistic expectations about their abilities in order to become commercially successful.

On the other hand, humanoid robots open up a fascinating and interdisciplinary research field. The large and growing body of literature in this field emphasizes the scientific interest. Moreover, a large amount of scientific journals and conferences about robotics in general and about humanoids in particular exist.

The following part is intended to give only a brief overview of relevant research on biped walking robots. Surveys of legged walking (and running) robots can be found in [84, 118, 119, 129]. GIENGER [60] and LOHMEIER [85] give detailed surveys of biped robot hardware design while BUSCHMANN [28] focuses on both historical and control aspects. Only more recent work, not found in the cited works above, are described here in more detail. Finally, individual chapters contain detailed reviews to relevant literature covering the according chapter topics.

In 1973, KATO developed at the Waseda University, Japan, the first full-scale human-like robot WABOT-1. Thereby, modern research on biped walking was initiated. In subsequent years a number of advanced robots with innovative hardware concepts were developed at Waseda [106, 108].

At the JSK-LABORATORY, Tokyo University, the humanoid robots H6 and H7 were designed and built in cooperation with KAWADA INDUSTRIES. Actuated toe joints enable the robots to knee down, climb stairs and enlarge the foot steps [104]. Using vision systems, research on advanced path planning methods for discreet footholds in unknown and dynamically changing environments is conducted [103]. Furthermore, tendon-driven musculoskeletal humanoids with flexible spines are developed [101].

---

<sup>2</sup> [www.3ds.com/products-services/catia/capabilities/catia-systems-engineering/modelica-systems-simulation/dymola](http://www.3ds.com/products-services/catia/capabilities/catia-systems-engineering/modelica-systems-simulation/dymola)

More recently, URATA et al. [149] developed the biped robot prototype HRP3La-JSK with high power joints. An active cooling system and batteries combined with a high capacitance capacitor providing instantaneously high current rates enables the robot to walk at 5 km/h (cf. [150]).

Within the “Humanoid Robotics Project” (HRP) a number of sophisticated full-sized humanoid walking robots, HRP-2 to HRP-4, have been developed jointly by the NATIONAL INSTITUTE OF ADVANCED INDUSTRIAL SCIENCE AND TECHNOLOGY (AIST), KAWADA INDUSTRIES and a number of Japanese universities. Copies of the HRP-2 are sold to different research groups mostly working on software aspects. While the first robots developed in the program (HRP-2 and HRP-3) had a “Transformers-like” appearance, the newest models HRP-4 and HRP-4C presented in 2009 have a more slender shape with both anthropometry and appearance of a young Japanese female. One goal of the newest development HRP-4 is to reduce robot production costs significantly making the robot more affordable to research groups [74].

While several years ago *service robotics* was considered being the main application area for humanoid robots, a current trend is the research on robotic systems able to work or help in harsh environments. This is, without doubt, a reaction to the problems encountering after the FUKUSHIMA nuclear disaster.

In the DARPA ROBOTICS CHALLENGE<sup>3</sup> (DRC), which is currently in process, a number of teams develop new walking robots with manipulation abilities. The goal of DRC is to initiate research for future humanoid robots able to perform hazardous activities in disaster response operations, together with humans, in order to reduce casualties, avoid further destruction, and save lives. The challenge is divided in three increasingly demanding events (in June 2013, Dec. 2013 and Dec. 2014). The winning team will be awarded with 2,000,000 US\$ [41].

The U.S. company BOSTON DYNAMICS recently presented the biped robot Petman capable of walking at 7 km/h. Unlike other robots with electrically driven joints, the robots from BOSTON DYNAMICS use hydraulic actuation driven by a combustion engine. Based on Petman, the company developed a robot named ATLAS specifically for DRC. Like Petman, the project is funded by DARPA. Several copies of the robot are given to DRC teams only working on software. The system has near-human anthropometry with two arms and legs with 28 hydraulically actuated joints. The system is electrically powered from off-board, weights 150 kg and is 188 cm tall. The head is equipped with a laser range finder and a stereo vision system for perception [3].

Starting in 1986, the Japanese HONDA MOTOR CORPORATION developed a series of advanced humanoid robots (E1 to E6 and P1 to P3). While research was secretly conducted in the beginning, HONDA presented the first fully self-contained humanoid robots in 1996. Starting in 2000, the company presented a series of autonomous humanoids under the name ASIMO.<sup>4</sup> Each generation has increasing capabilities and

---

<sup>3</sup> DARPA is the U.S. Defense Advanced Research Project Agency

<sup>4</sup> short for: “Advanced Step in Innovative Mobility”

is amongst the most advanced biped robots of their time. All ASIMO models have arms with hands and legs, are about 130 cm tall and weight approx. 50 kg. Being a company, Honda usually releases details about their systems in patents rather than in scientific papers. Only recently, more details of trajectory planning and control methods used for the ASIMO robots were published as scientific papers [136–139]. However, Honda's approach to building and controlling biped robots has been both very influential and successful, and hence, has been widely adopted. At 10 km/h, HONDA currently has the fastest running biped robot [137].

Recently, the GERMAN AEROSPACE CENTER (DLR) presented a biped robot named TORO.<sup>5</sup> The arms and legs of TORO are based on DLR's lightweight robots having joints with torque sensors which enable the robot to react flexibly to its environment [111].

Research on biped robots at the INSTITUTE OF APPLIED MECHANICS (AM), Technische Universität München, started with Johnnie [60, 84]. Using only on-board perception and control, the robot was the first biped able to navigate autonomously in an unknown environment. Based on experience gained from research on Johnnie and other robotics projects at AM, LOHMEIER [85] and BUSCHMANN [28] developed the biped robot Lola.<sup>6</sup> At 3.6 km/h, Lola is currently one of the fastest biped walking machines developed at a university [30]. A vision system, developed by VON HUNDELSHAUSEN, enables navigation among dynamically changing arbitrary obstacles without requiring color coding, surface texture or explicit models of the environment [31].

### 1.3 Overview of this Thesis

The first objective of this thesis is the development of an efficient method for simulating the multibody dynamics of mechatronic systems. Small kinematic loops inherent in the modeling of mechatronic systems must be solvable with the approach.

The second objective is the development of some methods for walking control contributing to both safety and human-like biped walking performance. The methods must allow for real-time application. Furthermore, the system should be sufficiently general to be applicable to biped robots similar to Lola. Thereby, the problem of avoiding self-collisions in real-time is addressed. Moreover, angular momentum tracking is integrated into walking control to reduce the modeling error of current real-time walking pattern generators.

The main contributions of this thesis are:

- The development of an efficient multibody simulation algorithm which extends

---

5 short for: "TORque controlled humanoid ROBot"; Press release: [http://www.dlr.de/dlr/en/desktopdefault.aspx/tabid-10080/150\\_read-6601/year-2013/150\\_page-4/#gallery/9208](http://www.dlr.de/dlr/en/desktopdefault.aspx/tabid-10080/150_read-6601/year-2013/150_page-4/#gallery/9208)

6 Many other people contributed to the development of Lola: Valerio Favot, Georg Mayr, Mathias Bachmayer, the author and students.



the most efficient  $\mathcal{O}(n)$ -algorithms for handling small kinematic loops inherent in the modeling of mechatronic systems.

- Provide run-time measurements, conducted on different computer systems, comparing and showing the efficiency of the dynamics algorithms when applied to a number of test systems including the multibody models of Lola.
- New and improved algorithms for real-time walking control for biped robots with redundant configurations, including:
  - Self-collision avoidance
  - Angular momentum tracking
  - Angular momentum minimization
- The development of a framework for fast distance computation between segments of the robot, thereby, enabling self-collision avoidance in real-time.
- Verifications showing the effectiveness of all methods applied to the robot Lola using simulations and experiments.

The developed methods should extend and improve the frameworks developed by BUSCHMANN (cf. [28]) for the robot Lola.

This thesis is structured as follows: In Chapter 2 an  $\mathcal{O}(n)$ -algorithm for the *simulation* of multibody systems with kinematic loops is derived. Equations for the calculation of quantities for *kinematics* mostly specific to the robot Lola as well as the method for *kinematic calibration* is described in Chapter 3. An introductory overview for subsequent chapters about relevant *inverse kinematics* algorithms is given in Chapter 4. The *self-collision avoidance* method developed in this work is presented in Chapter 5 and the used framework for *distance computation* in Chapter 6. Two methods for incorporating the *angular momentum* into walking control are described in Chapter 7. Chapter 8 concludes the thesis with a summary and suggestions for future work.

Finally, used mathematical operators and expressions, some implementation details, and a list of abbreviations is found in the *appendix*.



## 2 An $\mathcal{O}(n)$ -formalism for the Simulation of MBS with Small Kinematic Loops

### 2.1 Introduction

The simulation of mechatronic systems such as humanoid robots is an important tool for hardware and controller design in different stages of the development process. The mechanical design provides model parameters and requires the knowledge of loads to assess strains, analyze dynamic performance, support design decisions and enable optimization. Model-based controllers such as walking controllers need kinematic models and dynamic models. Finally, the evaluation of control system robustness and performance is enabled by simulation. Each step requires for different modeling depth and/or computational efficiency. BUSCHMANN developed in [28] a simulation environment written in C++ which was extended in this work. The software was developed to simulate the biped robots Johnnie and Lola. Therefore, it also contains specialized modeling elements which cannot be found in commercially available simulation software, neither for general purpose simulation nor for *multibody system* (MBS) simulation. The main building blocks are a modeling library and different integrators for *Ordinary Differential Equations* (ODE). Typically, they are combined to a dynamics simulation coupled to a controller. The modeling library consists of different rigid-body-types and special mechanisms, gear friction models for Harmonic Drive gears and roller screw drives, electrical motor models, different contact models and contact solvers as well as environment models. Although the library was developed to simulate biped robots other *ordinary* MBS, i. e., with holonomic constraints, can be simulated.

The growing computational power of computer systems enables the simulation of more detailed models. In order to exploit parallel processors, co-simulation becomes more important [38, 57]. Thereby, the evaluations of complex sub-models, such as FEM-contacts, exceed the computational cost of many rigid body dynamics algorithms. Obviously, in such cases the computational efficiency of MBS simulation is of secondary interest. Current trends in commercial simulation software tools are therefore to integrate the Functional Mock-up Interface<sup>1</sup> (FMI) standard [18]. This enables multi-physics co-simulation and model file-exchange via FMUs<sup>2</sup> between FMI-conformant software.

---

<sup>1</sup> FMI specifies standardized interfaces used by simulation software [www.fmi-standard.org](http://www.fmi-standard.org)

<sup>2</sup> Functional Mockup Units are singular model components used by the FMI architecture.

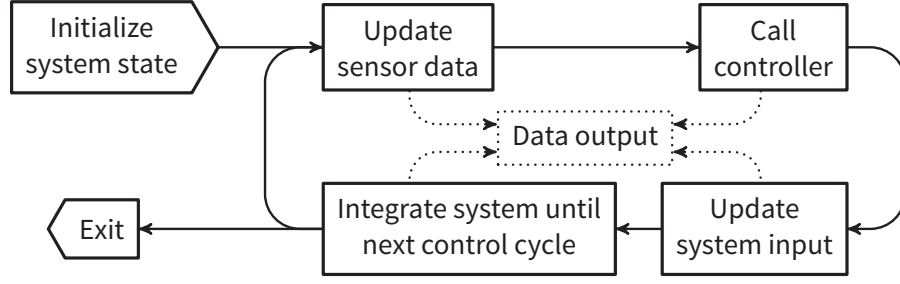


Figure 2.1: Simulation of mechatronic systems.

### 2.1.1 Basic Dynamics Equation

Using the Newton-Euler-Jourdain principle [27, 148], the equations of motion (EoM) for the rigid-body MBS can always be formulated as second order ordinary differential equations (ODE) in the form

$$M\ddot{\mathbf{q}} + \mathbf{h} = \mathbf{Q}_{\text{ext}}, \quad (2.1)$$

with the mass-matrix  $M$ , the vector of generalized coordinates  $\mathbf{q}$  and its second time derivative  $\ddot{\mathbf{q}}$ , the vector of Coriolis, centrifugal and gravitational forces  $\mathbf{h}$  and the impressed generalized forces  $\mathbf{Q}_{\text{ext}}$ .

For calculating the time evolution of the positions and velocities, (2.1) has to be solved for  $\ddot{\mathbf{q}}$  and integrated with respect to time. Rewriting (2.1) as a standard first order ordinary differential equation leads to

$$\dot{\mathbf{z}} = \mathbf{f}(\mathbf{z}, \mathbf{u}), \quad (2.2)$$

where vectors  $\mathbf{z}$  and  $\mathbf{u}$  denote state and input, respectively. Depending on the type of simulation,  $\mathbf{z}$  contains generalized positions and velocities and other states to describe the system. On the other hand, for the EoM input  $\mathbf{u}$  typically contains physical quantities such as forces/torques, motor voltage/current etc. Assuming only dynamics from (2.1) and holonomic constraints, (2.2) may be written as

$$\dot{\mathbf{z}}_{\text{MBS}} := \begin{bmatrix} \ddot{\mathbf{q}} \\ \dot{\mathbf{q}} \end{bmatrix} = \begin{bmatrix} M^{-1}(\mathbf{Q}_{\text{ext}} - \mathbf{h}) \\ \dot{\mathbf{q}} \end{bmatrix}. \quad (2.3)$$

Integration finally leads to

$$\mathbf{z}(t) = \mathbf{z}_0 + \int_0^t \dot{\mathbf{z}} \, d\tau, \quad (2.4)$$

which is evaluated using a suitable numerical integrator.

Figure 2.1 illustrates the simulation of a mechatronic system using a feedback

controller: the initialized system state is converted to sensor output used by the controller. Data output by the controller is used as input quantities to the dynamic system. After time integration until the next control cycle, sensor data is again given to the controller. This loop is repeated until an exit-condition is met. For cascade control, the block for time-integration contains multiple levels of this picture.

Solving the EoM for  $\dot{\mathbf{q}}$  is also known as *forward dynamics* (FD) calculation.<sup>3</sup> Different efficient FD algorithms have been developed since the 1960's, when computational resources were a limiting factor. Some of them are briefly presented in the following section.

### 2.1.2 Related Work

Traditionally, robotics researchers have focused on the computational efficiency of their dynamics algorithms. In fact, many of the most efficient algorithms in dynamics, that are applicable to a broad range of mechanisms, were developed with a focus on robotics [1, 21, 54, 152]. A good overview of available algorithms can be found in [53]. SCHIEHLEN [122] presents historical remarks on the development of MBS from classical mechanics to modern aspects in MBS simulation. A detailed derivation of relevant methods and further historical aspects of classical mechanics is presented by BREMER in [25]. FEATHERSTONE gives a historical overview on the development of dynamics algorithms specialized to computers [53].

Most FD algorithms can be classified into two main types: *propagation algorithms* and *inertia-matrix (mass-matrix) algorithms*. The latter calculate the mass-matrix explicitly and solve for the accelerations. In contrast, propagation algorithms propagate quantities over joints to calculate the accelerations in a recursive manner without calculating the mass-matrix  $\mathbf{M}$  of the system. Typically, for open-loop systems the computational complexity of propagation algorithms is of  $\mathcal{O}(n)$ , i. e., the computation time grows linearly with the number of *degrees of freedom* (DoF). On the other hand, general inertia-matrix algorithms have  $\mathcal{O}(n^3)$  algorithmic complexity. The algorithmic complexity becomes an important factor when dealing with a large number of DoFs. Hence, the algorithms are also often called  $\mathcal{O}(n)$  algorithms and  $\mathcal{O}(n^3)$  algorithms respectively. However, the algorithmic complexity of some algorithms is highly dependent on the kinematic configuration, or topology, of the MBS, i. e.,  $\mathcal{O}(n)$  algorithms perform best with an open-loop structure without branches, i. e., a strict chain. On the other hand, topologies containing closed loops are not directly supported by recursive  $\mathcal{O}(n)$ -algorithms. Several extensions to recursive  $\mathcal{O}(n)$ -formalisms have been developed to cope with such topologies [6, 22]. One possible solution is to convert the loops into a tree by cutting the loops open at suitable joints and introducing Lagrange multipliers. These Lagrange multipliers are equivalent to the joint constraint forces which satisfy the kinematic constraints of the joint. However, this approach has some

---

<sup>3</sup> In contrast to the *inverse dynamics*, where forces are computed from known accelerations.

drawbacks. First, depending on the loop dimension,<sup>4</sup> the increased computational complexity may become larger than  $\mathcal{O}(n^3)$  (cf. [22]) which cancels the advantage of the method. More importantly, due to numerical drift, the resulting *differential algebraic equations* (DAEs) become stiff which leads to an inefficient integration [122]. In order to overcome these inherent instabilities several methods such as BAUMGARTE stabilization [9] and projection methods [2] have been developed.

The modeling process for mechatronic systems and mechanical systems with closed kinematic loops is more efficient on the basis of implicit constraint equations for coupling components and closing the loop [122].

BREMER presents in [26] a technique for the efficient modeling of large mechanical systems with (ideally) repeating structures called *sub-systems*. The technique was further developed as  $\mathcal{O}(n)$  algorithm which is able to handle elastic bodies [24, 130] and systems with contacts [59] (see Section 2.6).

Under certain conditions, mainly due to branches in the kinematic tree, the mass-matrix  $\mathbf{M}$  becomes sparse.<sup>5</sup> FEATHERSTONE [52] proposes a specialized sparse matrix solver for  $\mathbf{M}$ . The method consists of a *reversed order* sparse Cholesky decomposition such that  $\mathbf{M} = \mathbf{L}^T \mathbf{L}$  (instead of  $\mathbf{M} = \mathbf{L} \mathbf{L}^T$ , cf. e. g., [116]). The sparsity is exploited by using an index vector containing the child-parent relations. The resulting lower left triangular matrix  $\mathbf{L}$  then inherits the sparsity pattern of  $\mathbf{M}$ . Depending on the branching topology of the kinematic tree the algorithmic complexity is in the range of  $\mathcal{O}(n)$  ( $n$  decoupled pendulums) to  $\mathcal{O}(n^3)$  (unbranched chain). A strict chain topology implies a fully dense mass-matrix. According to [52], for a robot with four limbs and 30 DoFs the computational cost is only approx. 15% higher when compared to an  $\mathcal{O}(n)$  algorithm.

## Chapter Structure

Based on the rigid body kinematic quantities introduced in Section 2.2, the dynamics equations of a single body are derived in Section 2.3. In Section 2.4 the *relative* kinematics of rigid multibody systems is formulated and exploited in Section 2.5 to derive the equations of motion of a multibody system. Furthermore, the concept of sub-systems is introduced in Section 2.6 and a suitable  $\mathcal{O}(n)$ -algorithm, able to cope with kinematic loops, is derived in Section 2.7. Section 2.8 presents an algorithmic form of the derivation. To facilitate a simplified modeling, a method to reconfigure a MBS into sub-systems is presented in Section 2.9. Finally, a run-time comparison studying certain effects is presented in Section 2.10.

4 term “loop dimension” refers to the number of bodies or joints involved in the loop

5 Zero elements are introduced systematically due to branches. Moreover, depending on the configuration and other mass parameters, other entries in the mass-matrix can become zero.

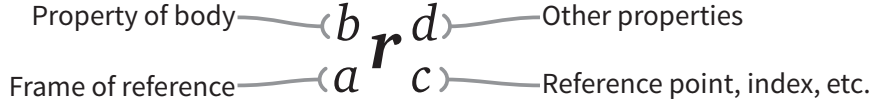


Figure 2.2: Meaning of sub- and superscripts for a variable “ $\mathbf{r}$ ”.

## 2.2 Rigid Body Kinematics

Throughout this thesis, sub- and superscripts for variables are used to define other properties. Leading subscripts denote the *frame of reference* (FoR) and leading superscripts the reference body. Normal subscripts are used for reference points and indices and normal superscripts for further descriptive properties. The meaning of sub- and superscripts is illustrated in Figure 2.2. In this nomenclature,  ${}^b_a \dot{\mathbf{r}}_o$  denotes the velocity of body  $b$ 's origin  $o$  described in FoR  $a$ .

**Positions** With reference to Figure 2.3 the position of a rigid body can be expressed using vector notation as

$$\mathbf{r} = \mathbf{r}_O + \mathbf{r}_{OP}, \quad \mathbf{r} \in \mathbb{R}^3. \quad (2.5)$$

By letting point  $O$  and FoR  $K$  be fixed to the rigid body, we can rewrite above relation using coordinate notation to

$${}_I \mathbf{r}_P = {}_I \mathbf{r}_O + \mathbf{A}_{IK} {}_K \mathbf{r}_{OP}, \quad (2.6)$$

where matrix  $\mathbf{A}_{IK}$  defines the transformation from FoR  $K$  to an inertial FoR  $I$ . The logic is illustrated in Figure 2.4. Thereby, axis unit-directions  ${}_I \mathbf{e}_{K,i}$  for  $i \in (x, y, z)$  of FoR  $K$  represented in FoR  $I$  are the columns of  $\mathbf{A}_{IK}$ :

$$\mathbf{A}_{IK} = [{}_I \mathbf{e}_{K,x} \quad {}_I \mathbf{e}_{K,y} \quad {}_I \mathbf{e}_{K,z}], \quad \|\mathbf{e}_{K,i}\| = 1, \quad \mathbf{e}_{K,i} \cdot \mathbf{e}_{K,j} = 0 \text{ for } i \neq j. \quad (2.7)$$

Hence,  $\mathbf{A}_{IK}$  is orthonormal and the following useful identities hold:

$$\mathbf{A}_{IK} = \mathbf{A}_{KI}^{-1} = \mathbf{A}_{KI}^T, \quad (2.8)$$

$$\mathbf{A}_{IK} \mathbf{A}_{IK}^T = \mathbf{E}, \quad (2.9)$$

where  $\mathbf{E} \in \mathbb{R}^{3 \times 3}$  is the unit matrix.

**Velocities** The *absolute* velocity of point  $P$  is the *total* derivative of position vector  $\mathbf{r}_P$  with respect to time in (2.6)

$${}_I \dot{\mathbf{r}}_P = \frac{d}{dt} {}_I \mathbf{r}_P = {}_I \dot{\mathbf{r}}_O + \dot{\mathbf{A}}_{IK} {}_K \mathbf{r}_{OP} + \mathbf{A}_{IK} {}_K \dot{\mathbf{r}}_{OP}. \quad (2.10)$$

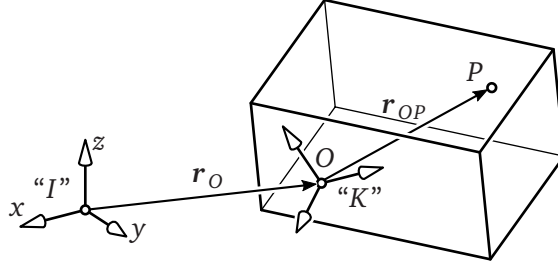


Figure 2.3: Kinematics of a point in a rigid body (adopted from [27]).

The total derivative ( $\dot{\phantom{x}}$ ) implies derivatives of the FoR, which obviously vanish for the non-moving FoR  $I$ . However, this must be considered, when transforming to a moving FoR, e. g.,  $K$  in Figure 2.3. Since only rigid bodies are considered, we have  ${}_K \dot{\mathbf{r}}_{OP} \stackrel{!}{=} \mathbf{0}$ . Additionally, using the transformation  ${}_K \mathbf{r}_{OP} = \mathbf{A}_{KI} \mathbf{r}_{OP}$  gives

$${}_I \dot{\mathbf{r}}_P = {}_I \dot{\mathbf{r}}_O + \dot{\mathbf{A}}_{IK} \mathbf{A}_{KI} \mathbf{r}_{OP} . \quad (2.11)$$

The term  $\dot{\mathbf{A}}_{IK} \mathbf{A}_{KI}$  is a skew-symmetric tensor, which is easily shown by time derivation of (2.9) yielding

$$\dot{\mathbf{A}}_{IK} \mathbf{A}_{IK}^T + \mathbf{A}_{IK} \dot{\mathbf{A}}_{IK}^T = \mathbf{0} \quad \Leftrightarrow \quad \dot{\mathbf{A}}_{IK} \mathbf{A}_{KI} = -(\dot{\mathbf{A}}_{IK} \mathbf{A}_{KI})^T . \quad (2.12)$$

Moreover, it is related to the angular velocity vector  $\boldsymbol{\omega}_{KI}$ , describing the angular velocity of FoR  $K$  relative to FoR  $I$ , via

$$\dot{\mathbf{A}}_{IK} \mathbf{A}_{IK}^T = {}_I \tilde{\boldsymbol{\omega}}_{KI} \quad \text{and} \quad \mathbf{A}_{IK}^T \dot{\mathbf{A}}_{IK} = {}_K \tilde{\boldsymbol{\omega}}_{IK} =: \tilde{\boldsymbol{\omega}}_K . \quad (2.13)$$

The operator  $\tilde{\mathbf{a}}$  represents the matrix-equivalent to the cross-product, i. e.,  $\tilde{\mathbf{a}} \mathbf{b} = \mathbf{a} \times \mathbf{b}$ . Equation (2.11) then becomes

$${}_I \dot{\mathbf{r}}_P = {}_I \dot{\mathbf{r}}_O + {}_I \tilde{\boldsymbol{\omega}}_{KI} \mathbf{r}_{OP} . \quad (2.14)$$

**Accelerations** Repeating the time derivative of  $\dot{\mathbf{r}}_P$  from (2.11) and using (2.13) we obtain the acceleration:

$${}_I \ddot{\mathbf{r}}_P = \frac{d}{{}_I dt} \dot{\mathbf{r}}_P = {}_I \ddot{\mathbf{r}}_O + \underbrace{({}_I \dot{\boldsymbol{\omega}} + \tilde{\boldsymbol{\omega}} \tilde{\boldsymbol{\omega}})}_{\dot{\mathbf{A}}_{IK} \mathbf{A}_{IK}^T} \mathbf{r}_{OP} . \quad (2.15)$$

**Transformation** By applying a coordinate transformation, above quantities can always be expressed relative to a different FoR, e. g., for the body-fixed frame  $K$  we



$${}_I\dot{\mathbf{r}} = \mathbf{A}_{IK} {}_K\dot{\mathbf{r}}$$

**Figure 2.4:** Notation for transformation matrix  $\mathbf{A}_{IK}$  transforming a vector from FoR  $K$  to  $I$

have

$${}_K\dot{\mathbf{r}}_P = {}_K\dot{\mathbf{r}}_O + {}_K\tilde{\boldsymbol{\omega}}_{IK} {}_K\mathbf{r}_{OP}, \quad (2.16)$$

$${}_K\ddot{\mathbf{r}}_P = {}_K\ddot{\mathbf{r}}_O + ({}_K\dot{\tilde{\boldsymbol{\omega}}} + \tilde{\boldsymbol{\omega}}\tilde{\boldsymbol{\omega}})_{IK} {}_K\mathbf{r}_{OP}, \quad (2.17)$$

with

$${}_K\dot{\mathbf{r}}_O = \mathbf{A}_{KI} {}_I\dot{\mathbf{r}}_O, \quad {}_K\tilde{\boldsymbol{\omega}}_{IK} = \mathbf{A}_{KI} {}_I\tilde{\boldsymbol{\omega}}_{KI} \mathbf{A}_{IK},$$

$${}_K\ddot{\mathbf{r}}_O = \mathbf{A}_{KI} {}_I\ddot{\mathbf{r}}_O, \quad {}_K\dot{\tilde{\boldsymbol{\omega}}}_{IK} = \mathbf{A}_{KI} {}_I\dot{\tilde{\boldsymbol{\omega}}}_{KI} \mathbf{A}_{IK}.$$

Since absolute velocities and accelerations are independent of the FoR, leading subscripts are neglected in the following where appropriate.

## 2.3 Rigid Body Dynamics

The equations of motion (EoM) of a multibody system can be derived in various ways. BREMER leads a detailed discussion about this topic including historical aspects on dynamics equations in [25, 27]. Basically, two main approaches can be distinguished: *analytical methods* work by analyzing energy [114], while *synthetic methods* start with linear and angular momentum for each body [148]. Obviously, both ways lead to the same result. The derivation effort, however, might differ. Since synthetic methods are easier to represent in the form of an algorithm, they are dominant in computer programs.

With reference to Figure 2.5, where impressed forces let a body move, the EoM of a single rigid body are derived. Due to NEWTON translational motion is described using the *linear momentum theorem*:

$$\int_B \ddot{\mathbf{r}}_P dm = \sum_k^{n_l} \mathbf{F}_k^i, \quad (2.18)$$

where  $\mathbf{F}_k^i$  are the  $n_l$  impressed forces (denoted by superscript  $i$ ). The integral over body  $B$  in (2.18) is solved by substituting (2.15) and using both mass ( $m = \int_B dm$ )

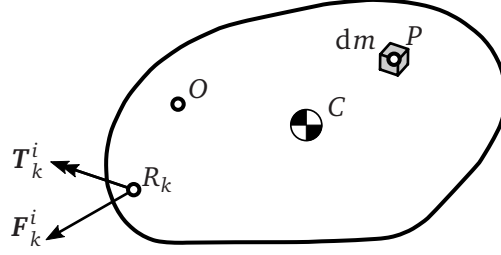


Figure 2.5: Impressed forces acting on a body (adopted from [56])

and CoG position ( $m\mathbf{r}_{OC} = \int_B \mathbf{r}_{OP} dm$ ) of a rigid body

$$m \left( \ddot{\mathbf{r}}_O + \left( \dot{\tilde{\boldsymbol{\omega}}} + \tilde{\boldsymbol{\omega}} \tilde{\boldsymbol{\omega}} \right) \mathbf{r}_{OC} \right) = \sum_k^{n_l} \mathbf{F}_k^i. \quad (2.19)$$

This effectively transforms body  $B$  to a particle with mass  $m$ , located at CoG  $C$  where all impressed forces are acting.

Due to EULER rotational motion is described using the *angular momentum theorem* with respect to the moving point  $O$  (cf. Fig. 2.5):

$$\int_B (\tilde{\mathbf{r}}_{OP} \ddot{\mathbf{r}}_P) dm = \sum_k^{n_l} (\tilde{\mathbf{r}}_{OR,k} \mathbf{F}_k^i + \mathbf{T}_k^i), \quad (2.20)$$

where  $\mathbf{T}_k^i$  are the impressed moments and  $\mathbf{r}_{OR,k}$  denotes the vector from point  $O$  to reference point  $R_k$  where force  $\mathbf{F}_k^i$  acts on the body. Again, the integral is solved by substitution of (2.15) and assuming a rigid body:

$$m\tilde{\mathbf{r}}_{OC} \ddot{\mathbf{r}}_O + \mathbf{I}_O \dot{\boldsymbol{\omega}} + \tilde{\boldsymbol{\omega}} \mathbf{I}_O \boldsymbol{\omega} = \sum_k^{n_l} (\tilde{\mathbf{r}}_{OR,k} \mathbf{F}_k^i + \mathbf{T}_k^i), \quad (2.21)$$

where  $\mathbf{I}_O := - \int_B (\tilde{\mathbf{r}}_{OP} \tilde{\mathbf{r}}_{OP}) dm$  is the *mass-moment of inertia* tensor of the rigid body with respect to point  $O$ . Common 3-D CAD programs provide functionality to evaluate the integral to calculate  ${}_K \mathbf{I}_O$  with respect to an arbitrary point  $O$  and FoR  $K$  of a part or an assembly. Similarly, the computation for mass and CoG position is provided by CAD programs. Obviously, for a body-fixed FoR  $K$  these quantities are constant.

Combining (2.19) and (2.21) leads to the NEWTON-EULER equation of motion (EoM) for a single rigid body:

$$\begin{bmatrix} \mathbf{I}_O & m\tilde{\mathbf{r}}_{OC} \\ m\tilde{\mathbf{r}}_{OC}^\top & m\mathbf{E}_3 \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{\omega}} \\ \ddot{\mathbf{r}}_O \end{bmatrix} + \begin{bmatrix} \tilde{\boldsymbol{\omega}} \mathbf{I}_O \boldsymbol{\omega} \\ m\tilde{\boldsymbol{\omega}} \tilde{\boldsymbol{\omega}} \mathbf{r}_{OC} \end{bmatrix} = \sum_k^{n_l} \begin{bmatrix} \tilde{\mathbf{r}}_{OR,k} \mathbf{F}_k^i + \mathbf{T}_k^i \\ \mathbf{F}_k^i \end{bmatrix}, \quad (2.22)$$

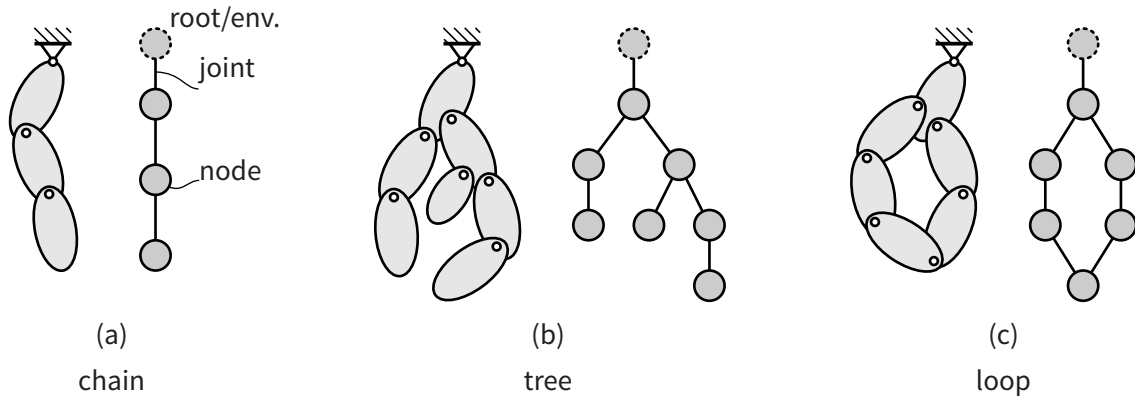


Figure 2.6: Examples of mechanisms (left) and their topological graph (right).

$$M\mathbf{a} + \mathbf{h} = \mathbf{f} , \quad (2.23)$$

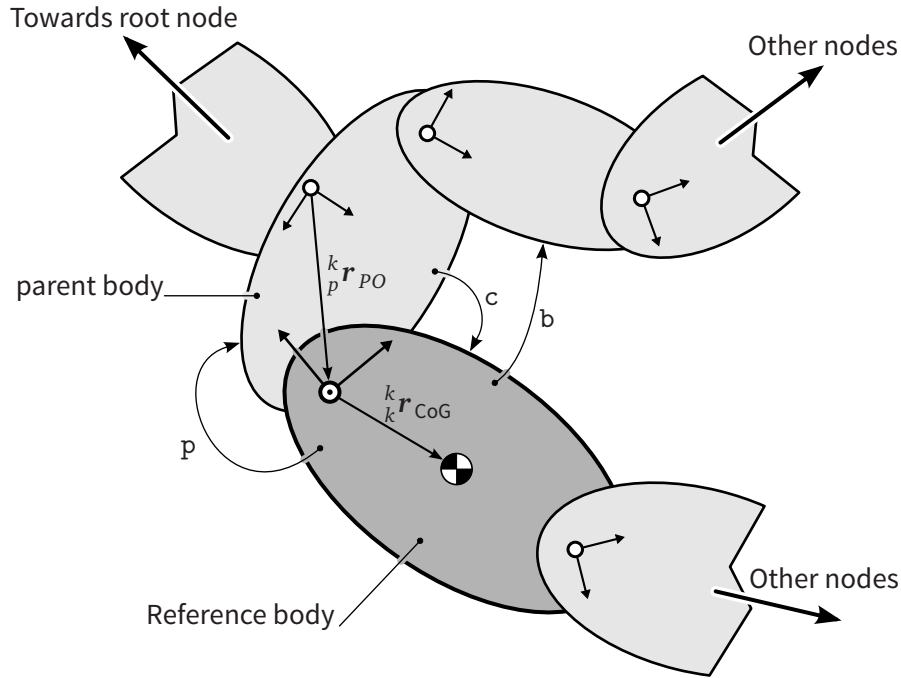
where  $M \in \mathbb{R}^{6 \times 6}$  is the mass-matrix,  $\mathbf{a} \in \mathbb{R}^6$  contains the accelerations,  $\mathbf{h} \in \mathbb{R}^6$  and  $\mathbf{f} \in \mathbb{R}^6$  are the vector of velocity-dependent inertial forces and impressed forces, respectively.

## 2.4 Relative Kinematics of Rigid Multibody Systems

The special topology of robotic systems facilitate an efficient recursive calculation of kinematic quantities. Thereby, bodies are connected by joints which exhibit motion constraints.

### 2.4.1 Topology of Multibody Systems

Figure 2.6 illustrates different kinematic configurations of multibody systems. Formally, topologies in Figure 2.6 are *graphs* where bodies and joints correspond to *nodes* and *edges*, respectively. A *tree* is a *directed acyclic graph* where each node has exactly one predecessor, called *parent* (cf. Fig. 2.6(b)). On the other hand, a *loop* is a *cyclic graph* where the parent relations are not uniquely described (cf. Fig. 2.6(c)). Exactly one node, the *root* node, exists which has no parent. Typically, the root node is fixed, i. e., the environment. For systems having only a single tree, the root node might also be defined as the first body in the graph. In general, nodes can have multiple successors or *child* nodes. The *chain* topology, shown in Fig. 2.6(a), is a special tree where nodes have at most one child node. A *branch* is defined as a sub-graph of a tree spanned by a node, that is, the root node of the branch. A *leaf* node is a node which has no children, i. e., the “last” nodes of a tree. Loop topologies can be converted to a tree by cutting one suitable joint of the loop open and replacing the joint by constraint forces. In general, loops increase the computation effort compared to tree topologies.



**Figure 2.7:** Local description of tree structure with pointers to parent (p), child (c) and brother (b) nodes and vectors to the center of gravity  ${}^k_k \mathbf{r}_{CoG}$  and from parent to child node  ${}^k_p \mathbf{r}_{PO}$  (adopted from [28]).

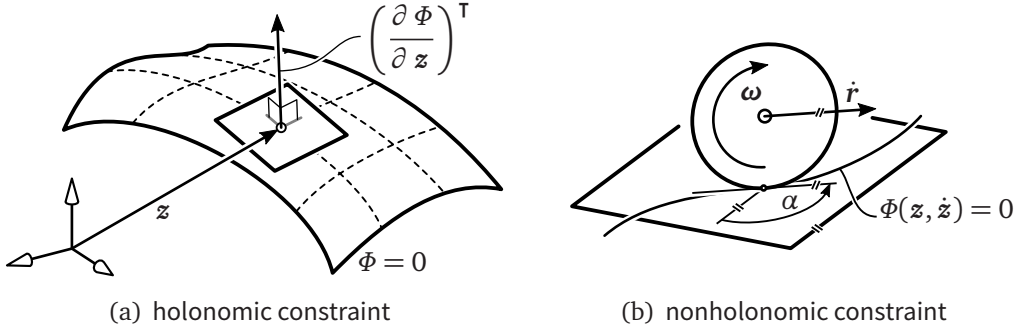
Different methods for describing the connectivity relations suitable for computer representation are known from graph theory [8, 19]. Most graph representations for modeling MBS are based on continuous numbering of bodies  $i \in \mathcal{N}$  where  $\mathcal{N} = \{0, \dots, N_{\text{bodies}}-1\}$  is the set of nodes from the whole system. By using the assigned number as index in a list, provides easy access to bodies. The numbering works by starting with 0 for the root node and assigning successors numbers in ascending order, thereby, each node  $i \in \mathcal{N}$  has a larger number than its parent [25, 53, 68]. Additionally, joints can be numbered in a similar fashion.

Based on the “lower body array”, proposed by HUSTON [68], a topology matrix can be built [25]. Each body is associated to a column containing the path along parents leading to the root node.

One of the simplest ways to describe arbitrary topologies is to generate an index list of predecessors  $p \in \mathbb{N}^{N_{\text{bodies}}}$  and successors  $s \in \mathbb{N}^{N_{\text{bodies}}}$  [53, 59]. The parent of body  $i$  is then easily obtained by  $p(i)$ . On the other hand, each node  $i$  holds a set of children denoted by  $c(i)$ .

Another elegant way for describing tree topologies by means of C/C++ pointers<sup>6</sup> is presented by YAMANE [156] and BUSCHMANN [28]. Locally, each node keeps three

<sup>6</sup> A C/C++ pointer contains a memory address, hence it “points” to data (structures) located in computer memory.



**Figure 2.8:** Graphical interpretation of allowed motion subject to constraints.

references: to a parent ‘p’, child ‘c’ and brother ‘b’ node. Thereby, ‘p’ points towards the root node, ‘c’ points to leaf nodes and ‘b’ points to the next child of the parent. Obviously, a node and its brother ‘b’ share the same parent node. Figure 2.7 illustrates the local pointer-based description of the tree structure.

In this thesis, the pointer approach is combined with a simple list of bodies, having a strict ordering, i.e., each body appears always after its parent in the list. This enables fast access to bodies in forward- and backward-recursions to evaluate quantities.

### 2.4.2 Motion Constraints and Minimal Coordinates

Typically, multibody mechanisms, such as robots, are subject to nonlinear constraints induced by joints. Nonlinear constraints limit the motion represented by  $\mathbf{z}$ :

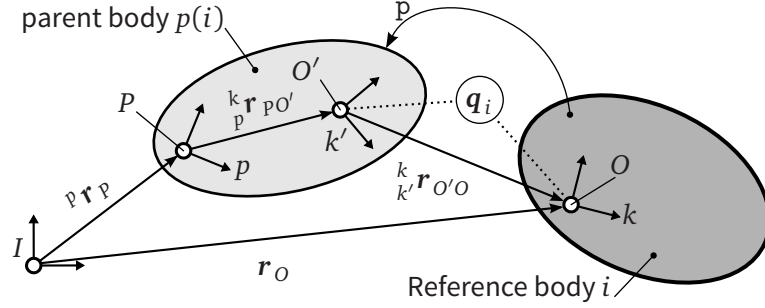
$$\Phi = \mathbf{0}, \quad \Phi := \Phi(\mathbf{z}) : \mathbb{R}^{6N_{\text{Bodies}}} \rightarrow \mathbb{R}^{N_{\text{DoFs}}}. \quad (2.24)$$

Here,  $N_{\text{Bodies}}$  and  $N_{\text{DoFs}}$  denote the number of bodies and DoFs, respectively. Vector  $\mathbf{z} \in \mathbb{R}^{6N_{\text{Bodies}}}$  contains position and orientation for each body. Figure 2.8a illustrates this: motion is only allowed tangentially to the constraint manifold  $\Phi$  on the current configuration  $\mathbf{z}$ . A set of minimal (or: generalized) coordinates  $\mathbf{q}$ , which satisfy the constraints implicitly, are found by eliminating the constraints. Therefore, minimal coordinates  $\mathbf{q}$  uniquely describe the motion in a non-redundant way. Since we typically have  $\dim(\mathbf{z}) \gg \dim(\mathbf{q})$  for constrained MBS, the computational effort is reduced significantly by using a minimal representation.

For nonholonomic constraints  $\Phi(\mathbf{z}, \dot{\mathbf{z}}) = \mathbf{0}$ , generalized velocities  $\dot{\mathbf{s}}$  are introduced:

$$\dot{\mathbf{s}} = \frac{\partial \mathbf{s}}{\partial \mathbf{q}} \dot{\mathbf{q}} = \frac{\partial \dot{\mathbf{s}}}{\partial \dot{\mathbf{q}}} \dot{\mathbf{q}}. \quad (2.25)$$

Nonholonomic constraints limit velocities without affecting positions [25]. A typical example are rolling (non-slipping) wheels, where sufficient friction prohibits motion orthogonal to the rolling direction (cf. Fig. 2.8b).



**Figure 2.9:** Relative joint kinematics of successive bodies. Intermediate FoR  $k'$  and point  $O'$  are thereby fixed to parent  $p(i)$ .

In the special case of holonomic constraints Jacobian matrix  $\partial s / \partial \mathbf{q}$  degenerates to the identity matrix  $\mathbf{E}$  and hence, we get  $\dot{\mathbf{s}} = \dot{\mathbf{q}}$  and  $\mathbf{s} = \mathbf{q}$  [25, 27]. In this thesis, only holonomic constraints are considered. However, by including relationship (2.25) nonholonomic constraints can easily be integrated. For a detailed discussion of MBS with nonholonomic constraints see [25, 53]. Furthermore, only *scleronomic* constraints (without explicit time dependency) are considered in this thesis.

The parent-child-relations in tree structures (cf. Section 2.4.1) are used to generate a strict ordering of  $\mathbf{q}$ . Moreover, each joint  $i$  having  $n_{q,i}$  DoFs is associated to a subset of  $\mathbf{q}$  such that

$$\mathbf{q}_i = [q_i \ \dots \ q_{i+n_{q,i}}]^T \in \mathbb{R}^{n_{q,i}} \quad \text{and} \quad \mathbf{q}_i \in \mathbf{q}. \quad (2.26)$$

### 2.4.3 Recursive Kinematics Calculation

The minimal coordinate representation allows us to express kinematic quantities as functions of generalized coordinates  $\mathbf{q}$  and therefore via the relative motion between parent and child bodies.

With reference to Figure 2.9, body  $i$  moves relative to a reference position  $O'$  and reference orientation  $k'$  both located on parent  $p(i)$ . Relative motion is prescribed using the set of minimal coordinates  $\mathbf{q}_i$ . Depending on the joint type,  $\mathbf{q}_i$  might contain a translational part  $\mathbf{q}_{T,i}$  and a rotational part  $\mathbf{q}_{R,i}$ . Hence, for the relative position and orientation we have

$$\mathbf{r}_{O'O} = \mathbf{r}_{O'O}(\mathbf{q}_{T,i}) \quad \text{and} \quad \mathbf{A}_{k'k} = \mathbf{A}_{k'k}(\mathbf{q}_{R,i}). \quad (2.27)$$

Using the relative Jacobians of translation and rotation

$$\mathbf{J}_{T,i} = \frac{\partial \dot{\mathbf{r}}_{\text{rel},i}}{\partial \dot{\mathbf{q}}_{T,i}} \in \mathbb{R}^{3 \times n_{q,T}} \quad \text{and} \quad \mathbf{J}_{R,i} = \frac{\partial \boldsymbol{\omega}_{\text{rel},i}}{\partial \dot{\mathbf{q}}_{R,i}} \in \mathbb{R}^{3 \times n_{q,R}}, \quad (2.28)$$

we can rewrite kinematic quantities for velocity as

$$\boldsymbol{\omega}_i = \boldsymbol{\omega}_{p(i)} + \mathbf{J}_{R,i} \dot{\mathbf{q}}_{R,i}, \quad \dot{\mathbf{r}}_P = \dot{\mathbf{r}}_P + \tilde{\boldsymbol{\omega}}_{p(i)} \mathbf{r}_{OP} + \mathbf{J}_{T,i} \dot{\mathbf{q}}_{T,i}, \quad (2.29)$$

and acceleration as

$$\dot{\boldsymbol{\omega}}_i = \dot{\boldsymbol{\omega}}_{p(i)} + \mathbf{J}_{R,i} \ddot{\mathbf{q}}_{R,i} + \dot{\mathbf{J}}_{R,i} \dot{\mathbf{q}}_{R,i} + \tilde{\boldsymbol{\omega}}_{p(i)} \mathbf{J}_{R,i} \dot{\mathbf{q}}_{R,i}, \quad (2.30)$$

$$\ddot{\mathbf{r}}_O = \ddot{\mathbf{r}}_P + \dot{\tilde{\boldsymbol{\omega}}}_{p(i)} \mathbf{r}_{PO} + \mathbf{J}_{T,i} \ddot{\mathbf{q}}_{T,i} + \dot{\mathbf{J}}_{T,i} \dot{\mathbf{q}}_{T,i} + \tilde{\boldsymbol{\omega}}_{p(i)} (2\mathbf{J}_{T,i} \dot{\mathbf{q}}_{T,i} + \tilde{\boldsymbol{\omega}}_{p(i)} \mathbf{r}_{PO}). \quad (2.31)$$

#### 2.4.4 Recursive Kinematics using Spatial Vector Notation

A closer look at the EoM (2.22) suggests the use of 6-D vectors and matrices to describe dynamics quantities. This spatial vector notation uses 6-D vectors and 6-D tensors for describing rigid-body velocity, acceleration and inertia. The notation can be extended in a natural way to include other quantities, e. g., for elastic robots and/or sub-systems (cf. [58, 130]).

Three-dimensional angular and translational quantities for velocity and acceleration are combined to vectors in  $\mathbb{R}^6$  such that

$$\mathbf{v} = \begin{bmatrix} \boldsymbol{\omega} \\ \dot{\mathbf{r}} \end{bmatrix} \in \mathbb{R}^6 \quad \text{and} \quad \mathbf{a} = \begin{bmatrix} \dot{\boldsymbol{\omega}} \\ \ddot{\mathbf{r}} \end{bmatrix} \in \mathbb{R}^6, \quad (2.32)$$

where  $\mathbf{v}$  and  $\mathbf{a}$  are the spatial velocity and spatial acceleration, respectively.

This facilitates a compact formulation of the EoM. However, in a real implementation, when computational performance is important, operations involving multiplications with zeros may be pruned. This, in turn, leads to the classical representation with separate vectors in  $\mathbb{R}^3$  describing translations and rotations.

Using spatial vector notation, kinematic quantities in (2.29) to (2.31) for the  $i$ -th body denoted in FoR  $k$  write to

$$\mathbf{v}_i = \mathbf{C}_i \mathbf{v}_{p(i)} + \mathbf{J}_i \dot{\mathbf{q}}_i, \quad (2.33)$$

$$\mathbf{a}_i = \frac{d \mathbf{v}_i}{dt} = \mathbf{C}_i \mathbf{a}_{p(i)} + \mathbf{J}_i \ddot{\mathbf{q}}_i + \underbrace{\dot{\mathbf{C}}_i \mathbf{v}_{p(i)} + \dot{\mathbf{J}}_i \dot{\mathbf{q}}_i}_{\mathbf{b}_i}. \quad (2.34)$$

Here, the subscripts denote the associated body  $i$  and its parent  $p(i)$ , respectively. Thereby, all quantities are written with respect to their own body-fixed FoR. The relative Jacobian and its rate of change are defined as

$$\mathbf{J}_i := \frac{\partial \mathbf{v}_{\text{rel},i}}{\partial \dot{\mathbf{q}}_i} = \begin{pmatrix} {}^{k'}\mathbf{J}_{R,i} \\ {}^{k'}\mathbf{J}_{T,i} \end{pmatrix} \in \mathbb{R}^{6 \times N_{q_i}}, \quad \dot{\mathbf{J}}_i := \frac{d \mathbf{J}_i}{dt} = \frac{\partial \mathbf{v}_{\text{rel},i}}{\partial \mathbf{q}_i} \in \mathbb{R}^{6 \times N_{q_i}}, \quad (2.35)$$

denoted here with respect to FoR  $k'$  which is fixed to the parent (cf. Figure 2.9). Matrix

$\mathbf{C}_i$  is the spatial transformation matrix

$$\mathbf{C}_i := \mathbf{C}_{kp} = \begin{bmatrix} \mathbf{A}_{kp} & \mathbf{0} \\ -\mathbf{A}_{kp} \tilde{\mathbf{r}}_{PO} & \mathbf{A}_{kp} \end{bmatrix}, \quad (2.36)$$

where  $\mathbf{A}_{kp} \in \mathbb{R}^{3 \times 3}$  is the rotation matrix from parent FoR to body-fixed FoR, i.e.,  $\mathbf{A}_{kp} = \mathbf{A}_{kk'} \mathbf{A}_{k'p}$ , and vector  $\mathbf{r}_{PO}$  is the vector between the origins of parent and child (cf. Fig. 2.7). The velocity-dependent part of the acceleration (2.34) is written using identity (2.13) to

$$\mathbf{b}_i = \left[ \tilde{\boldsymbol{\omega}}_{p(i)} \left( 2 \mathbf{J}_{T,i} \dot{\mathbf{q}}_{T,i} + \tilde{\boldsymbol{\omega}}_{p(i)} \mathbf{r}_{PO} \right) \right] + \mathbf{J}_{J,i} \dot{\mathbf{q}}_i. \quad (2.37)$$

Quantities  $\mathbf{v}_i$  and  $\mathbf{a}_i$  in (2.33) and (2.34) are absolute quantities denoted in the body-fixed FoR  $k$ .

### Joint Model Examples

Examples for the quantities used in Equations (2.33) – (2.36) are given here for some joint types. Where appropriate, body index  $i$  is omitted for conciseness.

**6-DoF Joint** Unconstrained (free) motion is described using a 6-DoF joint with three rotations and translations respectively. Different representations are known to describe the orientation of a rigid body. An obvious way is to use a set of three parameters (minimal coordinates)  $\mathbf{q}_{R,i} := [q_1 \ q_2 \ q_3]^T$ , e. g., representing angles of subsequent elementary rotations about different axes. Some of the most widely used rotation representations include

**Euler angles** three subsequent elementary rotations where the first and last (resulting) axis of rotation are identical, e. g.,  $\mathbf{A}_{kl} = \mathbf{A}_z(q_3) \mathbf{A}_x(q_2) \mathbf{A}_z(q_1)$

**Tait-Bryan angles** (sometimes called *Cardan angles*, *nautical angles*, etc.) three subsequent elementary rotations each about different axes (often used in flight dynamics [120])

Since the use of three angle parameters inherently leads to singularities in the mapping between minimal velocities and angular velocities, alternative representations with redundant parameters might be necessary.<sup>7</sup> Some of them are *rotation vector representations* (e. g., *Euler-Rodrigues parameters* and *axis-angle representation*) [25] or *six-parameter methods* [120]. However, in dynamics applications the computational cost is higher with increasing redundancy, as pointed out by ROBERSON et al. in [120].

<sup>7</sup> Obviously, the dimension of  $\mathbf{q}_{R,i}$  increases with the degree of redundancy.



On the other hand, translations typically have a direct mapping between coordinates and parameters, i.e.,  ${}_I \mathbf{r} = \mathbf{q}_{T,i} := [q_4 \ q_5 \ q_6]^\top$ .

In this thesis, the *Euler-ZXZ angle representation* (cf. Appendix C, Fig. C.2) is used to describe rotations for unconstrained motion. The set of minimal coordinates is then defined by  $\mathbf{q}_i := [\mathbf{q}_{R,i}^\top \ \mathbf{q}_{T,i}^\top]^\top$ , leading to the following quantities:

$$\mathbf{J} = \begin{bmatrix} {}_k \mathbf{J}_R & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{kI} \mathbf{J}_T \end{bmatrix} \in \mathbb{R}^{6 \times 6}, \quad (2.38)$$

with

$${}_k \mathbf{J}_R = \frac{\partial \boldsymbol{\omega}}{\partial \dot{\mathbf{q}}_{R,i}} = \begin{bmatrix} s_2 s_3 & c_3 & 0 \\ s_2 c_3 & -s_3 & 0 \\ s_2 & 0 & 1 \end{bmatrix}, \quad {}_I \mathbf{J}_T = \frac{\partial \dot{\mathbf{r}}}{\partial \dot{\mathbf{q}}_{T,i}} = \mathbf{E} \in \mathbb{R}^{3 \times 3}, \quad (2.39)$$

$$(2.40)$$

and

$$\mathbf{A} = \mathbf{A}_{kI} = \mathbf{A}_{\text{EulerZXZ}}^\top(\mathbf{q}_{R,i}) = \begin{bmatrix} c_1 c_3 - s_1 c_2 s_3 & s_1 c_3 + c_1 c_2 s_3 & s_2 s_3 \\ -c_1 s_3 - s_1 c_2 c_3 & -s_1 s_3 + c_1 c_2 c_3 & s_2 c_3 \\ s_1 s_2 & -c_1 s_2 & c_2 \end{bmatrix}. \quad (2.41)$$

Here,  $s_j := \sin(q_j)$  and  $c_j := \cos(q_j)$  for  $j = \{1, 2, 3\}$  are used as abbreviations. Furthermore, for the time derivative of the joint Jacobian and velocity-dependent acceleration we have

$$\mathbf{j} = \left[ \begin{array}{ccc|c} \dot{q}_2 c_2 s_3 + \dot{q}_3 s_2 c_3 & -\dot{q}_3 s_3 & 0 & \mathbf{0} \\ \dot{q}_2 c_2 c_3 - \dot{q}_3 s_2 s_3 & -\dot{q}_3 c_3 & 0 & \mathbf{0} \\ -\dot{q}_2 s_2 & 0 & 0 & \mathbf{0} \\ \hline \mathbf{0} & & & \mathbf{0} \end{array} \right], \quad (2.42)$$

$$\mathbf{b} = \mathbf{j} \dot{\mathbf{q}}_i. \quad (2.43)$$

Finally, by setting  $\mathbf{r}_{PO} = \mathbf{0}$  and defining the initial position of the joint via  $\mathbf{q}_{T,i}$ , the spatial transformation matrix  $\mathbf{C}$  in (2.36) reduces to

$$\mathbf{C} = \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{A} \end{bmatrix}, \quad (2.44)$$

which can be exploited for computational efficiency.

**Revolute Joint** The revolute joint has one rotational DoF about the  $z$ -axis denoted by  $q$ . Hence, quantities used in (2.33) to (2.36) become

$$\mathbf{J} = [0 \ 0 \ 1 \ 0 \ 0 \ 0]^\top = \text{const.}, \quad \Rightarrow \quad \dot{\mathbf{J}} = \mathbf{0} \in \mathbb{R}^{6 \times 1}, \quad (2.45)$$

$$\mathbf{A}_{kp} = \mathbf{A}_z(q) \mathbf{A}_{k'p} = \begin{bmatrix} \cos q & \sin q & 0 \\ -\sin q & \cos q & 0 \\ 0 & 0 & 1 \end{bmatrix} \mathbf{A}_{k'p}, \quad (2.46)$$

where  $\mathbf{A}_{k'p}$  is a constant initial transformation matrix (cf. Figure 2.9). Matrix  $\mathbf{A}_{k'p}$  (and vector  $\mathbf{r}_{pO}$ ) can be obtained from given DENAVIT-HARTENBERG parameters.<sup>8</sup>

**Prismatic Joint** Similar to the revolute joint, the prismatic joint has one translational DoF along the  $z$ -axis denoted by  $q$  and we get

$$\mathbf{J} = [0 \ 0 \ 0 \ 0 \ 0 \ 1]^\top = \text{const.}, \quad \Rightarrow \quad \dot{\mathbf{J}} = \mathbf{0} \in \mathbb{R}^{6 \times 1}, \quad (2.47)$$

$$\mathbf{A} = \mathbf{A}_{k'p} = \text{const.}, \quad (2.48)$$

$$\mathbf{b} = \begin{bmatrix} \mathbf{0} \\ \tilde{\boldsymbol{\omega}}_{p(i)} \left( 2 \mathbf{J}_{T,i} \dot{\mathbf{q}}_{T,i} + \tilde{\boldsymbol{\omega}}_{p(i)} \mathbf{r}_{pO} \right) \end{bmatrix}. \quad (2.49)$$

BUSCHMANN implemented some of above joint models used for the robots Johnnie and Lola directly as body-type classes Body3R3T and Body1R having 6 DoFs and 1 DoF, respectively [28]. These classes were extended in this work.

## Further Reading

For more examples of joint types, the reader is referred to one of the many text books about rigid body dynamics. Therein, the quantities might have different names, e. g., the columns of the relative spatial Jacobian  $\mathbf{J}$  are called “mode vectors” [120] or “motion subspace” [53]. Due to its formal derivation, however, the term “relative spatial Jacobian” seems more accurate.

Moreover, in Chapter 3 quantities for special joint types used for the robot Lola are presented.

8 1) rotation and translation about the  $x$ -axis of parent’s FoR using angle  $\alpha$  and length  $a$ , respectively; 2) translation along the new  $z$ -axis using length  $d$ . Hence, we also get vector  $\mathbf{r}_{pO}$ . For details, see [40].

## 2.5 Dynamics of Rigid Multibody Systems

Using EoM (2.23) for each body  $i$  of the whole system leads to

$$\mathbf{M}_i \mathbf{a}_i + \mathbf{h}_i = \mathbf{f}_i^i + \mathbf{f}_i^c, \quad \text{for } i = 1 \dots N_{\text{Bodies}}. \quad (2.50)$$

Impressed forces  $\mathbf{f}^i$  include external forces such as from contacts, gravity, motor torque and gear friction. Due to joint constraints, constraint forces  $\mathbf{f}_i^c$  can be separated from impressed forces. Equation (2.50) involves solving for 12 unknowns in  $\mathbf{a}_i$  and  $\mathbf{f}_i^c$  for each of the  $N_{\text{Bodies}}$  bodies of the system.

Rewriting (2.50) gives

$$\overline{\overline{\mathbf{M}}} \overline{\overline{\mathbf{a}}} - \overline{\overline{\mathbf{f}}} = \mathbf{0}, \quad (2.51)$$

with the mass-matrix  $\overline{\overline{\mathbf{M}}} = \text{block diag}(\mathbf{M}_i) \in \mathbb{R}^{6N_b \times 6N_b}$  and vectors

$$\overline{\overline{\mathbf{a}}} = \text{stack}(\mathbf{a}_i), \quad \overline{\overline{\mathbf{f}}} = \text{stack}(\mathbf{f}_i^i + \mathbf{f}_i^c - \mathbf{h}_i), \quad \overline{\overline{\mathbf{a}}}, \overline{\overline{\mathbf{f}}} \in \mathbb{R}^{6N_b},$$

where operator  $\text{stack}(\mathbf{a}_i) = [\mathbf{a}_1^\top \ \mathbf{a}_2^\top \ \dots \ \mathbf{a}_{N_b}^\top]^\top$  stacks individual quantities above each other.

By multiplying (2.51) from the left with the *global* Jacobian matrix

$$\overline{\overline{\mathbf{J}}} = \begin{pmatrix} \frac{\partial \mathbf{v}_1}{\partial \mathbf{q}} \\ \vdots \\ \frac{\partial \mathbf{v}_{N_b}}{\partial \mathbf{q}} \end{pmatrix} = \begin{pmatrix} \overline{\overline{\mathbf{J}}}_1 \\ \vdots \\ \overline{\overline{\mathbf{J}}}_{N_b} \end{pmatrix} \in \mathbb{R}^{6N_b \times N_q}, \quad (2.52)$$

we obtain the minimal form

$$\overline{\overline{\mathbf{J}}}^\top (\overline{\overline{\mathbf{M}}} \overline{\overline{\mathbf{a}}} - \overline{\overline{\mathbf{f}}}^*) = \mathbf{Q}. \quad (2.53)$$

Since the pre-multiplication of  $\overline{\overline{\mathbf{J}}}^\top$  is a projection onto the unconstrained directions, this effectively eliminates constraint forces:

$$\overline{\overline{\mathbf{J}}}^\top \overline{\overline{\mathbf{f}}}^c = \mathbf{0}, \quad \text{with} \quad \overline{\overline{\mathbf{f}}}^c = \text{stack}(\mathbf{f}_i^c). \quad (2.54)$$

Hence, the modified force vector  $\overline{\overline{\mathbf{f}}}^*$  becomes

$$\overline{\overline{\mathbf{f}}}^* = [\mathbf{f}_1^\top \ \mathbf{f}_2^\top \ \dots \ \mathbf{f}_{N_b}^\top]^\top, \quad (2.55)$$

with

$$\mathbf{f}_i = \mathbf{f}_i^i - \mathbf{h}_i = \underbrace{\begin{bmatrix} m \tilde{\mathbf{r}}_{OC} \mathbf{g}_e \\ m \mathbf{g}_e \end{bmatrix}}_{\mathbf{f}_{g,i}} + \begin{bmatrix} \sum_j^{N_T} \mathbf{T}_j^i + \sum_k^{N_R} \tilde{\mathbf{r}}_{OR,k} \mathbf{F}_k^i \\ \sum_k^{N_F} \mathbf{F}_k^i \end{bmatrix}_i - \underbrace{\begin{bmatrix} \tilde{\boldsymbol{\omega}} \mathbf{I}_O \boldsymbol{\omega} \\ m \tilde{\boldsymbol{\omega}} \tilde{\boldsymbol{\omega}} \mathbf{r}_{OC} \end{bmatrix}}_{\mathbf{h}_i}. \quad (2.56)$$

Force vector  $\mathbf{f}_i$  combines different forces such as the gravity forces  $\mathbf{f}_{g,i}$  and other impressed forces (cf. Fig. 2.5) as well as the velocity-dependent mass-forces  $\mathbf{h}_i$ .

The vector of generalized forces  $\mathbf{Q}$  is used in (2.53) for convenience if the left hand side of  $\mathbf{Q} = \bar{\mathbf{J}}^T \bar{\mathbf{f}}^q$  is calculated directly. Hence, forces are applied directly into unconstrained directions, e. g., for drive and/or friction forces.

Formally, (2.53) can be derived using JORDAIN's principle [27, 148] leading to<sup>9</sup>

$$\sum_i^{N_b} \left\{ \left[ \left( \frac{\partial \boldsymbol{\omega}}{\partial \dot{\mathbf{q}}} \right)^T \left( \frac{\partial \dot{\mathbf{r}}_{OC}}{\partial \dot{\mathbf{q}}} \right)^T \right] \begin{bmatrix} \dot{\mathbf{L}} + \tilde{\boldsymbol{\omega}}_{IR} \mathbf{L} - \mathbf{T}^i \\ \dot{\mathbf{p}} + \tilde{\boldsymbol{\omega}}_{IR} \mathbf{p} - \mathbf{F}^i \end{bmatrix} \right\}_i = \mathbf{Q}, \quad (2.57)$$

where  $\mathbf{L}$  and  $\mathbf{p}$  are the angular momentum about the CoG and the linear momentum, respectively.

## 2.6 Sub-Systems

Multibody systems can often be modeled using sub-systems by combining several neighboring nodes of the graph. BREMER introduces sub-systems to reduce the modeling effort of repeating structures in a MBS [25, 27]. Formally, EoM (2.57) is brought into sub-system form by splitting the sum on sub-system boundaries and applying the chain rule of differentiation on the Jacobians leading to [25]

$$\sum_n^{N_{sub}} \left( \frac{\partial \dot{\mathbf{y}}_n}{\partial \dot{\mathbf{q}}} \right)^T \sum_i^{N_{b,sub,i}} \left\{ \left[ \left( \frac{\partial \boldsymbol{\omega}}{\partial \dot{\mathbf{y}}_n} \right)^T \left( \frac{\partial \dot{\mathbf{r}}_{OC}}{\partial \dot{\mathbf{y}}_n} \right)^T \right] \begin{bmatrix} \dot{\mathbf{L}} + \tilde{\boldsymbol{\omega}}_{IR} \mathbf{L} - \mathbf{T}^i \\ \dot{\mathbf{p}} + \tilde{\boldsymbol{\omega}}_{IR} \mathbf{p} - \mathbf{F}^i \end{bmatrix} \right\}_i = \mathbf{Q}. \quad (2.58)$$

Here,  $\dot{\mathbf{y}}$  are the non-minimal coordinates of a sub-system  $S$  containing associated minimal velocities  $\dot{\mathbf{q}}_S$  and reference velocities  $\mathbf{v}_{p(S)}$  from its parent (cf. [25] for more details).

Sub-systems are used in this work to facilitate the application of  $\mathcal{O}(n)$  algorithms with (small) kinematic loops. In this context, kinematic loops are defined as a group of joints which share a set of generalized coordinates  $\mathbf{q}_S$ . Formally, kinematic loops can always be cut open to restore tree topology. Loop closure is then assured by introducing Lagrangian multipliers equivalent to the joint constraint forces together with algebraic loop-closure conditions. This, however, leads to EoMs in form of DAEs. On the other

<sup>9</sup> This form is only introduced here for reference in Section 2.6.

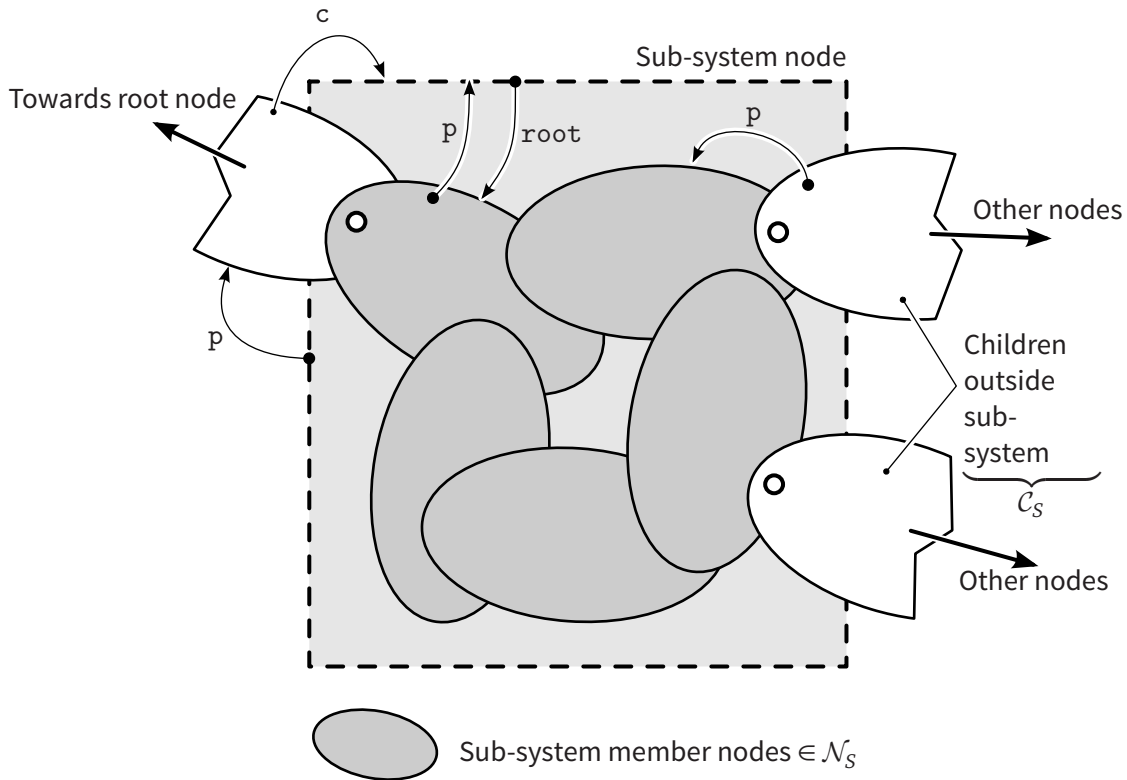


Figure 2.10: Sub-system embedded into a MBS.

hand, if a closed form solution to the constraint equations exists, the loop can be solved directly. This is achieved by building a sub-system “around” those DoF-sharing joints, and hence, maintain the basic form of  $\mathcal{O}(n)$ -algorithms by means of interface quantities. Thereby, algebraic constraint equations are solved inside sub-systems, which leads to a local  $\mathcal{O}(N_{q,\text{sub}}^3)$  complexity, where  $N_{q,\text{sub}} := \dim(\mathbf{q}_S)$  is the number of DoFs associated to sub-system  $S$ . Therefore, both the size of the loop and the effort for algebraic joint equations strongly influence the computational performance of the  $\mathcal{O}(n)$ -algorithm with sub-systems. A detailed derivation of this  $\mathcal{O}(n)$ -algorithm with sub-systems is presented in Section 2.7.

Section 2.9 gives an overview of how groups of joints, which share generalized coordinates, can be converted into sub-systems. This step allows to reconfigure the MBS structure in order to apply the  $\mathcal{O}(n)$ -algorithm.

With reference to Figure 2.10, sub-systems are special bodies on MBS-level. Thereby, sub-systems combine DoFs, and have an internal dynamics, i. e., are comparable to elastic bodies. By following the pointer logic introduced in Section 2.4.1 and Figure 2.7, sub-systems are both a body and a small MBS. Thereby, the first member body is the root of the sub-system-MBS and the sub-system-body is the root’s parent. Successor nodes of a sub-system are directly linked to bodies inside the sub-system. This is mainly

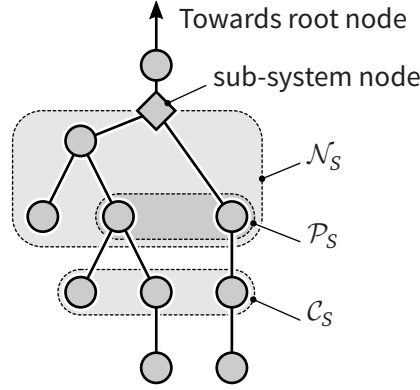


Figure 2.11: Node-sets of a sub-system

to facilitate a recursive calculation of kinematic quantities. A more formal solution is to let the sub-system provide necessary quantities for successors on MBS-level which in turn, however, leads to unnecessary overhead.

With reference to Figure 2.11, each sub-system holds a number of node-sets. Analogously to the node-set  $\mathcal{N}$  of the MBS, each sub-system holds a set of member bodies  $\mathcal{N}_S$ . In addition, a set of child nodes  $\mathcal{C}_S$  exists, which are not part of the sub-system such that  $\mathcal{C}_S = \{i \in \mathcal{N} \wedge i \notin \mathcal{N}_S \mid \exists j \in \mathcal{N}_S : i \in c(j)\}$ . In Figure 2.10, nodes marked as “Children outside sub-system” are in  $\mathcal{C}_S$ . Moreover, the parent nodes from nodes in  $\mathcal{C}_S$  are collected in the set  $\mathcal{P}_S = \{j \in \mathcal{N}_S \mid \exists i \in \mathcal{C}_S : j \in p(i)\}$ .

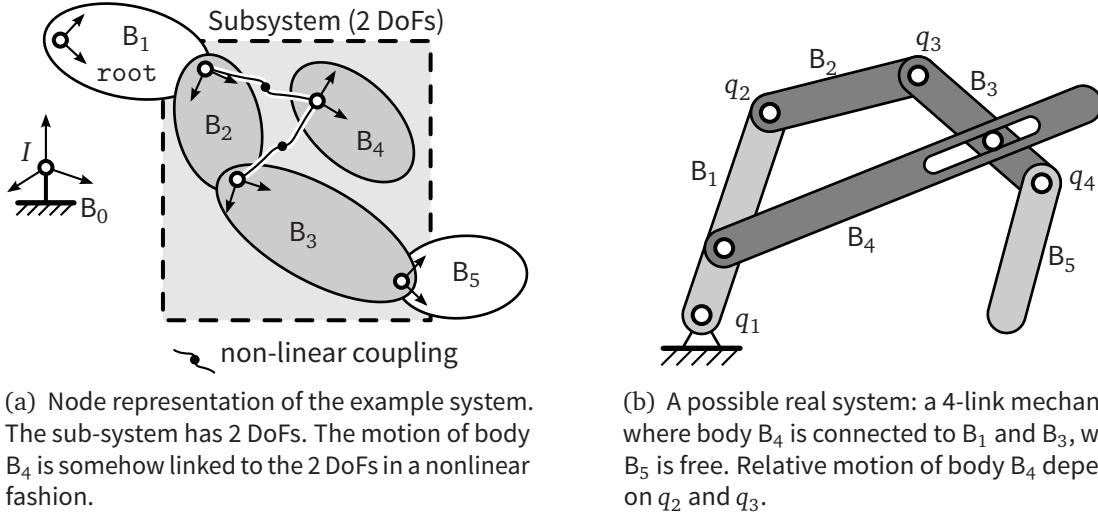
## 2.7 Detailed Derivation of the $\mathcal{O}(n)$ -Algorithm with Sub-Systems

Figure 2.12a is key to a detailed derivation of an  $\mathcal{O}(n)$  formalism for calculating the accelerations of a MBS with integrated sub-systems. The system allows for showing: (a) a sub-system embedded in a MBS, (b) a sub-system with an internal hierarchy, (c) a sub-system with a predecessor and a successor, (d) a sub-system with multiple DoFs, and, (e) a sub-system which contains a joint with multiple DoFs.

This abstract system can be seen as a generalization of the 4-link mechanism shown in Figure 2.12b where the last body  $B_5$  is free. The relative motion of body  $B_4$  is related to the joint angles  $q_2$  and  $q_3$  in a nonlinear fashion.

The vector of generalized coordinates  $\mathbf{q}$  is defined as

$$\mathbf{q} = \begin{pmatrix} q_1^* \\ q_2^* \\ \vdots \\ q_{N_{\text{DoFs}}}^* \end{pmatrix} = \begin{pmatrix} \mathbf{q}_1 \\ \mathbf{q}_2 \\ \mathbf{q}_3 \\ \mathbf{q}_5 \end{pmatrix} \mathbf{q}_4, \quad (2.59)$$



**Figure 2.12:** Example system used for deriving the  $\mathcal{O}(n)$  formalism for a MBS containing a sub-system.

where the relative motion of each body  $B_i$  is influenced by the set  $\mathbf{q}_i$ . By that logic,  $\mathbf{q}_4$  of body  $B_4$  is influenced by  $\mathbf{q}_2$  and  $\mathbf{q}_3$ . Since we want to analyze the general case, quantities  $\mathbf{q}_i$  are denoted in vector notation.

Using (2.34) the stacked absolute accelerations of the *whole* MBS are written as

$$\bar{\mathbf{a}} = \mathbf{C}\bar{\mathbf{a}} + \mathbf{J}\dot{\mathbf{q}} + \mathbf{b}, \quad \bar{\mathbf{a}}, \mathbf{b} \in \mathbb{R}^{6N_{\text{bodies}}}, \quad \mathbf{C} \in \mathbb{R}^{6N_{\text{bodies}} \times 6N_{\text{bodies}}}, \quad \mathbf{J} \in \mathbb{R}^{6N_{\text{bodies}} \times N_{\text{DoFs}}}, \quad (2.60)$$

where  $\mathbf{C}$  is the “geometry matrix” which defines the parent-child-relations and  $\mathbf{J}$  is a block-matrix which contains all the relative Jacobian matrices  $\mathbf{J}_i$ . In the case of a *chain structure*  $\mathbf{C}$  is a block-matrix containing the spatial transformation matrices in the lower secondary diagonal and  $\mathbf{J}$  is a block-diagonal matrix. An example for a chain structure can be found in [123]. Here, we have

$$\mathbf{C} = \begin{array}{ccccc} & \text{parents} & & & \\ & B_1 & B_2 & B_3 & B_4 & B_5 \\ \begin{array}{c} \mathbf{0} \\ \mathbf{C}_2 \\ \mathbf{0} \\ \mathbf{C}_4 \\ \mathbf{0} \end{array} & \begin{array}{c} \mathbf{0} \\ \mathbf{0} \\ \mathbf{C}_3 \\ \mathbf{0} \\ \mathbf{0} \end{array} & \begin{array}{c} \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{C}_5 \end{array} & \begin{array}{c} \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{array} & \begin{array}{c} \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{array} & \begin{array}{c} B_1 \\ B_2 \\ B_3 \\ B_4 \\ B_5 \end{array} \\ & & & & & \text{children} \end{array}$$

$$\mathbf{J} = \begin{array}{cccc} & \text{gen. coord. sets} & & & \\ & \mathbf{q}_1 & \mathbf{q}_2 & \mathbf{q}_3 & \mathbf{q}_5 \\ \begin{array}{c} \mathbf{J}_1 \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{array} & \begin{array}{c} \mathbf{0} \\ \mathbf{J}_2 \\ \mathbf{0} \\ \mathbf{J}_{4,1} \\ \mathbf{0} \end{array} & \begin{array}{c} \mathbf{0} \\ \mathbf{0} \\ \mathbf{J}_3 \\ \mathbf{J}_{4,2} \\ \mathbf{0} \end{array} & \begin{array}{c} \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{J}_5 \end{array} & \begin{array}{c} B_1 \\ B_2 \\ B_3 \\ B_4 \\ B_5 \end{array} \\ & & & & \text{bodies} \end{array}$$

The entries of  $\mathbf{J}$  and  $\mathbf{C}$  have a clear placement. In  $\mathbf{C}$  relative relations are visible:  $B_2$  and  $B_4$  are children of  $B_1$ , body  $B_3$  is a child of  $B_2$ , etc. On the other hand, in  $\mathbf{J}$  relative Jacobians  $\mathbf{J}_i$  span along associated generalized coordinates  $\mathbf{q}_i$  for each body.

Solving the recursive form (2.60) for  $\bar{\mathbf{a}}$  yields the non-recursive form

$$\bar{\mathbf{a}} = (\mathbf{E} - \mathbf{C})^{-1} (\mathbf{J} \ddot{\mathbf{q}} + \mathbf{b}) \quad (2.61a)$$

$$= \underbrace{(\mathbf{E} - \mathbf{C})^{-1} \mathbf{J}}_{\bar{\mathbf{J}}} \ddot{\mathbf{q}} + \bar{\mathbf{b}}, \quad (2.61b)$$

where  $\mathbf{E}$  is the unit matrix with according dimensions. However, it is not intuitive to see what the matrix  $(\mathbf{E} - \mathbf{C})^{-1}$  is, especially for block-matrices. Here, we have

$$(\mathbf{E} - \mathbf{C})^{-1} = \begin{bmatrix} \mathbf{E}_6 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{C}_2 & \mathbf{E}_6 & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{C}_3 \mathbf{C}_2 & \mathbf{C}_3 & \mathbf{E}_6 & \mathbf{0} & \mathbf{0} \\ \mathbf{C}_4 & \mathbf{0} & \mathbf{0} & \mathbf{E}_6 & \mathbf{0} \\ \mathbf{C}_5 \mathbf{C}_3 \mathbf{C}_2 & \mathbf{C}_5 \mathbf{C}_3 & \mathbf{C}_5 & \mathbf{0} & \mathbf{E}_6 \end{bmatrix}, \quad (2.62)$$

which is necessary for calculating both  $\bar{\mathbf{b}} = (\mathbf{E} - \mathbf{C})^{-1} \mathbf{b}$  in (2.61b) and matrix  $\bar{\mathbf{J}}$ :

$$\bar{\mathbf{J}} = (\mathbf{E} - \mathbf{C})^{-1} \mathbf{J} = \begin{array}{c} \text{gen. coord. sets} \\ \mathbf{q}_1 \quad \mathbf{q}_2 \quad \mathbf{q}_3 \quad \mathbf{q}_5 \\ \left[ \begin{array}{cccc|c} \mathbf{J}_1 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \bar{\mathbf{J}}_1 \\ \mathbf{C}_2 \mathbf{J}_1 & \boxed{\mathbf{J}_2} & \mathbf{0} & \mathbf{0} & \bar{\mathbf{J}}_2 \\ \mathbf{C}_3 \mathbf{C}_2 \mathbf{J}_1 & \mathbf{C}_3 \mathbf{J}_2 & \mathbf{J}_3 & \mathbf{0} & \bar{\mathbf{J}}_3 \\ \mathbf{C}_4 \mathbf{J}_1 & \boxed{\mathbf{J}_{4,1}} & \boxed{\mathbf{J}_{4,2}} & \mathbf{0} & \bar{\mathbf{J}}_4 \\ \mathbf{C}_5 \mathbf{C}_3 \mathbf{C}_2 \mathbf{J}_1 & \mathbf{C}_5 \mathbf{C}_3 \mathbf{J}_2 & \mathbf{C}_5 \mathbf{J}_3 & \mathbf{J}_5 & \bar{\mathbf{J}}_5 \end{array} \right] \begin{array}{l} \bar{\mathbf{J}}_1 \\ \bar{\mathbf{J}}_2 \\ \bar{\mathbf{J}}_3 \\ \bar{\mathbf{J}}_4 \\ \bar{\mathbf{J}}_5 \end{array} \end{array} \quad \begin{array}{l} \text{body} \\ \text{Jacobians} \end{array} \quad (2.63)$$

where  $\bar{\mathbf{J}}$  is the global Jacobian matrix formally defined in (2.52). Here, the framed part of the matrix marks the portion associated to the sub-system (which will be introduced later) for future reference.

Again, both matrices from (2.62) and (2.63) have a clear structure related to the system's topology. In fact, the matrix  $(\mathbf{E} - \mathbf{C})^{-1}$  must be seen as part of the formal derivation. We will find recursive formulations, e. g., to calculate  $\bar{\mathbf{b}}$ .

Substitution of (2.63) into (2.53) yields the EoM in block-matrix notation:

$$\underbrace{\begin{bmatrix} \mathbf{J}_1^\top & \mathbf{J}_1^\top \mathbf{C}_2^\top & \mathbf{J}_1^\top \mathbf{C}_2^\top \mathbf{C}_3^\top & \mathbf{J}_1^\top \mathbf{C}_4^\top & \mathbf{J}_1^\top \mathbf{C}_2^\top \mathbf{C}_3^\top \mathbf{C}_5^\top \\ \mathbf{0} & \boxed{\mathbf{J}_2^\top} & \boxed{\mathbf{J}_2^\top \mathbf{C}_3^\top} & \mathbf{J}_{4,1}^\top & \mathbf{J}_2^\top \mathbf{C}_3^\top \mathbf{C}_5^\top \\ \mathbf{0} & \mathbf{0} & \mathbf{J}_3^\top & \mathbf{J}_{4,2}^\top & \mathbf{J}_3^\top \mathbf{C}_5^\top \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{J}_5^\top \end{bmatrix}}_{\bar{\mathbf{J}}^\top} \begin{bmatrix} \mathbf{M}_1 \mathbf{a}_1 - \mathbf{f}_1 \\ \mathbf{M}_2 \mathbf{a}_2 - \mathbf{f}_2 \\ \mathbf{M}_3 \mathbf{a}_3 - \mathbf{f}_3 \\ \mathbf{M}_4 \mathbf{a}_4 - \mathbf{f}_4 \\ \mathbf{M}_5 \mathbf{a}_5 - \mathbf{f}_5 \end{bmatrix} = \begin{bmatrix} \mathbf{Q}_1 \\ \mathbf{Q}_2 \\ \mathbf{Q}_3 \\ \mathbf{Q}_4 \end{bmatrix}. \quad (2.64)$$

The upper triangular form of (2.64) facilitates some of the most efficient  $\mathcal{O}(n)$  for-



malisms for MBS, such as from BRANDL et al. [21]. These exploit the fact that: (a) the impressed forces acting on leaf-nodes, i.e.  $\mathbf{f}_5$ , contain only external forces, and, (b) the motion of a reference body is known, i.e.  $\mathbf{a}_0 = \mathbf{0}$  for the inertially-fixed environment.

Equation (2.64) can be solved for  $\ddot{\mathbf{q}}$  by successively evaluating each line. By starting from the last line and substituting the acceleration from (2.34) on page 21 for body  $B_5$

$$\mathbf{a}_5 = \mathbf{C}_5 \mathbf{a}_3 + \mathbf{J}_5 \ddot{\mathbf{q}}_5 + \mathbf{b}_5, \quad (2.65)$$

we get

$$\mathbf{J}_5^T (\mathbf{M}_5 (\mathbf{C}_5 \mathbf{a}_3 + \mathbf{J}_5 \ddot{\mathbf{q}}_5 + \mathbf{b}_5) - \mathbf{f}_5) = \mathbf{Q}_4 .$$

Solving above equation for  $\ddot{\mathbf{q}}_5$  gives

$$\ddot{\mathbf{q}}_5 = \mathbf{P}_5 \mathbf{a}_3 + \mathbf{p}_5 \quad \Rightarrow \quad \boxed{\ddot{\mathbf{q}}_i = \mathbf{P}_i \mathbf{a}_{p(i)} + \mathbf{p}_i}, \quad (2.66)$$

with

$$\mathbf{P}_5 = -(\mathbf{J}_5^T \mathbf{M}_5 \mathbf{J}_5)^{-1} \mathbf{J}_5^T \mathbf{M}_5 \mathbf{C}_5 \quad \Rightarrow \quad \boxed{\mathbf{P}_i = -(\mathbf{J}_i^T \bar{\mathbf{M}}_i \mathbf{J}_i)^{-1} \mathbf{J}_i^T \bar{\mathbf{M}}_i \mathbf{C}_i}, \quad (2.67)$$

$$\begin{aligned} \mathbf{p}_5 &= -(\mathbf{J}_5^T \mathbf{M}_5 \mathbf{J}_5)^{-1} (\mathbf{J}_5^T (\mathbf{M}_5 \mathbf{b}_5 - \mathbf{f}_5) - \mathbf{Q}_4) \\ &\Rightarrow \quad \boxed{\mathbf{p}_i = -(\mathbf{J}_i^T \bar{\mathbf{M}}_i \mathbf{J}_i)^{-1} (\mathbf{J}_i^T (\bar{\mathbf{M}}_i \mathbf{b}_i - \bar{\mathbf{f}}_i) - \mathbf{Q}_i)}, \end{aligned} \quad (2.68)$$

where  $\mathbf{P}_i$  and  $\mathbf{p}_i$  are *forward* propagation quantities of body  $i$ . The boxed equations on the right side are the derived general equations. Thereby, quantities which did not appear yet are about to be derived later along the way, e. g.,  $\bar{\mathbf{M}}_i$  and  $\bar{\mathbf{f}}_i$ . Equation (2.66) is not solvable yet, since two unknown quantities  $\ddot{\mathbf{q}}_5$  and  $\mathbf{a}_3$  appear. By continuing with the third line of (2.64) we have

$$\mathbf{J}_3^T (\mathbf{M}_3 \mathbf{a}_3 - \mathbf{f}_3) + \mathbf{J}_{4,2}^T (\mathbf{M}_4 \mathbf{a}_4 - \mathbf{f}_4) + \mathbf{J}_3^T \mathbf{C}_5^T (\mathbf{M}_5 \mathbf{a}_5 - \mathbf{f}_5) = \mathbf{Q}_3 ,$$

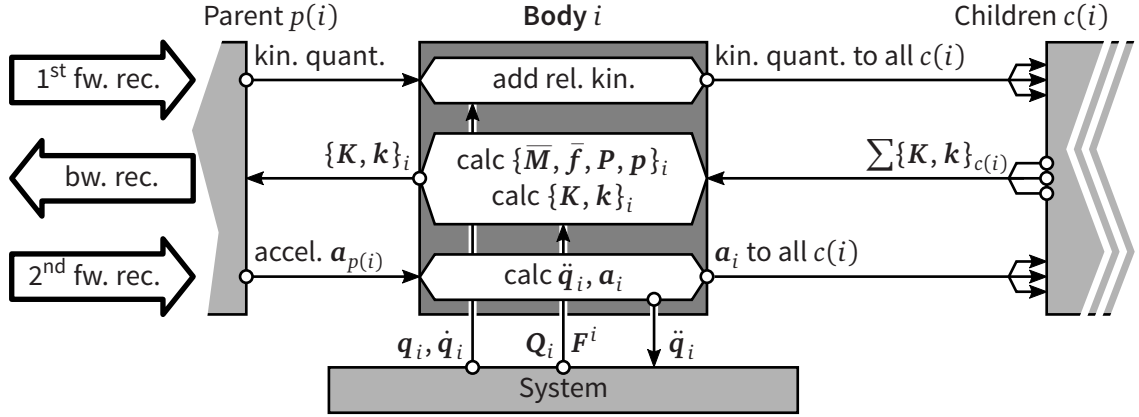
and substitution of (2.65) and (2.66) gives

$$\mathbf{J}_3^T (\bar{\mathbf{M}}_3 \mathbf{a}_3 - \bar{\mathbf{f}}_3) + \mathbf{J}_{4,2}^T (\mathbf{M}_4 \mathbf{a}_4 - \mathbf{f}_4) = \mathbf{Q}_3 ,$$

with

$$\bar{\mathbf{M}}_3 = \mathbf{M}_3 + \mathbf{C}_5^T \mathbf{K}_5 \quad \Rightarrow \quad \boxed{\bar{\mathbf{M}}_i = \mathbf{M}_i + \sum_{j \in c(i)} \mathbf{C}_j^T \mathbf{K}_j}, \quad (2.69)$$

$$\bar{\mathbf{f}}_3 = \mathbf{f}_3 - \mathbf{C}_5^T \mathbf{k}_5 \quad \Rightarrow \quad \boxed{\bar{\mathbf{f}}_i = \mathbf{f}_i - \sum_{j \in c(i)} \mathbf{C}_j^T \mathbf{k}_j}. \quad (2.70)$$



**Figure 2.13:** Schematic local view of the propagation mechanism for quantities in forward, backward and second forward recursion of the  $\mathcal{O}(n)$ -formalism.

Here,  $\bar{M}_i$  and  $\bar{f}$  are the projected mass-matrix and projected forces with the *backward* propagation quantities

$$K_5 = M_5 (C_5 + J_5 P_5) \quad \Rightarrow \quad \boxed{K_i = \bar{M}_i (C_i + J_i P_i)}, \quad (2.71)$$

$$k_5 = M_5 (J_5 p_5 + b_5) - f_5 \quad \Rightarrow \quad \boxed{k_i = \bar{M}_i (J_i p_i + b_i) - \bar{f}_i}. \quad (2.72)$$

This allows us to rewrite the global EoM (2.64) as a reduced system to

$$\begin{bmatrix} J_1^T & J_1^T C_2^T & J_1^T C_2^T C_3^T & J_1^T C_4^T \\ \mathbf{0} & J_2^T & J_2^T C_3^T & J_{4,1}^T \\ \mathbf{0} & \mathbf{0} & J_3^T & J_{4,2}^T \end{bmatrix} \cdot \begin{bmatrix} M_1 a_1 - f_1 \\ M_2 a_2 - f_2 \\ \bar{M}_3 a_3 - \bar{f}_3 \\ M_4 a_4 - f_4 \end{bmatrix} = \begin{bmatrix} Q_1 \\ Q_2 \\ Q_3 \end{bmatrix}. \quad (2.73)$$

Note that quantities associated to body  $B_5$  are now contained in  $\bar{M}_3$  and  $\bar{f}_3$ . The boxed equations (2.66) to (2.72) are the basic building blocks for the  $\mathcal{O}(n)$ -formalism from BRANDL et al. [21], although, presented in a different notation. Figure 2.13 illustrates the basic scheme of the algorithm. In a first forward recursion, kinematic quantities dependent on  $q$  and  $\dot{q}$  are calculated for each body. Second, in a backward recursion—that is, starting from leaf nodes iterating up to the root node—(2.67) to (2.72) are evaluated. Finally, accelerations  $\ddot{q}_i$  and  $a_i$  in (2.66) and (2.34) are computed in a second forward recursion. However, the original  $\mathcal{O}(n)$ -formalism cannot cope with kinematic loops.

### Introducing Sub-Systems

By applying the relative kinematics (2.34) for the remaining spatial accelerations we get

$$\mathbf{a}_4 = \mathbf{C}_4 \mathbf{a}_1 + \underbrace{\begin{bmatrix} \mathbf{J}_{4,1} & \mathbf{J}_{4,2} \end{bmatrix}}_{\mathbf{J}_4} \cdot \underbrace{\begin{bmatrix} \ddot{\mathbf{q}}_2 \\ \ddot{\mathbf{q}}_3 \end{bmatrix}}_{\ddot{\mathbf{q}}_4} + \mathbf{b}_4, \quad (2.74a)$$

$$\mathbf{a}_3 = \mathbf{C}_3 \mathbf{a}_2 + \mathbf{J}_3 \ddot{\mathbf{q}}_3 + \mathbf{b}_3, \quad (2.74b)$$

$$\mathbf{a}_2 = \mathbf{C}_2 \mathbf{a}_1 + \mathbf{J}_2 \ddot{\mathbf{q}}_2 + \mathbf{b}_2, \quad (2.74c)$$

$$\mathbf{a}_1 = \mathbf{J}_1 \ddot{\mathbf{q}}_1 + \mathbf{b}_1. \quad (2.74d)$$

Since (2.74a) contains two sets of unknown generalized accelerations  $\ddot{\mathbf{q}}_2$  and  $\ddot{\mathbf{q}}_3$  (also used by bodies  $B_2$  and  $B_3$  in (2.74b) and (2.74c), respectively), we have to find a closed form solution for the last two lines of (2.73). Ultimately, this step leads to equations to obtain the propagation quantities for a sub-system. Here, bodies  $B_2$  to  $B_4$  are combined into a sub-system.

Substitution of (2.74) into the last two lines of (2.73) and reordering gives the EoM of the sub-system:

$$\mathbf{M}_S \cdot \begin{bmatrix} \ddot{\mathbf{q}}_2 \\ \ddot{\mathbf{q}}_3 \end{bmatrix} = -\mathbf{B}_S \mathbf{a}_1 - \mathbf{f}_S + \underbrace{\begin{bmatrix} \mathbf{Q}_2 \\ \mathbf{Q}_3 \end{bmatrix}}_{\mathbf{Q}_S}, \quad (2.75)$$

with quantities specific to sub-system  $S$ :

$$\boxed{\mathbf{M}_S = \sum_j^{N_{b,S}} \widehat{\mathbf{J}}_j^\top \widehat{\mathbf{M}}_j \widehat{\mathbf{J}}_j} \quad \mathbf{M}_S \in \mathbb{R}^{N_{\text{DoF},S} \times N_{\text{DoF},S}}, \quad (2.76a)$$

$$\boxed{\mathbf{B}_S = \sum_j^{N_{b,S}} \widehat{\mathbf{J}}_j^\top \widehat{\mathbf{M}}_j \widehat{\mathbf{C}}_j} \quad \mathbf{B}_S \in \mathbb{R}^{N_{\text{DoF},S} \times 6}, \quad (2.76b)$$

$$\boxed{\mathbf{f}_S = \sum_j^{N_{b,S}} \widehat{\mathbf{J}}_j^\top (\widehat{\mathbf{M}}_j \widehat{\mathbf{b}}_j - \widehat{\mathbf{f}}_j)} \quad \mathbf{f}_S \in \mathbb{R}^{N_{\text{DoF},S}}. \quad (2.76c)$$

Here,  $\mathbf{M}_S$ ,  $\mathbf{B}_S$  and  $\mathbf{f}_S$  are the generalized quantities of the sub-system for mass-matrices and forces, respectively. The dimension  $N_{\text{DoF},S} = \dim(\mathbf{q}_S)$  is the number of DoFs spanning the sub-system. Moreover, quantities denoted by  $\widehat{(\cdot)}$  are associated to

sub-system member bodies such that

$$\boxed{\widehat{\mathbf{M}}_j = \mathbf{M}_j + \sum_{k \in \{c(j) \wedge \mathcal{C}_S\}} \mathbf{C}_k^\top \overline{\mathbf{M}}_k}, \quad (2.77a)$$

$$\boxed{\widehat{\mathbf{f}}_j = \mathbf{f}_j + \sum_{k \in \{c(j) \wedge \mathcal{C}_S\}} \mathbf{C}_k^\top \overline{\mathbf{f}}_k}. \quad (2.77b)$$

In contrast to (2.69) and (2.70) the sum applies here only to member bodies having children outside of the sub-system, i.e.,  $j \in \mathcal{P}_S$  (cf. Figure 2.10 and Figure 2.12a, hence,  $\widehat{\mathbf{M}}_3 = \overline{\mathbf{M}}_3$  and  $\widehat{\mathbf{M}}_2 = \mathbf{M}_2$ ).<sup>10</sup> Moreover, for sub-system member body quantities we have (in recursive form)

$$\boxed{\widehat{\mathbf{b}}_j = \mathbf{C}_j \widehat{\mathbf{b}}_{p(j)} + \mathbf{b}_j} \quad \text{for the sub-system: } \boxed{\widehat{\mathbf{b}}_S = \mathbf{0}}, \quad (2.78a)$$

$$\boxed{\widehat{\mathbf{J}}_j = \frac{\partial \mathbf{v}_j}{\partial \mathbf{q}_S} = \mathbf{C}_j \widehat{\mathbf{J}}_{p(j)} + \widehat{\mathbf{J}}_{\text{rel},j}} \quad \text{for the sub-system: } \boxed{\widehat{\mathbf{J}}_S = \mathbf{0}}, \quad (2.78b)$$

$$\boxed{\widehat{\mathbf{C}}_j = \mathbf{C}_j \widehat{\mathbf{C}}_{p(j)}} \quad \text{for the sub-system: } \boxed{\widehat{\mathbf{C}}_S = \mathbf{E}}. \quad (2.78c)$$

Note, that  $\widehat{\mathbf{J}}_j$  is the framed portion (associated to the sub-system) of  $\overline{\mathbf{J}}_j$  in (2.63).

Solving (2.75) for  $[\ddot{\mathbf{q}}_2^\top, \ddot{\mathbf{q}}_3^\top]^\top$  and following the notation scheme of the propagation quantities in (2.66) gives

$$\boxed{\begin{bmatrix} \ddot{\mathbf{q}}_2 \\ \ddot{\mathbf{q}}_3 \end{bmatrix}} = \ddot{\mathbf{q}}_S = \mathbf{P}_S \mathbf{a}_1 + \mathbf{p}_S, \quad (2.79)$$

with

$$\boxed{\mathbf{P}_S = -\mathbf{M}_S^{-1} \mathbf{B}_S}, \quad (2.80)$$

$$\boxed{\mathbf{p}_S = -\mathbf{M}_S^{-1} (\mathbf{f}_S - \mathbf{Q}_S)}. \quad (2.81)$$

Finally, by evaluating the first line of (2.73) and substituting the accelerations in (2.74) and (2.79) we get

$$\mathbf{J}_1^\top \overline{\mathbf{M}}_1 \mathbf{J}_1 \ddot{\mathbf{q}}_1 + \mathbf{J}_1^\top (\overline{\mathbf{M}}_1 \mathbf{b}_1 - \overline{\mathbf{f}}_1) = \mathbf{Q}_1, \quad (2.82)$$

<sup>10</sup>  $\widehat{(\cdot)}$  quantities are related to  $\overline{(\cdot)}$  quantities and only marked differently to distinguish quantities for bodies inside sub-systems and normal bodies from MBS

with

$$\bar{\mathbf{M}}_1 = \mathbf{M}_1 + \mathbf{K}_S, \quad (2.83a)$$

$$\bar{\mathbf{f}}_1 = \mathbf{f}_1 - \mathbf{k}_S. \quad (2.83b)$$

This leads to the propagation quantities of the sub-system

$$\boxed{\mathbf{K}_S = \bar{\mathbf{M}}_S + \mathbf{B}_S^\top \mathbf{P}_S}, \quad (2.84)$$

$$\boxed{\mathbf{k}_S = \mathbf{B}_S^\top \mathbf{p}_S - \bar{\mathbf{f}}_S}, \quad (2.85)$$

with

$$\begin{aligned} \bar{\mathbf{M}}_S &= \mathbf{C}_2^\top (\mathbf{M}_2 + \mathbf{C}_3^\top \bar{\mathbf{M}}_3 \mathbf{C}_3) \mathbf{C}_2 + \mathbf{C}_4^\top \mathbf{M}_4 \mathbf{C}_4 \\ &\Rightarrow \boxed{\bar{\mathbf{M}}_S = \sum_j^{N_{b,S}} \{\widehat{\mathbf{C}}^\top \widehat{\mathbf{M}} \widehat{\mathbf{C}}\}_j} \quad \bar{\mathbf{M}}_S \in \mathbb{R}^{6 \times 6}, \end{aligned} \quad (2.86)$$

$$\begin{aligned} \bar{\mathbf{f}}_S &= \mathbf{C}_2^\top (\mathbf{M}_2 \mathbf{b}_2 - \bar{\mathbf{f}}_2 + \mathbf{C}_3^\top (\bar{\mathbf{M}}_3 \bar{\mathbf{b}}_3 - \bar{\mathbf{f}}_3)) + \mathbf{C}_4^\top (\mathbf{M}_4 \mathbf{b}_4 - \mathbf{f}_4) \\ &\Rightarrow \boxed{\bar{\mathbf{f}}_S = \sum_j^{N_{b,S}} \{\widehat{\mathbf{C}}^\top (\widehat{\mathbf{M}} \widehat{\mathbf{b}} - \widehat{\mathbf{f}})\}_j} \quad \bar{\mathbf{f}}_S \in \mathbb{R}^6. \end{aligned} \quad (2.87)$$

Generalized acceleration  $\ddot{\mathbf{q}}_1$  is obtained from (2.66) using the known acceleration of the environment, i.e.  $\mathbf{a}_0 = \mathbf{0}$ .

In order to fit sub-systems into the propagation scheme illustrated in Figure 2.13 some properties are still missing. In (2.84) and (2.85) backward propagation quantities  $\mathbf{K}_i$  and  $\mathbf{k}_i$  are already defined for sub-systems. By comparing (2.83) with (2.71) and (2.72) we have

$$\mathbf{C}_S = \mathbf{E} \in \mathbb{R}^{6 \times 6}. \quad (2.88)$$

Therefore, the sub-system moves with its parent, i.e., FoRs are identical. Interestingly, this results in a generic kinematic interface between a sub-system and its member bodies which is independent of specific joint models. Analogously to the joint model examples on pages 22 ff., we get the following quantities for a sub-system node  $S$

$$\mathbf{v}_S = \mathbf{v}_{p(S)}, \quad \mathbf{a}_S = \mathbf{a}_{p(S)} \quad \Rightarrow \quad \mathbf{b}_S = \mathbf{0}, \quad \mathbf{J} = \dot{\mathbf{J}} = \emptyset. \quad (2.89)$$

Hence, velocities and accelerations are directly inherited from the parent of the sub-system and joint Jacobians are empty, since there are no relative DoFs.

In order to evaluate (2.66) for bodies in  $\mathcal{C}_S$ , accelerations  $\mathbf{a}_j$  of a sub-system member

**Algorithm 2.1:** General form of the  $\mathcal{O}(n)$ -algorithm.**Input:**  $t, \mathbf{q}, \dot{\mathbf{q}}$ **Output:**  $\ddot{\mathbf{q}}$ , (if desired:  $\lambda^c$ )*Step 1: first forward recursion (cf. Algorithm 2.2):*

update kinematic quantities for all bodies of the MBS

*Step 2: compute impressed forces and torques  $\mathbf{F}^i, \mathbf{T}^i$  (e.g., from contacts) and generalized forces  $\mathbf{Q}_{\text{ext}}$  (e.g., from motor dynamics and gear friction)**Step 3: backward recursion (cf. Algorithm 2.3):*update propagation quantities  $\mathbf{P}, \mathbf{p}, \mathbf{K}$  and  $\mathbf{k}$  for bodies of the MBS*Step 4: second forward recursion (cf. Algorithm 2.4):*calculate generalized accelerations  $\ddot{\mathbf{q}}$  (and if desired:  $\lambda^c$ ) for bodies of the MBSbody  $j$  are required:

$$\boxed{\mathbf{a}_j = \widehat{\mathbf{C}}_j \mathbf{a}_S + \widehat{\mathbf{J}}_j \ddot{\mathbf{q}}_S + \widehat{\mathbf{b}}_j} \quad \text{for } j \in \mathcal{P}_S. \quad (2.90)$$

Note that, (2.90) must only be evaluated for bodies in  $\mathcal{P}_S$ , which improves efficiency.

## 2.8 Resulting Formalism

This section summarizes the derived  $\mathcal{O}(n)$ -algorithm into an implementation-friendly form. The basic recursions of the  $\mathcal{O}(n)$  formalism are shown in Algorithm 2.1 (cf. also Figure 2.13 on page 32). Algorithms of the individual recursions are shown in Algorithm 2.2 (forward 1), Algorithm 2.3 (backward) and Algorithm 2.4 (forward 2).

Evidently, the formulas for updating the formalism quantities are not identical for sub-systems and rigid bodies. Therefore, the individual algorithms contain conditions

“if bodytype( $i$ ) = sub-system then ... else ... end if”

for choosing the respective evaluation paths. As a consequence, this adds additional complexity and computational overhead. However, in object oriented programming languages such as C++ this specialization can easily be achieved by *overloading* class-specific member functions. Similarly, sub-system quantities which are summed from member bodies can be specialized in the body-specific code to improve efficiency from special structures of certain matrices.

---

**Algorithm 2.2:** First forward recursion
 

---

**Input:**  $t, \mathbf{q}, \dot{\mathbf{q}}$ **Output:** updated kinematic quantities for each body of the MBS

```

for  $i \leftarrow 1$  to  $N_{\text{bodies}}$  do // first forward recursion
  if  $\text{bodytype}(i) = \text{sub-system}$  then // sub-system
    copy time-varying variables from parent  $\hat{\mathbf{A}}_{kp}, \boldsymbol{\omega}, \dot{\mathbf{r}}_o, \mathbf{g}$ , e. g.,  $\mathbf{g}_i = \mathbf{g}_{p(i)}$ 
     $\bar{\mathbf{b}}_i \leftarrow \mathbf{0}$ 
  5:   for  $j \leftarrow 1$  to  $N_{b,S}$  do // iteration over all members of the sub-system
      compute time-varying variables  $\mathbf{A}_{kp,j}, {}^I\mathbf{r}_{o,j}, \dot{\mathbf{r}}_{o,j}, \boldsymbol{\omega}_j, \mathbf{g}_j, \mathbf{J}_j, \dot{\mathbf{J}}_j, \mathbf{C}_j, \mathbf{b}_j$ 
       $\hat{\mathbf{b}}_j \leftarrow \mathbf{C}_j \hat{\mathbf{b}}_{p(j)} + \mathbf{b}_j$ 
       $\hat{\mathbf{M}}_j \leftarrow \mathbf{M}_j$ 
       $\hat{\mathbf{C}}_j \leftarrow \mathbf{C}_j \hat{\mathbf{C}}_{p(j)}$ 
  10:  end for
    else // rigid body
      compute time-varying variables  $\mathbf{A}_{kp,i}, {}^I\mathbf{r}_{o,i}, \dot{\mathbf{r}}_{o,i}, \boldsymbol{\omega}_i, \mathbf{g}_i, \mathbf{J}_i, \dot{\mathbf{J}}_i, \mathbf{C}_i, \mathbf{b}_i$ 
    end if
  end for

```

---

**Algorithm 2.3:** Backward recursion

---

**Input:** updated kinematic quantities, impressed and generalized forces of the MBS  
**Output:** propagation quantities  $\mathbf{P}$ ,  $\mathbf{p}$ ,  $\mathbf{K}$  and  $\mathbf{k}$  for each body/sub-system

```

for  $i \leftarrow N_{\text{bodies, MBS}}$  to 1 do // backward recursion
  if  $\text{bodytype}(i) = \text{sub-system}$  then // sub-system
    for  $j \leftarrow 1$  to  $N_{b,s}$  do // recursion on sub-system member bodies
      compute  $\mathbf{f}_j$ ,  $\hat{\mathbf{f}}_j \leftarrow \mathbf{f}_j$ 
5:   end for
    for all  $j \in \mathcal{C}_{S,i}$  do // border bodies outside sub-system
       $\widehat{\mathbf{M}}_{p(j)} \leftarrow \widehat{\mathbf{M}}_{p(j)} + \mathbf{C}_j^T \mathbf{K}_j$ ,  $\hat{\mathbf{f}}_{p(j)} \leftarrow \hat{\mathbf{f}}_{p(j)} - \mathbf{C}_j^T \mathbf{k}_j$ 
    end for
     $\bar{\mathbf{M}}_i \leftarrow \sum_j^{N_{b,s}} \widehat{\mathbf{J}}_j^T \widehat{\mathbf{M}}_j \widehat{\mathbf{J}}_j$ ,  $\bar{\mathbf{f}}_i \leftarrow \sum_j^{N_{b,s}} \widehat{\mathbf{J}}_j^T (\widehat{\mathbf{M}}_j \widehat{\mathbf{b}}_j - \hat{\mathbf{f}}_j)$ 
10:   $\bar{\mathbf{B}}_i \leftarrow \sum_j^{N_{b,s}} \widehat{\mathbf{J}}_j^T \widehat{\mathbf{M}}_j \widehat{\mathbf{C}}_j$ ,  $\bar{\mathbf{f}}_i \leftarrow \sum_j^{N_{b,s}} \widehat{\mathbf{C}}_j^T (\widehat{\mathbf{M}}_j \widehat{\mathbf{b}}_j - \hat{\mathbf{f}}_j)$ 
     $\bar{\mathbf{M}}_i \leftarrow \sum_j^{N_{b,s}} \widehat{\mathbf{C}}_j^T \widehat{\mathbf{M}}_j \widehat{\mathbf{C}}_j$ 
     $\bar{\mathbf{P}}_i \leftarrow \bar{\mathbf{M}}_i^{-1} \bar{\mathbf{B}}_i$ ,  $\bar{\mathbf{K}}_i \leftarrow \bar{\mathbf{M}}_i + \bar{\mathbf{B}}_i^T \bar{\mathbf{P}}_i$ 
     $\bar{\mathbf{p}}_i \leftarrow \bar{\mathbf{M}}_i^{-1} (\bar{\mathbf{f}}_i - \bar{\mathbf{Q}}_i)$ ,  $\bar{\mathbf{k}}_i \leftarrow \bar{\mathbf{B}}_i^T \bar{\mathbf{p}}_i + \bar{\mathbf{f}}_i$ 
  else // rigid body
15:  compute  $\mathbf{f}_i$ 
     $\bar{\mathbf{M}}_i \leftarrow \bar{\mathbf{M}}_i + \sum_{j \in N_{\text{children}}} \mathbf{C}_j^T \mathbf{K}_j$ ,  $\bar{\mathbf{f}}_i \leftarrow \bar{\mathbf{f}}_i - \sum_{j \in N_{\text{children}}} \mathbf{C}_j^T \mathbf{k}_j$ 
     $\bar{\mathbf{P}}_i \leftarrow (\mathbf{J}_i^T \bar{\mathbf{M}}_i \mathbf{J}_i)^{-1} (-\mathbf{J}_i^T \bar{\mathbf{M}}_i \mathbf{C}_i)$ ,  $\bar{\mathbf{K}}_i \leftarrow \bar{\mathbf{M}}_i (\mathbf{C}_i + \mathbf{J}_i \bar{\mathbf{P}}_i)$ 
     $\bar{\mathbf{p}}_i \leftarrow (\mathbf{J}_i^T \bar{\mathbf{M}}_i \mathbf{J}_i)^{-1} (-\mathbf{J}_i^T (\bar{\mathbf{M}}_i \mathbf{b}_i - \bar{\mathbf{f}}_i) - \bar{\mathbf{Q}}_i)$ ,  $\bar{\mathbf{k}}_i \leftarrow \bar{\mathbf{M}}_i (\mathbf{b}_i + \mathbf{J}_i \bar{\mathbf{p}}_i) - \bar{\mathbf{f}}_i$ 
  end if
20: end for

```

---

**Algorithm 2.4:** Second forward recursion

---

**Input:** updated propagation quantities  $\mathbf{P}$ ,  $\mathbf{p}$ ,  $\mathbf{K}$  and  $\mathbf{k}$   
**Output:**  $\bar{\mathbf{q}}$

```

for  $i \leftarrow 1$  to  $N_{\text{bodies}}$  do // second forward recursion
   $\bar{\mathbf{q}}_i \leftarrow \bar{\mathbf{P}}_i \mathbf{a}_{p(i)} + \bar{\mathbf{p}}_i$ 
  if  $\text{bodytype}(i) = \text{sub-system}$  then // sub-system
     $\bar{\mathbf{a}}_i \leftarrow \bar{\mathbf{a}}_{p(i)}$ 
5:   for all  $j \in \mathcal{P}_S$  do // recursion on sub-system member bodies with children outside
      $\bar{\mathbf{a}}_j \leftarrow \widehat{\mathbf{C}}_{S,j} \bar{\mathbf{a}}_i + \widehat{\mathbf{J}}_j \bar{\mathbf{q}}_i + \widehat{\mathbf{b}}_j$ 
   end for
  else // rigid body
     $\bar{\mathbf{a}}_i \leftarrow \mathbf{J}_i \bar{\mathbf{q}}_i + \mathbf{b}_i + \mathbf{C}_i \bar{\mathbf{a}}_{p(i)}$ 
10:  end if
end for

```

---



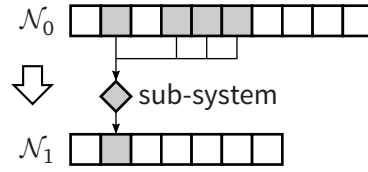


Figure 2.14: Modification of the system node list  $\mathcal{N}$  when nodes are replaced by a sub-system.

## 2.9 Automatic Sub-System Generation

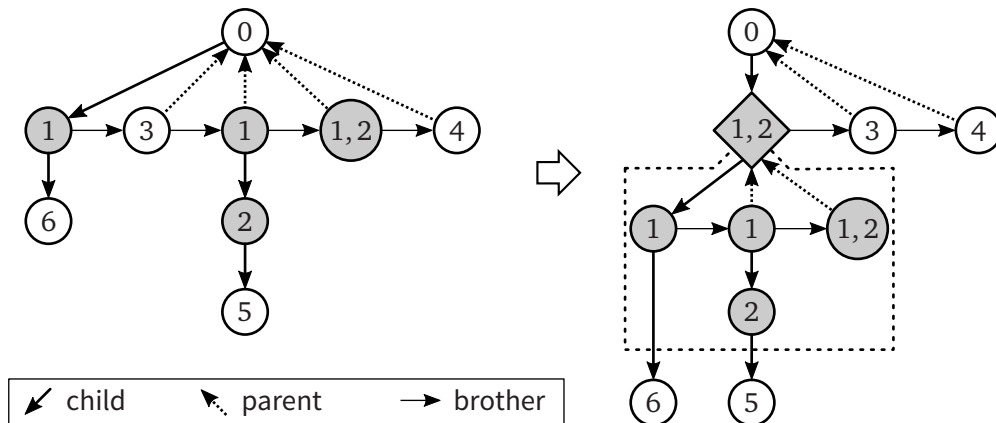
In order to apply the  $\mathcal{O}(n)$ -algorithm presented above to systems containing joints with overlapping generalized coordinates, these joints (and respective bodies) must be combined into sub-systems. However, it is favorable to generate only *one* model of a system and be able to solve it with *any* implemented algorithm without thinking about the modeling method used. Hence, an appropriate conversion scheme of a given MBS into sub-systems is presented here.

The conversion process basically boils down to detecting groups of bodies with overlapping sets of generalized coordinates  $\mathbf{q}_S$ , “wrapping” individual groups into a sub-system  $S$  and replacing member bodies in the MBS by the sub-system. Depending on the system representation used, the replacement task might be more complex. Since in this work a body list and a pointer-based tree representation are deployed, these two types are considered.

Figure 2.14 illustrates the process for a body list  $\mathcal{N}$ . In an iterative process, for each sub-system  $S$  an initial body list of the system  $\mathcal{N}_0$  is modified by replacing all nodes  $j \in \mathcal{N}_S$  by the sub-system  $S$  leading to the new list  $\mathcal{N}_1$ . Since forward and backward recursions rely on a correct ordering of  $\mathcal{N}$ , sub-system  $S$  is inserted at the former lowest position of its member bodies.

On the other hand, for the tree representation, all connections must be reconfigured such that the pointer logic is correct. An example is presented in Figure 2.15: On the left, a tree containing bodies with overlapping  $\mathbf{q}$ -indices  $\in 1,2$  is shown. The corresponding MBS with a sub-system replacing those bodies is depicted on the right. Thereby, the following steps are taken ( $\mathcal{N}_0$  is the initial set of system nodes):

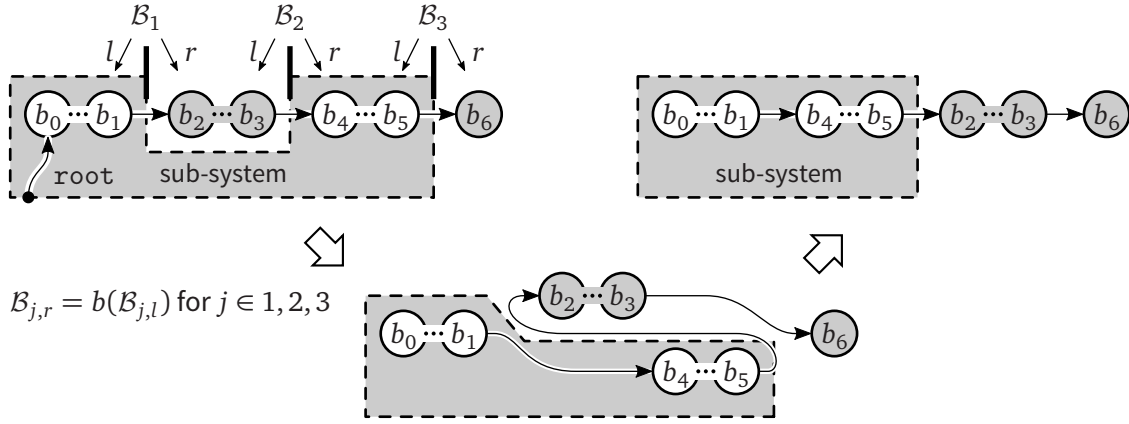
1. Find a group of nodes  $\mathcal{N}_S \in \mathcal{N}_0$  with overlapping  $\mathbf{q}$ -indices.
2. Find a common parent  $p_S \in \mathcal{N} \wedge p_S \notin \mathcal{N}_S \wedge \exists j \in \mathcal{N}_S : p_S = p(j)$ .
3. Generate a sub-system node  $S$  from  $\mathcal{N}_S$ .
4. Set new parent node for direct children:  $p(\{j \in \mathcal{N}_S \mid p(j) = p_S\}) \leftarrow S$ .
5. Modify the pointer sequence to generate a continuous sequence of brothers for  $S$  (cf. Algorithm 2.5 and Fig. 2.16).
6. Finally, replace member bodies by  $S$  in the pointer logic (cf. Figure 2.14).



**Figure 2.15:** *Left:* Pointer-based tree of nodes where the numbers represent associated  $q$ -indices. Nodes with overlapping  $q$ -indices are marked. *Right:* conversion result of a group of nodes with overlapping  $q$ -indices (1, 2) into a sub-system.

The first line of sub-system member bodies shown in Figure 2.15 is not a continuous sequence of brother pointers, since it is interrupted by the node with index 3. Hence, it must be moved after the sequence which closes the gap.

Algorithm 2.5 solves this problem for an arbitrary sequence of brother pointers with multiple interrupting sequences by studying inward and outward borders  $\mathcal{B}_i$ . Thereby, bodies on both sides of the border,  $\mathcal{B}_{i,l}$  and  $\mathcal{B}_{i,r}$  for the left and right side, respectively, are tracked and pointers are reset. Figure 2.16 illustrates the presented algorithm.



**Figure 2.16:** Modification of the pointer sequence to generate a continuous line of brothers for a given sub-system with only one inward and one outward boundary.

**Algorithm 2.5:** Generate a continuous sequence of brothers from a given set of sub-system member bodies  $\mathcal{N}_S$  cf. Figure 2.16

---

**Input:**  $\mathcal{N}_S$   
 $\mathcal{B}_{1,l} \leftarrow \mathcal{N}_S(0)$  // initialize with sub-system's root  
**while**  $\mathcal{B}_{1,l} \neq \emptyset \wedge \mathcal{B}_{1,l} \in \mathcal{N}_S$  **do**  
  **if**  $b(\mathcal{B}_{1,l}) \neq \emptyset \wedge b(\mathcal{B}_{1,l}) \notin \mathcal{N}_S$  **then** // outward boundary  $\mathcal{B}_1$  found  
     $\mathcal{B}_{2,l} \leftarrow \mathcal{B}_{1,r} \leftarrow b(\mathcal{B}_{1,l})$   
    **while**  $\mathcal{B}_{2,l} \neq \emptyset \wedge b(\mathcal{B}_{2,l}) \in \mathcal{N}_S$  **do** // find next inward boundary  
       $\mathcal{B}_{2,l} \leftarrow b(\mathcal{B}_{2,l})$   
    **end while**  
    **if**  $\mathcal{B}_{2,l} \neq \emptyset$  **then** // inward boundary  $\mathcal{B}_2$  found  
       $\mathcal{B}_{3,l} \leftarrow \mathcal{B}_{2,r} \leftarrow b(\mathcal{B}_{2,l})$   
      **while**  $b(\mathcal{B}_{3,l}) \in \mathcal{N}_S$  **do** // find next outward boundary (including end)  
         $\mathcal{B}_{3,l} \leftarrow b(\mathcal{B}_{3,l})$   
      **end while**  
       $b(\mathcal{B}_{1,l}) \leftarrow \mathcal{B}_{2,r}$  // remove the gap by ...  
       $b(\mathcal{B}_{2,l}) \leftarrow b(\mathcal{B}_{3,l})$  // resetting brother pointers  
       $b(\mathcal{B}_{3,l}) \leftarrow \mathcal{B}_{1,r}$  // cf. Fig. 2.16  
    **end if**  
  **end if**  
   $\mathcal{B}_{1,l} \leftarrow b(\mathcal{B}_{1,l})$  // study next node  
**end while**

---

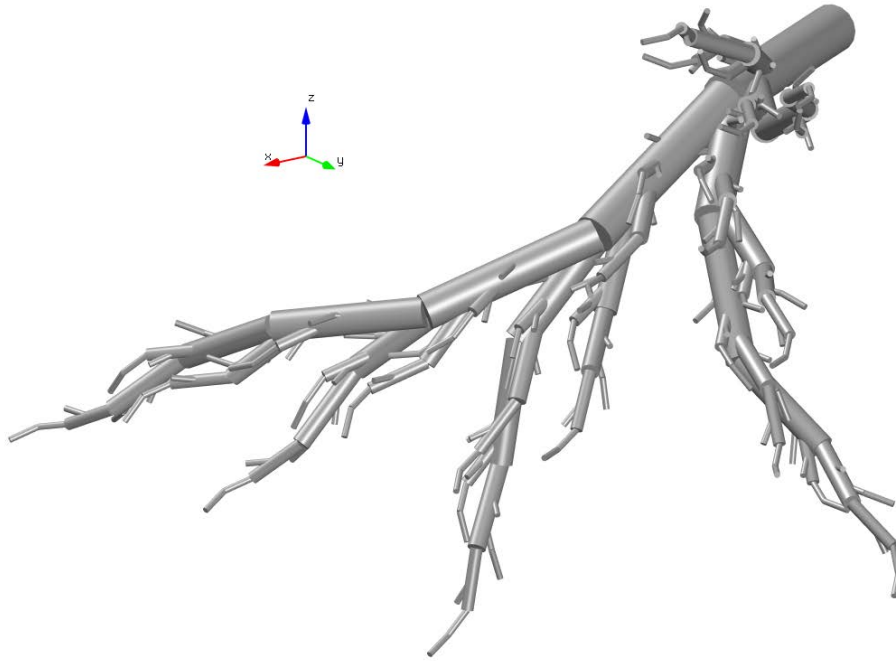


Figure 2.17: Resulting figure of Dill(7) with 128 DoFs.

## 2.10 Results

This section presents performance comparisons between the  $\mathcal{O}(n^3)$  and  $\mathcal{O}(n)$ -algorithm with sub-systems. The algorithms are applied to several test systems. Under certain circumstances the mass-matrix  $\mathbf{M}$  can become sparse. Therefore, a specialized solver for MBS exploiting the sparsity of the mass-matrix as described by FEATHERSTONE and ORIN (cf. [52]) is also implemented and added to the comparison where its application makes sense. The performance was measured by sampling the run-time of one specific FD-algorithm and taking the median value of the measurements. Finally, the results are discussed.

### 2.10.1 Run-Time Comparisons for the Dill Example

This section presents the results of run-time measurements of different FD-algorithms implemented in this work applied to the Dill( $n$ ) example. The Dill( $n$ ) example is a fictitious robot mechanism which contains branches. It is adopted from [51] by FEATHERSTONE:

‘Dill( $n$ ) is a fractal kinematic tree with  $2^n$  links and a maximum branching factor of  $n$ . Dill(0) consists of a single link, connected to its parent (or fixed base) by a revolute joint. The link inertia parameters are  $mass = 0.1$ ,  $CoM = (0.05, 0, 0)$ ,  $I_{xx} = 1/200000$ , and  $I_{yy} = I_{zz} = 403/1200000$ , which

**Table 2.1:** Some numbers on the Dill( $n$ ) example with  $n_D$  is the Dill-level,  $n_{\text{DoFs}}$  is the number of DoFs,  $n_{\text{nz}}$  the number of non-zero elements,  $n_{\text{dense}}$  the number of dense elements when a Cholesky solver is used which doesn't exploit the sparsity, the sparsity measure  $\alpha$  and the resulting "possible speedup" when using the sparse matrix solver described by FEATHERSTONE in [52].

Dill level	number of DoFs	non-zero elements	no. of ele. of tri. mat.	triangular sparsity	possible speedup
$n$	$n_q$	$n_{\text{nz}}$	$n_{\text{dense}}$	$\alpha$	$1/\alpha^2$
1	2	3	3	1.000000	1.00
2	4	8	10	0.800000	1.56
3	8	20	36	0.555556	3.24
4	16	48	136	0.352941	8.02
5	32	112	528	0.212121	22.22
6	64	256	2080	0.123077	66.01
7	128	576	8256	0.069767	205.44
8	256	1280	32896	0.038911	660.49
9	512	2816	131328	0.021442	2174.95
10	1024	6144	524800	0.011707	7296.01

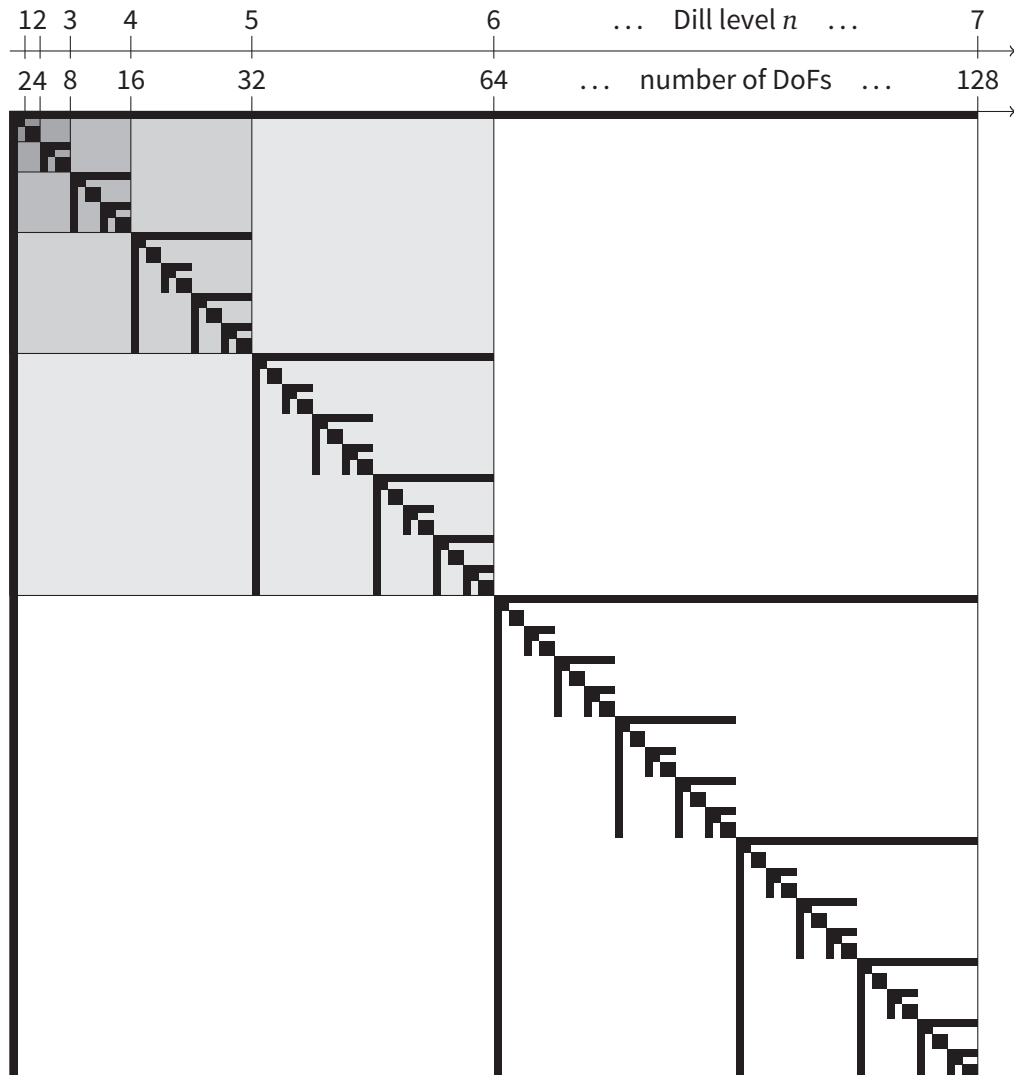
are the parameters of a cylinder of length 0.1 and radius 0.01, centered on the  $x$ -axis. Dill( $n$ ),  $n > 0$ , consists of a Dill(0) mechanism scaled by a factor of  $n$  (so the mass goes up by a factor of  $n^3$  and the moments of inertia by a factor of  $n^5$ ), to which  $n$  sub-trees are attached. Sub-Tree  $i$  is a Dill( $i - 1$ ) mechanism and is attached at a displacement of  $i \times 0.1$  along the  $x$ -axis and a skew angle of  $i/3$  about the  $x$ -axis.

In the test configuration,  $q_i = \pi/6$ , which produces a treelike posture of the mechanism, and  $\dot{q}_i = 1$ .

While Dill( $n$ ) was originally developed to study the influence of the kinematic structure on the numerical accuracy of different FD-algorithms, it is also suitable to benchmark their performance.

Figure 2.17 shows the resulting mechanism for a Dill(7) with 128 DoFs and reveals the structure and the origin of the name. Note that not all bodies are visible because they are possibly enclosed inside of a larger body.

Figure 2.18 shows the branch induced sparsity pattern of the mass-matrix for the Dill( $n$ ) example for different Dill-levels  $n \in \mathbb{N}$  from 1 to 7. All elements marked as ■ are non-zero. In contrast to a strict chain topology—where the resulting mass-matrix is fully dense—the Dill( $n$ ) features a high level of sparsity. Especially for larger  $n$  we observe a growing number of zeros in the mass-matrix. Therefore, Dill( $n$ ) and other real world mechanisms which comprise a larger level of sparsity would greatly benefit from a sparse matrix solver. However, traditional sparse matrix solvers based



**Figure 2.18:** Density plot of the mass-matrix for the Dill example for different Dill levels. We have  $2^{n-1}(2+n)$  non-zero elements, when only taking the lower or upper triangular matrix into account. The differences in background color show the different levels.

on Cholesky decomposition do not preserve the sparsity level in the resulting lower left matrix  $L$  making them unusable for MBS.

FEATHERSTONE and ORIN present in [52] such a specialized sparse matrix solver for MBS. Strictly speaking, the solver consists of a collection of algorithms for reversed sparse Cholesky (RSC) decomposition and factorization which uses an index array describing the child-parent relations of the kinematic tree.<sup>11</sup> The index array is key

<sup>11</sup> In this work the property of the preservation of the sparsity of the mass-matrix in the triangular matrix  $L$  by reversing the order of the Cholesky decomposition was discovered independently

**Table 2.2:** Equations to calculate the values found in Table 2.1.

$n_{\text{nz, Dill}}(n) = 2^{n-1}(2 + n)$	$n_{\text{dense}}(n_q) = (1 + n_q) n_q / 2$
$n_{q, \text{Dill}} = 2^n$	$n_{\text{dense}}(n) = (1 + 2^n) 2^n / 2$
$\alpha = n_{\text{nz}} / n_{\text{dense}}$	$\text{speedup} = 1 / \alpha^2$

to exploit the sparsity in all stages of the decomposition and factorization of the mass-matrix. Furthermore, the sparsity pattern is identical for the mass-matrix and the lower triangular matrix  $L$  resulting from the Cholesky decomposition.

In Table 2.1 some properties of  $\text{Dill}(n)$  for different Dill-levels  $n$  are listed. Where  $n_q$  is the number of DoFs,  $n_{\text{nz}}$  is the number of non-zero elements,  $n_{\text{dense}}$  is the number of elements of a lower (or upper) triangular symmetric matrix which a conventional Cholesky decomposition and factorization algorithm would use,  $\alpha \in [0, 1]$  is a number describing the density of the lower triangular part of the symmetric mass-matrix  $\mathbf{M} \in \mathbb{R}^{n_q \times n_q}$ . According to [52] the theoretical speedup gained by the RSC solver when compared to the conventional Cholesky decomposition and factorization (see e. g., [116]) can be calculated via  $1/\alpha^2$ . The last column of Table 2.1 shows the theoretical speedup for different  $\text{Dill}(n)$  levels. All values are calculated by the equations found in Table 2.2.

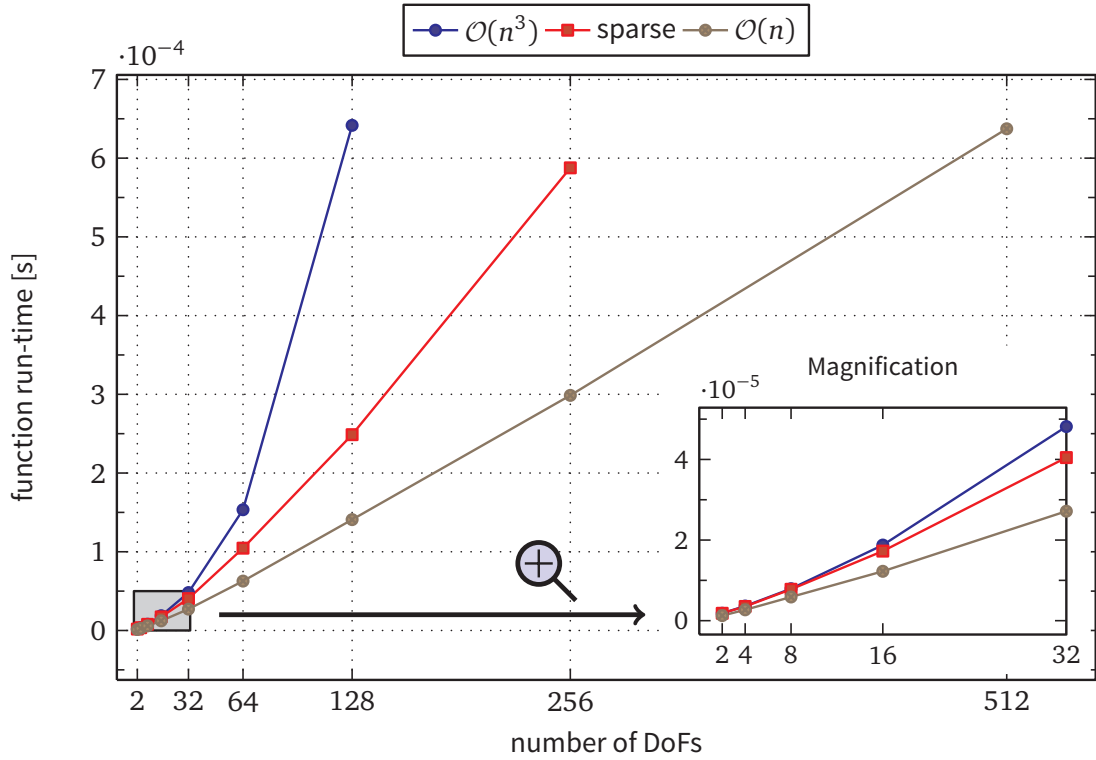
In this work, the mentioned algorithms found in [52] have been implemented to compare the run-time with the  $\mathcal{O}(n)$ -algorithm. Figure 2.19 shows the resulting run-times for different FD-algorithms over the  $\text{Dill}(n)$  levels. The magnification reveals that the  $\mathcal{O}(n)$ -algorithm is still the fastest method when applied to  $\text{Dill}(n)$ . However, the  $\mathcal{O}(n^3)$ -method greatly benefits from the RSC decomposition and factorization. It has to be pointed out that the results depend strongly on the used computer architecture. This not only affects the scaling but also the run-time of the individual algorithms due to implementation details, compiler optimizations and different CPU optimizations influencing the run-time of individual CPU instruction sets. This behavior will be discussed in Section 2.11.

### 2.10.2 Run-Time Comparison for the Lola Model

In this section the FD-algorithms implemented in this work are applied to the Lola models of Type 2 and Type 3 (cf. Table 2.3)<sup>12</sup> originally developed by BUSCHMANN [28]. Furthermore, the run-time measurements are conducted on different computers with

from [52]. However, the observation was made years after the elegant method using the index array described by FEATHERSTONE and ORIN in 2005.

<sup>12</sup> The model Type 1 is incompatible with this study. The models Type 4 and Type 5 differ only in the contact modeling and are therefore omitted in this study, since the contact solver is not influenced by the method.

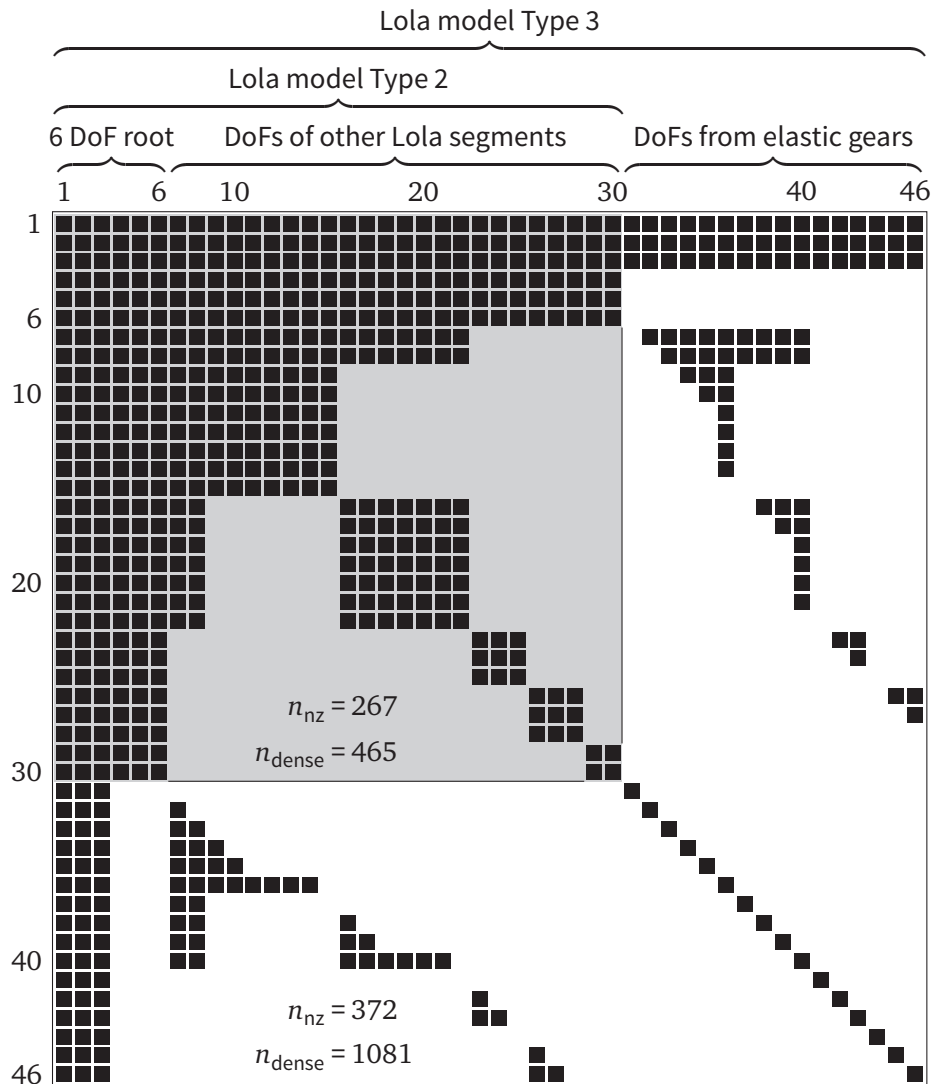


**Figure 2.19:** Run-time comparison for different FD-algorithms of the Dill( $n$ ) example (cf. [51]). Each mark corresponds to one Dill level  $n_D$  such that the number of DoFs  $n_{\text{DoFs}} = 2^{n_D}$ . The magnification shows the DoFs from  $n_{\text{DoFs}} = 2, \dots, 32$  or Dill levels  $n_D = 1, \dots, 5$ , respectively. The run-time corresponds to one function call for calculating the accelerations  $\ddot{\mathbf{q}}$  of the system. Beside the different scaling of the  $y$ -values, the results depend on the computer architecture used. The measurements were conducted on machine 2 (cf. Table 2.5).

**Table 2.3:** Overview of simulation models for Lola with different modeling depth. Symbols  $\blacksquare$  and  $\square$  mean “yes” and “no”, respectively (from: [28]).

Model component	Reduced model		Full models		
	Type 1	Type 2	Type 3	Type 4	Type 5
Rigid body dynamics	$\blacksquare$	$\blacksquare$	$\blacksquare$	$\blacksquare$	$\blacksquare$
Gear elasticity	$\square$	$\square$	$\blacksquare$	$\square$	$\blacksquare$
Drive dynamics	$\square$	$\blacksquare$	$\blacksquare$	$\blacksquare$	$\blacksquare$
Contact model	Independent	Indep.	Indep.	FEM	FEM
<b>Mechanical DoFs</b>	6	<b>30</b>	<b>46</b>	30	46
Contact layer DoFs	48	48	48	864	864
Electrical DoFs	0	24	24	24	24
No. of 1 <sup>st</sup> order ODEs	60	132	164	948	980





**Figure 2.20:** Sparsity pattern of the mass-matrix of Lola’s model Type 2 (gray background) and Type 3. (HD-based gears from the head and roller screw drives from knees and ankle joints are not modeled elastically, therefore, the number of additional DoFs is not  $30 - 6$  but 16).

different Linux kernel and compiler versions and compared with each other. Finally, some remarks on the influence of the computer architecture and the compiler are given.

Figure 2.20 shows the branch induced sparsity pattern of the mass-matrix for both Lola models investigated. The mass-matrix is notably sparser in areas assigned to the motor-DoFs, only found in the Type 3 model. Since motor shafts have no children in the kinematic tree and because of the decoupled modeling of motors and segments the lower right part is the unit matrix. Furthermore, motor shafts are modeled mass-less

**Table 2.4:** Relevant values for calculating the possible speedup gained from RSC factorization when applied on both Lola models.

model type	number of DoFs $n_q$	triangle elements $n_{\text{dense}}(n_q)$	non-zero tri. elem. $n_{\text{nz}}$	triangle sparsity $\alpha$	possible speedup $1/\alpha^2$
Type 2	30	465	267	0.57419	3.0331
Type 3	46	1081	420	0.38853	6.6245

which only influences the rotational part of the 6-DoF root (the first 3 DoFs).<sup>13</sup>

Table 2.4 presents relevant values when applying the RSC factorization (see previous section) on the sparse mass-matrices. The values for the number of triangle elements  $n_{\text{dense}}(n_q)$ , sparsity factor  $\alpha \in [0, 1]$  and the possible speedup are calculated with the formulas shown in Table 2.2. The number of non-zero triangle elements  $n_{\text{nz}}$  is gained from the sparsity pattern in Figure 2.20. Especially for the Type 3 model a distinctive solver speedup of more than 6.6 compared to the  $\mathcal{O}(n^3)$ -algorithm could be gained.

Figure 2.21 compares the run-times of the  $\mathcal{O}(n^3)$  and  $\mathcal{O}(n)$ -methods when applied to Lola model Type 2 and 3. Furthermore, the run-time of each individual step is shown. The plot reveals several interesting facts about the algorithms. First, as expected the overall run-times of the  $\mathcal{O}(n^3)$  increase with the larger model. However, this is not the case for the  $\mathcal{O}(n)$ -algorithm with sub-systems. Especially the run-time of the 1<sup>st</sup> forward recursion (FW1) is significantly larger for the smaller model type 2. Hence, the overhead due to the sub-systems is large. Furthermore, the speedup of the sparse solver is approx. 2 and not 6.6 as expected from Table 2.4.

The computer system on which the calculations are conducted also play an important role. Therefore, the performance measurements are conducted on five different computers. Key specifications for each computer are listed in Table 2.5. All computers run the GNU/Linux Operating System (OS). All programs were compiled using the C++ compiler of the GNU Compiler Collection (GCC) with optimization enabled.

Table 2.6 shows the results gathered on these computers. In order to compare the run-times of different FD-algorithms and study the influence of different computer systems at the same time, a relative run-time measure or efficiency is used. The relative run-time is normalized to the  $\mathcal{O}(n)$ -method such that

$$\eta(X) := \frac{\Delta t_{\text{eval}}(X)}{\Delta t_{\text{eval}}(\mathcal{O}(n))} \quad \text{with } X \in \{\mathcal{O}(n^3), \text{ sparse } \mathcal{O}(n^3)\},$$

where  $\eta(X)$  is the relative efficiency of method  $X = \{\mathcal{O}(n^3), \text{ sparse } \mathcal{O}(n^3)\}$  compared to the  $\mathcal{O}(n)$ -algorithm and  $\Delta t_{\text{eval}}(Y)$  denotes the run-time of one function call using

<sup>13</sup> FEATHERSTONE's algorithm however, cannot exploit sparsity due to special mass and/or mass-moment of inertia values and only sparsity from branching topology is considered.

**Table 2.5:** Test System Specifications ordered by the product launch date of the CPU.

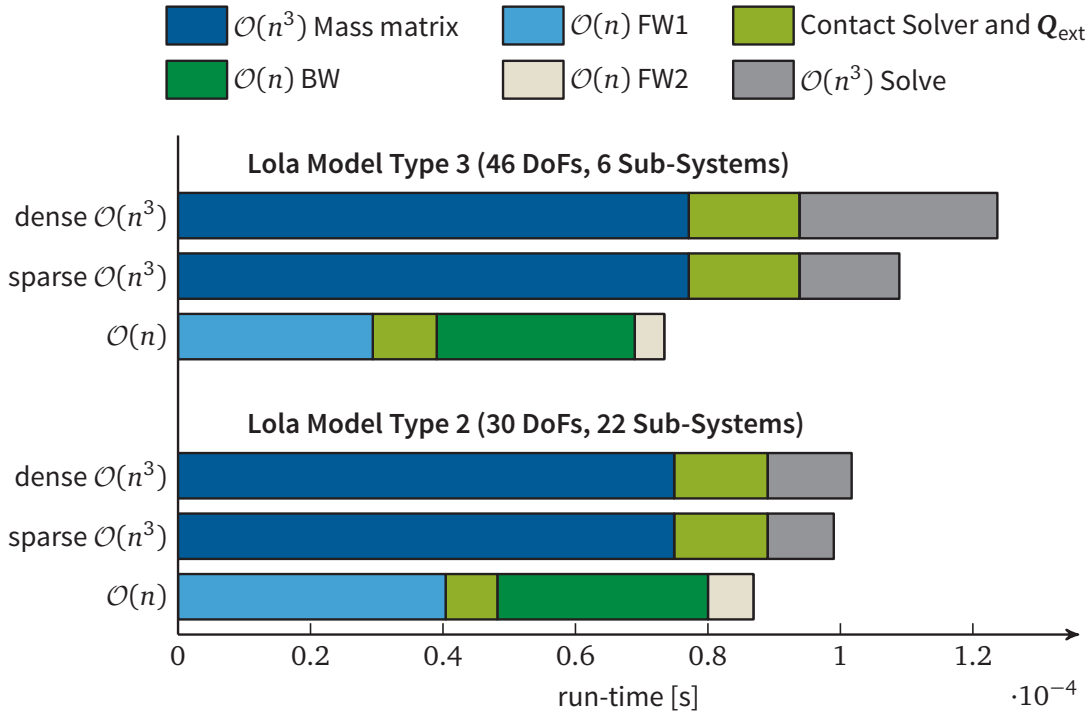
Machine	CPU					Software		
	vendor and product name	launch date <sup>a</sup> [-]	cores [-]	fre- quency [GHz]	cache size [KB]	RAM [GB]	Linux kernel version	GCC compiler version
1	Intel Core2 Duo E8600	Q3 2008	2	3.33	6144	8	3.5.0	4.7.2
2	Intel Core i7 Q 820	Q3 2009	4	1.73	8192	4	3.5.0	4.7.2
3	AMD Opteron 6172	Q1 2010	12	2.10	512	32	2.6.37	4.5.1
4	Intel Core i7 2600	Q1 2011	4	3.40	8192	8	3.6.11	4.6.3
5	Intel Core i7 3770K	Q2 2012	4	4.90 <sup>b</sup>	8192	8	3.4.11	4.7.1

<sup>a</sup> The quarter the processor was first introduced.

<sup>b</sup> overclocked from nominal 3.90 GHz

**Table 2.6:** Comparison of the run-time of one FD-algorithm function call obtained with Lola's models Type 2 and 3 and on different computer systems (cf. Table 2.5).

Model	Machine	run-time func. eval. $\Delta t_{\text{eval}}(\mathcal{O}(n))$ [s]	relative run-time <sup>a</sup> $\eta(\mathcal{O}(n^3))$ [-]	relative run-time <sup>b</sup> $\eta(\text{sparse})$ [-]
Type 2	1	8.916e-05	1.162	1.125
	2	1.217e-04	1.054	1.038
	3	1.893e-04	1.005	0.952
	4	7.771e-05	0.941	0.904
	5	5.367e-05	1.011	0.979
Type 3	1	7.517e-05	1.653	1.477
	2	9.831e-05	1.561	1.417
	3	1.410e-04	1.671	1.437
	4	6.403e-05	1.411	1.243
	5	4.612e-05	1.472	1.247



**Figure 2.21:** Run-time comparison of the FD-algorithms applied to Lola Models Type 2 and 3. The individual steps of each algorithm are presented. This reveals several things: (a) the overhead due to sub-systems is large in Type 2 especially in the 1<sup>st</sup> forward dynamics (FD1), (b) the run-time of Model Type 3 is shorter for the  $\mathcal{O}(n)$ -algorithm because of fewer sub-systems, and, (c) the speedup due to the sparse solver on model Type 3 is approx. 2 not 6.6 as expected from Table 2.4. The measurements were conducted on machine 1 (see Table 2.5).

method  $Y$ . For  $\eta(X) > 1$  the  $\mathcal{O}(n)$ -algorithm is faster and for  $\eta(X) < 1$  method  $X$  is faster than the  $\mathcal{O}(n)$ -algorithm. Moreover, the absolute run-time of the  $\mathcal{O}(n^3)$ -algorithms can simply be calculated by  $\Delta t_{\text{eval}}(X) = \eta(X)\Delta t_{\text{eval}}(\mathcal{O}(n))$ .

**Discussion** The relative run-times in Table 2.6 reveals an interesting factor. Since the same computations are involved, all relative run-times in a column associated to one model should be identical or at least in a close neighborhood due to measurement errors. However, the efficiency varies significantly between computers. Hence, there must be another influence on the run-time which also depends on the properties of the computer itself—a fact that will be discussed in the next section.

The model Type 2 is too small to gain any distinctive speedup by the  $\mathcal{O}(n)$ -algorithm with sub-systems. In some cases, the  $\mathcal{O}(n)$ -algorithm with sub-system is even slower (cf. Table 2.6, machine 4). Evidently, by comparing absolute run-time values for both models, the overhead caused by sub-system calculations is large. Therefore, for model Type 3, which has fewer sub-systems, a distinctive speedup is visible. The speedup of

the  $\mathcal{O}(n)$ -algorithm could even become larger with more independent DoFs.

Finally, the numbers in the columns associated to the sparse  $\mathcal{O}(n^3)$  method are always smaller than the  $\mathcal{O}(n^3)$  method. This is not surprising since the method exploits the sparsity of the mass-matrix.

## 2.11 Discussion

### The Influence of Different Computer Platforms

Due to the observations made here, some components which influence the performance most can be identified:

- CPU, with its frequency and cache memory size,
- compiler optimizations,<sup>14</sup>
- amount and bandwidth of the RAM.<sup>15</sup>

Moreover, if file I/O plays an important role, the OS and memory storage (along with its file system) also influence the overall performance.

Modern CPUs have a series of dynamic frequency scaling technologies implemented. This techniques enables the processor to better exploit the thermal capacity of the cooling system by increasing both voltage and frequency of one core and throttling the rest. This can lead to significantly faster execution of single threaded programs.

Due to Ohm's law the dissipated power of a CPU can be approximated by:

$$P \approx C V^2 f , \quad (2.91)$$

where  $P$  is the power,  $C$  the capacitance,  $V$  the voltage, and  $f$  is the frequency [46].

In order to avoid damage due to a overheating CPU, the frequency and voltage are lowered which in turn reduces the performance. Therefore, for real-time applications and/or single threaded applications this technique can distort performance measurements. The "Turbo Boost Technology"<sup>16</sup> and "Turbo Core Technology"<sup>17</sup> from Intel and AMD, respectively, are examples for this technique.

A discussion of architectural differences of a diverse set of multicore CPU configurations applied to high performance computing is found in [155]. The results therein suggest that the memory bandwidth can become a significant performance bottleneck for large models.

<sup>14</sup> in this work, only the GCC C++ compiler was tested

<sup>15</sup> RAM is important especially for larger systems, when the program does not fit into CPU cache

<sup>16</sup> <http://www.intel.com/technology/turboboost/>

<sup>17</sup> [www.amd.com](http://www.amd.com)

### Counting Operations

In earlier work on efficient dynamics algorithms, a classical approach to approximate the run-time of a method —and hence, to compare the efficiency— was to count operations for multiplications and additions [21, 53, 55, 152]. This approach still gives valid ordinal numbers, i. e.,  $\mathcal{O}(n^3)$ -algorithms will always take more time to compute than  $\mathcal{O}(n)$ -algorithms for sufficiently large  $n$ . However, with more refined computer architectures the approximation is quite inaccurate for current (and probably future) computer architectures.

### Differences and Similarities to Bremer’s Sub-System Representation

Finally, some important differences to the method developed by BREMER are outlined here. The method developed in this thesis uses 6-D vectors throughout. This enables an easier integration to possibly existing  $\mathcal{O}(n)$  or  $\mathcal{O}(n^3)$  algorithms. Moreover, fast SIMD operations can be exploited more naturally. SIMD operations are used in this work when reasonable, which gains approx. 20 % run-time performance. On the other hand, BREMER and GATTRINGER use an extended state vector  $\mathbf{y}$  with  $\dim(\mathbf{y}) > 6$ .

By using the automatic restructuring of MBS into sub-systems (cf. Section 2.9), sub-systems are only created where necessary to resolve closed loops. Hence, no special model must be built and the optimal  $\mathcal{O}(n)$  algorithmic complexity is maintained when no loops exist. On the other hand, BREMER’s method gains efficiency only when all equations are pre-processed, e.g., using a computer algebra system (possibly with automatic code export capabilities to reduce the implementation effort).

BREMER writes the EoM in a generalized form (cf. Eq. (2.58)) and gives examples. Hence, it can be seen as a more formal approach.

## 2.12 Chapter Summary

This chapter gave an overview of the implemented  $\mathcal{O}(n)$ -algorithm to solve the multi-body system dynamics. The algorithm is able to handle (small) kinematic loops quite efficiently. Sub-systems are used to group loop-bodies. This effectively restores tree topology required by the  $\mathcal{O}(n)$ -algorithm.

Since a system modeled with and without sub-systems exposes a different underlying structure, an automatic conversion of an existing regular MBS into sub-systems is applied. This drastically reduces the modeling effort to build a separate model using sub-systems. The basic approach is based on grouping bodies with overlapping degrees of freedom and converting these groups into sub-systems.

Finally, a study comparing the run-time performance for different dynamics algorithms applied to different models is presented. For the algorithm developed in this work the study shows good results, especially for systems comprising only few

sub-systems. Therefore, sub-systems must be seen as a compromise to enable the use of the very efficient  $\mathcal{O}(n)$ -algorithms to systems with closed kinematic loops with an algebraic solution.





## 3 Kinematics

In this chapter some special kinematic calculations used for the robot Lola are given. Here, we concentrate on the derivation of adequate quantities used in dynamics calculations and/or inverse kinematics.

Drive kinematics for ankle joint and knee joint of Lola are described by LOHMEIER [85] and according calculations of kinematic quantities by BUSCHMANN [28]. In order to apply those joints mechanisms to the  $\mathcal{O}(n)$ -algorithm using sub-systems (cf. Chapter 2) additional time derivatives of quantities are needed and derived in this chapter.

Moreover, the kinematic calibration for Lola's absolute angular encoders is described here.

### 3.1 Harmonic Drive Gears

The quantities for Harmonic Drive gear kinematics are similar to the ones from revolute joint kinematics presented on page 23.

The Jacobian  $J_i$  for drive body  $i$  in (2.33) is

$$J = [0 \ 0 \ N \ 0 \ 0 \ 0]^T = \text{const.} \quad \Rightarrow \quad \dot{J} = \mathbf{0} \in \mathbb{R}^{6 \times 1}, \quad (3.1)$$

$$A = A_z(Nq) A_{k'p} = \begin{bmatrix} \cos(Nq) & \sin(Nq) & 0 \\ -\sin(Nq) & \cos(Nq) & 0 \\ 0 & 0 & 1 \end{bmatrix} A_{k'p}. \quad (3.2)$$

Here,  $N$  is the gear ratio and  $A_{k'p}$  is a constant transformation matrix denoting the initial orientation (cf. Fig. 2.9 on page 20)

### 3.2 Knee Joint Drive Kinematics

Lola's knee joint rotates the lower leg segment relative to the thigh segment and is actuated via the nonlinear mechanism illustrated in Figure 3.1. Rotation is generated by the nonlinear 4-bar mechanism where linear motion from the planetary roller screw drive is converted into knee joint rotation. A motor attached to the screw drive actuates the mechanism. Since rotational dynamics about the screw axis is dominant, dynamic effects of the rotation orthogonal to the screw axis are neglected.

The following derivation is adopted from [28] and extended here.

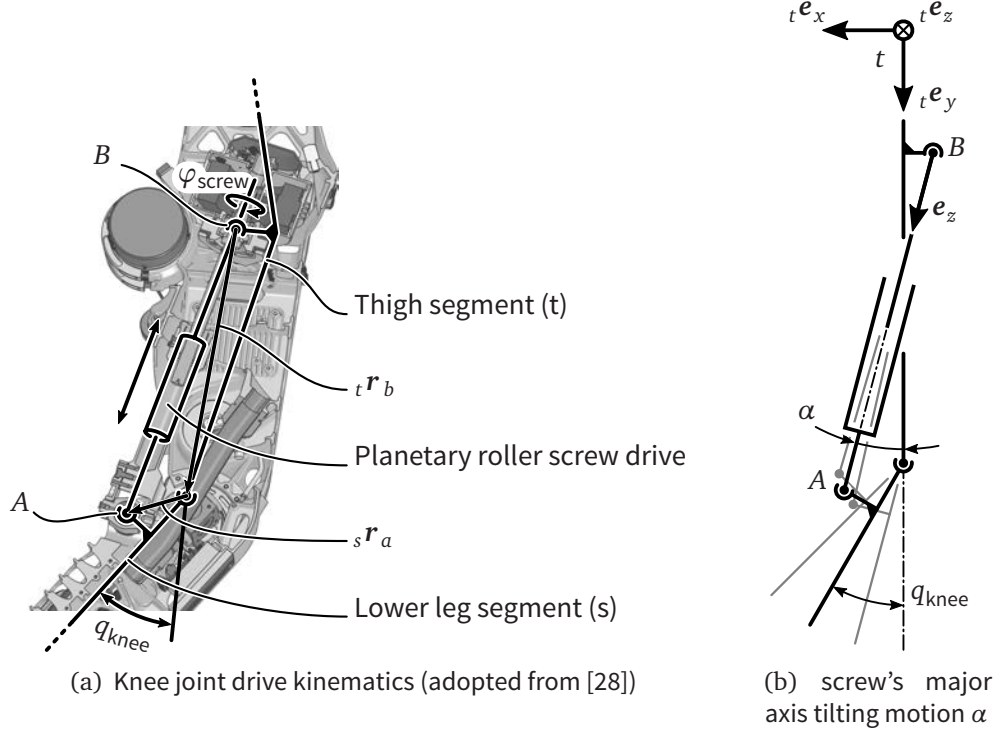


Figure 3.1: Knee joint drive kinematics

The screw's rotation is proportional to the screw lead  $P_{\text{lead}}$  and planetary roller screw drive length  $l_{\text{screw}}$ , which varies due to nut's displacement:

$$\varphi_{\text{screw}} = \frac{2\pi}{\underbrace{P_{\text{lead}}}_{N_{\text{screw}} = \text{const.}}} l_{\text{screw}}. \quad (3.3)$$

Calculating the relative kinematics therefore requires computing  $l_{\text{screw}}$ . The vector from the hinging point on the thigh to the pivoting point on the lower leg ( $\mathbf{r}_{\text{screw}}$ ) and the distance between these two points ( $l_{\text{screw}}$ ) and their time derivatives are given by:

$$\mathbf{r}_{\text{screw}} = {}_t\mathbf{r}_b + \mathbf{A}_{ts} {}_s\mathbf{r}_a, \quad l_{\text{screw}} = \|\mathbf{r}_{\text{screw}}\| = \sqrt{\mathbf{r}_{\text{screw}}^T \mathbf{r}_{\text{screw}}}, \quad (3.4)$$

$$\dot{\mathbf{r}}_{\text{screw}} = \dot{\mathbf{A}}_{ts} {}_s\mathbf{r}_a, \quad \dot{l}_{\text{screw}} = \frac{\mathbf{r}_{\text{screw}}^T \dot{\mathbf{r}}_{\text{screw}}}{l_{\text{screw}}}, \quad (3.5)$$

$$\ddot{\mathbf{r}}_{\text{screw}} = \ddot{\mathbf{A}}_{ts} {}_s\mathbf{r}_a, \quad \ddot{l}_{\text{screw}} = \frac{\dot{\mathbf{r}}_{\text{screw}}^T \dot{\mathbf{r}}_{\text{screw}} + \mathbf{r}_{\text{screw}}^T \ddot{\mathbf{r}}_{\text{screw}} - \dot{l}_{\text{screw}}^2}{l_{\text{screw}}}. \quad (3.6)$$

This yields:

$$\dot{\varphi}_{\text{screw}} = N_{\text{screw}} \dot{l}_{\text{screw}}, \quad \ddot{\varphi}_{\text{screw}} = N_{\text{screw}} \ddot{l}_{\text{screw}}, \quad (3.7)$$

$$\boldsymbol{\omega}_{\text{rel}} = \mathbf{e}_z \dot{\varphi}_{\text{screw}}, \quad (3.8)$$

and the spatial velocity vector becomes

$$\mathbf{v} = \mathbf{C} \mathbf{v}_p + \begin{bmatrix} \boldsymbol{\omega}_{\text{rel}} \\ \mathbf{0} \end{bmatrix}, \quad (3.9)$$

where  $\mathbf{C}$  is the spatial transformation and  $\mathbf{v}_p$  the reference velocity from parent  $p$ . Finally, the relative joint Jacobians are obtained by partial differentiation:

$$\mathbf{J} = \begin{bmatrix} \mathbf{e}_z \\ \mathbf{0} \end{bmatrix} \frac{N_{\text{screw}}}{l_{\text{screw}}} \mathbf{r}_{\text{screw}}^T \nabla_{\dot{q}} \dot{\mathbf{r}}_{\text{screw}}, \quad (3.10)$$

$$\dot{\mathbf{J}} = \begin{bmatrix} \mathbf{e}_z \\ \mathbf{0} \end{bmatrix} \frac{N_{\text{screw}}}{\nabla_q l_{\text{screw}}} \nabla_q \mathbf{r}_{\text{screw}}^T \dot{\mathbf{r}}_{\text{screw}}, \quad (3.11)$$

using the abbreviations

$$\nabla_q \dot{\mathbf{r}}_{\text{screw}} = \frac{\partial \dot{\mathbf{r}}_{\text{screw}}}{\partial \dot{q}_{\text{knee}}} = \frac{\partial \dot{\mathbf{A}}_{ts}}{\partial \dot{q}_{\text{knee}}} {}_s \mathbf{r}_a, \quad (3.12)$$

$$\nabla_q \mathbf{r}_{\text{screw}} = \frac{\partial \mathbf{r}_{\text{screw}}}{\partial q_{\text{knee}}} = \frac{\partial \mathbf{A}_{ts}}{\partial q_{\text{knee}}} {}_s \mathbf{r}_a, \quad (3.13)$$

$$\nabla_q l_{\text{screw}} = \|\nabla_q \mathbf{r}_{\text{screw}}\|. \quad (3.14)$$

The tilting motion of the screw axis about pivot point  $B$  is neglected in the model above, i.e.,  $\mathbf{e}_z := {}_t \mathbf{e}_y = \text{const}$ . This is feasible for Lola since the geometric relations, i.e.,  $\|\mathbf{r}_a\| / \|\mathbf{r}_b\| \ll 1$ , and the high gear ratio  $N_{\text{screw}}$  do allow for this simplification.

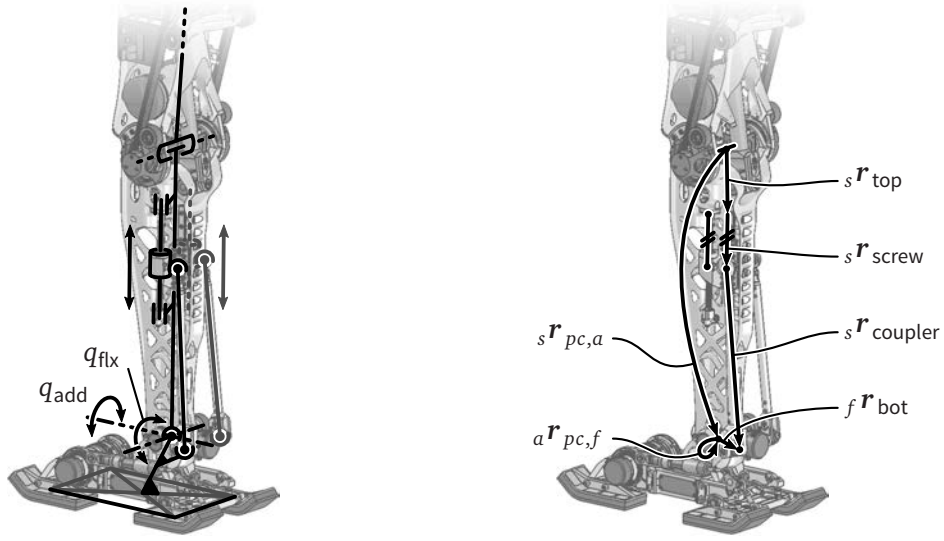
If, however, tilting motion is not negligible, angle  $\alpha$  between screw major axis  $\mathbf{e}_z$  and  ${}_t \mathbf{e}_y$ -axis in Figure 3.1b has to be considered:

$$\alpha = \text{atan} \left( \frac{\mathbf{r}_{\text{screw},x}}{\mathbf{r}_{\text{screw},y}} \right), \quad (3.15)$$

$$\boldsymbol{\alpha} = {}_t \mathbf{e}_z \alpha, \quad (3.16)$$

$$\dot{\boldsymbol{\alpha}} = \frac{\dot{\mathbf{r}}_{\text{screw}} \times \mathbf{r}_{\text{screw}}}{l_{\text{screw}}^2}, \quad (3.17)$$

$$\ddot{\boldsymbol{\alpha}} = \frac{\ddot{\mathbf{r}}_{\text{screw}} \times \mathbf{r}_{\text{screw}}}{l_{\text{screw}}^2} - \frac{2\dot{l}_{\text{screw}}}{l_{\text{screw}}^3} \dot{\mathbf{r}}_{\text{screw}} \times \mathbf{r}_{\text{screw}}. \quad (3.18)$$



**Figure 3.2:** Kinematics of the ankle drive mechanism to drive adduction and flexion motion of the foot. (adopted from [28])

Which yields:

$$\boldsymbol{\omega}_{\text{rel}} = \dot{\boldsymbol{\alpha}} + \mathbf{A}_z(\alpha) {}_t\mathbf{e}_y \dot{\varphi}_{\text{screw}}, \quad (3.19)$$

$$\mathbf{J}_{R,\text{rel}} = \frac{\partial \boldsymbol{\omega}_{\text{rel}}}{\partial \dot{q}_{\text{knee}}} = \frac{\nabla_{\dot{q}} \dot{\mathbf{r}}_{\text{screw}} \times \mathbf{r}_{\text{screw}}}{l_{\text{screw}}^2} + \frac{N_{\text{screw}}}{l_{\text{screw}}} \mathbf{r}_{\text{screw}}^{\top} \nabla_{\dot{q}} \dot{\mathbf{r}}_{\text{screw}} \mathbf{A}_z(\alpha) {}_t\mathbf{e}_y, \quad (3.20)$$

$$\dot{\mathbf{J}}_{R,\text{rel}} = \frac{\partial \boldsymbol{\omega}_{\text{rel}}}{\partial q_{\text{knee}}} = \frac{\dot{\mathbf{r}}_{\text{screw}} \times \nabla_q \mathbf{r}_{\text{screw}}}{(\nabla_q \mathbf{r}_{\text{screw}})^2} + \frac{N_{\text{screw}}}{\nabla_q l_{\text{screw}}} \nabla_q \mathbf{r}_{\text{screw}}^{\top} \dot{\mathbf{r}}_{\text{screw}} \frac{\partial \mathbf{A}_z(\alpha)}{\partial q_{\text{screw}}} {}_t\mathbf{e}_y, \quad (3.21)$$

where  $\mathbf{A}_z(\alpha)$  is an elementary rotation matrix about  $z$  using  $\alpha$ . Again, since gear ratio  $N_{\text{screw}}$  is quite large for Lola, i.e.,  $N_{\text{screw}} \approx 1256.6 \frac{1}{\text{m}}$ , the second term dominates above equations.

### 3.3 Ankle Joint Drive Kinematics

Each ankle joint of Lola is driven by a pair of parallel, spatial slider-crank mechanisms shown in Figure 3.2. The linear motion of a carriage is produced by planetary roller screw drives. The movement of the coupler link connecting the linear carriage to the foot segment relative to the lower leg is comparatively slow and the coupler link is very lightweight. Inertial effects due to the coupler link's motion are therefore neglected and only the much larger effects due to the screw's rotation are taken into account [28].

Similar to the procedure for the knee joint drive, ankle joint kinematics requires

the displacement of the nut  $l_{\text{screw}}$  due to screw's rotation  $\varphi_{\text{screw}}$  as a function of the generalized coordinates  $\mathbf{q}_{\text{ankle}} = [q_{\text{ankle,flexion}} \quad q_{\text{ankle,adduction}}]^T$ :

$$\varphi_{\text{screw}} = \frac{2\pi}{\underbrace{P_{\text{lead}}}_{N_{\text{screw}}}} l_{\text{screw}}(q_{\text{flx}}, q_{\text{add}}). \quad (3.22)$$

Referring to Figure 3.2, the vector chain along the closed kinematic loop can be written as

$${}_s \mathbf{r}_{\text{coupler}} = \underbrace{{}_s \mathbf{r}_{pc,a} + \mathbf{A}_{sa} (\mathbf{A}_{af} {}_f \mathbf{r}_{\text{bot}} + {}_a \mathbf{r}_{pc,f}) - {}_s \mathbf{r}_{\text{top}}}_{{}_s \mathbf{r}_{\text{coupler},0}} - \underbrace{l_{\text{screw}} \mathbf{e}_{\text{screw}}}_{{}_s \mathbf{r}_{\text{screw}}}. \quad (3.23)$$

Here,  ${}_s \mathbf{e}_{\text{screw}}$  is the unit vector along the screw's major axis and indices  $s$ ,  $a$ ,  $f$  and  $c$  denote the shank, ankle, foot and coupler link respectively. Vector  ${}_a \mathbf{r}_{pc,f}$  connecting ankle frame and foot frame, is zero for Lola and hence, shows up as loop in Figure 3.2.

The coupler length is known

$$l_{\text{coupler}}^2 = {}_s \mathbf{r}_{\text{coupler}}^T {}_s \mathbf{r}_{\text{coupler}} = \text{const.} \quad (3.24)$$

which is exploited to solve for the roller screw nut displacement  $l_{\text{screw}}$ :

$$l_{\text{screw}} = {}_s \mathbf{r}_{\text{coupler},0}^T \mathbf{e}_{\text{screw}} (\pm) \sqrt{l_{\text{coupler}}^2 - {}_s \mathbf{r}_{\text{coupler},0}^2 + ({}_s \mathbf{r}_{\text{coupler},0}^T \mathbf{e}_{\text{screw}})^2}. \quad (3.25)$$

Equation (3.2) solves the intersection between a line along  ${}_s \mathbf{e}_{\text{screw}}$  and a sphere with radius  $l_{\text{coupler}}$  centered at the according coupler point on the foot. Therefore, for the individual vector directions defined in Figure 3.2 the minus sign gives the correct solution.<sup>1</sup>

From  $l_{\text{screw}}$  the rotation relative to the shank  $\varphi_{\text{screw}}$  and the relative angular velocity  $\boldsymbol{\omega}_{\text{screw,rel}}$  are computed as:

$$\dot{\varphi}_{\text{screw}} = N_{\text{screw}} \dot{l}_{\text{screw}}, \quad \boldsymbol{\omega}_{\text{screw,rel}} = \mathbf{e}_z \dot{\varphi}_{\text{screw}}, \quad (3.26)$$

$$\ddot{\varphi}_{\text{screw}} = N_{\text{screw}} \ddot{l}_{\text{screw}}. \quad (3.27)$$

By using the following abbreviations

$$\mathbf{e} := {}_s \mathbf{e}_{\text{screw}}, \quad \mathbf{r} := {}_s \mathbf{r}_{\text{coupler},0}, \quad (3.28a)$$

$$l_s := l_{\text{screw}}, \quad l_c := l_{\text{coupler}}, \quad (3.28b)$$

$$\mathbf{P} := \mathbf{e} \mathbf{e}^T - \mathbf{E} \Rightarrow \mathbf{P} = \mathbf{P}^T, \quad (3.28c)$$

<sup>1</sup> The plus sign would give the point below the foot, which is infeasible.

where  $\mathbf{P}$  is a symmetric matrix projecting a vector to be perpendicular to  ${}_s\mathbf{e}_{\text{screw}}$ , (3.25) is rewritten to

$$l_s = \mathbf{e}^\top \mathbf{r} - \underbrace{\sqrt{l_c^2 + \mathbf{r}^\top \mathbf{P} \mathbf{r}}}_d, \quad (3.29)$$

and velocity, accelerations and relative Jacobians are determined by straightforward differentiation:

$$\dot{l}_s = \mathbf{p}^\top \dot{\mathbf{r}}, \quad (3.30)$$

$$\ddot{l}_s = \mathbf{p}^\top \ddot{\mathbf{r}} + \dot{\mathbf{p}}^\top \dot{\mathbf{r}}, \quad (3.31)$$

$$\frac{\partial \dot{l}_s}{\partial \dot{\mathbf{q}}_{\text{ankle}}} = \mathbf{p}^\top \frac{\partial \dot{\mathbf{r}}}{\partial \dot{\mathbf{q}}_a}, \quad (3.32)$$

$$\frac{d}{dt} \left( \frac{\partial \dot{l}_s}{\partial \dot{\mathbf{q}}_{\text{ankle}}} \right) = \mathbf{p}^\top \frac{\partial \dot{\mathbf{r}}}{\partial \mathbf{q}_a} + \dot{\mathbf{p}}^\top \frac{\partial \mathbf{r}}{\partial \mathbf{q}_a}. \quad (3.33)$$

Here,  $\mathbf{p}$  and  $\dot{\mathbf{p}}$  are projection vectors defined as

$$\mathbf{p} := \mathbf{e} - \frac{\mathbf{P} \mathbf{r}}{d}, \quad (3.34)$$

$$\dot{\mathbf{p}} := \frac{d\mathbf{p}}{dt} = \frac{(\dot{\mathbf{r}}^\top \mathbf{P} \mathbf{r}) \mathbf{P} \mathbf{r}}{d^3} - \frac{\mathbf{P} \dot{\mathbf{r}}}{d}. \quad (3.35)$$

Finally, the Jacobians of the screw (which is driven by the motor) become

$${}^{\text{screw}}\mathbf{J}_{R,rel} = \frac{\partial \boldsymbol{\omega}_{\text{screw,rel}}}{\partial \dot{\mathbf{q}}_{\text{ankle}}} = N_{\text{screw}} \frac{\partial \dot{l}_{\text{screw}}}{\partial \dot{\mathbf{q}}_{\text{ankle}}}, \quad (3.36)$$

$${}^{\text{screw}}\dot{\mathbf{J}}_{R,rel} = \frac{\partial \boldsymbol{\omega}_{\text{screw,rel}}}{\partial \mathbf{q}_{\text{ankle}}} = N_{\text{screw}} \frac{\partial \dot{l}_{\text{screw}}}{\partial \mathbf{q}_{\text{ankle}}}. \quad (3.37)$$

### 3.4 Camera Vergence Kinematics

Lola is equipped with a camera head with three degrees of freedom: pan, tilt and camera vergence. Pan and tilt joints are Harmonic Drive based rotary joints. The camera vergence angle  $q_{\text{verg}}$ , denoting the angle between the viewing directions  $\mathbf{v}_r$  and  $\mathbf{v}_l$  of the right and left camera respectively, however, is adjusted by a nonlinear spatial crank mechanism shown in Figure 3.3. The mechanism is integrated in the tilt segment  $t$  and designed to ensure symmetric motion of the camera angle using a single actuator M (cf. [85] for a detailed description).

Since the vergence mechanism works in a symmetric fashion, only the left part of the

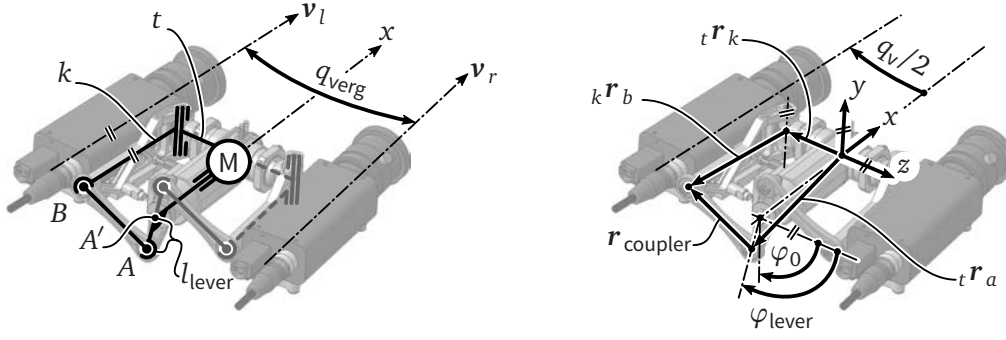


Figure 3.3: Camera vergence drive kinematics

mechanism is illustrated in Figure 3.3. Actuator M drives the double lever arm about the  ${}_t x$ -axis. The camera is mounted on the receptacle  $k$  which can rotate about the  ${}_t y$ -axis. Two coupler links connect both ends of the double lever arm to the receptacles. Thereby, rotation of the lever arm denoted by  $\varphi_{\text{lever}}$  is converted into camera vergence motion.

Since vergence motion is slow in normal operation, dynamics effects are neglected in simulation. For real-time control however, the nonlinear relation between desired vergence angle  $q_{\text{verg}}$  and actuator angle  $\varphi_{\text{lever}}$  must be known.

With reference to Figure 3.3 vectors  ${}_t \mathbf{r}_a$  and  ${}_t \mathbf{r}_b$  are the vectors from tilt segment origin to pivot points A and B, respectively. The vector chain is closed by vector  $\mathbf{r}_{\text{coupler}}$  connecting points A and B and we get

$${}_t \mathbf{r}_a = \begin{bmatrix} r_{a,x} \\ l_{\text{lever}} \sin(\varphi_{\text{lever}}) \\ l_{\text{lever}} \cos(\varphi_{\text{lever}}) \end{bmatrix}, \quad {}_t \mathbf{r}_b = {}_t \mathbf{r}_k + \mathbf{A}_{tk}(q_{\text{verg}}) {}_k \mathbf{r}_b, \quad (3.38)$$

$$\mathbf{r}_{\text{coupler}} = \mathbf{r}_a - \mathbf{r}_b, \quad l_{\text{coupler}} = \|\mathbf{r}_{\text{coupler}}\| = \text{const.} \quad (3.39)$$

Here,  $l_{\text{lever}}$  and  $l_{\text{coupler}}$  are the constant lengths of the lever arm and the coupler link, respectively.

In order to get a symmetric motion of the viewing vectors with respect to the  $x$ - $y$ -plane, the mechanism is designed such that point  $A'$ , denoting the rotation center of A, and point B are located on the  $x$ - $z$ -plane. Hence, we have  ${}_t r_{b,y} = 0$  and we get the simplified solution

$$l_{\text{coupler}} = \sqrt{(r_{a,x} - r_{b,x})^2 + l_{\text{lever}}^2 - 2l_{\text{lever}} r_{b,z} \cos \varphi_{\text{lever}} + r_{b,z}^2}, \quad (3.40)$$

$$\Rightarrow \varphi_{\text{lever}} = \arccos \left( \frac{s}{2l_{\text{lever}} r_{b,z}} \right), \quad (3.41)$$

where scalar  $s$  is introduced for abbreviation:

$$s := (r_{a,x} - r_{b,x})^2 + l_{\text{lever}}^2 - l_{\text{coupler}}^2 + r_{b,z}^2. \quad (3.42)$$

The motor angle  $\varphi_{\text{mot}}$  is calculated using the gear ratio  $N_{\text{verg}}$  from actuator M to

$$\varphi_{\text{mot}} = N_{\text{verg}} (\varphi_{\text{lever}} - \varphi_0), \quad (3.43)$$

where  $\varphi_0$  is the lever angle obtained by solving (3.41) with  $q_{\text{verg}} = 0$ .

The Jacobian acting as the transmission ratio of the mechanism is gained via straight-forward differentiation to

$$\frac{\partial \varphi_{\text{lever}}}{\partial q_{\text{verg}}} = \frac{2 r_{b,z} (r_{a,x} - r_{b,x}) \left( \nabla_{q_{\text{verg}}} {}^t \mathbf{r}_b \right)_x - s \left( \nabla_{q_{\text{verg}}} {}^t \mathbf{r}_b \right)_z}{2 l_{\text{lever}} r_{b,z}^2 \sqrt{1 - \left( \frac{s}{2 l_{\text{lever}} r_{b,z}} \right)^2}}, \quad (3.44)$$

with

$$\nabla_{q_{\text{verg}}} {}^t \mathbf{r}_b = \frac{\partial \mathbf{A}_{tk}}{\partial q_{\text{verg}}} {}^k \mathbf{r}_b. \quad (3.45)$$

Figure 3.4 shows the lever angle and its Jacobian as a function of the vergence angle of the camera vergence joint drive.

### 3.5 Kinematics Calibration

Kinematics calibration is the process of finding correct physical *kinematic parameters* for the robot's joints. Geometric errors due to tolerances from manufacturing lead to deviations of relative joint orientations and locations from ideal CAD data. Additionally, sensor gain and bias errors exist. All these errors are amplified in a serial link configuration leading to loss in manipulator accuracy.<sup>2</sup> Obviously, accuracy greatly influences walking performance and model based collision avoidance relies on a correct robot description.

Since accuracy plays an important role in industrial manipulators such as serial robots, many sophisticated methods for kinematic calibration have been developed with a focus on manipulators. Typically, industrial robots are quite heavy-weight machines whose stiff links combined with stiff gears provide a high degree of repeatability.<sup>3</sup> However, according to MOORING et al. [95] manipulator accuracies are often several orders of magnitudes worse than the repeatability. An adequate characterization of

<sup>2</sup> *Accuracy* is a measure of how well a robot is able to reach a predefined position in task space.

<sup>3</sup> *Repeatability* is a measure of how well a robot is able to return to a previously achieved pose.



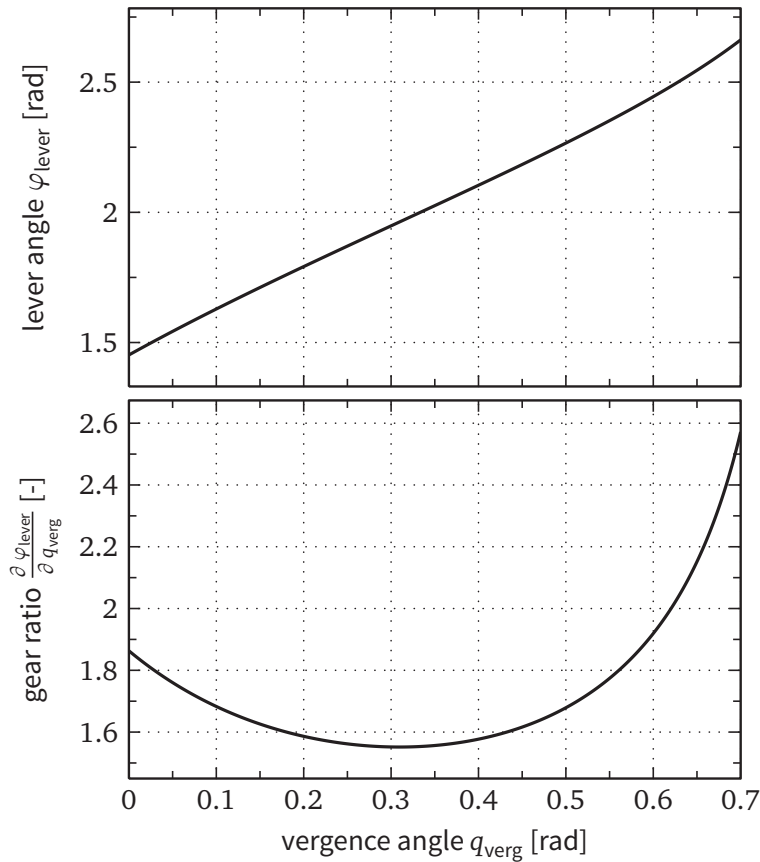


Figure 3.4: Nonlinear transmission of the camera vergence joint drive

kinematic modeling errors can improve accuracy up to the same order of magnitude as the repeatability [95].

A good review of state-of-art methods for kinematic calibration is given in [67]. They can be categorized in two general methods:

**Open-loop calibration** an external metrology system, such as coordinate measurement machines, theodolites, laser interferometers or vision systems, measures the robot pose without physical contact

**Closed-loop calibration** physical constraints, such as contacts or end-effector position fixtures, are applied forcing the robot into a predefined pose, hence, the name “closed-loop”

Each joint of Lola contains an incremental rotary encoder mounted on the motor shaft and an absolute angular encoder used as link position sensor (cf. [85] for more information). The absolute angular encoder eliminates the need for a homing routine, making start-up faster and easier<sup>4</sup> [87]. Thereby, correct motor angles are set by

<sup>4</sup> Only head joints have no absolute angular encoders and need a homing routine via limit switches.

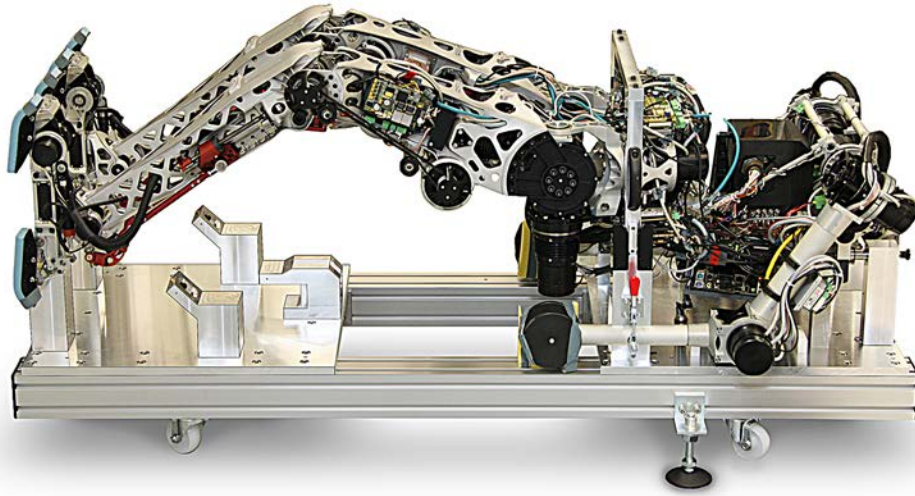


Figure 3.5: Photograph of Lola mounted on the kinematic calibration jig.

measuring absolute joint angles which are then converted to incremental encoder positions at start-up.<sup>5</sup> Since absolute angular encoders are mounted in an *arbitrary* angular position, sensor mounting misalignment must be calibrated.

The equipment for open-loop calibration requires high precision measurement systems which are very expensive [45]. Due to lack of such a high precision measurement system a closed-loop calibration procedure using a calibration jig is chosen.

Several assumptions are made in the developed of the calibration jig:<sup>6</sup>

- Geometric deviations are negligible, since special care was taken in design, manufacturing and assembly of the robot to ensure small geometric tolerances.
- Link and gear deformations due to dynamics and gravity effects during operation are assumed to be larger than geometric tolerances. Furthermore, load dependent deformations can only be measured with large effort.
- Sensor gain from absolute angular encoders (digital interface) is calibrated by the commercial manufacturer.
- Deformations due to thermal effects are negligible, since calibration is conducted under laboratory conditions.

Figure 3.5 shows a photograph of Lola mounted on the developed calibration jig. The jig is mostly made of aluminum alloy, is  $1510 \times 500 \times 560 \text{ mm}^3$  in size and weighs approx. 60 kg. It is designed to have a stiff and geometrically precise framing

5 For nonlinear joint mechanisms, such as knee and ankle joints, a Newton-Raphson based approach is used to find the according incremental encoder positions from given joint angles.

6 The calibration jig was developed in cooperation with SEBASTIAN LOHMEIER, cf. [85].

and provide easy mounting of the robot. Furthermore, it has swivel casters for fast positioning on the floor and three threaded legs and a bubble level for leveling out the jig before calibration (hidden in Fig. 3.5).

Six segments of the robot are fixed to the jig: pelvis, upper body, both feet and both lower arms. Precision locating pins mounted on the robot, a receptacle mounted on the pelvis prior to calibration and toggle clamps for the arms are used to position the robot. Thereby, the robot is set into a unique pose. Moreover, additional jigs are used to calibrate the toe and head joints.<sup>7</sup>

Since accuracy of the leg joints is vital for walking performance, the first six joints can be set into three different poses. This enables the verification of the assumed low geometric tolerances for the legs. As shown in Figure 3.6 the poses are set by mounting the feet to different positions.<sup>8</sup> Figure 3.7 shows the angles set by the poses for the right leg along with the according joint workspaces. Except for the hip adduction and flexion joints the workspace is covered quite well by the jig.

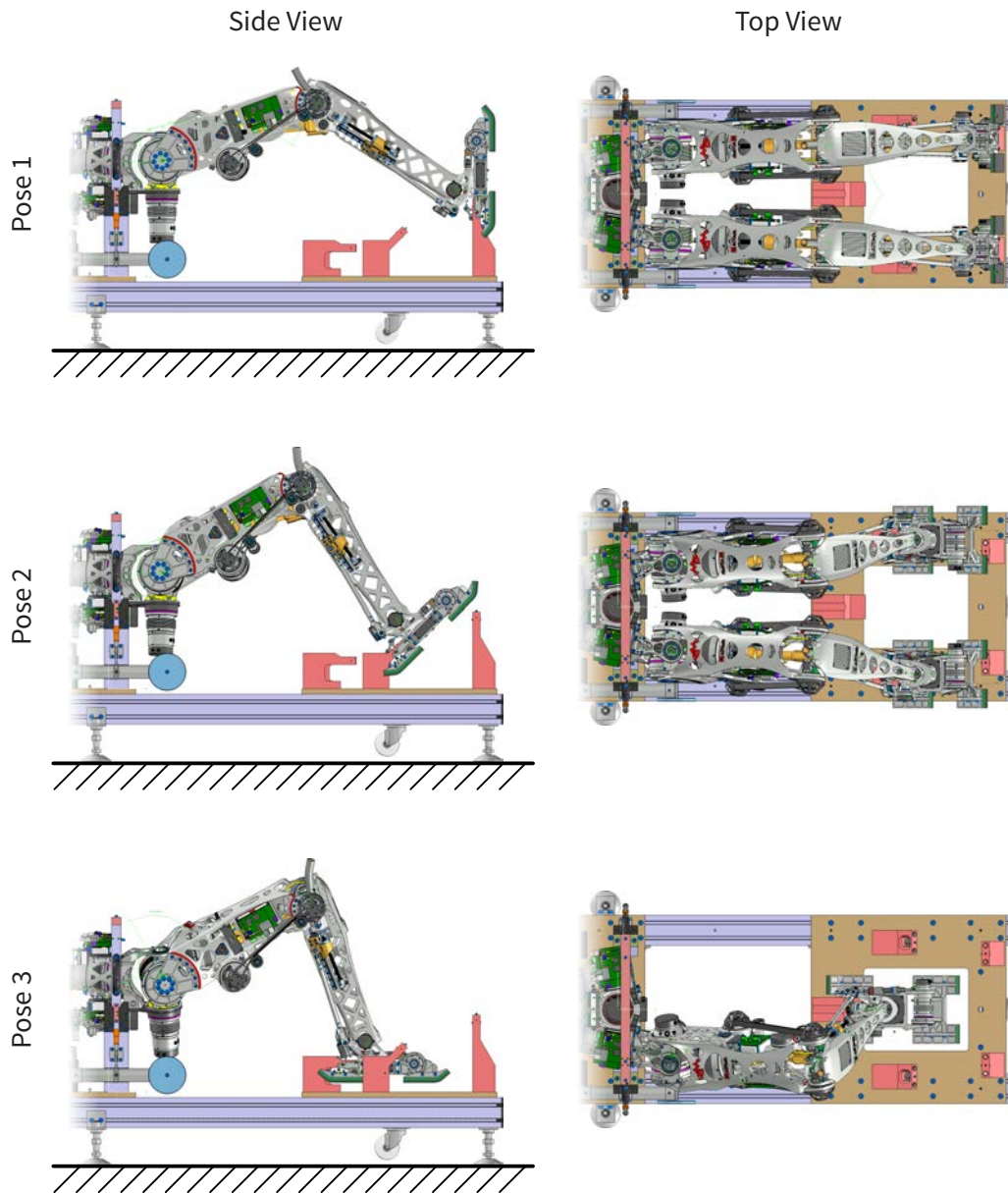
If, however, identification of geometric tolerances is desired, three poses are not enough. Assuming four Denavit-Hartenberg parameters to define the relative position and orientation for all six joint FoRs, at least 24 different foot positions per leg must be set to identify them. In the case of nearly parallel neighboring axes, such as the flexion joints from hip and knee, an additional parameter must be introduced [66], further increasing the number of poses. Moreover, in order to avoid an ill conditioned calibration problem, poses should be chosen carefully. The pose distribution can be optimized on a relevant subset of the joint's workspace by studying observability and applying sensitivity analysis of parameters. Therefore, a much more complex calibration jig has to be built to recover all parameters for the legs. On the other hand, if adequate equipment is available, open-loop calibration might be the better solution for the large set of free parameters. In any case, kinematic calibration effort increases strongly when attempting to identify more parameters than just the absolute angular encoder misalignment.

Experiments measuring sensor misalignment in all three poses show a good agreement. Therefore, it can be assumed that geometric deviations due to tolerances are negligible which was assumed initially. Furthermore, measurement uncertainties are present from small misalignments while mounting the robot to the jig. Finally, a one-pose calibration including mounting and dismounting of the robot takes about one hour.

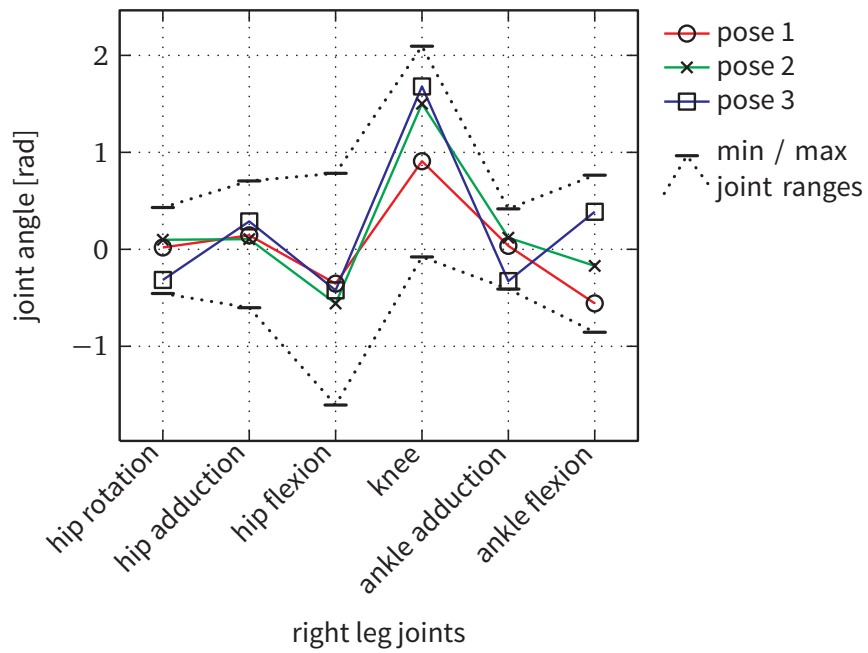
---

7 Relative limit switch positions used for the homing routine are identified by separate head joint calibration jigs not presented here.

8 For conciseness the term "feet position" is used here to define position *and* orientation of the feet



**Figure 3.6:** CAD drawings viewing from the side and top of all three calibration poses for Lola's legs. Pose 3 is used to calibrate one leg at a time only.



**Figure 3.7:** Joint angles of the right leg affected by the three calibration poses and the according joint angle workspaces. Depending on the joint definition, the angles for the left leg are identical or symmetric.

### 3.6 Chapter Summary

In this chapter, equations for the joint kinematics of Lola have been presented. These equations are used in both dynamics and inverse kinematics of the robot. Furthermore, the calibration of the robot is addressed. A closed-loop kinematic calibration is conducted by using a hardware jig.



## 4 Inverse Kinematics

This chapter reviews some existing methods to solve the inverse kinematics problem. The information is also intended as an introduction to the next chapters. Good surveys on the topic can be found in [40, 97, 128, 129, 145].

### 4.1 Problem Statement

Inverse Kinematics (IK) solvers are important tools in different fields like robotics, motion capturing and computer animation.

Formally, the goal of IK is to map a desired task space  $\mathbf{x}$  to generalized coordinates  $\mathbf{q}$ . Figure 4.1 illustrates this in the context of robotics. Typically, task space trajectories  $\mathbf{x}$  are the output of a planning and/or control stage and contain tasks like end-effector positions/orientations, CoG positions, gaze directions etc. Finally, generalized positions  $\mathbf{q}$  obtained from inverse kinematics are sent to the robot hardware. Depending on the setup, more (or other) quantities than positions might be interchanged, i. e., velocities, accelerations (cf. Fig. 4.1, quantities in parentheses).

Formally, the relationship between task space trajectories  $\mathbf{x} \in \mathbb{R}^m$  and joint coordinates  $\mathbf{q} \in \mathbb{R}^n$  is given by the direct kinematics equation

$$\mathbf{x} = f(\mathbf{q}), \tag{4.1}$$

where  $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$  is the forward kinematics function. Especially if rotatory joints are involved,  $f$  is nonlinear and a closed form solution to (4.1) exists only in rare cases. Such solutions are based on algebraic and geometric methods [40, 97, 129] and hence, are specific to a certain kinematic structure. Therefore, we focus on more generic approaches [97]. Moreover, only open kinematic chains with holonomic constraints are considered in this thesis. The general IK problem is given by

$$\{\mathbf{q} \mid f(\mathbf{q}) = \mathbf{x}\}. \tag{4.2}$$

Depending on the dimensions of  $\dim(\mathbf{x})$  and  $\dim(\mathbf{q})$  and the target pose  $\mathbf{x}$  there may be one, multiple or an infinite number of solutions [129]. Note that, due to numerous physical constraints, e. g., joint angle/velocity/acceleration/torque limits, it is always possible to define unfeasible tasks leading to singularities which degrade movement abilities. The problem of singularities is discussed in Section 4.4.

Methods to solve the nonlinear problem (4.2) can be categorized in two general

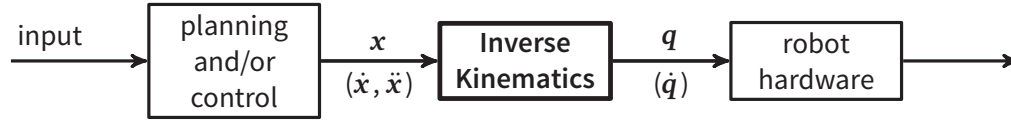


Figure 4.1: Inverse kinematics in the context of real-time robot control.

groups: (a) position based algorithms and (b) algorithms based on differential kinematics, i. e., time derivatives of (4.1). Thereby, group (a) is solved in an iterative manner while differential methods are often based on solutions to local optimization problems.

## 4.2 Position Based Inverse Kinematics

Algorithm 4.1 outlines an approach based on Newton-Raphson root finding of a residual  $\mathbf{r}(\mathbf{q}) := f(\mathbf{q}) - \mathbf{x} \stackrel{!}{=} \mathbf{0}$ . Here,  $\mathbf{J}_k^\#$  denotes the generalized inverse [14] of the Jacobian matrix  $\mathbf{J}_k := \partial f(\mathbf{q}_k)/\partial \mathbf{q} \in \mathbb{R}^{m \times n}$ :

$$\mathbf{J}_k^\# := \begin{cases} \mathbf{J}_k^\top (\mathbf{J}_k \mathbf{J}_k^\top)^{-1} & \text{if } m < n \\ \mathbf{J}_k^{-1} & \text{if } m = n. \end{cases} \quad (4.3)$$

Adequate stopping criteria, listed in Table 4.1, are combined via disjunction, i. e.,  $\sigma = \sigma_1 \wedge \sigma_2$ . Moreover,  $\mathbf{q}$  from the last time step is used for  $\mathbf{q}_0$  as starting point. PIEPER [115] was among the first to adopt this method to IK.

Especially when  $\mathbf{x}$  is defined close to a singularity, Algorithm 4.1 might not converge, since no root exists. However, it is still possible to minimize the residual leading to the minimization problem

$$\text{minimize } \phi(\mathbf{q}) \quad \text{with} \quad \phi(\mathbf{q}) := \frac{1}{2} \|f(\mathbf{q}) - \mathbf{x}\|^2. \quad (4.4)$$

Algorithm 4.1: Inverse kinematics using Newton-Raphson root finding.

---

**Input:**  $\mathbf{x} \in \mathbb{R}^m, \mathbf{q}_0 \in \mathbb{R}^n$  for  $m \leq n$

**Output:**  $\mathbf{q}$

- 1:  $k \leftarrow 0$
  - 2: **while** stopping criteria  $\sigma$  not satisfied **do**
  - 3:    $\mathbf{r}_k \leftarrow f(\mathbf{q}_k) - \mathbf{x}$
  - 4:    $\mathbf{q}_{k+1} \leftarrow \mathbf{q}_k - \mathbf{J}_k^\# \mathbf{r}_k$
  - 5:    $k \leftarrow k + 1$
  - 6: **end while**
  - 7:  $\mathbf{q} \leftarrow \mathbf{q}_k$
-



**Algorithm 4.2:** Inverse kinematics as optimization problem.

---

**Input:**  $\mathbf{x} \in \mathbb{R}^m, \mathbf{q}_0 \in \mathbb{R}^n$  for  $m \leq n$   
**Output:**  $\mathbf{q}$

- 1:  $k \leftarrow 0$
- 2: **while** stopping criteria  $\sigma$  not satisfied **do**
- 3:     direction  $\mathbf{d}_k$  from gradient  $\nabla f_k$
- 4:     determine step size  $\alpha_k$
- 5:      $\mathbf{q}_{k+1} \leftarrow \mathbf{q}_k + \alpha_k \mathbf{d}_k$
- 6:      $k \leftarrow k + 1$
- 7: **end while**
- 8:  $\mathbf{q} \leftarrow \mathbf{q}_k$

---

Algorithm 4.2 outlines an optimization based approach for solving (4.4). Again, stopping criteria from Table 4.1 are combined and  $\mathbf{q}_0$  is initialized with the result from last time step.

Descent direction vector  $\mathbf{d}_k$  is generated in Algorithm 4.2:3 using any adequate optimization method [105] (cf. [4] for examples). Optimization methods such as Conjugate Gradient (CG), Levenberg-Marquardt (LM) or Quasi-Newton (QN) have been applied to the inverse kinematics problem (CG [80], LM [98], QN [160]). Thereby, the search direction often has the form

$$\mathbf{d}_k = -\mathbf{B}_k^{-1} \nabla f_k, \quad (4.5)$$

where  $\mathbf{B}_k$  is a symmetric, positive definite matrix<sup>1</sup> and  $\nabla f_k$  is a gradient, e. g.,  $\nabla f_k := \partial \phi(\mathbf{q}_k) / \partial \mathbf{q}$ . For the Steepest Descent method  $\mathbf{B}_k$  is simply the identity matrix  $\mathbf{E}$ , while for Newton's method  $\mathbf{B}_k$  is the exact Hessian  $\nabla^2 f_k$  and for Quasi-Newton methods  $\mathbf{B}_k^{-1}$  is approximated iteratively [105]. On the other hand, the Conjugate Gradient method uses a different formula than (4.5) for updating direction  $\mathbf{d}_k$ . The convergence rate as well as the computational effort depends on the optimization method used. AYUSAWA et al. study in [4] the performance of some optimization methods applied to IK. Thereby, an approach to efficiently calculate the gradient  $\nabla f_k$  is presented. The difference between  $\mathbf{x}_k$  and goal  $\mathbf{x}$  is utilized to gain virtual generalized joint forces used as gradient  $\nabla f_k$ . Moreover, the method exploits fast dynamics algorithms (cf. Chapter 2) to gain efficiency [4].

Finally, step size  $\alpha_k$  in Algorithm 4.2:4 is either set constant or obtained from line search algorithms minimizing  $\phi(\mathbf{q}_k + \alpha_k \mathbf{d}_k)$  exactly or inexactly by satisfying Armijo or Wolfe-Powell conditions [105]. Simple experiments have shown that Wolfe conditions (with  $\alpha_0 = 1$ ) are superior to Armijo's rule.

---

<sup>1</sup> For the exact Newton's method,  $\mathbf{B}_k$  is the Hessian  $\mathbf{B}_k := \nabla^2 f_k = \mathbf{J}_k^T \mathbf{J}_k$ . For redundant configurations ( $m < n$ ), however, matrix  $\mathbf{J}_k^T \mathbf{J}_k$  is rank deficient. Adding a small diagonal matrix  $\mathbf{B}_k := \mathbf{J}_k^T \mathbf{J}_k + \delta \mathbf{E}$  facilitates convergence.

**Table 4.1:** Stopping criteria which can be combined via  $\sigma := \sigma_1 \wedge \sigma_2 \wedge \dots$ 

Criterion	Formulation
iteration limit exceeded	$\sigma_1 = k \geq k_{\max}$
convergence reached	$\sigma_2 = \ f(\mathbf{q}_k) - \mathbf{x}\  < \varepsilon_c$ for $\varepsilon_c > 0$
progress too small	$\sigma_3 = \ f(\mathbf{q}_{k+1}) - f(\mathbf{q}_k)\  < \varepsilon_p$ for $\varepsilon_p > 0$
gradient too small	$\sigma_4 = \ \nabla f_k\  < \varepsilon_g$ for $\varepsilon_g > 0$

Moreover, BADLER et al. [5] propose the use of different potential functions (called goals) whose gradients are combined and used to solve a nonlinear quadratic programming problem with inequality constraints. Goals are defined for position and/or orientation, aiming a vector towards a point as well as moving a point onto a line, plane or half-space. The method is used in their software package JACK for simulating human figures for ergonomic assessment [5]. The software is now sold by SIEMENS for assembly planning and validation [69].

In summary, for typical robotic systems with less than 50 DoFs and when singularities are not an issue, the Newton-Raphson based approach is the fastest method. Moreover,  $\mathbf{J}_k$  might be computed only each  $n^{\text{th}}$  iteration which gains efficiency. On the other hand, optimization based approaches are favorable for simulating human figures with very many DoFs. However, since run-time of iterative methods is non-deterministic, they are not directly applicable in real-time systems and are more importance for the simulation of human figures.

### 4.3 Differential Inverse Kinematics

Differentiating (4.1) leads to a *linear* equation for  $\dot{\mathbf{q}}$

$$\dot{\mathbf{x}} = \mathbf{J}_x \dot{\mathbf{q}}, \quad (4.6)$$

where  $\mathbf{J}_x := \partial f(\mathbf{q})/\partial \mathbf{q}$  is the Jacobian matrix. Since (4.6) is linear, it can be solved directly making it more suitable for real-time applications.

#### 4.3.1 Resolved Motion Rate Control and Redundancy Resolution

The well-established *resolved motion rate control* method, originally proposed by WHITNEY [154], solves a constrained quadratic programming problem

$$\begin{aligned} &\text{minimize} && \frac{1}{2} \dot{\mathbf{q}}^T \mathbf{W} \dot{\mathbf{q}} \\ &\text{subject to} && \dot{\mathbf{x}} - \mathbf{J}_x \dot{\mathbf{q}} = \mathbf{0}, \end{aligned} \quad (4.7)$$

**Algorithm 4.3:** Efficient evaluation of Equation (4.12), where  $W$  is positive definite.

---

**Input:**  $\dot{\mathbf{x}} \in \mathbb{R}^m, m \leq n$   
**Output:**  $\dot{\mathbf{q}} \in \mathbb{R}^n$   
 1:  $\mathbf{z} \leftarrow -\alpha_N W^{-1} (\nabla_{\mathbf{q}} H)^\top$   
 2:  $\mathbf{p} \leftarrow \dot{\mathbf{x}} - J_x \mathbf{z}$   
 3:  $\mathbf{B} \leftarrow J_x W^{-1} J_x^\top$   
 4: solve  $\mathbf{B} \boldsymbol{\lambda} = \mathbf{p}$  for  $\boldsymbol{\lambda}$   
 5:  $\dot{\mathbf{q}} \leftarrow W^{-1} J_x^\top \boldsymbol{\lambda} + \mathbf{z}$

---

which calculates (locally) minimal joint velocities  $\dot{\mathbf{q}}$  required to track  $\dot{\mathbf{x}}$ . Individual joint speeds are weighted by the (usually diagonal) positive definite matrix  $W$ . Especially for robots consisting of different joint types, such as mixed linear and rotary joints,  $W$  is used to account for the different physical meaning of the joint variables (see [11] for an example). Equation (4.7) is solved for  $\dot{\mathbf{q}}$  using the method of Lagrange multipliers:

$$\frac{\partial L}{\partial \dot{\mathbf{q}}} = \frac{\partial L}{\partial \boldsymbol{\lambda}} = \mathbf{0} \quad \text{with} \quad L := \frac{1}{2} \dot{\mathbf{q}}^\top W \dot{\mathbf{q}} + \boldsymbol{\lambda}^\top (\dot{\mathbf{x}} - J_x \dot{\mathbf{q}}), \quad (4.8)$$

$$\Rightarrow \dot{\mathbf{q}} = J_{x,W}^\# \dot{\mathbf{x}}, \quad (4.9)$$

where  $J_{x,W}^\#$  is the  $W$ -weighted generalized inverse, i. e.,  $J_{x,W}^\# := W^{-1} J_x^\top (J_x W^{-1} J_x^\top)^{-1}$ .

For redundant configurations, LIÉGEOIS proposed a modification for (4.7) to minimize an additional cost function  $H(\mathbf{q})$  [81]. The change in  $H$  during one time step  $\Delta t$  is:

$$\Delta H = \nabla_{\mathbf{q}} H(\mathbf{q}) \dot{\mathbf{q}} dt \approx \nabla_{\mathbf{q}} H(\mathbf{q}) \dot{\mathbf{q}} \Delta t, \quad (4.10)$$

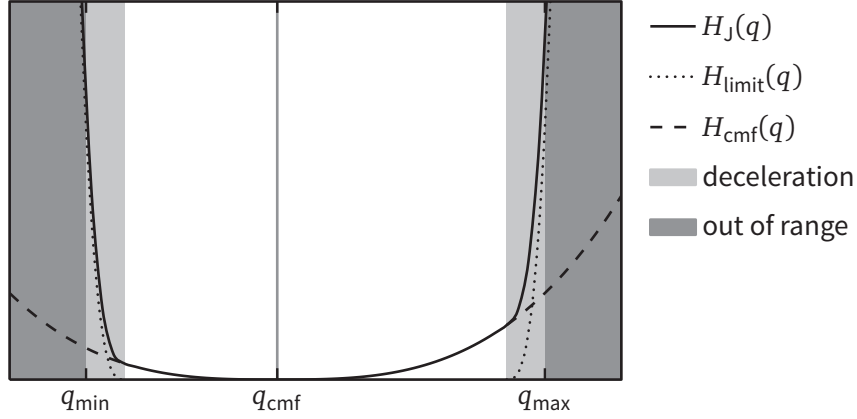
which is used to get an augmented optimization problem:

$$\begin{aligned} & \text{minimize} \quad \frac{1}{2} \dot{\mathbf{q}}^\top W \dot{\mathbf{q}} + \beta \Delta t (\nabla_{\mathbf{q}} H(\mathbf{q}))^\top \dot{\mathbf{q}} \\ & \text{subject to} \quad \dot{\mathbf{x}} - J_x \dot{\mathbf{q}} = \mathbf{0}. \end{aligned} \quad (4.11)$$

The scalar parameter  $\beta > 0$  determines the balance of priorities between both terms. (4.11) is solved for  $\dot{\mathbf{q}}$  using the method of Lagrange multipliers which gives:

$$\dot{\mathbf{q}} = J_{x,W}^\# \dot{\mathbf{x}} - \alpha_N N W^{-1} (\nabla_{\mathbf{q}} H(\mathbf{q}))^\top. \quad (4.12)$$

Here,  $\mathbf{N} := \mathbf{E} - J_{x,W}^\# J_x$  is the null-space projection matrix and  $\alpha_N := \beta \Delta t$ . As originally proposed by KLEIN and HUANG (see [97]), it is significantly more efficient to compute  $\dot{\mathbf{q}}$  from the optimization problem and thereby, avoiding the calculation of  $J_{x,W}^\#$  and  $\mathbf{N}$  (see [28, Appendix F] for an adaptation to the  $W$ -weighted Jacobian from (4.12)). The method is outlined in Algorithm 4.3.



**Figure 4.2:** Schematic representation of the cost function  $H_J = H_{\text{limit}} + H_{\text{cmf}}$  (for a single joint) used for Lola.  $H_{\text{limit}}$  and  $H_{\text{cmf}}$  are the terms for joint limit avoidance and convergence towards a preferred joint angle  $q_{\text{cmf}}$  respectively (adopted from [28]).

In practical implementations, cost function  $H$  is used for joint limit avoidance [40, 81], pose convergence [28], manipulability optimization [37, 158], momentum minimization [126], collision avoidance [125, 133, 134] and obstacle avoidance [77, 129]. Figure 4.2 illustrates the cost functions for joint limit avoidance and pose convergence used for Lola.

### Numerical Integration and Task Space Tracking

If generalized positions  $\mathbf{q}$  are required, velocities  $\dot{\mathbf{q}}$  can be integrated with respect to time. Because of the digital implementation of robot control systems only discrete time-samples are available, i. e.,  $\dot{\mathbf{q}}_i = \dot{\mathbf{q}}(t_i)$  and hence, numerical integration is used. This, however, leads to an unbounded error in the resulting task trajectories:

$$\Delta \mathbf{x}_i = \mathbf{x}_i - \mathbf{f}(\mathbf{q}_i) \quad \text{and} \quad \Delta \dot{\mathbf{x}}_i = \dot{\mathbf{x}}_i - \mathbf{J}_x \dot{\mathbf{q}}_i, \quad (4.13)$$

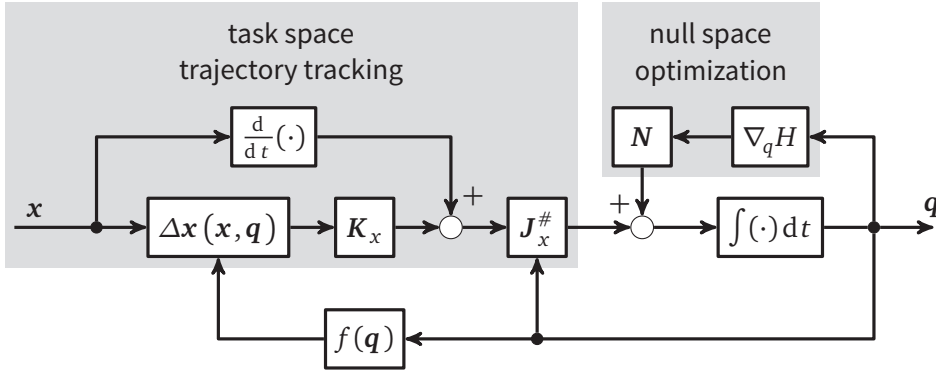
where  $\Delta \mathbf{x}_i$  is an expression of such task space tracking errors at time  $t_i$  and  $\Delta \dot{\mathbf{x}}$  denotes the according error dynamics. Therefore, an adequate feedback term added to the desired task trajectories ensures asymptotic tracking behavior:

$$\dot{\mathbf{q}}_i = \mathbf{J}_x^\# \dot{\hat{\mathbf{x}}}_i, \quad (4.14a)$$

$$\dot{\hat{\mathbf{x}}}_i := \dot{\mathbf{x}}_i + \mathbf{K}_x \Delta \mathbf{x}_i(\mathbf{x}_i, \mathbf{q}_i). \quad (4.14b)$$

Here, (4.14a) is representative for a differential inverse kinematics solution, i. e., (4.9) or (4.12). In (4.14b) task space velocities  $\dot{\mathbf{x}}$  are modified by feedback term  $\mathbf{K}_x \Delta \mathbf{x}$  which leads to the equivalent linear error dynamics [128]:

$$\Delta \dot{\mathbf{x}}_i + \mathbf{K}_x \Delta \mathbf{x}_i = \mathbf{0}, \quad (4.15)$$



**Figure 4.3:** First order differential inverse kinematics with task space trajectory tracking and null-space optimization.

where  $\mathbf{K}_x$  is a positive definite matrix which ensures asymptotic stability. Due to the finite step size in discrete time domain, however, an upper bound of the gains in  $\mathbf{K}_x$  exists. For a task defining positions the error simply calculates to  $\Delta \mathbf{x}_{\text{pos}} = \mathbf{x}_{\text{pos}} - \mathbf{f}_{\text{pos}}(\mathbf{q})$ . On the other hand, orientation errors  $\Delta \mathbf{x}_{\text{rot}}$  are not gained straightforwardly. See Appendix C for a discussion.

Finally, assuming sufficiently short time steps  $\Delta t$ , a simple recursive explicit Euler integration scheme is typically used to calculate joint positions  $\mathbf{q}_{i+1}$  for the next time instant  $t_{i+1}$ :

$$\mathbf{q}_{i+1} = \mathbf{q}_i + \dot{\mathbf{q}}_i \Delta t . \quad (4.16)$$

FALCO and NATALE study the stability of the closed-loop inverse kinematics algorithm for redundant robots in discrete time domain (presented above). Sufficient conditions for the stability and an estimation of the region of convergence are provided [49].

Figure 4.3 illustrates the resulting inverse kinematics algorithm using Eqs. (4.12)–(4.16) with task space trajectory tracking and null-space optimization. This scheme is also used for Lola.

As an alternative to the feedback-based integration, CHENG and GUPTA [34] propose the use of an Adams-Moulton predictor–corrector scheme for performing joint velocity integration leading to a fourth-order trajectory following in joint space.

### 4.3.2 Jacobian Transpose

Instead of using the computationally expensive pseudoinverse  $\mathbf{J}^\#$  in (4.9) or (4.12) the simple Jacobian transpose is used in this approach leading to

$$\dot{\mathbf{q}} = \mathbf{J}_x^\top \mathbf{K}_x \Delta \mathbf{x} . \quad (4.17)$$

The justification for this approach is led by a Lyapunov stability analysis of the error dynamics leading to its asymptotic convergence under certain conditions. Beside the advantage of computational simplicity the method also does not suffer from local singularities. However, this comes at the cost of bad tracking behavior (see SICILIANO et al. [128]).

### 4.3.3 Resolved Acceleration Rate Control

Straightforward differentiation of (4.6) leads to the second-order differential kinematics:

$$\ddot{\mathbf{x}} = \mathbf{J}_x(\mathbf{q})\ddot{\mathbf{q}} + \dot{\mathbf{J}}_x(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}. \quad (4.18)$$

Analogously to the optimization based derivation of the resolved motion rate control method (see (4.6) to (4.12)), (4.18) is used to derive the *resolved acceleration rate control* method, originally proposed by LUH et al. [89], in an optimization problem similar to (4.11) and solved for  $\ddot{\mathbf{q}}$ :

$$\ddot{\mathbf{q}} = \mathbf{J}_{W,x}^\# \left( \ddot{\mathbf{x}} - \dot{\mathbf{J}}_x \dot{\mathbf{q}} \right) - \alpha_N \mathbf{N} \mathbf{W}^{-1} \left( \nabla_{\dot{\mathbf{q}}} H(\mathbf{q}, \dot{\mathbf{q}}) \right)^\top. \quad (4.19)$$

Furthermore, by adding feedback terms on position and velocity level we get

$$\ddot{\mathbf{q}} = \mathbf{J}_{W,x}^\# \left( \ddot{\mathbf{x}} + \mathbf{K}_{\dot{\mathbf{x}}} \Delta \dot{\mathbf{x}} + \mathbf{K}_x \Delta \mathbf{x} - \dot{\mathbf{J}}_x \dot{\mathbf{q}} \right) - \alpha_N \mathbf{N} \mathbf{W}^{-1} \left( \nabla_{\dot{\mathbf{q}}} H(\mathbf{q}, \dot{\mathbf{q}}) \right)^\top, \quad (4.20)$$

which can be integrated using

$$\dot{\mathbf{q}}_{i+1} = \dot{\mathbf{q}}_i + \ddot{\mathbf{q}}_i \Delta t \quad \text{and} \quad \mathbf{q}_{i+1} = \mathbf{q}_i + \frac{1}{2} (\dot{\mathbf{q}}_i + \dot{\mathbf{q}}_{i+1}) \Delta t. \quad (4.21)$$

Note that the trapezoidal rule is used to integrate  $\mathbf{q}_{i+1}$  more accurately.

Joint space accelerations  $\ddot{\mathbf{q}}$  obtained from (4.20) are often used in conjunction with torque controlled robots, where desired joint torques are computed from EoM [97].

BUSCHMANN et al. [29] propose a method for resolving a hybrid task description consisting of generalized forces and task-space trajectories into  $\ddot{\mathbf{q}}$ . The authors apply this *inverse kineto-dynamics* approach to walking control, since an exact tracking of desired contact forces is enabled in real-time [29].

### 4.3.4 Hierarchical Approaches

In the presence of many simultaneous tasks, less significant tasks might exist whose execution has a lower priority than others. NAKAMURA et al. present in [100] a method for resolving tasks with ordered priority. The method works by minimizing a secondary task in the null-space of the primary task. Analogously to solution (4.12) an

additional cost function  $H$  can be minimized, which acts in the null-space of both tasks simultaneously. The method has been extended to an arbitrary number of priority levels in a recursive manner [7, 127]. Thereby, the QR-decomposition is exploited to gain efficiency. More recently, ESCANDE et al. further refined the method towards faster computation [48]. It is worth noting, however, that prioritizing control objectives will often make lower tasks infeasible. Methods exist to reduce the effect of these *algorithmic singularities* [75].

In addition to the equality constraints, e. g., in (4.7), inequality constraints have been considered in optimization based IK [33, 75]. Recently, the hierarchical approach is used to resolve linear, convex inequality constraints in a local manner [48, 75] (as opposed to global optimization of trajectories in planning stage [99, 146]). Inequality constraints are subject to collision avoidance [131], stability and local motions [75].

Based on these hierarchical methods MANSARD et al. present a graph based software framework [92]. It allows a dynamic insertion and removal of predefined tasks into a “stack of tasks” and provides a scripting interface. The framework is used to control the humanoid robot HRP-2.

## 4.4 Singularities and Manipulability

In a robotics context, singularities represent reduced movement abilities mainly due to extreme configurations which in turn disables task space tracking. Moreover, task space representation singularities, e. g., inherent in some orientation definitions, exist. Furthermore, in the neighborhood of a singularity, small task space velocities may cause large velocities in joint space.

Since singularities are characterized by a rank deficient Jacobian  $\mathbf{J}_x$ , YOSHIKAWA proposed in [158] a continuous *measure of manipulability*

$$w(\mathbf{q}) := \sqrt{\det(\mathbf{J}_x \mathbf{J}_x^T)} = \prod_i^m \sigma_i, \quad (4.22)$$

which equals to the product of the singular values  $\sigma_i$  of  $\mathbf{J}_x$ . Therefore, if  $\mathbf{J}_x$  is rank deficient, at least one  $\sigma_i = 0$  and hence,  $w \rightarrow 0$  when approaching singularities. On the other hand, the condition number  $c = \sigma_{\max}/\sigma_{\min}$  would be a good criterion too. However, the  $\sqrt{\det(\cdot)}$ -form is computationally more efficient.

For redundant robots, PARK et al. propose in [113] the inclusion of the measure of manipulability  $w$  into cost function  $H(\mathbf{q})$  (cf. (4.12)) to maximize  $w$ . CHIU presents a similar method to maximize  $w$  along a predefined direction in task space [37].

By adopting the *damped least squares solution* to (4.9) robustness in the neighborhood of singularities is obtained. This approach, independently proposed by NAKAMURA et al. [98] and WAMPLER [153], minimizes the sum  $\|\dot{\mathbf{x}} - \mathbf{J}_x \dot{\mathbf{q}}\|^2 + \lambda^2 \|\dot{\mathbf{q}}\|^2$ . Therefore, the tracking error is weighted against the joint velocity norm using *damping factor*  $\lambda$ .

**Table 4.2:** Components of Lola’s task-space coordinates (adopted from: [28]).

Component	Description
$T\mathbf{r}_{\text{CoG-RF}}$	Right foot TCP relative to CoG in torso planning frame.
$T\mathbf{r}_{\text{CoG-LF}}$	Left foot TCP relative to CoG in torso planning frame.
$T\mathbf{s}_{\text{TRF}}$	Orientation of right foot relative to upper body.
$T\mathbf{s}_{\text{TLF}}$	Orientation of left foot relative to upper body.
$I\mathbf{r}_{\text{CoG}}$	Robot CoG in inertial planning frame.
$\varphi_T$	Upper Body orientation relative to inertial planning frame.
$\mathbf{r}_{\text{CoG,RACoG}}$	Right arm CoG relative to robot CoG in torso planning frame.
$\mathbf{r}_{\text{CoG,LACoG}}$	Left arm CoG relative to robot CoG in torso planning frame.
$q_{J,\text{PR}}$	Pelvis rotation.
$q_{J,\text{PA}}$	Pelvis adduction.
$q_{J,\text{RTF}}$	Right toe flexion.
$q_{J,\text{LTF}}$	Left toe flexion.

The damped least squares solution of (4.9) is  $\dot{\mathbf{q}}^{(\lambda)} = \mathbf{J}_x^\top (\mathbf{J}_x \mathbf{J}_x^\top + \lambda^2 \mathbf{E})^{-1} \dot{\mathbf{x}}$ . However, finding an optimal  $\lambda$ , which provides a good trade-off between robustness against singularities and tracking performance, is difficult [42]. Moreover, the null-space in (4.12) is no longer orthogonal to the primary task for redundant robots [75]. CHIAVERINI et al. propose the *weighted* damped least squares solution to achieve user-defined task-space accuracy in the neighborhood of singular configurations [35, 36]. MACIEJEWSKI et al. study in [91] the *singular value decomposition* (SVD) to compute an optimal  $\lambda$  for the damped least squares solution. Furthermore, the *truncated SVD* (where rows and columns, associated to critical singular values, i. e.,  $\sigma_i < \varepsilon$ , are removed from the decomposed matrices) is studied [90].

## 4.5 Task Description of Lola

For further reference in this thesis, the task description  $\mathbf{x}(\mathbf{q})$  currently used for Lola is briefly described. It includes *feet position and orientations, robot CoG position, upper body orientation, arm CoG positions and waist joint positions* (see [28] for a detailed description). If all tasks are defined active, we have a non-redundant system, i. e.,  $\dim(\mathbf{x}) = \dim(\mathbf{q})$ . Obviously, tasks can be removed from  $\mathbf{x}$  to obtain a redundant configuration. Redundancy is exploited for the methods described in Ch. 5 and 7.

## 4.6 Chapter Summary

This chapter reviewed the nonlinear problem of IK and gave a brief overview to some important solutions for real-time control. Position based approaches solve the



nonlinear problem in an iterative manner and are therefore not applicable to real-time control. On the other hand, differential kinematics and local optimization give rise to most real-time capable IK methods. Moreover, some important schemes for resolving kinematic redundancies were presented. Furthermore, the issue of singularities was addressed and some popular techniques were shown to enable singularity robustness.

Global optimization based methods were not considered, since they are subject to offline planning. This, however, might change in the future when more powerful computer systems become available enabling the integration of IK into real-time planning.



## 5 Self-Collision Avoidance

This chapter describes the self-collision avoidance scheme developed for this thesis [125]. Self-collisions occur when two segments of a robot come into contact. Therefore, the main objective is to avoid damage to the expensive robot hardware due to colliding motions. Moreover, computational efficiency is required to enable real-time execution.

The simplest approach for collision avoidance is to stop the motion when a collision is predicted. For a walking robot, however, this is not a feasible approach since it would destabilize and potentially damage the robot. Therefore, a more sophisticated approach must be chosen to maintain balance.

### 5.1 Background and Related Work

Several strategies to avoid collisions for biped robots have been proposed, e. g. [61, 73, 78, 107, 131, 133, 146]. To speed up calculations, the robot geometry as well as the collision pairs are typically reduced.

KUFFNER et al. propose a safe on-line walking pattern generation method which uses fast distance calculation of the links of the robot H7 modeled as convex protective hulls. Calculating three steps in advance, the method executes an emergency stopping sequence if a new walking pattern would cause self-collision [78].

OKADA et al. experimentally evaluate the performance of different interference detection libraries for a real-time self-collision detection applied to the robot HRP-2 [107]. They use as many collision pairs as possible with detailed geometric models of the segments. The authors suggest the use of *axis aligned bounding box*-based methods which are faster than conventional oriented bounding box-based methods.

In recent years many interesting approaches to reaching and grasping objects have been proposed and applied to the robot Asimo [61, 133, 146]. SUGIURA et al. simplify the geometry of the robot uses *swept-sphere-volumes* [79] enabling an efficient distance calculation between robot links. The authors use the method to actively avoid collisions on-line for reaching motions without stopping the robot. This is achieved by blending the solution for the reaching motion to a collision-free pose when two segments are too close [133]. The same collision avoidance scheme is used for offline optimization of a sequence of attractors to generate collision-free trajectories of the arms for reaching objects [61, 146].

Recently, KANEHIRO et al. proposed a fast offline walking pattern generation method which also takes self-collisions into account to generate feasible and collision-free walking trajectories [73].

The self-collision avoidance developed in this work, integrates to the resolved motion rate control method. By recalling the method presented in (4.12) for a redundant robot we have:

$$\dot{\mathbf{q}} = \mathbf{J}_{x,W}^\# \dot{\mathbf{x}} - \alpha_N \mathbf{N} \mathbf{W}^{-1} \left( \nabla_{\mathbf{q}} H(\mathbf{q}) \right)^\top,$$

where joint velocities  $\dot{\mathbf{q}}$  are obtained from desired task space velocities  $\dot{\mathbf{x}}$ . In the second term  $\nabla_{\mathbf{q}} H(\mathbf{q})$  minimizes a cost function  $H(\mathbf{q})$ . Because of the projection via  $\mathbf{N}$  (the null-space matrix of  $\mathbf{J}_x$ ) into joint space, it does not interfere with the main task  $\mathbf{x}$ .

Consequently, the robot must be kinematically redundant, i. e.,  $n > m$ . For Lola this is the case for the 7-DoF legs and 2-DoF waist. However, kinematic redundancy can also be achieved by removing certain tasks from  $\mathbf{x}$  which also reduces  $m$ . Note that it is possible to choose reference trajectories  $\mathbf{x}(t)$ , such that self-collisions cannot be avoided. Special care is necessary when a dynamic modification of task  $\mathbf{x}$  is desired during motion. When changing the dimension or definition of  $\mathbf{x}$ , the Jacobian  $\mathbf{J}_x$  also changes, possibly resulting in velocity jumps on the joint level. To overcome this problem GIENGER et al. propose to limit the jerk for null-space motion rates by scaling down the null-space velocity vector accordingly [62].

## 5.2 Self-Collision Avoidance

The cost function  $H_c$  and its gradient used for collision avoidance are derived in this section. Formally, the robot  $\mathcal{R}$  is modeled as a set of  $n_S$  segments  $\mathcal{S}_i$

$$\mathcal{R} := \{\mathcal{S}_i : 0 \leq i < n_S\}. \quad (5.1)$$

The collision environment  $\mathcal{C}$  is defined as a set of  $n_C$  collision pairs  $\mathcal{P}_k$ , which in turn consist of two different segments of the robot:

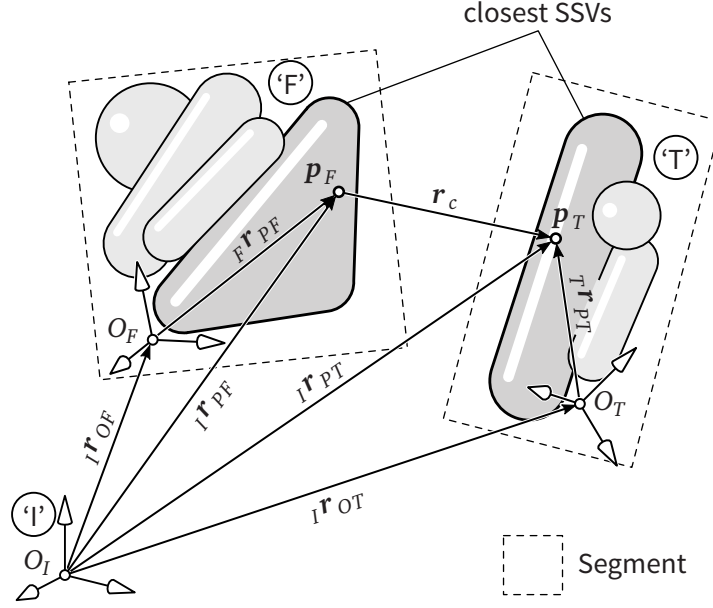
$$\mathcal{C} := \{\mathcal{P}_k : 0 \leq k < n_C\}, \quad (5.2)$$

$$\mathcal{P}_k := \{\mathcal{S}_F, \mathcal{S}_T : (\mathcal{S}_F \wedge \mathcal{S}_T \in \mathcal{R}) \wedge \mathcal{S}_F \neq \mathcal{S}_T\}. \quad (5.3)$$

Figure 5.1 illustrates the distance calculation between a collision pair  $\mathcal{P}_k$ . An adequate method is used to find the closest points  $\mathbf{p}_F$  and  $\mathbf{p}_T$ . We use the concept of *Swept Sphere Volumes* presented in Chapter 6. The connecting vector  $\mathbf{r}_c$  is defined from segment ‘F’ to segment ‘T’ and its length defines the separation distance

$$d = \sqrt{\mathbf{r}_c^\top \mathbf{r}_c} \quad \text{with} \quad \mathbf{r}_c := \mathbf{p}_T(\mathbf{q}) - \mathbf{p}_F(\mathbf{q}). \quad (5.4)$$

By using the chain rule of differentiation, the gradient of the collision cost function



**Figure 5.1:** A collision pair with the definition of the shortest distance vector  $\mathbf{r}_c$  from segment 'F' to segment 'T' (from: [125]).

$H_C$  for a single pair  $\mathcal{P}_k$  (with distance  $d_k$  and omitting  $k$ ) may be written as

$$\frac{\partial H_C(d)}{\partial \mathbf{q}} = \frac{\partial H_C(d)}{\partial d} \frac{\partial d}{\partial \mathbf{q}}. \quad (5.5)$$

The two partial derivatives on the right hand side are derived separately. The scalar function  $\nabla_d H_C$  is derived further below. The second part is derived using (5.4) to

$$\frac{\partial d}{\partial \mathbf{q}} = \frac{1}{d} (\mathbf{J}_{T,T} - \mathbf{J}_{T,F})^\top (\mathbf{p}_T - \mathbf{p}_F), \quad (5.6)$$

where  $\mathbf{J}_{T,TF}$  denotes the translational Jacobian of the closest points. In general the translational Jacobian for a point  $\mathbf{p}$  is given by

$$\mathbf{J}_T^p := \frac{\partial \mathbf{p}}{\partial \mathbf{q}} = \mathbf{J}_T^o + \mathbf{J}_R \times \mathbf{r}_{op}, \quad (5.7)$$

where  $\mathbf{J}_T^o$  and  $\mathbf{J}_R$  are the translational and rotational Jacobians of the segments, respectively. In our implementation the Jacobians are calculated analytically in the respective body-fixed frames 'F' and 'T'. Therefore, it is computationally more efficient to transform only  $\mathbf{r}_c$  and (5.6) becomes

$$\frac{\partial d}{\partial \mathbf{q}} = \frac{1}{d} (\mathbf{J}_{T,T}^\top \mathbf{r}_c - \mathbf{J}_{T,F}^\top \mathbf{r}_c). \quad (5.8)$$

Finally, for the whole robot gradient  $\nabla_{\mathbf{q}} H_C$  is simply the sum of the contributions of each pair  $\mathcal{P}_k$ :

$$\frac{\partial H_C}{\partial \mathbf{q}} = \sum_k^{n_c} \frac{\partial H_C(d_k)}{\partial \mathbf{q}}, \quad (5.9)$$

which is then added to  $\nabla_{\mathbf{q}} H$  in (4.12).

### Developing the Collision Cost Function

By studying (5.5) it is a good idea to directly design the scalar gradient function  $\nabla_d H_C$ , since  $H_C$  is of no particular interest in the final result. Furthermore, local convergence can be ensured by design.

Collision cost function  $H_C$  can be seen as a *repulsive potential* function to avoid collisions. Therefore, in order to maximize the distance between two segments, a continuously decreasing, positive gradient is required. Furthermore, since the term ‘‘avoidance’’ implies a predictive character, we require an activation of collision avoidance already at a distance  $d \leq d_a$  for  $d_a > 0$ . The gradient function is zero for  $d \geq d_a$  and hence, only active below the activation distance  $d_a$ .

A general approach is, therefore, to have a piece-wise function

$$\frac{\partial H_C}{\partial d} = \begin{cases} g(d) & \text{if } d < d_a \\ 0 & \text{else,} \end{cases} \quad (5.10)$$

where  $g(d)$  is a positive, continuously decreasing function with  $g(d_a) \stackrel{!}{=} 0$ . The simplest approach, shown in Figure 5.2, is to use a linear function

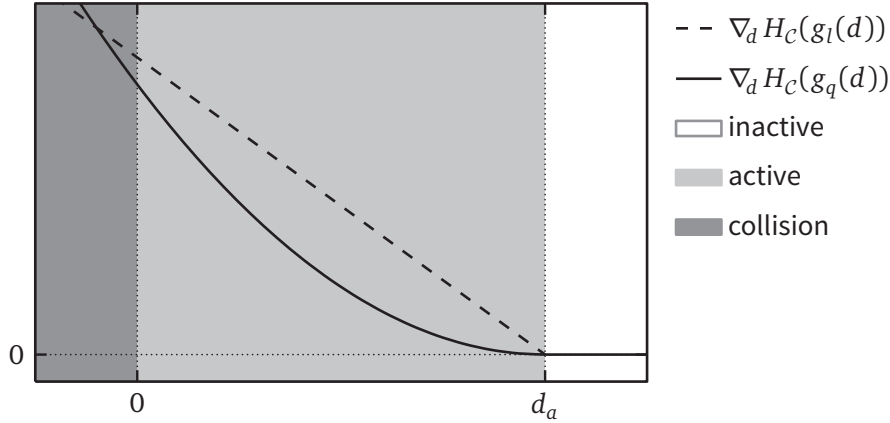
$$g(d) = g_l(d) := m(d - d_a), \quad (5.11)$$

where  $m < 0$  defines the gradient’s slope. In order to handle large velocities,  $m$  must have a sufficiently large negative value. The joint speed integration in (4.16), however, possibly leads to oscillations in the neighborhood of the kink at  $d_a$ .

As previously proposed by the author in [125] and others, e. g., [146], a higher order function is used to get a  $C^n$ , continuity ( $n > 0$ ) at  $d_a$ . For a quadratic function, illustrated in Figure 5.2, we get:

$$g(d) = g_q(d) := s(d - d_a)^2, \quad (5.12)$$

where  $s$  is a scaling factor, which is (along with  $d_a$ ) subject to tuning. Further experiments involving higher velocities, however, revealed that this approach leads to oscillations in joint space due to the simple Euler integration (cf. (4.16)). In order to avoid collisions in the presence of high joint velocities either  $s$  or  $d_a$  have to be chosen



**Figure 5.2:** Collision avoidance cost function gradient candidates: linear function  $g_l$  and quadratic  $g_q$  function.

large enough which in turn leads to degraded behavior with slower motion. On the other hand, by choosing higher order polynomials in (5.12) the situation gets even worse.<sup>1</sup>

Therefore, the use of a piece-wise function is proposed which mixes both gradients from (5.11) and (5.12). A parameter  $t \in [0, 1[$  defines the blending position, such that  $g_m(d, t = 0) = g_q(d)$  and  $g_m(d, t = 1) = g_l(d)$ . Rewriting (5.11) to

$$g_{l,2}(d) := m d + s_0, \quad (5.13)$$

and letting

$$g_{l,2} \Big|_{d=t d_a} = g_q \Big|_{d=t d_a} \quad \text{and} \quad \frac{\partial g_{l,2}}{\partial d} \Big|_{d=t d_a} = \frac{\partial g_q}{\partial d} \Big|_{d=t d_a}, \quad (5.14)$$

to get a  $C^1$ -continuous junction at  $d = t d_a$  gives

$$g(d) = g_m(d) := \begin{cases} -\frac{2s_0}{d_a(t+1)} d + s_0 & \text{if } d < t d_a \\ -\frac{s_0(d_a - d)^2}{d_a^2(t^2 - 1)} & \text{else.} \end{cases} \quad (5.15)$$

The resulting gradient  $\nabla_d H_C(d, t)$  is illustrated in Figure 5.3 for different values of  $t$ . We use  $t = [0.4 \dots 0.6]$  which is a good compromise. Parameter  $s_0$  defines the intersection with the  $y$ -axis. It is chosen sufficiently large to prevent self-collision in the presence of other null-space motions.

<sup>1</sup> This is due to the simple Euler integration with fixed time step.

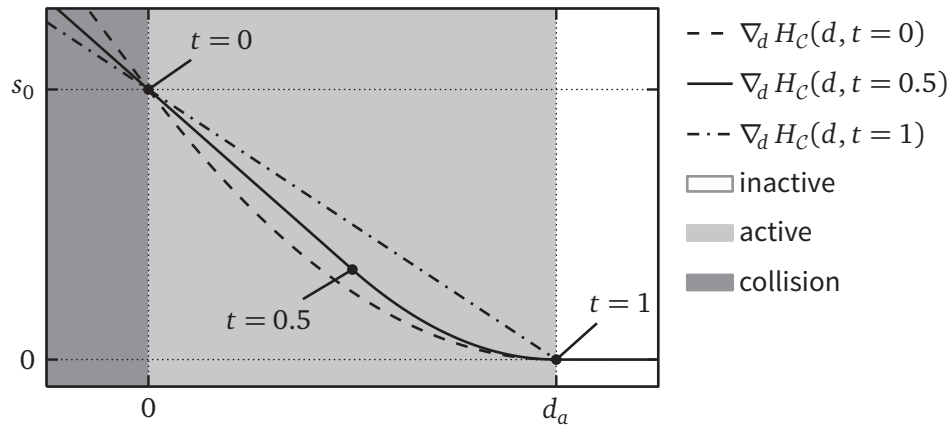
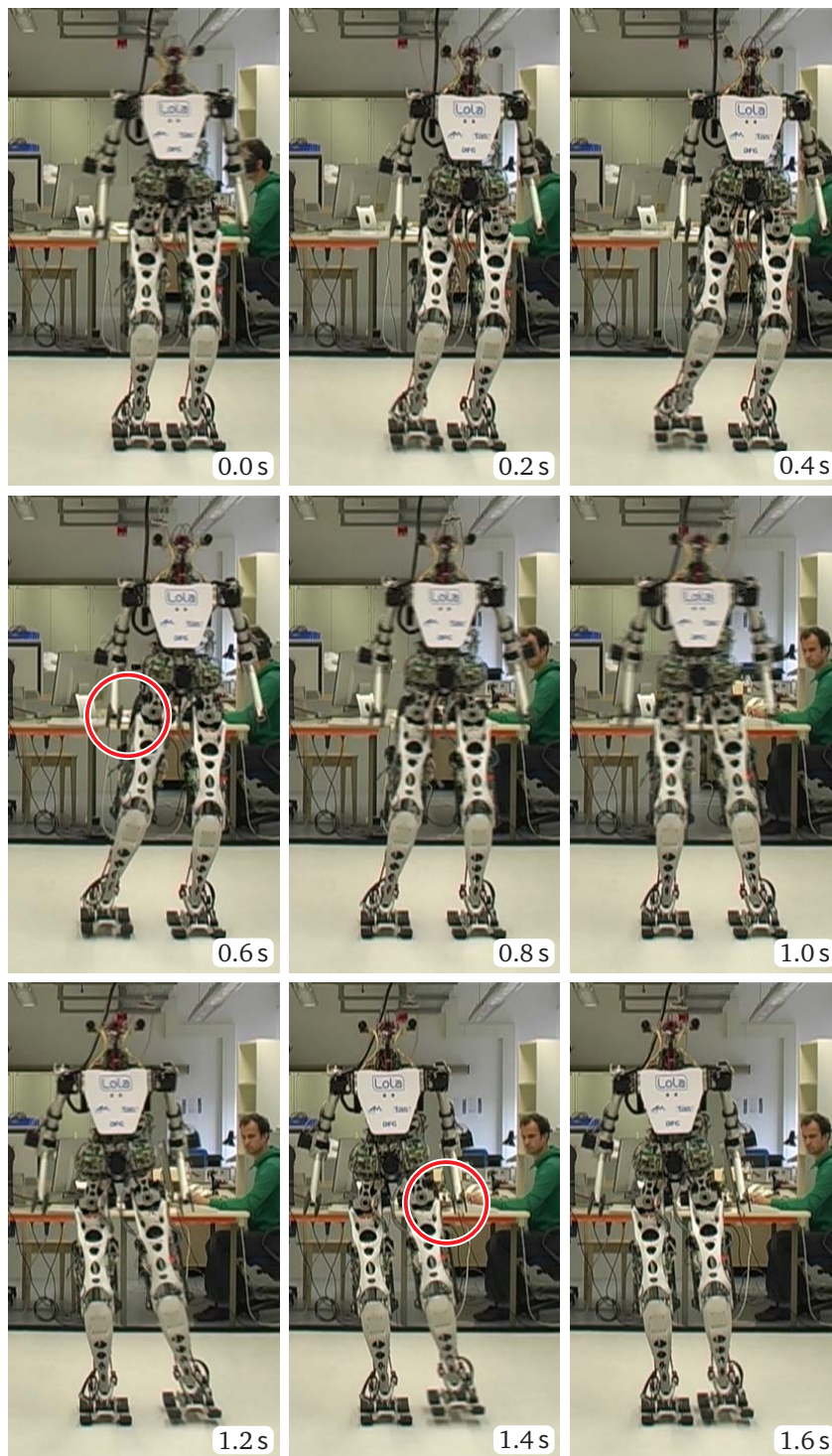


Figure 5.3: Schematic representation of the potential function  $H_C(d)$  for collision avoidance.

### 5.3 Chapter Summary

In this chapter a model-based approach for self-collision avoidance for redundant robots is presented. Figure 5.4 shows a video frame sequence of Lola while walking sideways. Thereby, the approach presented in this chapter avoids collisions between arms and upper legs. More experimental results are shown in the next chapters. The method works by minimizing a repulsive collision cost function in the null-space of the inverse kinematics. The cost function uses the closest points on two segments and applies a distance-based repulsive law to keep the segments separated. Different strategies for designing an adequate repulsive law of the cost function are discussed. The overall method proved to work reliably and effective in simulation and experiments for different redundant robots, including Lola (cf. [11, 125]).





**Figure 5.4:** Video frame sequence when walking sideways: the collision avoidance keeps the arms separated from the upper legs. The effect is especially visible at times 0.6 s and 1.4 s (see encircled regions).



## 6 Real-Time Distance Computation using Swept-Sphere-Volumes

To facilitate collision avoidance in the inverse kinematics algorithm, presented in Chapter 5, the closest points on the surface of two robot-segments must be calculated in real-time. This chapter provides a suitable method to accomplish this task.

### 6.1 Background and Related Work

Many applications, such as robotics, multibody contact dynamics, computer graphics, virtual reality, CAD/CAM, and others, require the knowledge of the minimum distance between two geometric objects. Most applications need a *fast* and *reliable* distance computation. In order to handle collisions in robot control, distances must be computed (among other things) within one control cycle, and hence, run faster than real-time.

A topic closely related to distance computation is the *detection* of collisions. Many algorithms and software libraries have been developed in this field. The GAMMA-group<sup>1</sup> at the University of North Carolina have designed and implemented many collision detection and proximity query<sup>2</sup> software packages. The libraries have been applied to large-scale interactive environments and simulations [132]. For a good review on available approaches, applicable for several geometric representations of objects, see [82, 83]. Overall, the *geometric representation* used has a strong influence on the runtime and accuracy of the applied method. In [32] the minimum distance computation between *implicit algebraic surfaces*, which is the basis for many CAD applications, is presented. *Polygons* are dominant in computer graphics and modeling, since they are versatile, have a simple representation and hardware-accelerated rendering is widely available [28]. Polygonal objects are either obtained by direct modeling or by tessellation of other representations. On the other hand, the *convexity* of shapes plays an important role. Some collision detection algorithms rely on the distance computation of convex, polygonal objects [70]. Hence, they are also applicable for distance calculation for that class of objects.

Simplex based algorithms calculate the distance between two convex polytopes by iteratively finding closer simplices. The Gilbert-Johnson-Keerthi (GJK) algorithm is one of the most widely used among them [63]. A “Revised GJK” (RGJK) algorithm

---

<sup>1</sup> Geometric Algorithms for Modeling, Motion, and Animation (GAMMA) website: <http://gamma.cs.unc.edu/research/collision/>

<sup>2</sup> Distance computation between two objects is referred as “proximity query” by some authors.

exists [109] which gains performance by exploiting the *adjacency* of the vertices of the polytopes.<sup>3</sup> The adjacency mapping can be generated in a pre-processing step and reused efficiently.

VORONOI-CLIP, or V-CLIP, is a more recent algorithm which tracks the closest pair of features between convex polyhedra to facilitate collision detection [94]. Once identified, these closest features can be used in minimal distance calculation between both objects.<sup>4</sup> When applied solely to collision detection, V-Clip can be computationally more efficient than the original GJK algorithm [94]. On the other hand, RGJK is faster than V-Clip, as studied in [109].

Many algorithms exploit the *temporal coherence* of the closest points on convex shapes. Based on a former solution, the location of the closest point is updated only locally. This approach works efficiently if the difference in relative object orientation within subsequent distance calculations is small.

OKADA et al. evaluate in [107] several publicly available collision detection libraries on the robot HRP-2 and study their run-time performance in a manipulation scenario.

Popular methods to reduce computational cost in collision detection and distance computation are *space partitioning* algorithms, *bounding volume hierarchies* and *sweep and prune* approaches. A common idea of these algorithms is to sort out potentially closest component-pairs which are then processed. This corresponds to a so called “broad-phase” algorithm combined with a “narrow-phase” distance (or collision) computation.

*Space partitioning algorithms*, such as kd-tree [16], Binary Space Partitioning Trees (BSP trees) [144], Octrees [93], etc., work by partitioning the space or complex geometry in a pre-processing step and later avoid most complex distance computations between the components of the shape. Hence, these methods are mostly applied in conjunction with direct distance computation algorithms. Similarly, *bounding volume hierarchies* (BVH) [157] are another elegant way to reduce computational cost. A hierarchy of bounding volumes with decreasing size contains gradually smaller portions of the geometry. A simple search routine finds closest bounding volumes on each hierarchy level. Consequently, the lowest level contains only smaller components of the full geometry. The final distance is then computed only between pairs of convex components. Efficiency is improved by: (a) the fast distance computation of the bounding volume primitives used, e. g., spheres, and, (b) by ignoring most pairs of components in distance computation in the narrow-phase [117]. The *sweep and prune* algorithm limits the number of collision checks by first testing overlapping bounding volumes projected into sub-spaces, i. e., individual axes or planes. Furthermore, temporal coherence of relatively slowly moving rigid bodies is exploited [39].

The Computational Geometry Algorithms Library (CGAL) [143] is a versatile library

3 RGJK software implementations are publicly available from:

Fortran language: <http://serve.me.nus.edu.sg/mpeongcj/?id=downloads>

C language: <http://www.cs.ox.ac.uk/people/stephen.cameron/distances/>

4 V-Clip software implementation in C language: <http://www.merl.com/areas/vclip/>

for processing geometry which also contains components for distance computation of  $n$ -dimensional convex polytopes in  $\mathbb{R}^n$  [76]. MUJA et al. [96] present a software library to approximately find the nearest neighbors of components called “Fast Library for Approximate Nearest Neighbors” (FLANN).<sup>5</sup> Although, originally developed for 2-D computer vision, the algorithm is general enough to work also in  $\mathbb{R}^n$ . FLANN is used in outlier filtering and feature extraction from point clouds, obtained from laser scans (see also: Point Cloud Library (PCL))<sup>6</sup> [121]).

For robotics applications, a  $C^1$ -continuity of the distance function is desired, since its gradient continuity ensures also continuity of low level velocity control while tracking tasks using proximity distance. ESCANDE et al. present in [47] the concept of “Sphere-Torus-Patches Bounding Volumes” (STP-BV) satisfying  $C^1$ -continuity of the distance function. This approach produces a tight bounding volume representation composed of convex patches of spheres and toruses. A distance computation from STP-BV to other convex polyhedra using the GJK algorithm is presented in [15] by the same authors.

More recently, TÄUBIG et al. proposed the concept of “sphere swept convex hulls” (SSCH) [141]. The main idea of SSCH is to increase the polygonal convex bounding volume hull by a buffer radius. The method is applied on the humanoid manipulation platform JUSTIN to avoid collisions on joint torque level for manipulation tasks and to catch a ball thrown by a human [10, 43].

Strategies for collision detection involving *deformable* objects in computer graphics are discussed in [142]. Approaches using the graphics processing unit (GPU) for deformable objects are found in [112, 140]. Methods exploiting the computational power of GPUs can be orders of magnitudes faster than other CPU-based approaches. However, since time spent for data compaction and data transfer from CPU to GPU and back are substantial, only large models with over  $10^5$  triangles benefit from these methods. However, since computer architectures are evolving and GPU-based approaches are an active research field, this might change in the future.

## 6.2 Formal Aspects of Distance Computation

Finding the minimal distance  $d$  between two manifolds  $S_0$  and  $S_1$  can be formulated by the minimization problem in the form

$$d = \min_{\substack{\mathcal{S}_{c,0} \in S_0 \\ \mathcal{S}_{c,1} \in S_1}} \|\mathcal{S}_{c,1} - \mathcal{S}_{c,0}\|, \quad (6.1)$$

where  $\mathcal{S}_{c,i}$  are the closest manifold subsets of  $S_i$ . Depending on the shape and configuration of the manifolds  $S_i$  the solution regions  $\mathcal{S}_{c,i}$  can be surfaces, curves or points in  $\mathbb{R}^3$ .

<sup>5</sup> FLANN is publicly available from: <http://mloss.org/software/view/143/>

<sup>6</sup> PCL is publicly available from: <http://pointclouds.org>

In many practical applications, such as collision avoidance, it is also important to know  $\mathcal{S}_{c,i}$ . Hence, problem (6.1) is cast to find the manifolds  $\mathcal{S}_{c,i}$  where the distance  $d$  is minimal

$$\begin{aligned} \mathcal{S}_{c,i}^* &= \begin{pmatrix} \mathcal{S}_{c,1}^* \\ \mathcal{S}_{c,0}^* \end{pmatrix}^\top = \arg \min_{\substack{\mathcal{S}_{c,0} \in \mathcal{S}_0 \\ \mathcal{S}_{c,1} \in \mathcal{S}_1}} \frac{1}{2} (\mathcal{S}_{c,1} - \mathcal{S}_{c,0})^2, \\ \Rightarrow d &= \|\mathcal{S}_{c,1}^* - \mathcal{S}_{c,0}^*\|. \end{aligned} \quad (6.2)$$

Solving problem (6.2) can be an extremely complex task. In most practical applications, however, when run-time is a critical factor and the exact knowledge of the solution manifolds is not important, a common practical approach is to reduce the problem to finding two singular points  $\mathbf{p}_i \in \mathcal{S}_{c,i}$  for  $i = \{0,1\}$ . Moreover, for convex manifolds which are rotating relative to each other, the solution  $\mathcal{S}_{c,i}$  almost always contains singular points. The reduced problem is then defined by

$$\begin{aligned} \mathbf{x}^* &= \begin{pmatrix} \mathbf{p}_0^* \\ \mathbf{p}_1^* \end{pmatrix} = \arg \min_{\substack{\mathbf{p}_0 \in \mathcal{S}_0 \\ \mathbf{p}_1 \in \mathcal{S}_1}} \frac{1}{2} (\mathbf{p}_1 - \mathbf{p}_0)^2, \\ \Rightarrow d &= \|\mathbf{p}_1^* - \mathbf{p}_0^*\|, \end{aligned} \quad (6.3)$$

where  $\mathbf{p}_i \in \mathcal{S}_{c,i} \in \mathcal{S}_i$  are the closest points between the manifolds  $\mathcal{S}_i$ .

### 6.3 SSV Primitives

Unlike some other biped robots, the surface geometry of Lola's segments is quite complex, because there is no smooth protective cover. Moreover, since not all cabling is modeled in CAD, the approximation of the geometry with a triangulated CAD-mesh is difficult. However, for collision avoidance the *exact* knowledge of the distances is not required. Therefore, the shape of the robot can be greatly simplified. In this work the concept of *swept-sphere-volumes* (SSV) [79, 125, 133] is adopted, since it is computational efficient, sufficiently accurate and allows for non-convex geometries.

Formally, a SSV is the result of the Minkowski sum obtained by adding a sphere to an underlying manifold. Hence, the resulting geometry inherits the  $\mathcal{C}^1$ -continuity of the sphere and convexity properties from the manifold.

The SSV modeling library developed in this work<sup>7</sup> consists of a family of three geometric primitives: the point-, line- and triangle-SSV. The use of the triangle-SSV (TSS) to model robotic segments for collision avoidance is proposed by the author in [125]. In contrast to the rectangle-SSV proposed by other researchers [79], complex shapes such as free-form surfaces can only be modeled using TSS.

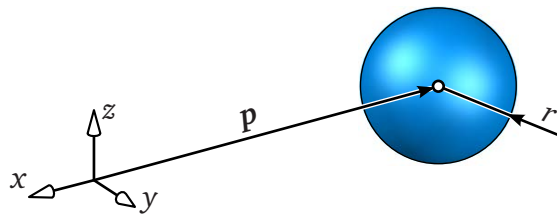
<sup>7</sup> The SSV modeling library is based on a prototype implementation by MATTIAS TRÄGER (cf. [147]).

Properties and algorithms to compute the distance between each class is derived in the following sections.

### 6.3.1 Point-Swept-Sphere Volume

A Point-Swept-Sphere (PSS) Volume is defined by

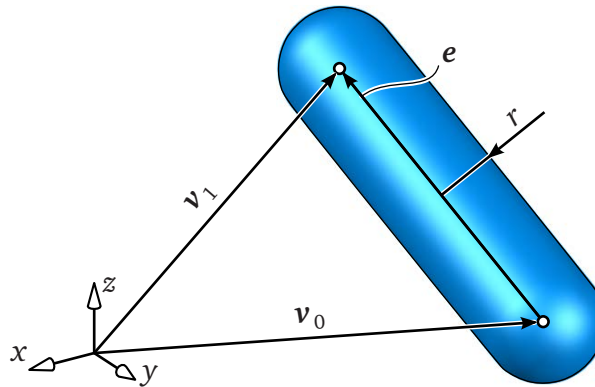
- the radius  $r$  of the PSS
- the position vector describing the center-point  $\mathbf{p}$



### 6.3.2 Line-Swept-Sphere Volume

A Line-Swept-Sphere (LSS) Volume is defined by

- the radius  $r$  of the LSS
- two vertices ( $\mathbf{v}_0, \mathbf{v}_1$ ) which define the start and end points of the line-segment



Each point on the line-segment is described by

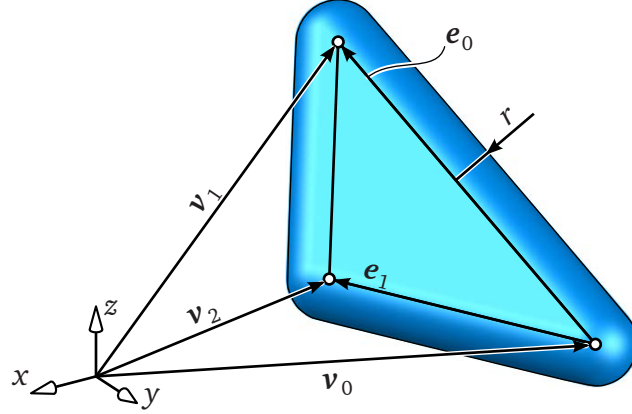
$$\mathcal{L}(s) = \mathbf{p}(s) = \mathbf{v}_0 + s\mathbf{e}, \quad (6.4)$$

with the parameter  $s \in [0, 1]$  and the edge vector  $\mathbf{e} := \mathbf{v}_1 - \mathbf{v}_0$ .

### 6.3.3 Triangle-Swept-Sphere Volume

A Triangle-Swept-Sphere (TSS) Volume is defined by

- the radius  $r$  of the TSS
- three vertices  $v_i$  for  $i \in \{0,1,2\}$  of the triangle



Each point  $p$  on the triangle is described by a linear combination of the edge vectors  $e_i$

$$\mathcal{T}(s) = p(s_0, s_1) = v_0 + s_0 e_0 + s_1 e_1, \quad (6.5)$$

using the parameters  $s_i$  which satisfy the conditions

$$0 \leq s_0 \leq 1 \quad \wedge \quad 0 \leq s_1 \leq 1 \quad \wedge \quad s_0 + s_1 \leq 1. \quad (6.6)$$

Here, the edge vectors are defined as  $e_0 = v_1 - v_0$  and  $e_1 = v_2 - v_0$ .

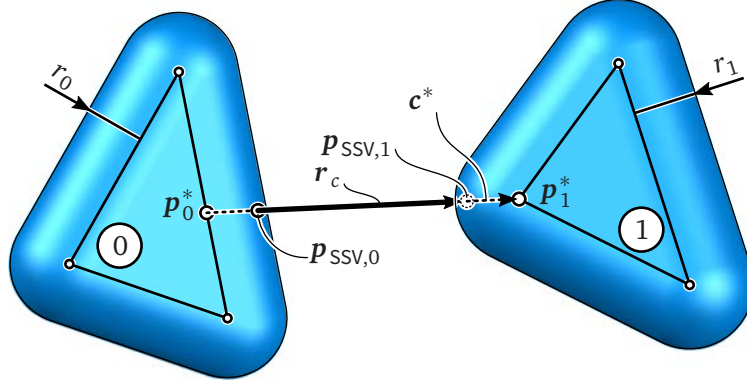
## 6.4 Distance Calculation between SSV Primitives

In this section, the efficient computation of the minimal distance between all three classes of SSV is addressed. Since a SSV is defined as the surface obtained by offsetting points, lines or triangles by its radius, we use those constructing primitives as the manifolds  $\mathcal{S}_i$  to solve the minimization problem (6.3). Due to the parameterizations of the manifolds, e. g.,  $\mathcal{L}(s)$  and  $\mathcal{T}(s)$  in (6.4) and (6.5), we have  $\mathcal{S}_i = \mathcal{S}_i(\mathbf{x}_i)$ . The minimization problem is then reformulated as:

$$\phi = \frac{1}{2} \mathbf{c}^T \mathbf{c}, \quad \text{with } \mathbf{c} = \mathcal{S}_1(\mathbf{x}_1) - \mathcal{S}_0(\mathbf{x}_0), \quad (6.7)$$

$$\mathbf{x}^* = \left( \mathbf{x}_0^* \quad \mathbf{x}_1^* \right)^T = \arg \min_{\mathbf{x}} \phi(\mathbf{x}), \quad (6.8)$$





**Figure 6.1:** Definitions and relations of resulting points and vectors from distance computation in the form of (6.11). Here, the distance is calculated from SSV “0” to SSV “1”.

where  $\phi$  is the objective function,  $\mathbf{c}$  is a vector connecting *any two feasible* points  $\mathbf{p}_0 \in \mathcal{S}_0(\mathbf{x}_0)$  and  $\mathbf{p}_1 \in \mathcal{S}_1(\mathbf{x}_1)$  and  $\mathbf{x}_i^*$  denotes the parameters of the manifold  $\mathcal{S}_i$  where  $\phi$  is minimal. The resulting closest points on the manifolds are then calculated by:

$$\mathbf{p}_0^* = \mathcal{S}_0(\mathbf{x}_0^*), \quad \mathbf{p}_1^* = \mathcal{S}_1(\mathbf{x}_1^*), \quad (6.9)$$

$$\mathbf{c}^* = \mathbf{p}_1^* - \mathbf{p}_0^*, \quad d^* = \|\mathbf{c}^*\|, \quad (6.10)$$

where  $\mathbf{c}^*$  is the vector describing the shortest distance  $d^*$ . Finally, the quantities defined on the surface of both SSVs are obtained via offsets along  $\mathbf{c}^*$  by their radii  $r_i$ :

$$\begin{aligned} \mathbf{p}_{\text{SSV},0} &= \mathbf{p}_0^* + \frac{\mathbf{c}^*}{\|\mathbf{c}^*\|} r_0, & \mathbf{p}_{\text{SSV},1} &= \mathbf{p}_1^* - \frac{\mathbf{c}^*}{\|\mathbf{c}^*\|} r_1, \\ \mathbf{r}_c &= \mathbf{p}_{\text{SSV},1} - \mathbf{p}_{\text{SSV},0}, & d &= d^* - r_0 - r_1 = \|\mathbf{r}_c\|, \end{aligned} \quad (6.11)$$

where  $\mathbf{p}_{\text{SSV},i}$  are the closest points located on the surface of SSV  $i$ ,  $\mathbf{r}_c$  is the connecting vector and  $d$  the minimal distance. Figure 6.1 illustrates the individual quantities presented above and their relations.

Therefore, the problem of finding the minimal distance between SSVs is effectively transformed into the problem of calculating the distance between points, lines and triangles. While the former problem may suffer from discontinuities or singularities in the SSV surface parameterizations, the latter problem uses simple geometric objects and can therefore be formulated in an efficient manner.

For the  $n = 3$  SSV primitives  $n^2 = 9$  distance calculation functions, listed in Table 6.1, are required to cope with any combination.

Given a function to calculate the quantities in (6.11) from SSV class  $\mathcal{A}$  to class  $\mathcal{B}$ :

$$\begin{pmatrix} d & \mathbf{p}_A & \mathbf{p}_B \end{pmatrix}^T \leftarrow \text{dist}(\mathcal{A}, \mathcal{B}),$$

**Table 6.1:** Distance calculation functions needed for the SSV library. Section and page where the specific derivation is found.

from \ to	point $\mathbf{p}_1$	line $\mathcal{L}_1(t)$	triangle $\mathcal{T}_1(t)$
point $\mathbf{p}_0$	PP( $\mathbf{p}_0, \mathbf{p}_1$ ) Sec. 6.4.1, p. 96	PL( $\mathbf{p}_0, \mathcal{L}_1(t)$ ) Sec. 6.4.2, p. 97	PT( $\mathbf{p}_0, \mathcal{T}_1(t)$ ) Sec. 6.4.3 p. 98
line $\mathcal{L}_0(s)$	LP( $\mathcal{L}_0(s), \mathbf{p}_1$ )	LL( $\mathcal{L}_0(s), \mathcal{L}_1(t)$ ) Sec. 6.4.4, p. 99	LT( $\mathcal{L}_0(s), \mathcal{T}_1(t)$ ) Sec. 6.4.5, p. 108
triangle $\mathcal{T}_0(s)$	TP( $\mathcal{T}_0(s), \mathbf{p}_1$ )	TL( $\mathcal{T}_0(s), \mathcal{L}_1(t)$ )	TT( $\mathcal{T}_0(s), \mathcal{T}_1(t)$ ) Sec. 6.4.6, p. 110

solving for the reversed problem is trivial, since the relation

$$\begin{array}{ccc}
 ( d \ \mathbf{p}_A \ \mathbf{p}_B \ \mathbf{r}_c ) & \leftarrow \text{dist}( \mathcal{A}, \mathcal{B} ) \\
 \downarrow \quad \times \quad \oplus & & \times \\
 ( d \ \mathbf{p}_B \ \mathbf{p}_A \ -\mathbf{r}_c ) & \leftarrow \text{dist}( \mathcal{B}, \mathcal{A} )
 \end{array}$$

between the results holds. The reversed problem can therefore be solved by swapping  $\mathbf{p}_{\text{SSV},0}$  and  $\mathbf{p}_{\text{SSV},1}$  and reversing  $\mathbf{r}_c$  in the result obtained from the known function  $\text{dist}(\mathcal{A}, \mathcal{B})$ . By exploiting this anti-symmetric property in Table 6.1 the implementation effort reduces to  $n(n+1)/2 = 6$  functions. Hence, only functions listed in the upper (or lower) triangular part must be implemented while the remaining functions use their existing symmetric counterpart and modify the result.

For the same reason, the derivations of the minimal distance calculation for the off-diagonal functions in Table 6.1 are given only for one variant. The objective function  $\phi(\mathbf{x})$  in (6.7) is the basis for all derivations. The result is always given as points  $\mathbf{p}^*$  on the manifolds point, line or triangle in the form of (6.9) which is then used to evaluate (6.11) to get the offset quantities.

The problem of finding the closest points between two manifolds usually is subject to domain restrictions in the form of inequality constraints (see, e. g., (6.6)). Therefore, in Section 6.4.7 a general framework using a minimization with inequality constraints is presented. The framework can be used to verify the implementations or as an alternative distance calculation method if explicit functions are too complex.

### 6.4.1 Point to Point Distance Computation

Since the point is not parameterized,  $\mathbf{p}_i$  are already the closest points in (6.9), i. e.,  $\mathbf{p}_i^* = \mathbf{p}_i$ . Hence, we have the solution to the minimal distance given as

$$\mathbf{c}^* = \mathbf{p}_1 - \mathbf{p}_0 \quad \Rightarrow \quad d = \|\mathbf{c}\| - r_0 - r_1. \quad (6.12)$$

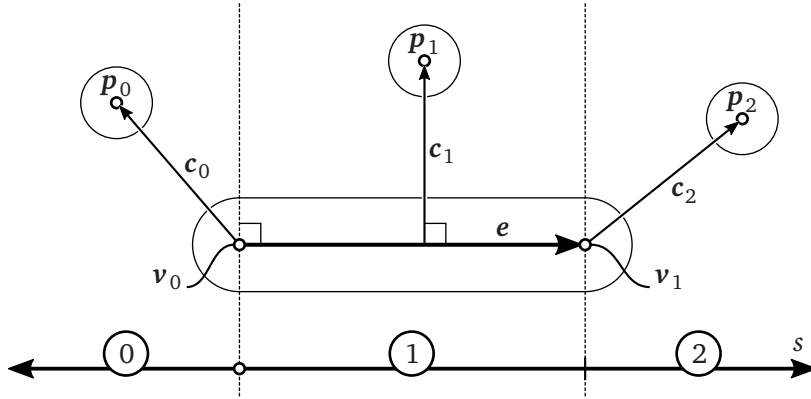


Figure 6.2: Definitions for the point to line segment minimal distance computation.

### 6.4.2 Point to Line-Segment Distance Computation

Referring to Figure 6.2, vector  $\mathbf{c}$  to calculate the minimal distance from line-segment to point is given by

$$\mathbf{c} = \mathbf{p} - (\mathbf{v}_0 + s\mathbf{e}), \quad (6.13)$$

where  $\mathbf{p}$  is the center point of the sphere,  $\mathbf{v}_0$  and is the first point,  $\mathbf{e}$  the edge vector and  $s \in [0, 1]$  is the parameter of the line. The optimization problem then writes to

$$s = \arg \min \frac{1}{2} \mathbf{c}^T \mathbf{c} = \arg \min \underbrace{\frac{1}{2} (\mathbf{p} - (\mathbf{v}_0 + s\mathbf{e}))^2}_{\phi(s)}. \quad (6.14)$$

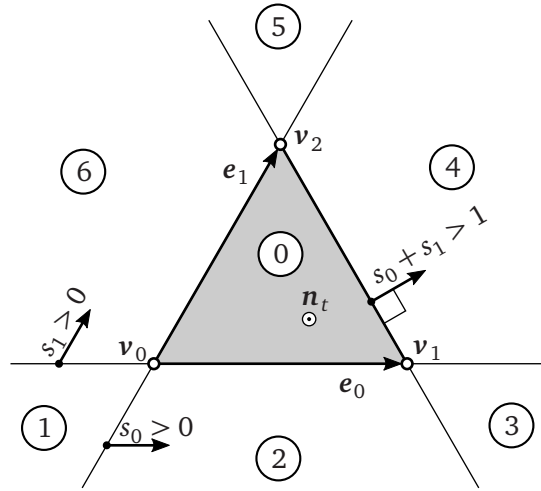
The problem is solved by finding the root of the partial derivative of the objective function  $\phi(s)$

$$\frac{\partial \phi}{\partial s} = -(\mathbf{p} - \mathbf{v}_0 - s\mathbf{e})^T \mathbf{e} \stackrel{!}{=} 0, \quad (6.15)$$

$$\Rightarrow s = \frac{(\mathbf{p} - \mathbf{v}_0)^T \mathbf{e}}{\mathbf{e}^T \mathbf{e}}. \quad (6.16)$$

Since  $s \in [0, 1]$ , the three cases illustrated in Figure 6.2 must be considered for  $s^*$ :

$$s^* = \begin{cases} 0 & \text{if } \mathbf{w}^T \mathbf{e} \leq 0 \\ 1 & \text{if } \mathbf{w}^T \mathbf{e} \geq \mathbf{e}^T \mathbf{e} \\ (\mathbf{w}^T \mathbf{e}) / (\mathbf{e}^T \mathbf{e}) & \text{otherwise,} \end{cases} \quad \text{with } \mathbf{w} = \mathbf{p} - \mathbf{v}_0. \quad (6.17)$$



**Figure 6.3:** Top view of the triangle showing solution regions 0-6 and important vector definitions for the point to triangle distance computation.

Finally, for the quantities used in (6.11) to get the SSV specific vectors, we have

$$\mathbf{p}_{\text{line}}^* = \mathbf{v}_0 + s^* \mathbf{e} \quad \text{and} \quad \mathbf{p}_{\text{point}}^* = \mathbf{p} . \quad (6.18)$$

### 6.4.3 Point to Triangle Distance Computation

The point to triangle distance computation is derived in two steps. First, the solution on an unbound plane spanned by the triangle is derived. Second, if necessary, the solution is modified to be within the feasible region of the triangle.

#### Minimal Distance between Point and Plane

Using point  $\mathbf{p}$  and  $\mathcal{T}(s)$  from (6.5) defining all points of plane  $\mathcal{P}(\mathcal{T})$  spanned by triangle  $\mathcal{T}(s)$  we get

$$\begin{aligned} \mathbf{c} &= \mathcal{T}(\mathcal{P}(s)) - \mathbf{p} = \mathbf{v}_0 + \mathbf{e}_0 s_0 + \mathbf{e}_1 s_1 - \mathbf{p} \\ &= \mathbf{v}_0 + \begin{pmatrix} \mathbf{e}_0 & \mathbf{e}_1 \end{pmatrix} \begin{pmatrix} s_0 & s_1 \end{pmatrix}^T - \mathbf{p} \\ &= \mathbf{v}_0 + \mathbf{D}\mathbf{s} - \mathbf{p} . \end{aligned} \quad (6.19)$$

The resulting optimization problem

$$\frac{1}{2} (\mathbf{v}_0 + \mathbf{D}\mathbf{s} - \mathbf{p})^2 \rightarrow \min! \quad (6.20)$$

**Table 6.2:** Point to triangle distance calculation: Region specific solution strategies to obtain the closest point on the triangle  $\mathbf{p}_{\mathcal{T}}^*$ . The regions are defined in Figure 6.3. The function denoted by “PL” is the point-line distance function found in Section 6.4.2.

Region	Location of $\mathbf{p}_{\mathcal{T}}^* \in \mathcal{T}$	Condition	Strategy to get $\mathbf{p}_{\mathcal{T}}^*$
0	inside triangle	$s_0 \geq 0 \wedge s_1 \geq 0 \wedge s_0 + s_1 < 1$	$\mathbf{v}_0 + \mathbf{D}\mathbf{s}$
1	on vertex	$s_0 < 0 \wedge s_1 < 0$	$\mathbf{v}_0$
2	on edge	$s_0 \geq 0 \wedge s_1 < 0 \wedge s_0 + s_1 < 1$	from PL( $\mathbf{p}, \mathbf{v}_0, \mathbf{v}_1$ )
3	on vertex	$s_1 < 0 \wedge s_0 + s_1 \geq 1$	$\mathbf{v}_1$
4	on edge	$s_0 \geq 0 \wedge s_1 \geq 0 \wedge s_0 + s_1 \geq 1$	from PL( $\mathbf{p}, \mathbf{v}_1, \mathbf{v}_2$ )
5	on vertex	$s_0 < 0 \quad \wedge \quad s_0 + s_1 \geq 1$	$\mathbf{v}_2$
6	on edge	$s_0 < 0 \wedge s_1 \geq 0 \wedge s_0 + s_1 < 1$	from PL( $\mathbf{p}, \mathbf{v}_2, \mathbf{v}_0$ )

is solved for  $\mathbf{s}$  which gives

$$\mathbf{s} = (\mathbf{D}^T \mathbf{D})^{-1} \mathbf{D}^T (\mathbf{p} - \mathbf{v}_0) = \mathbf{S} (\mathbf{p} - \mathbf{v}_0). \quad (6.21)$$

The resulting point  $\mathbf{p}_{\mathcal{P}} \in \mathcal{P}$  defined on plane  $\mathcal{P}$  calculates to

$$\mathbf{p}_{\mathcal{P}} = \mathcal{T}(\mathbf{s}) = \mathbf{v}_0 + \mathbf{e}_0 s_0 + \mathbf{e}_1 s_1. \quad (6.22)$$

Since  $\mathbf{D}$  and  $\mathbf{S}$  in (6.21) are constant for one triangle configuration, they can be pre-computed to gain efficiency.  $\mathbf{D}$  becomes singular if  $\mathbf{e}_i$  are either parallel or zero, which is the result from a triangle degenerated to a line or a point.

Here,  $\mathbf{p}_{\mathcal{P}} \in \mathcal{P}$  might be outside the feasible region of triangle  $\mathcal{T}$ , and therefore, the inequality constraints (6.6) for  $\mathbf{s}$  are not satisfied. In the following section, strategies to obtain a point  $\mathbf{p}_{\mathcal{T}}^* \in \mathcal{T}$  are presented.

### Minimal Distance between Point and Triangle

Figure 6.3 shows seven independent regions about the triangle where  $\mathbf{p}_{\mathcal{P}}$  can be located. Which region  $\mathbf{p}_{\mathcal{P}}$  lies in can be determined by studying  $\mathbf{s}$ . The different cases along with a strategy to obtain  $\mathbf{p}_{\mathcal{T}}^* \in \mathcal{T}$  are summarized in Table 6.2. If, e. g., region 4 is identified, the distance calculation between point and line segment presented in Section 6.4.2 is used.

Finally, in order to calculate the solution defined on the surface of the SSVs (cf. (6.11)),  $\mathbf{p}_{\mathcal{T}}^*$  is obtained by following a strategy from Table 6.2 based on  $\mathbf{s}$  and  $\mathbf{p}_{\text{point}}^* = \mathbf{p}$ .

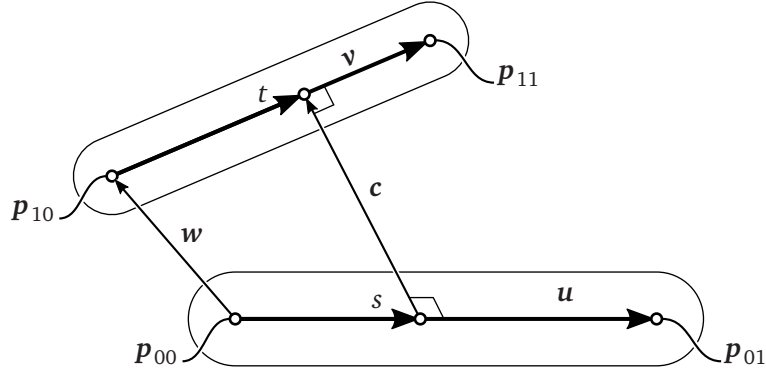


Figure 6.4: Definitions for the distance computation of Line to Line.

#### 6.4.4 Line-Segment to Line-Segment Distance Computation

Referring to Figure 6.4 each point on the two line-segments can be described by

$$\mathcal{L}_0(s) = \mathbf{p}_{00} + s\mathbf{u} \quad \text{and} \quad \mathcal{L}_1(t) = \mathbf{p}_{10} + t\mathbf{v}, \quad (6.23)$$

where  $s \in [0,1]$  and  $t \in [0,1]$  are the line parameters and

$$\mathbf{u} = \mathbf{p}_{01} - \mathbf{p}_{00} \quad \text{and} \quad \mathbf{v} = \mathbf{p}_{11} - \mathbf{p}_{10} \quad (6.24)$$

are the edge vectors. In the following sections, first, the distance calculation between two infinite lines is derived and second, the domain restrictions for  $s$  and  $t$  are incorporated.

##### Distance Between Two (Infinite) Lines

With the vector  $\mathbf{w} = \mathbf{p}_{10} - \mathbf{p}_{00}$  connecting the base points of the lines we have

$$\mathbf{c} = \mathbf{w} - s\mathbf{u} + t\mathbf{v} = \mathbf{w} + \mathbf{D}\mathbf{x}, \quad (6.25)$$

where  $\mathbf{D} = \begin{pmatrix} -\mathbf{u} & \mathbf{v} \end{pmatrix} \in \mathbb{R}^{3 \times 2}$  and  $\mathbf{x} = \begin{pmatrix} s & t \end{pmatrix}^\top$  is the vector of line parameters. The objective function  $\phi(\mathbf{x})$  in (6.7) to minimize the distance then writes to

$$\phi(\mathbf{x}) = \frac{1}{2}(\mathbf{w} + \mathbf{D}\mathbf{x})^2. \quad (6.26)$$

Minimization leads to

$$\frac{\partial \phi}{\partial \mathbf{x}} = \mathbf{D}^\top (\mathbf{w} + \mathbf{D}\mathbf{x}) \stackrel{!}{=} \mathbf{0}. \quad (6.27)$$

Solving (6.27) for  $\mathbf{x}$  yields parameters  $\mathbf{x}^*$  which minimize  $\phi$ :

$$\mathbf{x}^* = -(\mathbf{D}^\top \mathbf{D})^{-1} \mathbf{D}^\top \mathbf{w} . \quad (6.28)$$

Interestingly, when rewriting (6.27) by reusing (6.25) as

$$\mathbf{c}^\top \mathbf{D} = \begin{pmatrix} -\mathbf{c}^\top \mathbf{u} \\ \mathbf{c}^\top \mathbf{v} \end{pmatrix}^\top = \mathbf{0} , \quad (6.29)$$

and applying the well-known relation  $\mathbf{a}^\top \mathbf{b} = 0 \Leftrightarrow \mathbf{a} \perp \mathbf{b}$ , it is clear, that  $\mathbf{c}$  is also perpendicular to both  $\mathbf{u}$  and  $\mathbf{v}$  and hence,

$$\begin{aligned} \mathbf{c} \perp \mathbf{u} &\Leftrightarrow \mathbf{c}^\top \mathbf{u} = 0 &\Rightarrow & \mathbf{u}^\top \mathbf{w} - s \mathbf{u}^\top \mathbf{u} + t \mathbf{u}^\top \mathbf{v} = 0 , \\ \mathbf{c} \perp \mathbf{v} &\Leftrightarrow \mathbf{c}^\top \mathbf{v} = 0 &\Rightarrow & \mathbf{v}^\top \mathbf{w} - s \mathbf{v}^\top \mathbf{u} + t \mathbf{v}^\top \mathbf{v} = 0 . \end{aligned} \quad (6.30)$$

Obviously, this orthogonality condition can always be exploited as a starting point to derive *any* distance function between two manifolds, i. e., by constraining  $\mathbf{c}$  to be perpendicular to both manifolds. However, the cases arising from the boundary conditions restricting feasible regions are often far more complex to derive.

Finally, (6.28) can be written explicitly as:

$$\mathbf{x}^* = \begin{pmatrix} s^* \\ t^* \end{pmatrix} := \frac{1}{\mathbf{u}^\top \mathbf{v}^2 - (\mathbf{u}^\top \mathbf{v})^2} \begin{pmatrix} \mathbf{v}^\top \mathbf{u}^\top \mathbf{w} - \mathbf{u}^\top \mathbf{v} \mathbf{v}^\top \mathbf{w} \\ \mathbf{u}^\top \mathbf{v} \mathbf{u}^\top \mathbf{w} - \mathbf{u}^\top \mathbf{v} \mathbf{v}^\top \mathbf{w} \end{pmatrix} = \frac{1}{N} \begin{pmatrix} s_z \\ t_z \end{pmatrix} . \quad (6.31)$$

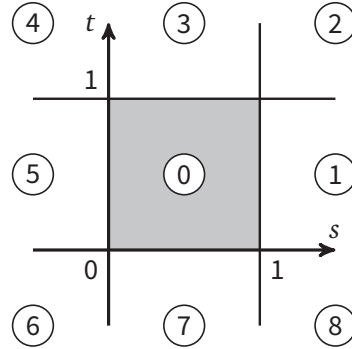
If the denominator  $N = 0$ , both lines are *parallel*. On the other hand, in the *non-parallel* case,  $s$  and  $t$  may not be bound to  $[0, 1]$ . General solutions to both cases are derived in the following sections. In this case,  $s_0 = s^*$  and  $t_0 = t^*$  from (6.28) or (6.31) are used.

### Distance Between Non-Parallel Line Segments

The feasible parameter-space  $s, t \in [0, 1]$  and nine possible domains where  $s_0$  and  $t_0$  can be located are shown in Figure 6.5. If  $s_0$  or  $t_0$  are outside their boundaries, further steps are required. Table 6.3 and Table 6.4 summarize all possible solutions for the two-step algorithm presented here. First,  $t$  is saturated to  $[0, 1]$  and  $s$  is recalculated using the equations found in Table 6.3. In a second step,  $s$  is saturated to  $[0, 1]$  and  $t$  is recalculated using Table 6.4.

The equations in Table 6.3 are obtained by substituting one boundary condition into (6.30) and solving for the other boundary condition. In Table 6.3 solutions 1 and 2 correspond to lines 1 and 2 of (6.30), respectively. Obviously, both solutions are equal and only a matter of the implementation. The equations found in Table 6.4 are the bounded version of the equations found in Table 6.3.

Figure 6.6 shows an example of the algorithm Table 6.3 and Table 6.4. For clear



**Figure 6.5:** Non-parallel line-segment to line-segment distance: feasible parameter space  $(s, t) \in [0,1]$  in gray for line segments and the nine cases for closest point locations.

**Table 6.3:** Equations for the first step to obtain a feasible solution when calculating the closest points between two line segments. Solution 1 and 2 are obtained from (6.30) from line 1 and line 2 by substitution of the saturation. The “Methods” are shown for consistency with Table 6.4.

Case	Saturation	Solution 1	Solution 2
$t_0 < 0$	$t_1 = 0$	$s_1 = \frac{\mathbf{u}^T \mathbf{w}}{\mathbf{u}^T \mathbf{u}}$	$s_1 = \frac{\mathbf{v}^T \mathbf{w}}{\mathbf{u}^T \mathbf{v}}$
$t_0 > 1$	$t_1 = 1$	$s_1 = \frac{\mathbf{u}^T (\mathbf{u} - \mathbf{w})}{\mathbf{u}^T \mathbf{u}}$	$s_1 = \frac{\mathbf{v}^T (\mathbf{u} - \mathbf{w})}{\mathbf{u}^T \mathbf{v}}$
else	$t_1 = t_0$	$s_1 = s_0$	$s_1 = s_0$

**Table 6.4:** Equations for the second step to obtain a feasible solution when calculating the closest points between two line segments (cf. Table 6.3).

Case	Sat.	Solution 1	Solution 2
$s_1 < 0$	$s_2 = 0$	$t_2 = \begin{cases} 0 & -\mathbf{u}^T \mathbf{w} < 0 \\ 1 & -\mathbf{u}^T \mathbf{w} > \mathbf{u}^T \mathbf{v} \\ \frac{-\mathbf{u}^T \mathbf{w}}{\mathbf{u}^T \mathbf{v}} & \text{otherwise} \end{cases}$	$t_2 = \begin{cases} 0 & -\mathbf{v}^T \mathbf{w} < 0 \\ 1 & -\mathbf{v}^T \mathbf{w} > \mathbf{v}^T \mathbf{v} \\ \frac{-\mathbf{v}^T \mathbf{w}}{\mathbf{v}^T \mathbf{v}} & \text{otherwise} \end{cases}$
$s_1 > 1$	$s_2 = 1$	$t_2 = \begin{cases} 0 & \mathbf{u}^T (\mathbf{u} - \mathbf{w}) < 0 \\ 1 & \mathbf{u}^T (\mathbf{u} - \mathbf{w}) > \mathbf{u}^T \mathbf{v} \\ \frac{\mathbf{u}^T (\mathbf{u} - \mathbf{w})}{\mathbf{u}^T \mathbf{v}} & \text{otherwise} \end{cases}$	$t_2 = \begin{cases} 0 & \mathbf{v}^T (\mathbf{u} - \mathbf{w}) < 0 \\ 1 & \mathbf{v}^T (\mathbf{u} - \mathbf{w}) > \mathbf{v}^T \mathbf{v} \\ \frac{\mathbf{v}^T (\mathbf{u} - \mathbf{w})}{\mathbf{v}^T \mathbf{v}} & \text{otherwise} \end{cases}$
else	$s_2 = s_1$	$t_2 = t_1$	$t_2 = t_1$



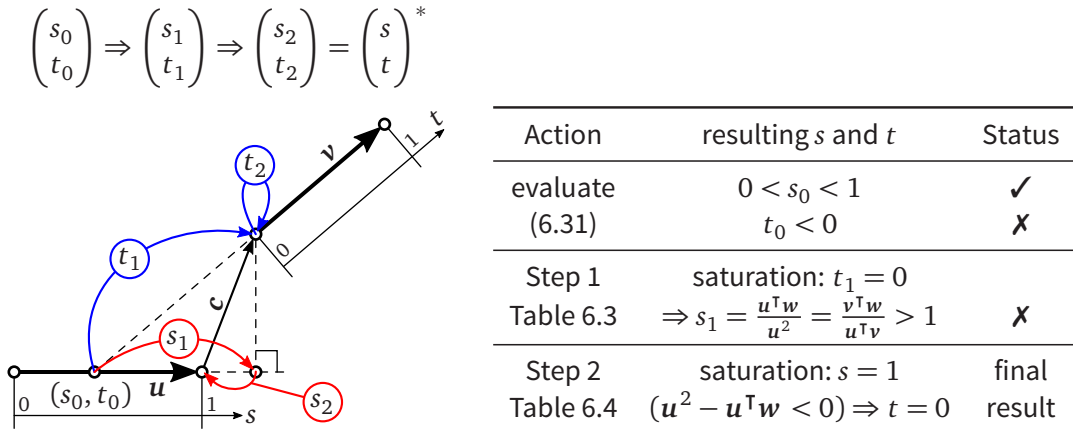


Figure 6.6: Example showing the steps involved in the distance computation between two non-parallel line segments. For clear visual appearance both line segments are defined in a plane.

visual appearance both lines are defined at the same plane.<sup>8</sup> Equation (6.31) gives  $0 < s < 1$  and  $t < 0$ . Therefore, in step 1,  $t$  is saturated to  $t_1 = 0$ . By choosing one of the two solutions, Table 6.3 gives the shortest distance on  $u$  with  $s > 1$ . In step 2,  $s$  is saturated to  $s_2 = 1$  and, according to Table 6.4,  $t$  is set to  $t_2 = 0$ . Therefore, the final result equals to  $(s \ t)^T = (1 \ 0)^T$  which can be proved visually from Figure 6.6.

### Distance Between Parallel Line Segments

By projecting the vectors  $v$  and  $w$  onto  $u$  the problem of finding the parameters  $s$  and  $t$  is effectively transformed into a one-dimensional problem which is solved graphically by studying distances along  $u$ . Table 6.5 summarizes all 11 independent solutions<sup>9</sup> for the parallel case.

If edge vectors  $u$  and  $v$  overlap, the correct solution of the closest points would expand to a line segment. However, the parallel case is expected to be rare and appear only for a short time period. Therefore, the resulting closest points are calculated to be located in the center of the result region. Hence, the result degenerates to a point representing the average value of the solution.

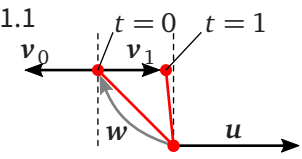
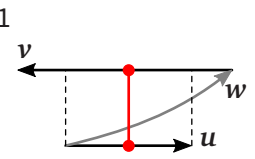
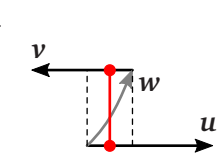
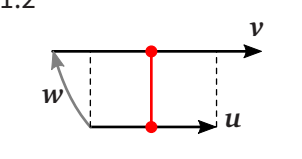
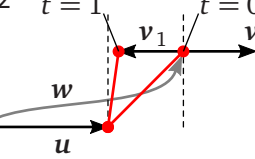
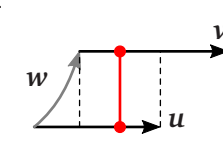
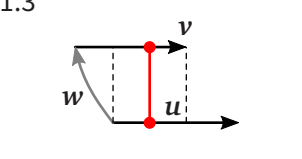
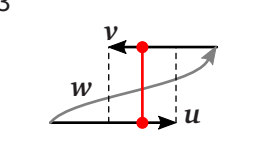
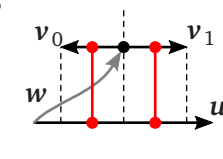
### Numerical Considerations to Identify the Parallel Case


To distinguish the parallel and non-parallel cases one could simply test the denominator in (6.31) for non-zero value to avoid division by zero. However, for numerical reasons it is unlikely to get an exact zero value. Therefore, since  $N$  is always positive, a common approach is to simply test for  $N < \epsilon$ . Here,  $\epsilon$  is a small number, which,

<sup>8</sup> Otherwise, the dashed line leading to the intersection point of  $u$  and  $v$  might not be parallel to  $v$ .

<sup>9</sup> Note that cells 1.1 and 2.2 contain two solutions for  $t$ .

**Table 6.5:** Solution table for  $s$  and  $t$  to compute the minimal distance between two parallel line segments. The results depend only on dot products between  $u$ ,  $v$  and  $w$ . The vectors  $u = p_{01} - p_{00}$  and  $v = p_{11} - p_{10}$  correspond to the first and second line segment respectively. Vector  $w$  connects the starting points  $p_{i,0}$  ( $i \in \{0,1\}$ ) of both lines. In the outermost ring of the table are the descriptions of the mathematical conditions (inner ring) which lead to the solution cell. The solutions are obtained by projections on  $u$ . If two fields overlap (the last column and last row), the connecting vector  $c$  is located in the center of the solution field.

		$p_{10}$ left of $p_{00}$	$p_{10}$ right of $p_{01}$	$p_{10}$ over $u$
$p_{11}$ left of $p_{00}$	$u^T w < 0$	1.1  $s = 0$ $t = \begin{cases} 0 & \text{if } u^T v < 0 \\ 1 & \text{otherwise} \end{cases}$	2.1  $s = 0.5$ $t = (u^2 - 2u^T w) / 2u^T v$	3.1  $s = v^T w / 2u^T v$ $t = -u^T w / 2u^T v$
	$u^T v + u^T w > u^2$	1.2  $s = 0.5$ $t = (u^2 - 2u^T w) / 2u^T v$	2.2  $s = 1$ $t = \begin{cases} 1 & \text{if } u^T v < 0 \\ 0 & \text{otherwise} \end{cases}$	3.2  $s = (u^T v + v^T w) / 2u^T v$ $t = (u^2 - u^T w) / 2u^T v$
	otherwise	1.3  $s = (v^2 + v^T w) / 2u^T v$ $t = (u^T v - u^T w) / 2u^T v$	2.3  $s = \frac{u^T v + v^2 + v^T w}{2u^T v}$ $t = \frac{u^T v + u^2 - u^T w}{2u^T v}$	3.3  $s = (v^2 + 2v^T w) / 2u^T v$ $t = 0.5$

  $\Rightarrow$  resulting connecting vector  $c$  pointing from  $u$  to  $v$

depending on the working precision of the operation/computer, is usually defined in the range of  $10^{-15} \leq \varepsilon \leq 10^{-6}$ . However, under certain circumstances this test will fail and select the code path for the parallel lines. Hence, a modified criterion is investigated.

Two vectors  $\mathbf{u}$  and  $\mathbf{v}$  are co-linear if we have

$$\frac{\mathbf{u}}{\|\mathbf{u}\|} \cdot \frac{\mathbf{v}}{\|\mathbf{v}\|} = 1.$$

Squaring and bringing the fraction to the other side gives

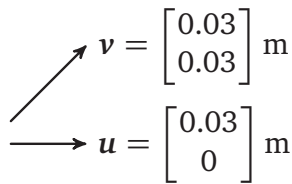
$$1 - \frac{(\mathbf{u}^\top \mathbf{v})^2}{\mathbf{u}^2 \mathbf{v}^2} = 0.$$

Which finally leads to

$$\frac{\mathbf{u}^2 \mathbf{v}^2 - (\mathbf{u}^\top \mathbf{v})^2}{\mathbf{u}^2 \mathbf{v}^2} = \frac{N}{\mathbf{u}^2 \mathbf{v}^2} = 0 \quad (6.32)$$

$$\Leftrightarrow N = 0. \quad (6.33)$$

From a mathematical point of view, (6.32) and (6.33) are equal and using  $N < \varepsilon$  should be good to test for co-linearity. However, from a numerical point of view (6.32) is favorable. A simple numerical example reveals the problem:



$$\mathbf{v} = \begin{bmatrix} 0.03 \\ 0.03 \end{bmatrix} \text{ m}$$

$$\mathbf{u} = \begin{bmatrix} 0.03 \\ 0 \end{bmatrix} \text{ m}$$

The two vectors shown on the left are definitely not parallel and (for a real-world example) have a reasonable length of a few cm. Evaluating the denominator  $N$  in (6.31) (resp. nominator  $N$  in (6.33)) and using  $\varepsilon = 10^{-6}$  to test for near-zero value of  $N$  gives

$$N = \mathbf{u}^2 \mathbf{v}^2 - (\mathbf{u}^\top \mathbf{v})^2 = 16.2 \cdot 10^{-7} - 8.1 \cdot 10^{-7} \\ = 8.1 \cdot 10^{-7} < 10^{-6} = \varepsilon.$$

Here, the numerical criterion from (6.33) fails and the vectors are identified to be co-linear, which is obviously wrong. Decreasing  $\varepsilon$  resolves the problem in this case but would lead to numerical instabilities for very long vectors.

However, when applying the numerical criterion from (6.32) we get  $N/(\mathbf{u}^2 \mathbf{v}^2) = 0.5 \geq 10^{-6} = \varepsilon$  and the non-parallel case is correctly identified.  $\square$

Therefore, replacing the simple criterion ( $N < \varepsilon$ ) by adopting (6.32)

$$\frac{\mathbf{u}^2 \mathbf{v}^2 - (\mathbf{u}^\top \mathbf{v})^2}{\mathbf{u}^2 \mathbf{v}^2} = \boxed{\frac{N}{\mathbf{u}^2 \mathbf{v}^2} < \varepsilon} \quad (6.34)$$

to identify the parallel case is numerically more robust.

A pseudo-code for the calculation of the minimum distance between two line-segments is presented in Algorithm B.1 (Section B.1 on page 141).

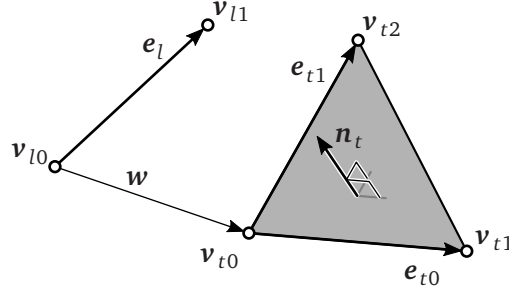


Figure 6.7: Quantities needed for the line-segment to triangle distance calculation.

### 6.4.5 Line-Segment to Triangle Distance Computation

With reference to Figure 6.7 each point on line  $\mathcal{L}(t)$  and triangle  $\mathcal{T}(s)$  is described by

$$\mathcal{L}(t) := \mathbf{v}_{l0} + t \mathbf{e}_l \quad \text{and} \quad \mathcal{T}(s) := \mathbf{v}_{t0} + s_0 \mathbf{e}_{t0} + s_1 \mathbf{e}_{t1}, \quad (6.35)$$

where

$$\mathbf{e}_l := \mathbf{v}_{l1} - \mathbf{v}_{l0}, \quad \mathbf{e}_{t0} := \mathbf{v}_{t1} - \mathbf{v}_{t0}, \quad \mathbf{e}_{t1} := \mathbf{v}_{t2} - \mathbf{v}_{t0}, \quad (6.36)$$

are the edge vectors of the line and triangle respectively.

#### Intersection Point Between (Infinite) Line and Plane

A line and a plane always intersect each other in  $\mathbb{R}^3$ , unless they are parallel. In order to find that intersection point, we define the unified connection vector

$$\mathbf{c} = \mathcal{T}(s) - \mathcal{L}(t) \quad (6.37)$$

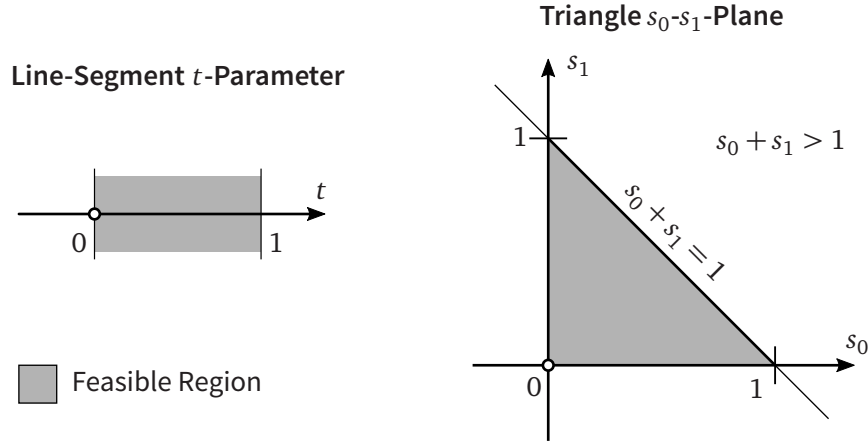
$$= \mathbf{w} + \mathbf{D} \mathbf{x} \quad \text{with} \quad \mathbf{D} = \begin{pmatrix} \mathbf{e}_{t0} & \mathbf{e}_{t0} & -\mathbf{e}_l \end{pmatrix} \in \mathbb{R}^{3 \times 3}. \quad (6.38)$$

Here, vector  $\mathbf{w} = \mathbf{v}_{t0} - \mathbf{v}_{l0}$  connects the base points,  $\mathbf{x} = \begin{pmatrix} s^\top & t \end{pmatrix}^\top = \begin{pmatrix} s_0 & s_1 & t \end{pmatrix}^\top$  is the vector of parameters and matrix  $\mathbf{D} = \begin{pmatrix} \mathbf{e}_{t0} & \mathbf{e}_{t0} & -\mathbf{e}_l \end{pmatrix} \in \mathbb{R}^{3 \times 3}$ . Objective function  $\phi(\mathbf{x})$  from (6.7) to minimize the distance then is written as:

$$\phi(\mathbf{x}) = \frac{1}{2} (\mathbf{w} + \mathbf{D} \mathbf{x})^2. \quad (6.39)$$

Minimization  $\mathbf{x}^* = \arg \min_{\mathbf{x}} \phi$  leads to

$$\frac{\partial \phi}{\partial \mathbf{x}} = \mathbf{D}^\top (\mathbf{w} + \mathbf{D} \mathbf{x}) \stackrel{!}{=} \mathbf{0}, \quad (6.40)$$



**Figure 6.8:** Feasible regions for the line-segment (parameter  $t$ ) to triangle distance calculation (parameters  $s_0, s_1$ ).

and solving for  $\mathbf{x}$  obtains the parameters  $\mathbf{x}^*$  which minimize  $\phi$ :

$$\mathbf{x}^* = -\mathbf{D}^{-1} \mathbf{w} \quad (6.41)$$

$$= \frac{1}{\mathbf{n}_t^\top \mathbf{e}_l} \left( \mathbf{e}_l \times \mathbf{e}_{t,1} \quad \mathbf{e}_{t,0} \times \mathbf{e}_l \quad \mathbf{n}_t \right)^\top \mathbf{w} \quad \text{with } \mathbf{n}_t := \mathbf{e}_{t,0} \times \mathbf{e}_{t,1}, \quad (6.42)$$

where  $\mathbf{n}_t$  is the normal vector of the triangle. Equation (6.42) is singular, if the denominator becomes zero:

$$\mathbf{n}_t^\top \mathbf{e}_l = 0 \quad \Leftrightarrow \quad \mathbf{n}_t \perp \mathbf{e}_l \quad \Leftrightarrow \quad \mathcal{L}(t) \parallel \mathcal{T}(s), \quad (6.43)$$

in which case line  $\mathcal{L}$  and plane  $\mathcal{T}$  are parallel.<sup>10</sup> Similar to the line to line distance calculation (see page 103), it is more robust to directly test for parallelism instead of watching for near-zero denominator in (6.42) to avoid division by zero.

A line defined along  $\mathbf{e}_l$  and a plane with the normal vector  $\mathbf{n}_t$  are parallel if

$$\frac{\mathbf{n}_t}{\|\mathbf{n}_t\|} \cdot \frac{\mathbf{e}_l}{\|\mathbf{e}_l\|} = 0, \quad (6.44)$$

and hence, a numerically robust criterion to identify the parallel case is given by

$$-\varepsilon \leq \frac{\mathbf{n}_t}{\|\mathbf{n}_t\|} \cdot \frac{\mathbf{e}_l}{\|\mathbf{e}_l\|} \leq \varepsilon \quad \Rightarrow \quad \left| \frac{\mathbf{n}_t}{\|\mathbf{n}_t\|} \cdot \frac{\mathbf{e}_l}{\|\mathbf{e}_l\|} \right| \leq \varepsilon, \quad (6.45)$$

where  $\varepsilon$  is a small, positive number (typically:  $10^{-15} \leq \varepsilon \leq 10^{-6}$ ).

<sup>10</sup> Obviously, in the trivial case of a degenerated line or triangle we also get  $\mathbf{n}_t \mathbf{e}_l = 0$ .

**Table 6.6:** Minimal enumeration of possible cases and corresponding constraint states for the non-parallel line-triangle distance function. The cases are combined from table **Line** and **Triangle** giving a total of  $3 \times 7 = 21$  cases. Constraint-state symbols are defined below, i. e., active means violation of the constraint.

Line			Triangle			
Cases	Condition <sup>a)</sup>		Cases	Condition <sup>a)</sup>		
[a-c]	$t \leq 0$	$t \geq 1$	[0-6]	$s_0 \leq 0$	$s_1 \leq 0$	$s_0 + s_1 \geq 1$
a	□	□	0	□	□	□
b	■		1	■	■	□
c		■	2	□	■	□
			3	■	■	
			4	□	□	■
			5	■		■
			6	□	■	■

a) constraint violation symbols: ■ → active, □ → inactive, “ ” → implicitly inactive

For the distance calculation between an infinite line parallel to a plane the reader is referred to Section 6.4.3 and using the first vertex of the line  $\mathbf{v}_{l0}$  as point  $\mathbf{p}$  therein.

### Line-Segment to Triangle Distance Calculation

Since (6.41) gives the unbound parameters  $\mathbf{x}^*$  of the intersection point between line and plane, the boundaries of the geometry are considered in this section for the non-parallel and parallel case, respectively.

**Non-Parallel Line and Triangle** Referring to Section 6.4.2 and 6.4.3, a total of  $n_{\text{cases, PL}} \times n_{\text{cases, PT}} = 3 \times 7 = 21$  cases must be considered here. Table 6.6 enumerates these based on the constraint states. If one of the three parameters in  $\mathbf{x}$  is outside their feasible region, a two-step approach is developed —analogously to the line-line distance calculation in Section 6.4.4. First,  $t_0$  is calculated from (6.42) to

$$t_0 = \frac{\mathbf{n}_t^\top \mathbf{w}}{\mathbf{n}_t^\top \mathbf{e}_l}. \quad (6.46)$$

If  $t_0$  is within the feasible region  $t \in [0, 1]$ ,  $t_0$  is accepted ( $\Rightarrow t_1 = t_0$ ) and  $s_1$  is also calculated using (6.42) to

$$\mathbf{s}_1 = \begin{pmatrix} s_0 \\ s_1 \end{pmatrix}_1 = \frac{1}{\mathbf{n}_t^\top \mathbf{e}_l} \left( \mathbf{e}_l \times \mathbf{e}_{t,1} \quad \mathbf{e}_{t,0} \times \mathbf{e}_l \right)^\top \mathbf{w}. \quad (6.47)$$

**Table 6.7:** Line-segment to triangle distance calculation: Region specific solution strategies to obtain the closest points  $\mathbf{p}_L$  and  $\mathbf{p}_T$  on the line and triangle respectively. The regions are defined in Figure 6.3 on page 98. Here, the solution from step 1, i. e.,  $\mathbf{s}_1$  is used. The function denoted by “LL” is the line-line distance function found in Section 6.4.4. For regions 1, 3 and 5, which are close to vertices of the triangle, the line-line distance is computed to both edges originating on that vertex and the result with the smaller distance is used.

Region	Condition based on $\mathbf{s}_1$	Strategy for the closest points
0	$s_0 \geq 0 \wedge s_1 \geq 0 \wedge s_0 + s_1 < 1$	$\mathbf{p}_T = \mathbf{v}_{t0} + (\mathbf{e}_{t0} \ \mathbf{e}_{t1}) \mathbf{s}_1$ $\mathbf{p}_L = \mathbf{v}_{l0} + t\mathbf{e}_l$
1	$s_0 < 0 \wedge s_1 < 0$	$\min_d \left( \begin{array}{l} \text{LL}(\mathbf{v}_{t0}, \mathbf{v}_{t1}, \mathbf{v}_{l0}, \mathbf{v}_{l1}) \\ \text{LL}(\mathbf{v}_{t0}, \mathbf{v}_{t2}, \mathbf{v}_{l0}, \mathbf{v}_{l1}) \end{array} \right)$
2	$s_0 \geq 0 \wedge s_1 < 0 \wedge s_0 + s_1 < 1$	$\text{LL}(\mathbf{v}_{t0}, \mathbf{v}_{t1}, \mathbf{v}_{l0}, \mathbf{v}_{l1})$
3	$s_1 < 0 \wedge s_0 + s_1 \geq 1$	$\min_d \left( \begin{array}{l} \text{LL}(\mathbf{v}_{t1}, \mathbf{v}_{t0}, \mathbf{v}_{l0}, \mathbf{v}_{l1}) \\ \text{LL}(\mathbf{v}_{t1}, \mathbf{v}_{t2}, \mathbf{v}_{l0}, \mathbf{v}_{l1}) \end{array} \right)$
4	$s_0 \geq 0 \wedge s_1 \geq 0 \wedge s_0 + s_1 \geq 1$	$\text{LL}(\mathbf{v}_{t1}, \mathbf{v}_{t2}, \mathbf{v}_{l0}, \mathbf{v}_{l1})$
5	$s_0 < 0 \quad \wedge \quad s_0 + s_1 \geq 1$	$\min_d \left( \begin{array}{l} \text{LL}(\mathbf{v}_{t2}, \mathbf{v}_{t0}, \mathbf{v}_{l0}, \mathbf{v}_{l1}) \\ \text{LL}(\mathbf{v}_{t2}, \mathbf{v}_{t1}, \mathbf{v}_{l0}, \mathbf{v}_{l1}) \end{array} \right)$
6	$s_0 < 0 \wedge s_1 \geq 0 \wedge s_0 + s_1 < 1$	$\text{LL}(\mathbf{v}_{t0}, \mathbf{v}_{t2}, \mathbf{v}_{l0}, \mathbf{v}_{l1})$

Now, if also candidate  $\mathbf{s}_1$  is located in the feasible region of the triangle, the line-segment intersects the triangle and the distance is zero. Since the shortest connection vector  $\mathbf{c}^* = \mathbf{0}$ , the closest points  $\mathbf{p}_{\text{SSV},i}$  on the SSV surfaces are not uniquely defined (cf. (6.11)).<sup>11</sup>

If, on the other hand,  $t_0$  from (6.46) is outside its bounds we set  $t_1 = \text{sat}(t_0, 0, 1)$  to saturate  $t$  on  $[0, 1]$ . Now, (6.21) from page 99 to calculate the distance between point

$$\mathbf{p} = \begin{cases} \mathbf{v}_{l0} & \text{for } t_1 = 0 \\ \mathbf{v}_{l1} & \text{for } t_1 = 1 \end{cases} \quad (6.48)$$

and plane  $\mathcal{P}(\mathcal{T})$  is applied to obtain  $\mathbf{s}_1$ .

By studying  $\mathbf{s}_1$ , seven independent regions about the triangle must be considered. Table 6.7 summarizes strategies to obtain the closest points  $\mathbf{p}_L$  and  $\mathbf{p}_T$  on the line and triangle respectively. The function denoted by “LL” is the line-line distance function from Section 6.4.4. For regions 1, 3 and 5, which spread out from the vertices of the

<sup>11</sup> Note that this problem exists also for all other SSV-based distance computations. For SSVs with radii of  $r_i > 0$  their volumes overlap—and collision is detected—before intersection between the primitive geometry occurs. In the context of robot control, however, a safety policy to switch of the robot becomes active. Therefore, the problem of intersecting primitives is only of theoretical interest.

triangle, the line-line distance is computed to both edges originating on that vertex and the result giving the smaller distance is used.

The presented approach can be summarized as follows:

---

Initialize  $t_0$  from (6.46)

$$\text{Step 1} \quad t_1 = \begin{cases} 0 & \text{for } t_0 < 0 \\ 1 & \text{for } t_0 > 1 \\ t_0 & \text{otherwise} \end{cases} \Rightarrow \mathbf{s}_1 \text{ from (6.21) using } \mathbf{p} = \mathbf{v}_{l0} + t_1 \mathbf{e}_l$$

$$\Rightarrow \mathbf{s}_1 \text{ from (6.47)}$$

Step 2 Finally, apply strategy from Table 6.7 using  $\mathbf{s}_1$  to identify the region

---

**Parallel Line and Triangle** Several strategies to cope with all possible cases arising for the parallel case can be developed. By e. g., applying the point-triangle distance calculation using the end points of the line-segment, seven cases for both points are to be considered resulting in a total of  $7^2 = 49$  cases. Since the amount of cases becomes quite large, an exhaustive search approach is quite efficient in this case, i. e., by calculating the distances between all possible edge to edge combinations (three in this case) and using the result with the shortest distance. Hence, the line-segment to line-segment distance calculation from Section 6.4.4 is used.

**Unified Approach** A unified approach, to cope with the parallel and non-parallel case at the same time, is to calculate the edge-edge and point-triangle distances and use the result giving the shortest distance.

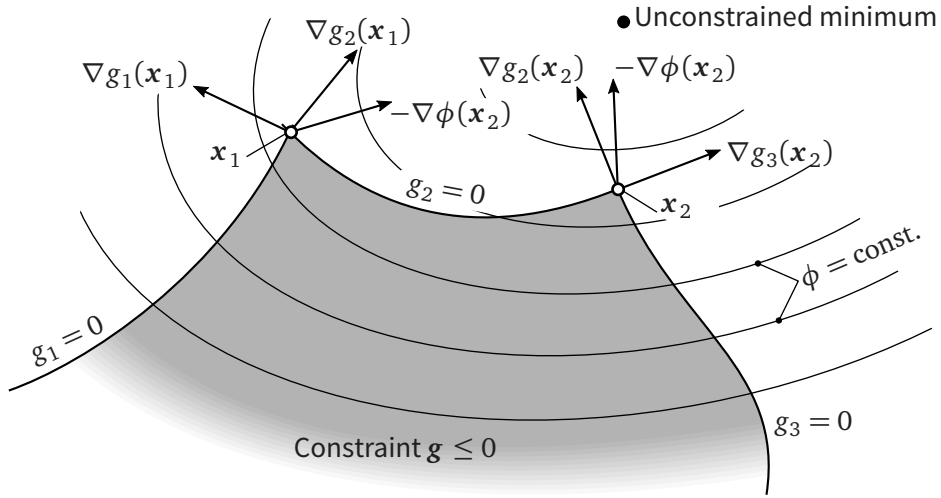
### 6.4.6 Triangle to Triangle Distance Computation

Since the combinational space of cases to be considered for the triangle-triangle distance computation is very large, the functions derived so far can be applied to generate an efficient algorithm. One approach is to calculate all nine possible edge-edge and six vertex-triangle combinations and use the result which gained the shortest distance.

### 6.4.7 General Framework of Minimization Using Inequality Constraints

The problem of finding the closest points between two manifolds can always be defined as an optimization problem in the form of (6.3). In this section, a more formal approach based on a minimization with inequality constraints is developed. The goal is to have a general framework for distance computation which is usable for both development and verification.





**Figure 6.9:** Geometric illustration of the Karush-Kuhn-Tucker optimality conditions with the gradients on two points  $x_1$  and  $x_2$ .

For this purpose, the manifolds  $\mathcal{S}$  are defined more rigorous in the form of

$$\mathcal{S}(\mathbf{x}) = \{ \mathbf{x} \in X : g_i(\mathbf{x}) \leq 0 \text{ for } i = 1, \dots, m \}, \quad (6.49)$$

where the inequality constraints  $g_i(\mathbf{x}) : \mathbb{R}^n \rightarrow \mathbb{R}$  for  $i = 1, \dots, m$  specify the feasible region and  $X$  is an open set in  $\mathbb{R}^n$ .

Strictly speaking, the mathematical representation of the manifold depends on a finite set of parameters  $\mathbf{x}$ . The manifold class plane  $\mathcal{P} := \mathcal{T}(\mathbf{x})$ , for example depends on the parameters  $\mathbf{x} = (s_0, s_1)^\top$  (cf. (6.5), Section 6.3.3). Additional inequality constraints  $\mathbf{g}(\mathbf{x}) \leq \mathbf{0}$  define the boundary of the manifold, e. g., the sub-manifold triangle  $\mathcal{T}$  is obtained from plane  $\mathcal{P}$  using (6.6) to  $\mathbf{g}(\mathbf{x}) = (-s_0, -s_1, s_0 + s_1 - 1)^\top \leq \mathbf{0}$ .

Problem (6.8) —which was the basis for all our derivations so far— is now cast as an optimization problem with inequality constraints. We define the objective function

$$\phi(\mathbf{x}) := \frac{1}{2} \mathbf{c}^\top \mathbf{c} \quad \text{with} \quad \mathbf{c} := S_1(\mathbf{x}_1) - S_0(\mathbf{x}_0). \quad (6.50)$$

Here,  $\mathbf{x} = (\mathbf{x}_0, \mathbf{x}_1)^\top$  is the parameter vector combining the parameter set from both manifolds  $\mathcal{S}_i(\mathbf{x}_i)$ . The inequality constraint problem (ICP) is now written as:

$$\begin{aligned} & \text{minimize} && \phi(\mathbf{x}) \\ & \text{subject to} && \mathbf{g}(\mathbf{x}) \leq \mathbf{0}. \end{aligned} \quad (6.51)$$

Using the method of Lagrange multipliers, we obtain the Lagrange function:

$$L(\mathbf{x}, \boldsymbol{\mu}) = \phi(\mathbf{x}) + \boldsymbol{\mu}^\top \mathbf{g}(\mathbf{x}), \quad (6.52)$$

where  $\boldsymbol{\mu} = (\mu_1, \dots, \mu_r)^\top$  is a Lagrange multiplier vector. By following the Karush-Kuhn-Tucker (KKT) necessary conditions for a local minimum,  $\mathbf{x}^*$  is a local minimum of above problem (6.51) if

$$\begin{aligned} \nabla_{\mathbf{x}} L(\mathbf{x}^*, \boldsymbol{\mu}^*) &= 0 \\ \mu_j^* &\geq 0, \quad j \in \{j \mid g_j(\mathbf{x}^*) = 0\}, \\ \mu_j^* &= 0, \quad j \in \{j \mid g_j(\mathbf{x}^*) < 0\}. \end{aligned} \tag{6.53}$$

This effectively transforms the inequality constraints into equality constraints for which sophisticated solver strategies exist (cf. one of the many textbooks on optimization e. g., [13, 17, 20, 105]).

Figure 6.9 illustrates the Karush-Kuhn-Tucker optimality conditions on two points  $\mathbf{x}_1$  and  $\mathbf{x}_2$ . Evaluation of (6.53) at  $\mathbf{x}_1$  suggests that it is not a local minimum, due to  $\mu_1 \nabla g_1(\mathbf{x}_1) < 0$ . Point  $\mathbf{x}_2$ , however, satisfies the KKT criterion, which is obvious from the figure: vector  $-\nabla \phi$  is located between  $\nabla g_2$  and  $\nabla g_3$  and hence,  $\boldsymbol{\mu} \geq \mathbf{0}$  holds.

In the field of optimization several algorithms have been developed which are able to solve ICP (6.51). Depending on the properties of the manifolds and their boundaries different algorithms must be applied. For e. g., a non-convex problem the KKT criterions may fail and a global search of the minimum is required. In most cases, however, ICP (6.51) can be solved by a sequence of quadratic programming problems (SQP). Many modern numerical software packages, such as MATLAB,<sup>12</sup> SCILAB,<sup>13</sup> MATHEMATICA<sup>14</sup> and MAPLE,<sup>15</sup> are able to solve SQPs quite efficiently. Table 6.8 summarizes algorithms applicable to different classes of manifolds and inequality constraints.

Since above ICP, in general, can't be solved directly, an iterative approach must be chosen. Starting from an initial guess  $\{\mathbf{x}_0, \boldsymbol{\mu}_0\}$  a *feasible search direction*  $\mathbf{d}$  is calculated to obtain  $\mathbf{x}_i = \mathbf{x}_{i-1} + \mathbf{d}_i$  such that  $\phi(\mathbf{x}_i)$  decreases in each iteration  $i$  until  $\{\mathbf{x}^*, \boldsymbol{\mu}^*\}$  is reached. In order to test for convergence, the KKT conditions are applied in each iteration  $i$ . Finding a feasible direction  $\mathbf{d}_i$  leads to a complementary problem, which is solved in each iteration. In [159] an optimization based approach to calculate the minimal distance between polyhedra is presented. Therein, the feasible direction  $\mathbf{d}$  is calculated by considering geometry.

However, for simple geometries, such as points, line segments or triangles, geometrical considerations and vector calculus is usually enough to derive a minimal distance algorithm. On the other hand, in order to obtain an efficient algorithm, sometimes

<sup>12</sup> MATLAB is a commercial numerical computing environment, developed by THE MATHWORKS, INC. (<http://www.mathworks.com/>)

<sup>13</sup> SCILAB is an open source software for numerical computation (<http://www.scilab.org/>)

<sup>14</sup> MATHEMATICA is a commercial product for technical computing, developed by WOLFRAM RESEARCH (<http://www.wolfram.com/mathematica/>)

<sup>15</sup> MAPLE is a commercial computer algebra system, developed by MAPLESOFT (<http://www.maplesoft.com/products/maple/>)

**Table 6.8:** Classification of the resulting optimization problem depending on the type of manifold and inequality constraints.

Manifold Function	Inequality Constraint	Solver Strategy
linear, convex	linear, convex	active set algorithms, line search, SQP
linear, convex	nonlinear, convex	active set algorithms, line-search, SQP
nonlinear	nonlinear	global minimization

many different algorithms must be explored or combined, which may also subject to tuning code for a specific computer platform.

Basically, the algorithms presented in Sections 6.4.2 . . . 6.4.6 are solvers to a specific optimization problem. Therein, the complementary problems are solved in an implicit manner and the number of “iterations” is fixed (e. g., two for the two-step algorithm of the line-line distance computation). Numerical experiments conducted with different optimization software packages show good convergence behavior. However, the general framework of optimization usually takes many iterations to find the minimal distance. Furthermore, the number of iterations is not predictable, which limits the usability in a real-time scenario. On the other hand, for some combinations of even simple shapes no analytic distance function is known and numerical solutions are the only option left [151].

## 6.5 Implementation Details and Run-Time Performance

Since the distance calculation is performed in real-time on the on-board PC of Lola, the implementation must be computationally efficient. Therefore, vector and matrix calculations are implemented using the SIMD<sup>16</sup> units of the CPU. Table 6.9 compares the run-time of the implemented distance computation functions between the three types of SSV. The measurements were performed on the on-board PC under the 32-bit<sup>17</sup> real-time OS QNX 6.4.1 on an Intel Core2 Duo T7600 mobile CPU running at 2.33 GHz. Depending on the level of detail the distance computation takes 50 – 500  $\mu$ s for the whole collision model (see Figure 6.15) which makes the integration in real-time control possible.

It is obvious that the complexity of the primitive also influences the computational effort of the distance calculation, i. e., the relative run-time  $t_{rel,i} := t_{abs,i}/t_{abs,PP}$  for TSS-TSS is over 10 times higher compared to PSS-PSS. Therefore, a good trade off between geometric representation and computational efficiency should be found.

<sup>16</sup> Single Instruction Multiple Data: an efficient way of performing one instruction on more than one number.

<sup>17</sup> The run-time improves significantly in 64-bit mode. However, currently no 64-bit QNX version for x64 hardware is available.

**Table 6.9:** Run-time comparison for the implemented distance calculation functions. The numbers are obtained from Lola’s on-board PC.

Distance calculation function <sup>a</sup>	abs. run-time ( $t_{\text{abs}}$ ) [ $\mu\text{s}$ ]	stdev( $t_{\text{abs}}$ ) [ns]	rel. run-time ( $t_{\text{rel}}$ ) [–]	stdev( $t_{\text{rel}}$ ) [–]
PP	0.0886	0.136	1.00	0.000
PL, LP	0.1256	0.155	1.42	0.001
LL	0.1555	0.624	1.76	0.007
PT, TP	0.1392	0.821	1.57	0.010
LT, LT	0.3614	1.519	4.08	0.015
TT	0.8887	0.300	10.03	0.015

<sup>a</sup> Point-, Line- and Triangle-swept-sphere volume, e. g., PL = distance computation function from a Point-SSV to a Line-SSV

Moreover, run-time performance is greatly influenced by the computer platform and compiler (cf. Section 2.10 for a discussion of the topic). Run-time experiments conducted on more recent Intel-CPU-based 64-bit computer systems using the GNU/Linux OS show results which are by an order of magnitude faster compared to Table 6.9. However, relative run-times are larger on Linux.

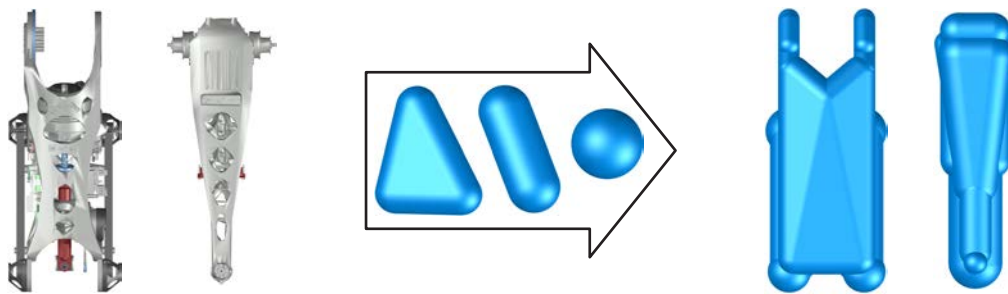
## 6.6 Modeling of the Robot Segments

### 6.6.1 Compounds of SSVs as Robot Segments

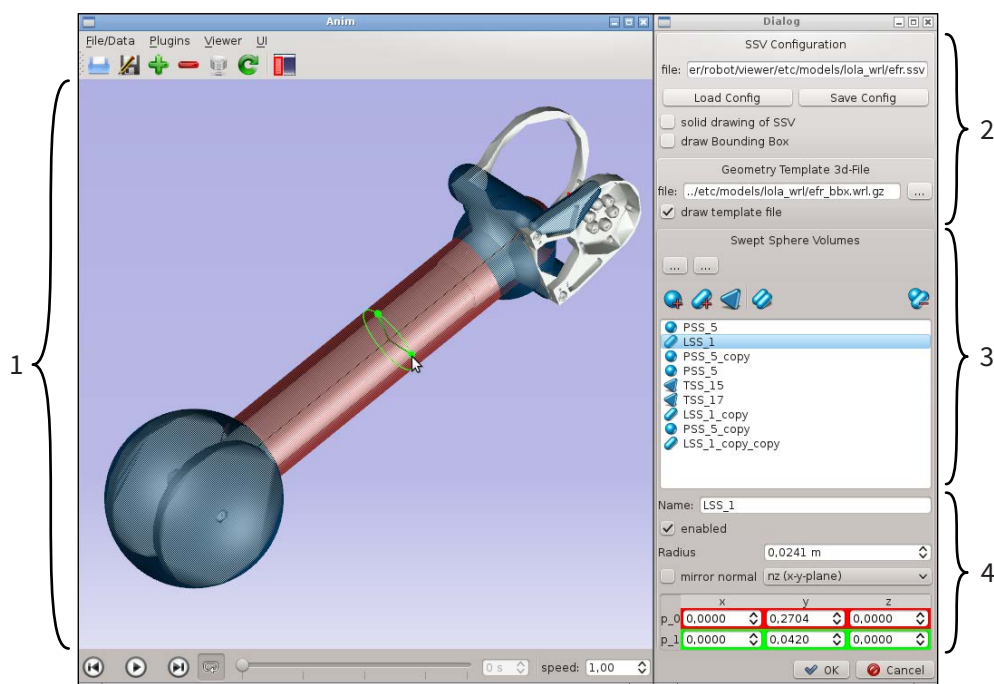
The complex shape of the robot’s segments is simplified by a set of SSVs. Using triangulated high-resolution CAD models of the segments as templates, the segments are remodeled manually. Figure 6.10 illustrates the manual remodeling process for the upper and lower leg. Non-convex simplified shapes of the segments are created with this method. The SSV representations are then used to calculate the closest points and distances between a pair of segments by computing the minimal distances between each SSV from both sets. Since several segments of the robot are symmetric, only one version has to be modeled while the opposite version is gained by affine transformations. This greatly reduces the manual modeling effort.

### 6.6.2 A Versatile Modeling Tool

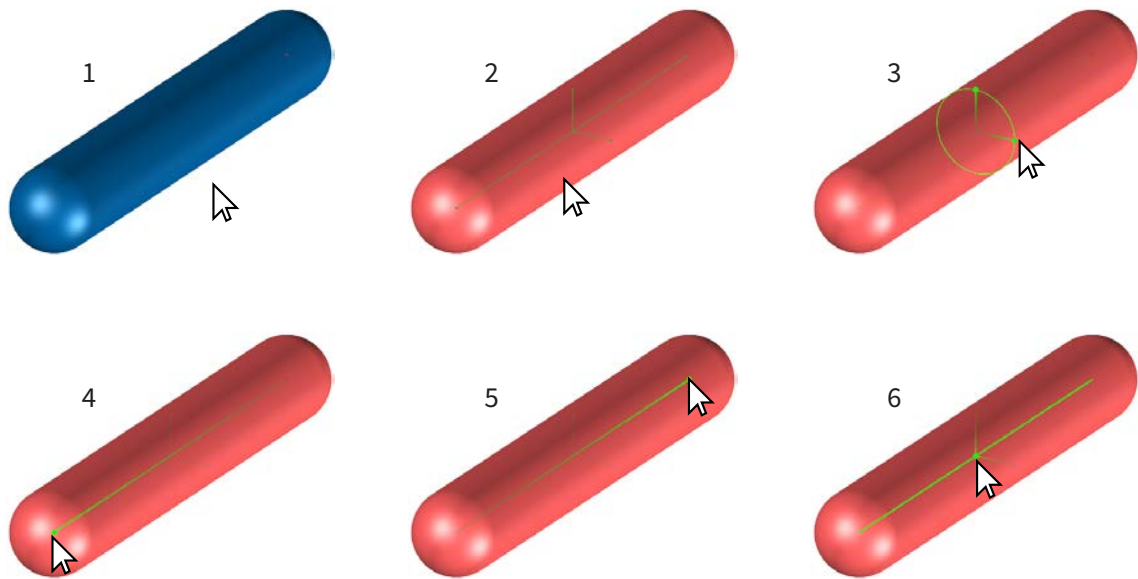
In order to remodel the complex CAD geometry using SSVs a versatile, simple to use modeling tool is developed. The tool is designed as plug-in for the simulation result viewer application. The viewer was developed in cooperation with THOMAS BUSCHMANN and provides software interfaces for animation and visualization usable



**Figure 6.10:** The complex CAD-geometry (upper and lower leg shown on the left) is remodeled using swept-sphere-volumes (center) enabling an efficient distance computation (adopted from [125]).



**Figure 6.11:** The SSV modeling tool (dialog on the right) integrated into the simulation results viewer ANIM, shown on the left. 1) interactive 3-D-viewport; 2) file handling and drawing appearance; 3) tools to create, copy, delete and select SSVs; 4) SSV class specific properties, like name, radius, vertex coordinates and mirror options.



**Figure 6.12:** Interactive and intuitive modeling of the SSVs by moving the handles for (3) radius, (4-5) vertices and (6) SSV using the mouse pointer. The visual appearance changes in selection mode (1 → 2). Handles are highlighted when hovering over their location (3-6).

by the plug-ins. This modular approach simplifies the development of small, self-contained and problem-specific plug-ins for all kinds of graphical visualizations. Other plugins include animation and visualization of results gathered during simulation, planning or experiments, as well as a real-time visualization of the robot's state. The interactive 3-D-viewport is based on OpenGL<sup>18</sup> and libQGLviewer<sup>19</sup> and provides basic functions for panning, zooming, object selection and manipulation, exporting still images, movies and static 3-D scenes [28].

Figure 6.11 shows a snapshot of the viewing application with the SSV modeling plug-in loaded. The tool provides functionality to:

- Load and save SSV compound files.<sup>20</sup>
- Load and render tessellated 3-D CAD-geometry files in VRML97/WRL<sup>21</sup> file format used as template for modeling.

18 OpenGL is a platform-independent API for developing 2-D and 3-D graphics applications (<http://www.opengl.org>)

19 libQGLviewer is a C++ library based on Qt that eases the creation of OpenGL 3-D viewers (<http://www.libqglviewer.com>)

20 The simple and compact libconfig C/C++ library is used to process text-based configuration files (<http://www.hyperrealm.com/libconfig/>).

21 Virtual Reality Modeling Language is a file format for 3-D models standardized as ISO/IEC 14772 (<http://www.web3d.org/x3d/specifications/vrml/ISO-IEC-14772-VRML97>). The loader developed in this work implements a relevant subset of the full ISO standard to load WRL-files exported from CATIA V5. CATIA was used to design Lola.

- Render and control the visualization of SSVs.
- Create, manipulate, copy and delete SSVs.
- Intuitively manipulate SSVs via handles in the 3-D-viewport using the mouse.
- Exploit symmetries within the segment by mirroring individual SSVs about axis-planes.

The intuitive concept for the handle-based modeling is illustrated in Figure 6.12. Visual appearance changes when a SSV is selected (2) and manipulation handles are enabled. Handle indicators are highlighted (3-6) when the mouse pointer approaches their location in the 3-D-viewport. Additional numerical entry fields in the configuration dialog of the plug-in enable a precise positioning of vertices and selection of radius values (see Figure 6.11-4).

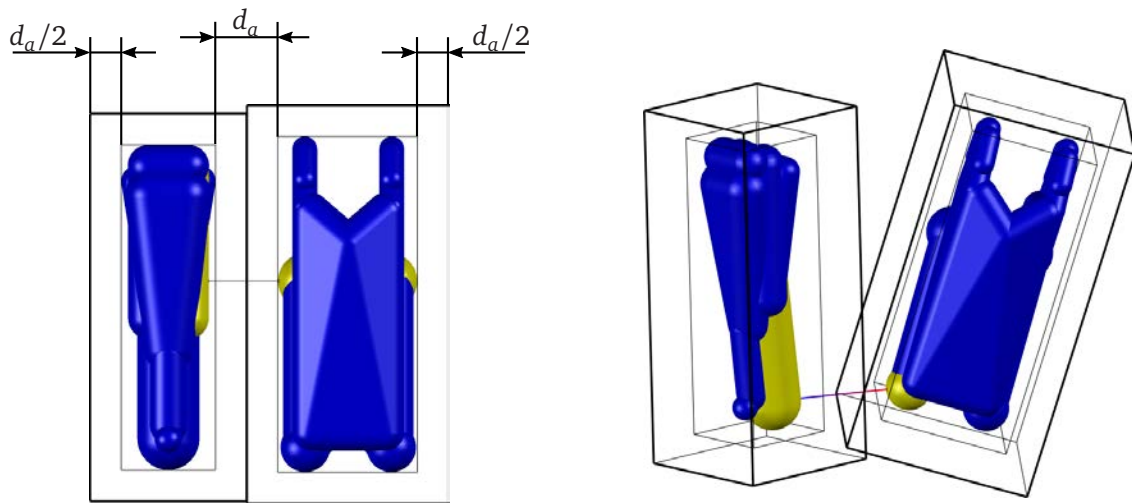
## 6.7 Integration of Bounding Boxes

Computing the distance between SSV segments having  $m$  and  $n$  SSV objects is of  $\mathcal{O}(mn)$  algorithmic complexity, since it is an exhaustive search algorithm. Hence, pruning distance calculations between sufficiently separated segments by a computationally cheap pre-selection can reduce costs significantly. Common *collision detection* libraries use space partitioning methods or bounding volume hierarchies to reject most of the time-consuming collision checks between complex geometries. For *distance computation*, however, these approaches can't be adopted efficiently and reliably [132]. On the other hand, methods like GJK can't be used either, since the SSV representation, in general, has a non-convex shape. In [117] a bounding sphere based hierarchical approach is presented for proximity queries.

A feasible approach, however, is to generate a bounding box (BB) for each segment enlarged by a tolerance and test for box-intersection prior to distance computation. In order to reliably compute distances below a predefined threshold value, or tolerance,  $d_a$ , the BB must be enlarged by  $d_a/2$ . Since collision avoidance becomes active only if segments are closer than a predefined threshold distance, i. e.,  $d_a$ , this is a viable approach. A similar approach is used in the collision detection libraries SWIFT and SWIFT++<sup>22</sup> [44].

Axis-aligned bounding boxes (AABB) are computationally efficient, since overlap testing reduces to one-dimensional problems in each axis direction. However, AABBs must be refitted after each configuration change. More importantly, the volume for potential overlap might increase substantially leading to more false positive intersection tests and distances are calculated unnecessarily [109]. On the other hand, oriented

<sup>22</sup> Swift++ is publicly available: <http://gamma.cs.unc.edu/SWIFT++>



**Figure 6.13:** Tight oriented bounding boxes are enlarged by  $d_a/2$  to reliably compute distances below a threshold distance  $d_a$  when intersection occurs.

bounding boxes (OBB) usually have a tighter fit to the wrapped geometry, but intersection tests are more complex [109]. Since the segments of the robot are relatively close to each other during walking motion, OBBs are implemented to increase chances of non-intersecting BBs. Figure 6.13 illustrates the approach of enlarging each OBB by  $d_a/2$  to reliably compute distances below  $d_a$ .

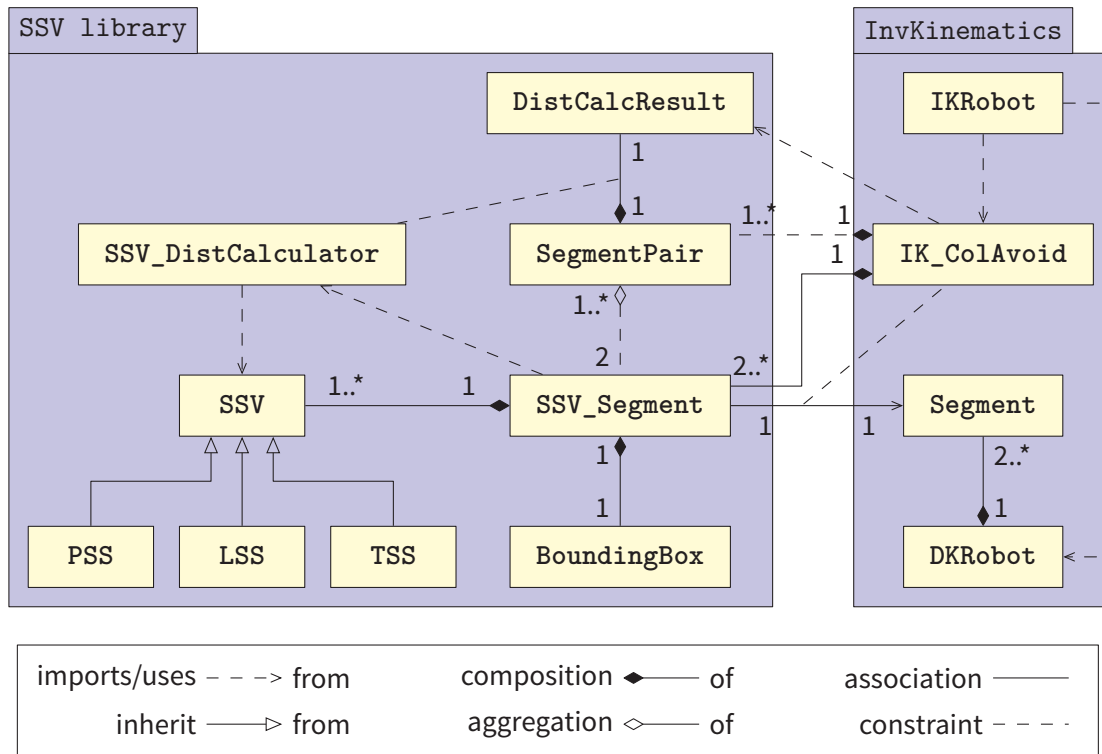
The computational cost for OBB intersection testing is about 2 to 3 times higher than line-line distance computation in Table 6.9. Therefore, this approach is well suited for more complex SSV-models. Depending on the threshold  $d_a$ , however, the state of intersection might not be left, i. e.,  $d_a$  is too large, and the approach is useless.

## 6.8 System Overview

The SSV library is implemented using the object oriented programming language C++. Figure 6.14 shows the structure and integration of the framework as a UML-diagram.<sup>23</sup> The basic framework consists of three classes of SSVs: PSS, LSS and TSS (point, line and triangle SSV, respectively) which can be used to model an SSV\_Segment (cf. Section 6.6.1). Each segment is therefore composed of several SSVs and one BoundingBox which surrounds the SSVs (cf. Section 6.7). A SegmentPair is an aggregation of two SSV\_Segments. A segment-pair defines between which segments distances are to be calculated. The main difference between *aggregation*, denoted by “◁—”, and *composition*, denoted by “◁—”, is that objects in a composition are created and destructed by the composition, and hence, do not exist without the composition.

<sup>23</sup> Unified Modeling Language (<http://www.uml.org/>) is developed and standardized by the Object Management Group (<http://www.omg.org>)

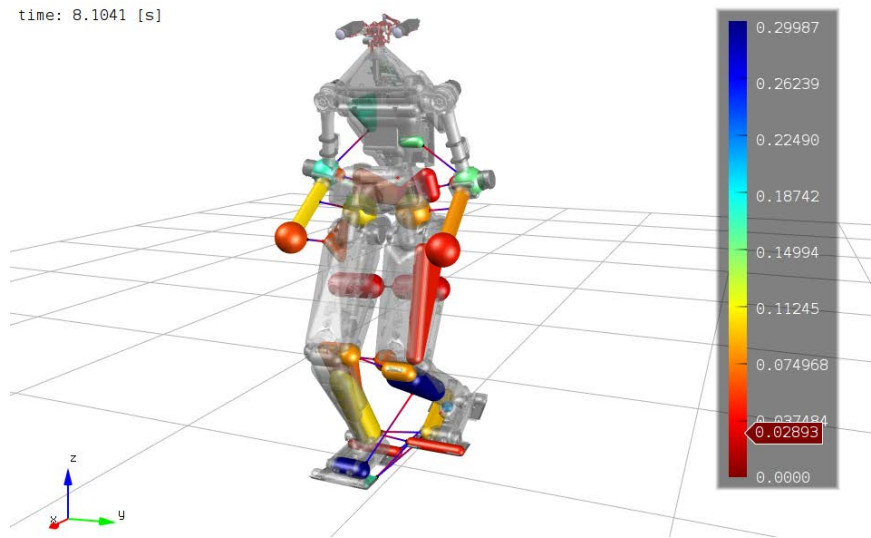




**Figure 6.14:** UML class diagram of the SSV library framework implementation and its integration into the inverse kinematics.

The inverse kinematics module implements a collision avoidance scheme denoted by `IK_ColAvoid` which is described in Chapter 5. During initialization, individual SSV-segments are created and linked to their direct kinematics counterpart `Segments`. A `DKRobot` object is composed of multiple segments and provides a mapping between joint positions to link positions and orientations. If collision between two segments is likely to happen, segment-pairs are predefined manually. Since collision between e. g., head and toes are impossible, such pairs can be excluded which reduces computational cost. SSV-segments are loaded from files produced by the modeling tool described in Section 6.6.2. This approach makes it easy to define multiple SSV versions of a segment enabling optimization. Therefore, if only a subset of the segment can collide in a segment-pair, a specialized version of the SSV-segment containing only this subset is built for inclusion in this segment-pair. E.g., since upper legs will only collide on the inside and arms collide only with the outside part of the upper legs, either the outside or the inside geometry is irrelevant for a segment-pair. Hence, the thigh is modeled in two versions each applied to a different segment-pair. Such model-reduction decreases the computational effort in real-time operation substantially.<sup>24</sup>

<sup>24</sup> The concept could even be extended by defining groups of SSVs inside a SSV-segment and only assign specific groups to a segment-pair. This approach reduces the number of files needed.



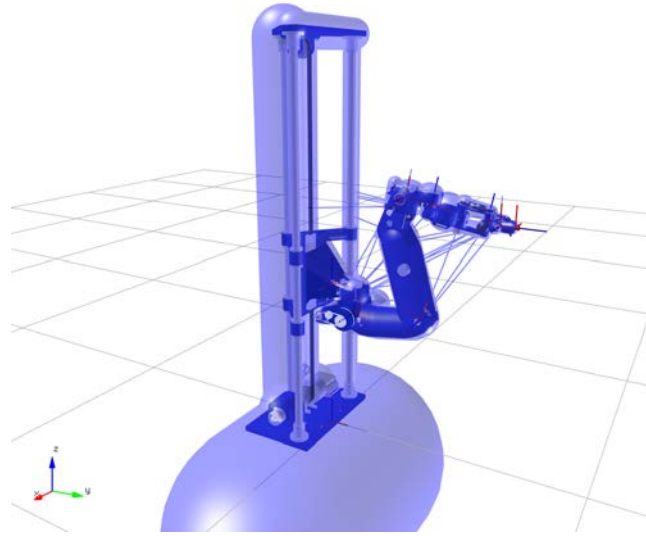
**Figure 6.15:** Visualization showing the collision robot model with the underlying CAD-geometry. The collision model is based on Swept-Sphere-Volumes (SSV). The closest SSVs within a segment are drawn opaque and the color indicates the distance. The colorbar on the right ranges from zero to the largest gap. The red indicator is dynamically changing and displays the smallest gap.

During real-time operation, the inverse kinematics module calls the collision avoidance module which updates SSV-segment positions and orientations from direct kinematics. Each SSV-segment holds a constant non-transformed state of the SSVs, which is then transformed to a common FoR, where distance computation is executed. Currently, the planning FoR of the inverse kinematics module is used. The `SSV_DistCalculator` contains all functions for SSV distance computation presented in this chapter. The `SSV_DistCalculator` is used by the SSV-segments to find the shortest connecting vector between a segment-pair. Each segment-pair then holds a distance result saved in a `DistCalcResult` structure. Collision avoidance then works based on these results (cf. Chapter 5).

Figure 6.15 presents a simulation snapshot showing the collision model of Lola while walking forwards.

The SSV library is used also in other current robotics projects at the Institute of Applied Mechanics. Figure 6.16 shows the collision avoidance model of a robot manipulator with nine DoFs [11, 12]. The manipulator is developed as a research platform for fruit harvesting and spraying within the research project CROPS<sup>25</sup> funded by the European Commission. The programming integration work took only a few hours, which demonstrates the flexibility of the framework.

<sup>25</sup> Clever Robots for Crops (<http://www.crops-robots.eu>)



**Figure 6.16:** The CROPS robot manipulation arm uses SSV models for collision detection and avoidance. The lines represent shortest distances between segment-pairs.

## 6.9 Chapter Summary

This chapter gave an overview of the framework for computing the minimal distance between the robot's segments. Since distances are used in real-time for collision avoidance, only a very short amount of time is permissible for distance calculation. Hence, all distance computations are designed for efficiency. The concept of Swept-Sphere-Volumes (SSVs) is adopted to model the segments of the robot. In this work, three classes of SSVs are implemented to enable flexibility in the modeling process and efficient computation. Equations and strategies for distance computation between all classes are derived. Furthermore, a general optimization based approach using inequality constraints to calculate the minimal distances between arbitrary manifolds is presented. The method can be used both as development tool and for implementation verification.

Since the complex outer shape of Lola's segments can be greatly simplified for collision avoidance, the presented modeling approach works surprisingly well. An intuitive 3-D program is developed for fast creation of SSV-representation of the robot segments by using triangulated high-resolution CAD-meshes as templates. The program provides a simple to use graphical user interface with an interactive 3-D-viewport where new SSVs can be added, shaped and positioned with the mouse. Additionally, symmetries within the link as well as between its left and right version can easily be exploited. This greatly reduces the manual modeling effort.

Finally, the framework integration into the collision avoidance module, along with an application in another current robotics research project is presented. This shows the flexibility of the framework.



# 7 Use of Angular Momentum in Walking Control

## 7.1 Introduction

In order to enable real-time execution, current real-time walking pattern generators (WPG) employ simplified dynamics models of the robot. A classical approach is to reduce the dynamics of the whole robot to a point mass. This “inverted pendulum” model of the robot can further be enhanced by using additional point masses for the feet (see [136] for a detailed review). The model is typically used to compute stable trajectories for the robot’s CoG and zero-moment-point for several steps in advance, which are then tracked during walking. However, mainly due to the differences between the simple mass model and the real robot, the walking pattern must further be stabilized using sensor feedback. Methods to compensate for these dynamics errors exist [29, 136].

Since the mass model used in planning neglects the angular momentum of the robot—and therefore, its full dynamics—several researchers proposed to incorporate the angular momentum into walking control to obtain improvements [64, 71, 72, 125]. Especially for running robots the conservation of angular momentum during flight phases is important [135, 137].

In this chapter, two different methods which incorporate the angular momentum into walking control are presented. The methods are designed to work in combination with the WPG algorithm presented by BUSCHMANN [28]. Therein, the angular momentum of the robot is not considered and, therefore, implicitly assumed to be zero. The methods proposed here, are embedded in the inverse kinematics framework of Lola and enable better tracking of the planned trajectories. Due to the integration into inverse kinematics, it is not confined to BUSCHMANN’s WPG but can be adapted to any kinematically redundant walking machine.

Furthermore, a simple method to plan vertical angular momentum trajectories is presented in this chapter.

## 7.2 Angular Momentum

The overall angular momentum of a rigid-body system about its CoG is given by

$$\mathbf{L}^{\text{CoG, sys}} = \sum_i^{n_{\text{Bodies}}} \mathbf{I}_i^{\text{C}} \boldsymbol{\omega}_i + m_i \mathbf{r}_i \times \dot{\mathbf{r}}_i \in \mathbb{R}^3. \quad (7.1)$$

Here  $\boldsymbol{\omega}_i$  is the angular velocity and  $I_i^C$  is the inertia matrix with respect to the CoG of the  $i^{\text{th}}$  body. Vectors  $\mathbf{r}_i$  and  $\dot{\mathbf{r}}_i$  denote position and velocity of the body CoG relative to the system's CoG, respectively.

Since tree-structured mechanisms, such as a biped walking robot, can be described using generalized coordinates  $\mathbf{q}$ , we can rewrite the right hand side of (7.1) as:

$$\mathbf{L}^{\text{CoG, sys}} = \mathbf{J}_L \dot{\mathbf{q}}, \quad (7.2)$$

where  $\mathbf{J}_L := \partial \mathbf{L}^{\text{CoG, sys}} / \partial \dot{\mathbf{q}} \in \mathbb{R}^{3 \times n_q}$  is the angular momentum Jacobian. Therefore, the angular momentum is *linear* to generalized velocities of the mechanism.

## 7.3 Angular Momentum Compensation via Null-Space Motion

With the collision avoidance presented in Chapter 5 it is possible to remove specific tasks from task-space vector  $\mathbf{x}$  which is used in the inverse kinematics. Rewriting the solution of the *resolved motion rate control* for *redundant robots* given in (4.12) we have

$$\dot{\mathbf{q}} = \mathbf{J}_w^\# \dot{\mathbf{x}} - \mathbf{N} \mathbf{W}^{-1} \mathbf{y}. \quad (7.3)$$

Here,  $\mathbf{y}$  is an arbitrary vector, which is projected via matrix  $\mathbf{N}$  into the null-space of task  $\mathbf{x}$ . Usually,  $\mathbf{y}$  consists of the gradient to an optimization criterion  $H$  (see Section 4.3.1 for more details).

Especially at higher walking speeds, fast leg motion produces fast changes in total angular momentum. The resulting vertical contact torques can lead to foot-ground-contact slippage, which in turn can destabilize the robot. A common approach (also observed in human beings) is to use the arms to reduce this effect. However, in general this approach is limited by physical constraints, mainly due to the mass ratio between legs and arms as well as joint limits.

Lola has no batteries on-board and the arms are quite lightweight, since they have only three drives and an end-mass. Therefore, the overall mass and inertia of the upper body is low compared to other biped robots. Consequently, it is not possible to completely compensate the influence of the heavier legs on the angular momentum over a full gait cycle.

### 7.3.1 Reference Method

In order to compare the performance of the newly developed method, a reference scheme is briefly described. The method was developed by BUSCHMANN to control Lola and is described in [28].

The center of gravity (CoG) of the arms is controlled relative to the robot CoG. Reference values are calculated from CoG position of the opposite leg relative to the robot CoG. Since the arms are shorter than the legs, the compensating motion is scaled down accordingly. This approach is implemented in task-space of the inverse kinematics and works efficiently. Because of its task-space implementation, however, singularities occur close to workspace boundaries. This has proved to be an issue, especially for larger step lengths.

### 7.3.2 Proposed Method

This section describes an extended algorithm to a method previously presented by the author in [125]. The method is extended to set a reference angular momentum about the vertical axis. For this approach the arm CoG positions are removed from task space description  $\mathbf{x}$ .

The main idea of the method is to split the robot DoFs into two groups: DoFs which produce and DoFs which reduce angular momentum, denoted by  $\mathbf{q}_p$  and  $\mathbf{q}_r$  respectively. Here, motion of the unconstrained DoFs is used to cancel the vertical angular momentum produced by the remaining DoFs, i. e.,  $\mathbf{q}_r := \mathbf{q}_{\text{free}}$ . The scalar equation for the angular momentum with respect to the robot CoG about the  $z$ -axis then becomes

$$L_{d,z}^{\text{CoG}} = L_{p,z}^{\text{CoG}} - L_{r,z}^{\text{CoG}}, \quad (7.4)$$

where  $L_{d,z}^{\text{CoG}}$  is a desired reference angular momentum. Again, indices 'p' and 'r' denote producing and reducing DoFs, respectively. For the sake of readability reference point 'CoG' is omitted in the following. Using (7.2), (7.4) is rewritten as:

$$L_{d,z} = \mathbf{J}_{L,p,z} \dot{\mathbf{q}}_p - \mathbf{J}_{L,r,z} \dot{\mathbf{q}}_r. \quad (7.5)$$

The optimization problem

$$\begin{aligned} & \text{minimize} \quad \|\dot{\mathbf{q}}_r\|^2 \\ & \text{subject to} \quad \mathbf{J}_{L,p,z} \dot{\mathbf{q}}_p - \mathbf{J}_{L,r,z} \dot{\mathbf{q}}_r - L_{d,z} = 0, \end{aligned} \quad (7.6)$$

is solved to obtain minimal  $\dot{\mathbf{q}}_r$  which gives

$$\dot{\mathbf{q}}_r = \mathbf{J}_{L,r,z}^{\#} (\mathbf{J}_{L,p,z} \dot{\mathbf{q}}_p - L_{d,z}), \quad (7.7)$$

with the generalized inverse of a vector  $\mathbf{a}^{\#} := \mathbf{a}^T / \|\mathbf{a}\|^2$ . The resulting  $\dot{\mathbf{q}}_r$  lead to strange arm motions when executed directly. Therefore, arm movement is removed

from the task description  $\mathbf{x}$  and added to the null-space of (7.3) instead:

$$\mathbf{y} = \alpha \left( \frac{\partial H}{\partial \mathbf{q}} \right)^\top + \dot{\mathbf{q}}_r, \quad (7.8)$$

where  $\alpha$  is a gain and  $\partial H / \partial \mathbf{q}$  contains other optimization criteria to account for joint limit avoidance, posture drift compensation and collision avoidance (cf. Chapters 4 and 5).

On the other hand, due to physical constraints, we typically have  $L_p - L_d > L_r$  for Lola. Hence, it is not possible to satisfy the equality constraint in (7.6) at all times which leads to a singularity and  $\dot{\mathbf{q}}_r \rightarrow \infty$ . Therefore, solution (7.7) is replaced by introducing an adequate normalization which proved to be stable:

$$\dot{\mathbf{q}}_r = \frac{1}{\|J_{L,r,z}\|} J_{L,r,z}^\# (J_{L,p,z} \dot{\mathbf{q}}_p - L_{d,z}). \quad (7.9)$$

This relatively simple approach works surprisingly well when combined with the presented joint limit avoidance and self-collision avoidance in the null-space of the inverse kinematics. Results from both simulation and experiments are shown in the following sections.

### 7.3.3 Simulations

The simulations are conducted with the multibody system (MBS) simulation framework presented in Chapter 2. Figure 7.1 compares the resulting vertical angular momentum  $L$  and its rate of change  $\dot{L}$  for the reference method and the proposed method. The effective angular momentum peaks are reduced by approximately 20% and those for  $\dot{L}$  by approximately the same amount. Especially the proposed approach proved to be effective when walking with larger step lengths.

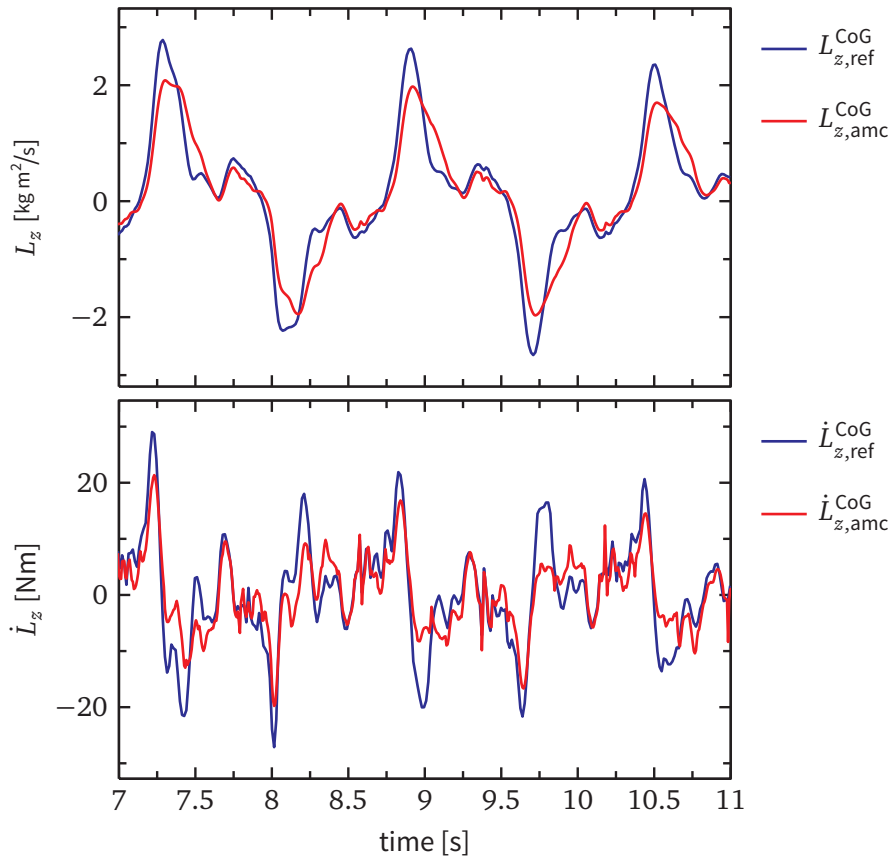
### 7.3.4 Experiments

This section shows experimental results with two examples showing both the effectiveness of the collision avoidance and the angular momentum compensation with the arms during walking.

#### Walking Forwards

The angular momentum about the CoG of the robot cannot be measured directly and calculation is difficult due to unknown or uncertain parameters. However, angular momentum compensation about the vertical axis should effectively reduce the reaction torque about the  $z$ -axis,  $\tau_z$ . Therefore,  $\tau_z$  is used to evaluate effectiveness of the proposed compensation. In order to obtain comparable results, Lola executes a



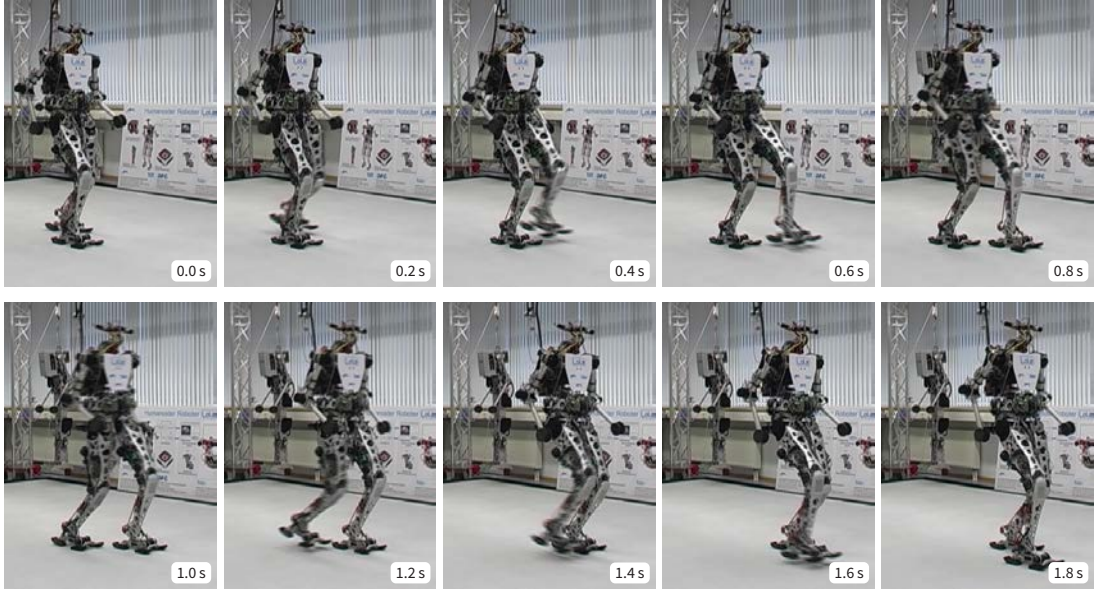


**Figure 7.1:** Vertical angular momentum  $L_z$  (top) and its rate of change  $\dot{L}_z$  (bottom) evaluated from MBS about the center of gravity in simulation. Indices ‘ref’ and ‘amc’ denote the results with the arm motion from the reference implementation and the proposed method respectively (step length: 0.4 m, step time: 0.8 s).

pre-defined step sequence. Figure 7.2 shows a frame sequence from a video of an experiment. The sequence consists of a full step cycle. Figure 7.3 shows the measured torque  $\tau_z$  during this gait cycle for the proposed method and the reference implementation.

## 7.4 Angular Momentum Trajectory

When incorporating the angular momentum into walking control a fundamental problem is to find an appropriate reference angular momentum trajectory  $L_{\text{ref}}(t)$ . KAJITA et al. develop an offline walking controller based on the total momentum of the robot [72]. Similar to the proposed method described above only the vertical component of the angular momentum is considered. The vertical angular momentum reference is set to zero to let the robot HRP-2 walk [72] and run forwards [71].



**Figure 7.2:** Video frame sequence when walking forwards with the proposed angular momentum tracking combined with collision avoidance.

TAJIMA et al. adopt the method to develop a running controller for the Toyota Partner Robot [135]. The angular momentum reference is calculated in planning stage from the linear inverted pendulum model of the robot CoG. In order to avoid unusual stepping motion, it is assumed that the robot is solid and has a diagonal inertia matrix. The angular momentum reference about the sagittal axis is then obtained from planned CoG velocities [135].

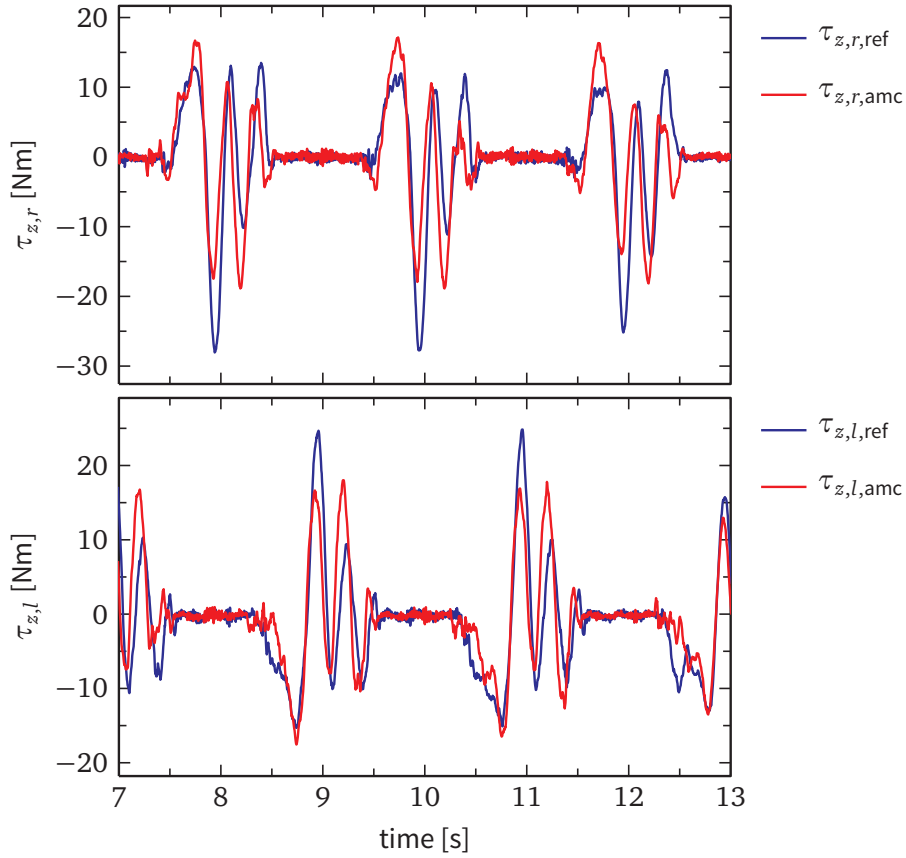
In the proposed approach presented above a simple cubic function is used for the angular momentum reference trajectory  $L_{\text{ref},z}$  for the duration of the current step  $T_{\text{step}}$ :

$$L_{\text{ref},z}(t) = \sum_{i=0}^3 a_i (t - t_b)^i \quad \text{for } t \in [t_b, t_e], \quad T_{\text{step}} = t_e - t_b. \quad (7.10)$$

Here, times  $t_b$  and  $t_e$  denote the beginning and ending times of a step, respectively. Coefficients  $a_i$  are calculated from boundary values of  $L_{\text{ref},z}$  and its rate of change  $\dot{L}_{\text{ref},z}$ , respectively. Thereby, current values for  $L_{\text{ref},z}$  and  $\dot{L}_{\text{ref},z}$  are used as initial values. Moreover, boundary end values are defined by:

$$L_{\text{ref},z}(t_e) := I_{z,z}^{\text{CoG}} \dot{\varphi}_u(t_e) \quad \text{and} \quad \dot{L}_{\text{ref},z}(t_e) := 0. \quad (7.11)$$

Here,  $I_{z,z}^{\text{CoG}} := I_{z,z}^{\text{CoG}}(\mathbf{q}_0)$  is the constant vertical mass moment of inertia of the whole robot, calculated from an initial configuration  $\mathbf{q}_0$ , e. g., while standing still before



**Figure 7.3:** Vertical contact torques for the right (top) and left (bottom) foot when walking forwards with 1.4 km/h in experiment. Indices ‘ref’ and ‘amc’ denote the results with the arm motion from the reference implementation and the proposed method respectively (step length: 0.4 m, step time: 1 s).

walking. The vertical rotation rate  $\dot{\varphi}_u$  of the upper body is approximated by

$$\dot{\varphi}_u(t_e) \approx \Delta\varphi_u / T_{\text{step}}, \quad (7.12)$$

where  $\Delta\varphi_u$  is the relative cornering angle of the upper body during one step and input to the walking controller. Effectively, this approach sets the reference value for the vertical angular momentum to zero while walking forwards ( $\Delta\varphi_u = 0$  in (7.12)). When cornering, however, a rigid robot is assumed which achieves a constant rotation rate  $\dot{\varphi}_u$  at the end of each step. This relatively simple approach works surprisingly well when combined with the walking controller presented by BUSCHMANN in [28].

## 7.5 Angular Momentum Minimization

Another approach to integrate the angular momentum into walking control, previously presented by the author in [126] and developed in cooperation with THOMAS BUSCHMANN, is presented in this section. The main idea here is to reduce the foot-ground slippage torque  $\tau_z$  by minimizing the angular momentum  $L_z$ , since its rate of change is  $\dot{L}_z \propto \tau_z$ .

By combining problem (4.11) and property (7.2) we obtain the optimization problem which fits the *resolved motion rate control method* [154] extended with the framework of redundancy resolution proposed by LIÉGEOIS [81] (cf. (4.11), pp. 73)

$$\begin{aligned} & \text{minimize} \quad \frac{1}{2} \dot{\mathbf{q}}^\top \underbrace{(\delta \mathbf{E} + \mathbf{J}_{L,z}^\top \mathbf{J}_{L,z})}_{\mathbf{W} \in \mathbb{R}^{n \times n}} \dot{\mathbf{q}} + \alpha_N (\nabla_{\mathbf{q}} H)^\top \dot{\mathbf{q}} \\ & \text{subject to} \quad \dot{\mathbf{x}} - \mathbf{J}_x \dot{\mathbf{q}} = \mathbf{0} . \end{aligned} \quad (7.13)$$

Since the weighting matrix  $\mathbf{W}$  must be invertible and  $(\mathbf{J}_{L,z}^\top \mathbf{J}_{L,z})$  is rank deficient, we add an additional term  $\delta \mathbf{E}$ . The variable  $\delta$  is a positive constant and  $\mathbf{E}$  is the unit matrix.

Analogously to (4.12) on page 73 solving (7.13) for  $\dot{\mathbf{q}}$  gives

$$\dot{\mathbf{q}} = \mathbf{J}_{x,W}^\# \dot{\mathbf{x}} - \alpha_N \mathbf{N} \mathbf{W}^{-1} \left( \nabla_{\mathbf{q}} H(\mathbf{q}) \right)^\top ,$$

with  $\mathbf{N} := \mathbf{E} - \mathbf{J}_{x,W}^\# \mathbf{J}_x$  is the null-space projection matrix of the  $\mathbf{W}$ -weighted generalized inverse  $\mathbf{J}_{x,W}^\# := \mathbf{W}^{-1} \mathbf{J}_x^\top \left( \mathbf{J}_x \mathbf{W}^{-1} \mathbf{J}_x^\top \right)^{-1}$ .

Since  $\mathbf{W}$  is dense, computing its inverse is not trivial. However, the special structure of the weighting matrix  $\mathbf{W}$  can be seen as a rank-1 update of a matrix  $\mathbf{A} = \delta \mathbf{E}$  with the dyadic product  $\mathbf{u} \mathbf{v}^\top = \mathbf{J}^\top \mathbf{J}$ . The inversion of  $\mathbf{W}$  then fits the *Sherman-Morrison-Woodbury identity* [65]:

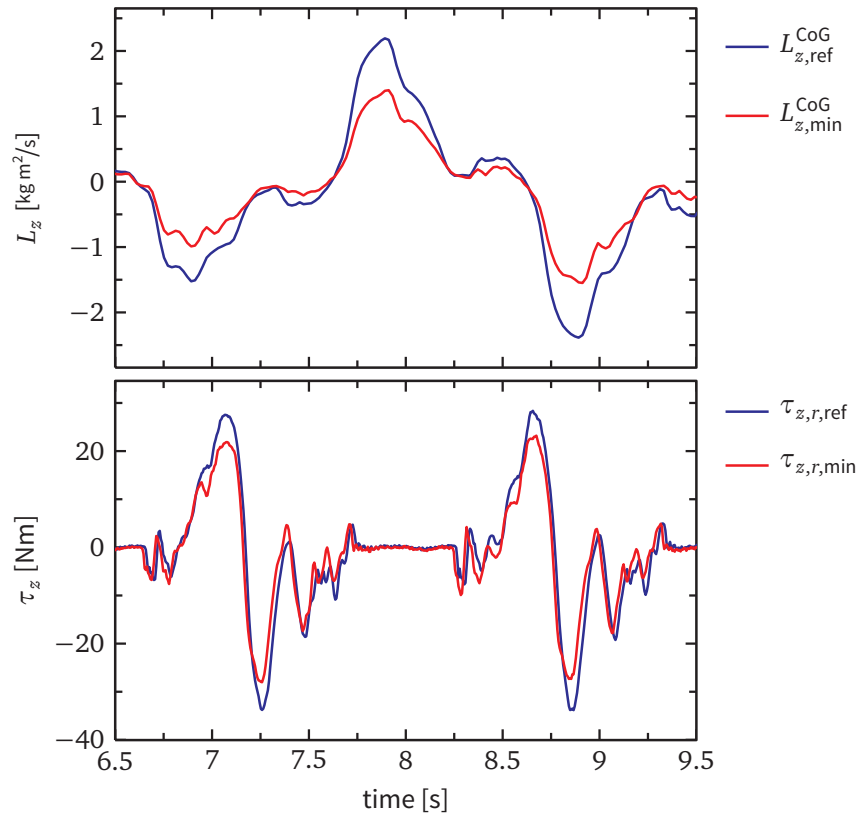
$$(\mathbf{A} + \mathbf{u} \mathbf{v}^\top)^{-1} = \mathbf{A}^{-1} - \frac{\mathbf{A}^{-1} \mathbf{u} \mathbf{v}^\top \mathbf{A}^{-1}}{1 + \mathbf{v}^\top \mathbf{A}^{-1} \mathbf{u}} . \quad (7.14)$$

Substitution of  $\mathbf{u} \mathbf{v}^\top = \mathbf{J}^\top \mathbf{J}$  and the trivial inverse of  $\mathbf{A}^{-1} = (\delta \mathbf{E})^{-1} = (1/\delta) \mathbf{E}$  into (7.14) and simplification leads to

$$\mathbf{W}^{-1} = \frac{1}{\delta} \left( \mathbf{E} - \frac{\mathbf{J}_{L,z}^\top \mathbf{J}_{L,z}}{\delta + \mathbf{J}_{L,z}^\top \mathbf{J}_{L,z}} \right) . \quad (7.15)$$

Therefore,  $\mathbf{W}^{-1}$  can be computed quite efficiently.<sup>1</sup>

<sup>1</sup> Thanks to Prof. dr. ir. DANIEL RIXEN for pointing out the relation between  $\mathbf{W}^{-1}$  and (7.14).



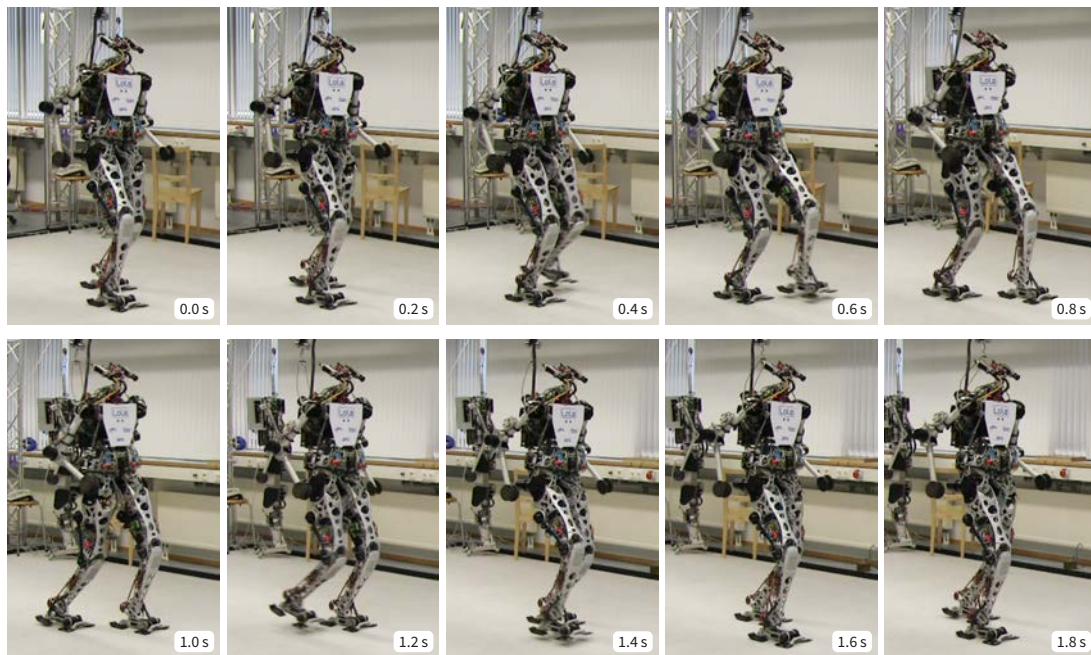
**Figure 7.4:** Simulation results comparing the vertical angular momentum about the CoG (above) and the vertical slippage torque acting on the right foot in experiment (below) of a reference solution (blue line; no arm motion) and the proposed method (red line).

## Experiments

The plots in Figure 7.4 compare the proposed method with a reference solution (i.e.  $\mathbf{W} = \mathbf{E}$ ). The vertical angular momentum about the CoG is obtained in simulation. The slippage torque  $\tau_z$  is measured from the force-torque sensor in the foot during experiment. Figure 7.5 was taken during the experiment using the proposed method (walking parameters: step time: 0.8 s, step length: 0.4 m). It can clearly be seen, that not only the angular momentum is reduced about 40 % but also the slippage torque peaks are lower by approximately 15 %.

Although, the method works in principle, the exact selection of parameter  $\delta$  is subject to tuning. In the experiments shown here we chose  $\delta = 0.5$ . The weighting of the joint velocities can also degrade the motion capabilities in general. Moreover, the collision avoidance in null-space is not guaranteed to work in all situations. The reason for this seems the dense weighting matrix  $\mathbf{W}$ , which also changes the relative weighting.

Overall, the method presented in Section 7.3 works more efficiently.



**Figure 7.5:** Video frame sequence when walking forwards with the proposed angular momentum minimization scheme combined with collision avoidance

## 7.6 Chapter Summary

This chapter deals with the important aspect of angular momentum in walking control. Two methods are presented.

The first method explicitly splits the robot into joints which *produce* and joints which can be used to *track* a desired reference. The former are typically important for the task, i. e., walk and maintain balance, while the latter are part of the null-space.

The second method is the output from a local optimization problem which attempts to minimize the joint velocities as well as their contribution to the total vertical angular momentum.

For both methods simulations and experiments are shown in this chapter. Especially the first method works effectively to track a desired angular momentum reference and produces natural looking arm movements at the same time.

The problem of choosing an adequate reference angular momentum which should be tracked during walking is discussed and a simple, general approach to generate a reference trajectory is presented.

Due to physical constraints (ratios between masses and between inertias of the robot segments and joint constraints) of the robot and the difficult nature of the angular momentum, a real tracking of a reference is not possible with Lola.

## 8 Conclusion and Outlook

Humanoid robots have great potential to operate in cluttered environments. In this scenario, bipedal locomotion is considered one of the core technologies, since wheel-based locomotion is often unfeasible. A brief summary and discussion of key ideas and contributions is given in the following section. The final section provides suggestions for future research based on the methods presented in the preceding chapters.

### 8.1 Summary

This thesis covered three aspects important to biped robots. In the first part, an efficient algorithm for the simulation of multibody systems (MBS) with small kinematic loops is presented. The modeling of mechatronic systems typically introduces (small) kinematic loops, since drive mechanisms are modeled as rigid elements. Rigid drive mechanisms reduce both the number of degrees of freedom (DoF) and the effort of time integration simultaneously (cf. Introduction). A straightforward implementation for solving the equations of motion (EoM) of MBS leads to an asymptotic algorithmic complexity of  $\mathcal{O}(n^3)$ . Specifically for tree-structured MBS,  $\mathcal{O}(n)$ -algorithms have been developed in the past decades. Since the algorithmic complexity grows only linearly with the number of DoFs, these algorithms are superior to  $\mathcal{O}(n^3)$ -algorithms even for a moderate number of DoFs.

In this work, a novel  $\mathcal{O}(n)$ -algorithm based on sub-systems is derived which is able to cope with kinematic loops. Sub-systems are only used to group loop-bodies and solve kinematic loops in a closed form. Hence, sub-systems can be interpreted as meta-bodies which effectively restore the tree-topology of the MBS. Consequently, this approach enables a unified application of the  $\mathcal{O}(n)$ -algorithm.

In order to reduce the modeling effort, an automatic conversion of existing MBS with kinematic loops into sub-systems is presented. Sub-systems are instantiated only where loops are detected. This reduces the computational overhead from sub-systems to a minimum. From a practical point of view, this conversion process is key to facilitate a unified application of both  $\mathcal{O}(n^3)$  and  $\mathcal{O}(n)$ -algorithm.

The proposed method is implemented in the framework for simulating biped robots developed by BUSCHMANN [28]. The original implementation uses an  $\mathcal{O}(n^3)$ -method already optimized for tree-like structures. Especially for small kinematic loops —found in mechatronic systems, such as Lola— the proposed algorithm works quite efficiently. The computational efficiency is also shown in a comparative study applied to different test systems.

The second part of this thesis presents approaches for self-collision avoidance and angular momentum tracking for biped robots. Both methods exploit the kinematic redundancy of Lola with respect to the chosen task space description. The framework proposed by LIÉGEOIS is adopted for local optimization of redundancy.

Self-collision avoidance works by minimizing a repulsive collision cost function in the null-space of the inverse kinematics. The cost function uses the *closest points* on two segments of the robot and applies a *distance-based repulsive law* to keep the segments separated. A new formulation for the repulsive law is developed which works effectively for both fast and slow motions. Because of the *local* optimization, a general *global* minimization of the cost function, and therefore, global collision free motion, is not guaranteed. However, simulations and experiments with Lola have shown the effectiveness of the system for a wide range of conditions.

To facilitate real-time execution of the proposed self-collision avoidance, a framework to compute the closest points between two segments efficiently is developed. The method adopts the concept of *swept-sphere-volumes* (SSV) to remodel the segments of the robot. Distance computation between three classes of SSVs is derived. The use of *triangle SSVs* is proposed which allows for a more versatile geometric modeling process. In addition, a general methodology based on optimization with inequality constraints is derived for effective distance computation for arbitrary shapes.

Due to run-time restrictions for real-time execution, the evaluation of a full dynamics model of the robot along with inverse kinematics during planning stage is not possible with current computer systems. Typically, reduced robot models which approximate the system dynamics using point masses are used for this purpose. As a consequence, angular momentum of the full model is neglected. To satisfy this approximation during trajectory tracking, two methods to incorporate the angular momentum into walking control are developed. Both methods work cooperatively with the proposed self-collision avoidance.

The first method explicitly splits the robot's joints into two groups where one group is used for tracking a vertical angular momentum reference trajectory. Moreover, an efficient method to generate feasible vertical angular momentum reference trajectories is presented. The second method solves a local optimization problem which minimizes both the joint velocities and their contribution to the total vertical angular momentum of the robot.

For both methods simulations and experiments are presented. Especially the first method works effectively to track a desired angular momentum reference. Interestingly, the kinematic redundancy chosen for these methods improves walking performance in general and produces more natural looking arm movements at the same time.

Due to physical constraints of the robot and the difficult nature of the angular momentum, a real tracking of a reference is not possible with Lola.



## 8.2 Recommendations for Future Work

Based on the experience gained during this thesis, several suggestions for future research in the field can be made. Especially future generations of more powerful computer systems could meet the requirements for the application of most methods developed in this work into real-time planning of biped walking robots.

Efficient simulation enables refined dynamics models of the robot used in walking pattern generation. This would reduce errors due to dynamics approximation and hence, lead to increased walking stability. However, the resulting global optimization problem is computationally heavy.

It is predicted that future generations of computer systems enable faster computation only through parallelism. Therefore, branches of the robot model, i. e., branches due to robot limbs, could be exploited to facilitate parallelism in simulation. Moreover, for globally optimal walking pattern generation (see above), computing the inverse kinematics and dynamics simulation in parallel improves run-time execution. This is feasible, since there is typically a delay in communication between gathered sensor data and control commands. Moreover, run-times for inverse kinematics and dynamics computation are almost identical for Lola.

The self-collision avoidance along with the fast distance computation developed in this work, can be extended towards environment obstacle avoidance. An adequate sensor system to gather a sufficiently accurate environment model would be required in this scenario.

Since the presented methods for self-collision avoidance and vertical angular momentum tracking are local methods, a global optimum is typically not achieved. Thus, self-collision avoidance is not guaranteed. Semi-global optimization which predicts system behavior for a certain time interval improves the situation.

On the other hand, hierarchical inverse kinematics could be applied to hierarchize tasks including collision avoidance.

In order to further develop walking pattern generators towards running controllers, tracking of angular momentum seems to be important, as already pointed out by others (cf. [71, 135, 137]). Especially during free-flight phases, conservation of angular momentum is key to control the robot's attitude while running. Ultimately, further research in this field might contribute also to advancements in walking control.

Finally, in order to improve angular momentum tracking, modifications to Lola's hardware would be favorable. Especially a heavier upper body and/or heavier arms could improve tracking significantly.



# Appendix A

## Mathematical Toolbox

### A.1 Notation and Operators

Notations and mathematical operators used in this thesis are defined in this section.

**Notation** In this work scalars, vectors and matrices are denoted by the following type face notation

Type	Denoted by	Example
scalar	normal face type	$x, L \in \mathbb{R}$
vector	bold face type, lower case	$\mathbf{r} \in \mathbb{R}^{3 \times 1}$
matrix	bold face type, upper case	$\mathbf{A} \in \mathbb{R}^{3 \times 3}$

**Transposed Vectors and Matrices** Vectors are always defined as column-vectors, while row-vectors are denoted by the transposed  $(\cdot)^T$  version:

$$\mathbf{a} = \begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix} \in \mathbb{R}^n, \quad \mathbf{a}^T = [a_1 \ a_2 \ \cdots \ a_n] \in \mathbb{R}^{1 \times n},$$

$$\mathbf{A} = \begin{bmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,m} \\ A_{2,1} & A_{2,2} & \cdots & A_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n,1} & A_{n,2} & \cdots & A_{n,m} \end{bmatrix} \in \mathbb{R}^{n \times m}, \quad \mathbf{A}^T = \begin{bmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,m} \\ A_{2,1} & A_{2,2} & \cdots & A_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n,1} & A_{n,2} & \cdots & A_{n,m} \end{bmatrix} \in \mathbb{R}^{m \times n}.$$

**Dot-Product** denoted by “ $\cdot$ ” of two vectors  $\mathbf{a} \in \mathbb{R}^n$  and  $\mathbf{b} \in \mathbb{R}^n$

$$c = \mathbf{a} \cdot \mathbf{b} = \mathbf{a}^T \mathbf{b} := \sum_{i=1}^n a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n, \quad (\text{A.1})$$

where  $c \in \mathbb{R}$  is the resulting scalar value. Furthermore, the form

$$\mathbf{a}^2 := \mathbf{a} \cdot \mathbf{a} \quad (\text{A.2})$$

is used for brevity.

**Cross-Product** of two vectors  $\mathbf{a} \in \mathbb{R}^3$  and  $\mathbf{b} \in \mathbb{R}^3$ :

$$\mathbf{a} \times \mathbf{b} := \begin{bmatrix} a_2 b_3 - a_3 b_2 \\ a_3 b_1 - a_1 b_3 \\ a_1 b_2 - a_2 b_1 \end{bmatrix}. \quad (\text{A.3})$$

**Tilde-Operator** of a vector  $\mathbf{a} \in \mathbb{R}^3$  generates a skew-symmetric matrix in the form

$$\tilde{\mathbf{a}} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}^{\sim} := \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}, \quad (\text{A.4})$$

such that the multiplication with a vector  $\mathbf{b} \in \mathbb{R}^3$

$$\tilde{\mathbf{a}}\mathbf{b} = \mathbf{a} \times \mathbf{b}, \quad (\text{A.5})$$

resembles the cross-product. The tilde operator is often used for brevity.

Moreover, the following useful identities hold [23]

$$\tilde{\mathbf{a}}\mathbf{b} = \tilde{\mathbf{b}}^T \mathbf{a} = -\tilde{\mathbf{b}}\mathbf{a}, \quad (\text{A.6})$$

$$\tilde{\mathbf{a}}\mathbf{a} = \mathbf{a} \times \mathbf{a} = \mathbf{0}, \quad (\text{A.7})$$

$$\tilde{\mathbf{a}}\tilde{\mathbf{b}} = \mathbf{b}\mathbf{a}^T - \mathbf{b}^T \mathbf{a}\mathbf{E}, \quad (\text{A.8})$$

$$\widetilde{\tilde{\mathbf{a}}\mathbf{b}} = \mathbf{b}\mathbf{a}^T - \mathbf{a}\mathbf{b}^T = \tilde{\mathbf{a}}\tilde{\mathbf{b}} - \tilde{\mathbf{b}}\tilde{\mathbf{a}}, \quad (\text{A.9})$$

$$\widetilde{\tilde{\mathbf{a}}\mathbf{b}\mathbf{c}} = \tilde{\mathbf{c}}\tilde{\mathbf{b}}\tilde{\mathbf{a}}, \quad (\text{A.10})$$

$$\tilde{\mathbf{a}}\tilde{\mathbf{a}}\tilde{\mathbf{a}} = -\tilde{\mathbf{a}} \|\mathbf{a}\|^2, \quad (\text{A.11})$$

$$\tilde{\mathbf{a}}\tilde{\mathbf{b}}\tilde{\mathbf{b}}\tilde{\mathbf{a}} = \tilde{\mathbf{b}}\tilde{\mathbf{a}}^T \tilde{\mathbf{a}}\tilde{\mathbf{b}}, \quad (\text{A.12})$$

$$\tilde{\mathbf{a}}\tilde{\mathbf{b}}\mathbf{c} + \tilde{\mathbf{b}}\tilde{\mathbf{c}}\mathbf{a} + \tilde{\mathbf{c}}\tilde{\mathbf{a}}\mathbf{b} = \mathbf{0}. \quad (\text{A.13})$$

**2-Norm** of a vector  $\mathbf{a} \in \mathbb{R}^n$

$$\|\mathbf{a}\|_2 = \|\mathbf{a}\| := \sqrt{\mathbf{a} \cdot \mathbf{a}} = \sqrt{\sum_{i=1}^n a_i^2} = \sqrt{a_1^2 + a_2^2 + \cdots + a_n^2}. \quad (\text{A.14})$$

## A.2 Coordinate Transformations

Rotation matrix  $\mathbf{A}_{KI} \in \mathbb{R}^{3 \times 3}$  is used to transform a vector denoted in frame of reference (FoR)  $I$  into FoR  $K$ . Since rotation matrices are orthonormal, following useful identities

can be exploited:

$$\mathbf{A}_{KI} = \mathbf{A}_{IK}^{-1} = \mathbf{A}_{IK}^T, \quad (\text{A.15})$$

$$\mathbf{A}_{KI} \mathbf{A}_{KI}^T = \mathbf{E}, \quad (\text{A.16})$$

with the unit matrix  $\mathbf{E} \in \mathbb{R}^{3 \times 3}$

**Vectors** vector  ${}_K \mathbf{a} \in \mathbb{R}^3$  is transformed from FoR  $I$  to FoR  $K$  using the transformation matrix  $\mathbf{A}_{KI} \in \mathbb{R}^{3 \times 3}$  to

$${}_K \mathbf{a} = \mathbf{A}_{KI} {}_I \mathbf{a} \quad \text{and} \quad {}_I \mathbf{a} = \mathbf{A}_{KI}^T {}_K \mathbf{a}. \quad (\text{A.17})$$

**Tensor Matrices** tensor matrix  ${}_K \mathbf{T} \in \mathbb{R}^{3 \times 3}$  is transformed from FoR  $K$  to FoR  $I$  using the transformation matrix  $\mathbf{A}_{KI} \in \mathbb{R}^{3 \times 3}$  to

$${}_I \mathbf{T} = \mathbf{A}_{KI}^T {}_K \mathbf{T} \mathbf{A}_{KI}. \quad (\text{A.18})$$

## A.3 Partial Derivatives

In this section,  $s \in \mathbb{R}$  is a scalar,  $\mathbf{a} \in \mathbb{R}^m$  and  $\mathbf{b} \in \mathbb{R}^n$  are vectors of functions and vectors of variables, respectively.

### Derivative of a Vector with Respect to a Scalar

$$\frac{\partial \mathbf{a}}{\partial s} := \begin{bmatrix} \frac{\partial a_1}{\partial s} \\ \frac{\partial a_2}{\partial s} \\ \vdots \\ \frac{\partial a_m}{\partial s} \end{bmatrix}, \quad (\text{A.19})$$

where  $\mathbf{a} = \mathbf{a}(s) : \mathbb{R} \rightarrow \mathbb{R}^m$  is a vector valued function

### Derivative of a Scalar with Respect to a Vector

$$\frac{\partial s}{\partial \mathbf{b}} := \left[ \frac{\partial s}{\partial b_1} \quad \cdots \quad \frac{\partial s}{\partial b_n} \right], \quad (\text{A.20})$$

where  $s = s(\mathbf{b}) : \mathbb{R}^n \rightarrow \mathbb{R}$  is a function of multiple variables  $\mathbf{b} \in \mathbb{R}^n$ .

**Derivative of a Vector with Respect to a Vector – Jacobian Matrix** The derivative of a vector using a vector is obtained by subsequently applying (A.19) and (A.20) yielding

$$\frac{\partial \mathbf{a}}{\partial \mathbf{b}} := \begin{bmatrix} \frac{\partial a_1}{\partial \mathbf{b}} \\ \vdots \\ \frac{\partial a_m}{\partial \mathbf{b}} \end{bmatrix} = \begin{bmatrix} \frac{\partial a_1}{\partial b_1} & \cdots & \frac{\partial a_1}{\partial b_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial a_m}{\partial b_1} & \cdots & \frac{\partial a_m}{\partial b_n} \end{bmatrix}, \quad (\text{A.21})$$

where  $\mathbf{a} = \mathbf{a}(\mathbf{b}) : \mathbb{R}^n \rightarrow \mathbb{R}^m$  is a vector field. The resulting matrix is called *Jacobian matrix* due to CARL GUSTAV JACOB JACOBI.

**Derivative of a Matrix Using a Scalar** Analogously to (A.19), for a matrix  $A(s) \in \mathbb{R}^{m \times n}$  we have

$$\frac{\partial A}{\partial s} := \begin{bmatrix} \frac{\partial A_{1,1}}{\partial s} & \cdots & \frac{\partial A_{1,n}}{\partial s} \\ \vdots & \ddots & \vdots \\ \frac{\partial A_{m,1}}{\partial s} & \cdots & \frac{\partial A_{m,n}}{\partial s} \end{bmatrix}. \quad (\text{A.22})$$

**Gradient Notation** The gradient notation is used for brevity and is equivalent to the partial derivative from (A.21):

$$\nabla_{\mathbf{b}} \mathbf{a} := \frac{\partial \mathbf{a}}{\partial \mathbf{b}}, \quad (\text{A.23})$$

where  $\mathbf{a} = \mathbf{a}(\mathbf{b}) : \mathbb{R}^n \rightarrow \mathbb{R}^m$  for  $n, m \in \mathbb{N}$ .

**Quadratic Equations** The partial derivative of a quadratic form

$$s = \mathbf{x}^T \mathbf{A} \mathbf{x}, \quad \mathbf{A} \in \mathbb{R}^{n \times n}, \quad \mathbf{A} \text{ is symmetric} \quad (\text{A.24})$$

is

$$\frac{\partial}{\partial \mathbf{x}} (\mathbf{x}^T \mathbf{A} \mathbf{x}) = \mathbf{x}^T \mathbf{A} + \mathbf{x}^T \mathbf{A}^T = 2\mathbf{x}^T \mathbf{A} \quad \text{for } \mathbf{A} = \mathbf{A}^T, \quad (\text{A.25})$$

and for the second partial derivative we have

$$\frac{\partial^2}{\partial \mathbf{x} \partial \mathbf{x}^T} (\mathbf{x}^T \mathbf{A} \mathbf{x}) = \frac{\partial}{\partial \mathbf{x}^T} (2\mathbf{x}^T \mathbf{A}) = 2\mathbf{A}. \quad (\text{A.26})$$

## Appendix B

### Pseudo-Code for Distance Calculation Algorithms

This chapter refers to Chapter 6 and contains pseudo-code for some distance calculation algorithms of the swept-sphere-volume library developed in this work.

#### B.1 Line-Segment to Line-Segment

In Algorithms B.1, B.2 and B.3 the pseudo-code for one possible solution to the calculation of the minimum distance between two line segments is shown. A detailed derivation can be found in Section 6.4.4 on page 99.

**Algorithm B.1:** Line to Line Distance calculation algorithm with references to Algorithms B.2 and B.3 below.

---

<b>Input:</b> $r_0, \mathbf{p}_{00}, \mathbf{p}_{01}, r_1, \mathbf{p}_{10}, \mathbf{p}_{11}$	7: <b>else</b> // <i>skewed lines</i>
<b>Output:</b> $d, \mathbf{c}, \mathbf{p}_0, \mathbf{p}_1$	8:   call Alg. B.3 for non-parallel case
1: $\mathbf{u} \leftarrow \mathbf{p}_{01} - \mathbf{p}_{00}$ // <i>precalculations</i>	9: <b>end if</b>
2: $\mathbf{v} \leftarrow \mathbf{p}_{11} - \mathbf{p}_{10}$	10: $\mathbf{p}_{\text{res},0} \leftarrow \mathbf{p}_{00} + s \mathbf{u}$ // <i>final calculations</i>
3: $\mathbf{w} \leftarrow \mathbf{p}_{10} - \mathbf{p}_{00}$	11: $\mathbf{p}_{\text{res},1} \leftarrow \mathbf{p}_{10} + t \mathbf{v}$
4: $N \leftarrow (\mathbf{u}^2)(\mathbf{v}^2) - (\mathbf{u}^T \mathbf{v})^2$	12: $\mathbf{c} \leftarrow \mathbf{p}_{\text{res},1} - \mathbf{p}_{\text{res},0}$
5: <b>if</b> parallel e. g., Eq. (6.34) <b>then</b>	13: $d \leftarrow \ \mathbf{c}\  - r_0 - r_1$
6:   call Alg. B.2 for parallel case	

---

**Algorithm B.2:** Line to Line Distance calculation algorithm — parallel case in Algorithm B.1

---

**Input:**  $u, v, w$   
**Output:**  $s, t$   
 $N \leftarrow 2u^T v$   
**if**  $u^T w < 0$  **then**  
  **if**  $u^T v + u^T w < 0$  **then**  
     $s \leftarrow 0$   
    5:  $t \leftarrow \begin{cases} 0 & \text{if } u^T v < 0 \\ 1 & \text{otherwise} \end{cases}$   
  **else if**  $u^T v + u^T w > u^2$  **then**  
     $s \leftarrow 0.5$   
     $t \leftarrow (u^2 - 2u^T v)/N$   
  **else**  
    10:  $s \leftarrow (v^2 + v^T w)/N$   
     $t \leftarrow (u^T v - u^T w)/N$   
  **end if**  
  **else if**  $u^T w > u^2$  **then**  
    **if**  $u^T v + u^T w < 0$  **then**  
      15:  $t \leftarrow 0.5$   
       $t \leftarrow (u^2 - 2u^T w)/N$   
    **else if**  $u^T v + u^T w > u^2$  **then**  
       $s \leftarrow 1$   
       $t \leftarrow \begin{cases} 1 & \text{if } u^T v < 0 \\ 0 & \text{otherwise} \end{cases}$   
    20: **else**  
       $s \leftarrow (u^T v + v^2 + v^T w)/N$   
       $t \leftarrow (u^T v + u^2 - u^T w)/N$   
    **end if**  
  **else**  
    25: **if**  $u^T v + u^T w < 0$  **then**  
       $s \leftarrow v^T w/N$   
       $t \leftarrow -u^T w/N$   
    **else if**  $u^T v + u^T w > u^2$  **then**  
       $s \leftarrow (u^T v + v^T w)/N$   
      30:  $t \leftarrow (u^2 - u^T w)/N$   
    **else**  
       $s \leftarrow (v^2 + 2v^T w)/N$   
       $t \leftarrow 0.5$   
    **end if**  
  35: **end if**

---

**Algorithm B.3:** Line to Line Distance calculation algorithm — non-parallel case in Algorithm B.1

---

**Input:**  $u, v, w, N$   
**Output:**  $s, t$   
 $t_Z \leftarrow u^T v u^T w - u^2 v^T w$   
 $t_N \leftarrow N$   
**if**  $t_Z < 0$  **then** // 1) saturate  $t$   
   $t_Z \leftarrow 0$   
  5:  $s_Z \leftarrow u^T w$   
   $s_N \leftarrow u^2$   
  **else if**  $t_Z > t_N$  **then**  
     $t_Z \leftarrow t_N$   
     $s_Z \leftarrow u^T v + u^T w$   
  10:  $s_N \leftarrow u^2$   
  **else**  
     $s_Z \leftarrow v^2 u^T w - u^T v v^T w$   
     $s_N \leftarrow N$   
  **end if**  
  15: **if**  $s_Z < 0$  **then** // 2) saturate  $s$   
     $s \leftarrow 0$   
     $t \leftarrow \begin{cases} 0 & -v^T w < 0 \\ 1 & -v^T w > v^2 \\ (-v^T w)/v^2 & \text{otherwise} \end{cases}$   
  **else if**  $s_Z > s_N$  **then**  
     $s \leftarrow 1$   
    20:  $t \leftarrow \begin{cases} 0 & u^T v - v^T w < 0 \\ 1 & u^T v - v^T w > v^2 \\ \frac{u^T v - v^T w}{v^2} & \text{otherwise} \end{cases}$   
  **else**  
     $s \leftarrow s_Z/s_N$   
     $t \leftarrow t_Z/t_N$   
  **end if**

---



## Appendix C

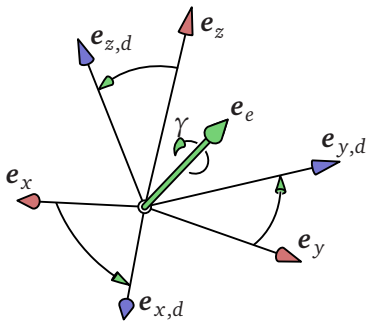
### Inverse Kinematics: Orientation Error in Task Space

For the closed-loop inverse kinematics algorithm in (4.14) on page 74 the position error  $\Delta \mathbf{x}$  at time instant  $t_i$  is required. For *linear positions*, this is simply the difference between desired and current position in task space, i. e.,  $\Delta \mathbf{x}_{\text{pos}}(t_i) = \mathbf{x}_{\text{pos}} - f_{\text{pos}}(\mathbf{q}(t_i))$ . Where  $f_{\text{pos}}(\mathbf{q})$  denotes the task space position gained via forward kinematics from current generalized positions  $\mathbf{q}$ . On the other hand, the *orientation error* of an orientation task depends on the specific orientation definition in task space. Since orientations can be defined in several ways (see also: Section 2.4.4, 6-DoF joints), it is a good idea to calculate  $\Delta \mathbf{x}_{\text{rot}}$  in a unified way from rotation matrices.

Figure C.1 illustrates a feasible approach to calculate  $\Delta \mathbf{x}_{\text{rot}}$  for an orientation task using *equivalent angle-axis representation* [40, 97]. Essentially, the misalignment between two FoRs, is expressed by an axis  $\mathbf{e}_e$  and an angle  $\gamma$ :

$$\Delta \mathbf{x}_{\text{rot}} = \gamma \mathbf{e}_e, \quad (\text{C.1})$$

which are calculated from axis directions of the desired and current orientation, respectively. These axis directions are the column vectors of the corresponding rotation matrices  $\mathbf{A}_d := (\mathbf{e}_{x,d}, \mathbf{e}_{y,d}, \mathbf{e}_{z,d})$  and  $\mathbf{A} := (\mathbf{e}_x, \mathbf{e}_y, \mathbf{e}_z)$  describing the desired and current FoR orientations. Moreover,  $\mathbf{A}_d$  is calculated from task space, i. e., Euler angles, and  $\mathbf{A}$  is gained through forward kinematics from  $\mathbf{q}$ .

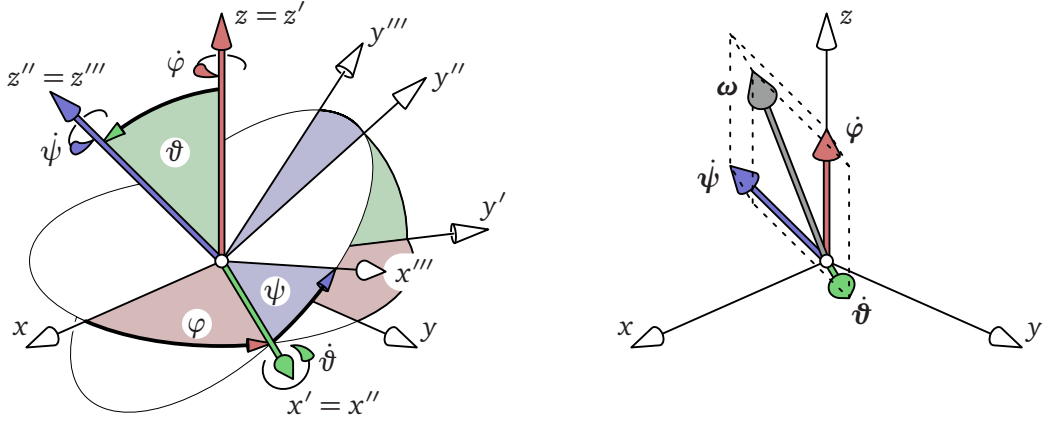


$$\gamma = \arccos \left( \frac{1}{2} \left( \mathbf{e}_x^T \mathbf{e}_{x,d} + \mathbf{e}_y^T \mathbf{e}_{y,d} + \mathbf{e}_z^T \mathbf{e}_{z,d} - 1 \right) \right)$$

$$\mathbf{e}_e = \frac{1}{2 \sin \gamma} \left( \mathbf{e}_x \times \mathbf{e}_{x,d} + \mathbf{e}_y \times \mathbf{e}_{y,d} + \mathbf{e}_z \times \mathbf{e}_{z,d} \right)$$

$$(\mathbf{e}_e = \mathbf{0} \quad \text{for} \quad \gamma = 0)$$

**Figure C.1:** Equivalent angle-axis representation is used to calculate the misalignment between two FoRs.



**Figure C.2:** Euler Z-X-Z angles to rotate FoR  $(x, y, z)$  to FoR  $(x''', y''', z''')$ ; *left*: rotation sequence: angle  $\varphi$  about  $z$ , angle  $\vartheta$  about rotated  $x'$  and angle  $\psi$  about rotated  $z''$ ; *right*: resulting angular velocity vector  $\omega = \dot{\varphi} + \dot{\vartheta} + \dot{\psi}$ , where  $\dot{\varphi}$  points along  $z$ -axis,  $\dot{\vartheta}$  points along  $x'$ -axis and  $\dot{\psi}$  points along  $z''$ -axis.

If a task defines desired orientations by means of three successive, independent elementary rotations, e. g., Euler Z-X-Z angles, the axes of rotation (and hence, also according angular velocities) are not orthogonal to each other. Figure C.2 illustrates this for the Euler Z-X-Z angles, where we have

$$\mathbf{x}_{\text{rot}} = \begin{pmatrix} \varphi \\ \vartheta \\ \psi \end{pmatrix} \quad \text{and} \quad \dot{\mathbf{x}}_{\text{rot}} = \begin{pmatrix} \dot{\varphi} \\ \dot{\vartheta} \\ \dot{\psi} \end{pmatrix}. \quad (\text{C.2})$$

The rotation matrix  $\mathbf{A}_d$  projecting FoR  $d$  into reference FoR  $I$  is written as<sup>1</sup>

$$\mathbf{A}_{Id}(\mathbf{x}_{\text{rot}}) = \begin{pmatrix} c\varphi c\psi - s\varphi c\vartheta s\psi & -c\varphi s\psi - s\varphi c\vartheta c\psi & s\varphi s\vartheta \\ s\varphi c\psi + c\varphi c\vartheta s\psi & -s\varphi s\psi + c\varphi c\vartheta c\psi & -c\varphi s\vartheta \\ s\vartheta s\psi & s\vartheta c\psi & c\vartheta \end{pmatrix}. \quad (\text{C.3})$$

Which can be used to recalculate the angles

$$\psi = \begin{cases} 0 & \text{if } A_{3,1} = 0 \wedge A_{3,2} = 0 \\ \text{atan2}(A_{3,1}, A_{3,2}) & \text{otherwise,} \end{cases} \quad (\text{C.4a})$$

$$\vartheta = \text{atan2}(A_{3,1} s\psi + A_{3,2} c\psi, A_{3,3}), \quad (\text{C.4b})$$

$$\varphi = \text{atan2}(A_{2,1} c\psi - A_{2,2} s\psi, A_{1,1} c\psi - A_{1,2} s\psi). \quad (\text{C.4c})$$

<sup>1</sup>  $s(\cdot) := \sin(\cdot)$  and  $c(\cdot) := \cos(\cdot)$

Since the components of  $\dot{\mathbf{x}}_{\text{rot}}$  are defined along their specific axis of rotation (cf. Figure C.1), a transformation is required in order to add feedback term  $\mathbf{K}_x \Delta \mathbf{x}$ . With reference to Figure C.1, the individual angular velocity vectors are transformed via

$$\boldsymbol{\omega}(\mathbf{x}, \dot{\mathbf{x}}) = \underbrace{\begin{bmatrix} 0 & c\varphi & s\varphi s\vartheta \\ 0 & s\varphi & -c\varphi s\vartheta \\ 1 & 0 & c\vartheta \end{bmatrix}}_{\mathbf{T}(\mathbf{x}_{\text{rot}})} \underbrace{\begin{bmatrix} \dot{\varphi} \\ \dot{\vartheta} \\ \dot{\psi} \end{bmatrix}}_{\dot{\mathbf{x}}_{\text{rot}}}, \quad (\text{C.5})$$

where matrix  $\mathbf{T}$  projects  $\dot{\mathbf{x}}_{\text{rot}}$  into reference FoR  $I$ . Conversely,  $\dot{\mathbf{x}}$  is obtained from  $\boldsymbol{\omega}$  by

$$\begin{bmatrix} \dot{\varphi} \\ \dot{\vartheta} \\ \dot{\psi} \end{bmatrix} = \frac{1}{s\vartheta} \begin{bmatrix} -s\psi c\vartheta & c\psi c\vartheta & s\vartheta \\ c\psi s\vartheta & s\psi s\vartheta & 0 \\ s\psi & -c\psi & 0 \end{bmatrix} \boldsymbol{\omega}, \quad (\text{C.6})$$

which has a singularity for  $\sin \vartheta = 0$ . For other orientation definitions see, e. g., [25].

With (C.5) we can now rewrite the differential kinematic equation (4.6) for a single orientation task to

$$\mathbf{T} \dot{\mathbf{x}}_{\text{rot}} = \mathbf{T} \partial f_{\text{rot}}(\mathbf{q}) / \partial \mathbf{q} \dot{\mathbf{q}}, \quad (\text{C.7})$$

or more general:

$$\boldsymbol{\omega} = \mathbf{J}_R \dot{\mathbf{q}}, \quad (\text{C.8})$$

where  $\mathbf{J}_R := \frac{\partial \boldsymbol{\omega}}{\partial \dot{\mathbf{q}}}$  is the Jacobian matrix of rotation. Note that (C.8) is now independent from the specific definition of the orientation task, since it uses angular velocity  $\boldsymbol{\omega}$ . Therefore, (C.1) and (C.8) fit also to orientation tasks defined by unit quaternions (see [128]), since a rotation matrix  $\mathbf{A}_d$  and desired angular velocity  $\boldsymbol{\omega}$  can be calculated.

**Example** For a task defining positions and an orientation suitable for a manipulator, i. e.,  $\mathbf{x} := (\mathbf{x}_{\text{pos}} \ \mathbf{x}_{\text{rot}})^T$ , the differential kinematic equation (4.6) writes to

$$\underbrace{\begin{bmatrix} \dot{\mathbf{x}}_{\text{pos}} \\ \boldsymbol{\omega} \end{bmatrix}}_{\dot{\mathbf{x}}_{\omega}} = \underbrace{\begin{bmatrix} \mathbf{J}_T(\mathbf{q}) \\ \mathbf{J}_R(\mathbf{q}) \end{bmatrix}}_{\mathbf{J}_{\omega}} \dot{\mathbf{q}}, \quad (\text{C.9})$$

where  $\boldsymbol{\omega}$  is calculated from (C.5) and  $\mathbf{J}_T := \partial f_{\text{pos}}(\mathbf{q}) / \partial \mathbf{q}$  is the Jacobian of translation. Because of the geometric interpretation of the task velocity  $\dot{\mathbf{x}}_{\omega}$  the Jacobian  $\mathbf{J}_{\omega}$  is often referred to as “geometric Jacobian”. An efficient computation of  $\mathbf{J}_{\omega}$  for this manipulator example is presented in [110]. Solving (C.9) for  $\dot{\mathbf{q}}$  (cf. (4.14) on page 74)

leads to

$$\dot{\mathbf{q}}_i = \mathbf{J}_\omega^\# \dot{\hat{\mathbf{x}}}_i, \quad (\text{C.10})$$

$$\dot{\hat{\mathbf{x}}}_i := \dot{\mathbf{x}}_\omega(t_i) + \mathbf{K}_x \Delta \mathbf{x}_i(\mathbf{x}_i, \mathbf{q}_i), \quad (\text{C.11})$$

$$\Delta \mathbf{x}_i := \begin{pmatrix} \Delta \mathbf{x}_{\text{pos}} & \Delta \mathbf{x}_{\text{rot}} \end{pmatrix}^\top. \quad (\text{C.12})$$

**Notes on Trajectory Generation for Orientation Tasks** Defining orientations or more importantly: orientation trajectories<sup>2</sup> is not always intuitive. Therefore, the framework using (C.1) and (C.8) is well suited in order to get a unified base for solving the IK problem.

Moreover, the equivalent angle-axis representation can be adopted for orientation interpolation purposes. The orientation between two FoRs, whose orientations are given by rotation matrices  $\mathbf{A}_0$  and  $\mathbf{A}_1$  valid at time instances  $t_0$  and  $t_1$ , can easily be interpolated using equivalent angle-axis representation and interpolating angle  $\gamma$ . This approach, known as SLERP<sup>3</sup>, is letting points move along circular paths about the axis of rotation. On the other hand, simple interpolation of angles from subsequent elementary rotations, i. e., Euler angles, lead to strange motion, especially around singular points.

---

2 That is, how a FoR should rotate during time.

3 SLERP: spherical linear interpolation

## List of Abbreviations

CAD	Computer Aided Design
CAM	Computer Aided Manufacturing
CLI	Command Line Interface
CoG	Center of Gravity
CoM	Center of Mass
CPU	Central Processing Unit
CROPS	Clever Robots for Crops (research project funded by the European Commission in the 7 <sup>th</sup> Framework Programme)
DAE	Differential Algebraic Equation
DoF	Degree of Freedom; “DoFs” often synonymous for “number of DoFs”
EoM	Equation of Motion
FD	Forward Dynamics
FMI	Functional Mock-up Interface
FMU	Functional Mock-up Unit
FoR	Frame of Reference
GCC	GNU Compiler Collection
GNU	GNU’s Not Unix
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HD	Harmonic Drive (gear)
ID	Inverse Dynamics
IK	Inverse Kinematics
I/O	Input and/or Output
ISO	International Organization for Standardization
MBS	Multibody System
ODE	Ordinary Differential Equation
RAM	Random Access Memory
RSC	Reverse Sparse Cholesky (includes decomposition and/or factorization)
SSV	Swept Sphere Volume



## Bibliography

- [1] ARMSTRONG, W. W. “Recursive solution to the equations of motion of an n-link manipulator”. In: *Proc. 5th World Congress on Theory of Machines and Mechanisms*. Vol. 2. 1979, pp. 1343–1346.
- [2] ASCHER, U. M., CHIN, H., PETZOLD, L. R., and REICH, S. “Stabilization of constrained mechanical systems with DAEs and invariant manifolds”. In: *Journal of Structural Mechanics* **23**, 2 (1995), pp. 135–157.
- [3] *Atlas - The Agile Anthropomorphic Robot*. Boston Dynamics. Oct. 2013. URL: [www.bostondynamics.com/robot\\_Atlas.html](http://www.bostondynamics.com/robot_Atlas.html).
- [4] AYUSAWA, K. and NAKAMURA, Y. “Fast Inverse Kinematics Algorithm for Large DOF System with Decomposed Gradient Computation Based on Recursive Formulation of Equilibrium”. In: *Proc. IEEE Int. Conf. Intell. Robots Syst. (IROS)*. 2012, pp. 3447–3452.
- [5] BADLER, N. I., PHILLIPS, C. B., and WEBBER, B. L. *Simulating Humans: Computer Graphics, Animation, and Control*. Oxford University Press, 1999.
- [6] BAE, D.-S. and HAUG, E. J. “A Recursive Formulation for Constrained Mechanical System Dynamics: Part II. Closed Loop Systems”. In: *Mechanics Based Design of Structures and Machines* **15** (1987), pp. 481–506.
- [7] BAERLOCHER, P. and BOULIC, R. “Task-Priority Formulations for the Kinematic Control of Highly Redundant Articulated Structures”. In: *Proc. IEEE Int. Conf. Intell. Robots Syst. (IROS)*. Vol. 1. IEEE. 1998, pp. 323–329.
- [8] BALAKRISHNAN, R. and RANGANATHAN, K. *A Textbook of Graph Theory*. Springer, 2012. DOI: 10.1007/978-1-4614-4529-6.
- [9] BAUMGARTE, J. “Stabilization of constraints and integrals of motion in dynamical systems”. In: *Computer methods in applied mechanics and engineering* **1**, 1 (1972), pp. 1–16.
- [10] BÄUML, B., BIRBACH, O., WIMBÖCK, T., FRESE, U., DIETRICH, A., and HIRZINGER, G. “Catching flying balls with a mobile humanoid: System overview and design considerations”. In: *Proc. IEEE Int. Conf. Humanoid Robots (Humanoids)*. 2011, pp. 513–520. DOI: 10.1109/Humanoids.2011.6100837.
- [11] BAUR, J., PFAFF, J., ULBRICH, H., and VILLGRATTNER, T. “Design and development of a redundant modular multipurpose agricultural manipulator”. In: *Advanced Intelligent Mechatronics (AIM), 2012 IEEE/ASME International Conference on*. 2012, pp. 823–830. DOI: 10.1109/AIM.2012.6265928.

- [12] BAUR, J., PFAFF, J., SCHÜTZ, C., and ULBRICH, H. “Dynamic modeling and realization of an agricultural manipulator”. In: *Proc. of XV International Symposium on Dynamic Problems of Mechanics, DINAME*. ABCM. Buzios, RJ, Brazil, Feb. 2013.
- [13] BAZARAA, M. S., SHERALI, H. D., and SHETTY, C. M. *Nonlinear Programming: Theory and Algorithms*. Wiley-interscience, 2006.
- [14] BEN-ISRAEL, A. and GREVILLE, T. N. *Generalized inverses: theory and applications*. Vol. 15. Springer, 2003.
- [15] BENALLEGUE, M., ESCANDE, A., MIOSSEC, S., and KHEDDAR, A. “Fast  $C^1$  proximity queries using support mapping of sphere-torus-patches bounding volumes”. In: *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*. IEEE. 2009, pp. 483–488.
- [16] BENTLEY, J. L. “Multidimensional Binary Search Trees Used for Associative Searching”. In: *Communications of the ACM* **18**, 9 (1975), pp. 509–517.
- [17] BERTSEKAS, D. P. *Nonlinear programming*. Athena Scientific, 1999.
- [18] BLOCHWITZ, T., OTTER, M., ÅKESSON, J., ARNOLD, M., et al. “Functional Mockup Interface 2.0: The standard for tool independent exchange of simulation models.” In: *Proc. of the 9th International Modelica Conference*. Ed. by OTTER, M. and ZIMMER, D. Munich, Germany, Sept. 2012. DOI: DOI : 10 . 3384 / ecp12076173.
- [19] BONDY, J. A. and MURTY, U. S. R. *Graph Theory*. Graduate texts in mathematics Vol. 244. New York: Springer, 2008.
- [20] BONNANS, J. F., GILBERT, J. C., and LEMARÉCHAL, C. *Numerical optimization*. Springer, 2007.
- [21] BRANDL, H., JOHANNI, R., and OTTER, M. “A Very Efficient Algorithm for the Simulation of Robots and Similar Multibody Systems without Inversion of the Mass Matrix.” In: *IFAC/IFIP/IMACS Symposium on Theory of Robots*. 1986, pp. 95–100.
- [22] BRANDL, H., JOHANNI, R., and OTTER, M. “An Algorithm for the Simulation of Multibody Systems with Kinematic Loops”. In: *Proc. of the IFToMM Seventh World Congress on the Theory of Machines and Mechanisms*. 1987, pp. 407–411.
- [23] BREMER, H. *Dynamik und Regelung mechanischer Systeme*. Leitfäden der angewandten Mathematik und Mechanik; Bd. 67. (German). Stuttgart: Teubner Verlag, 1988.
- [24] BREMER, H. “Dynamik von Mehrkörpersystemen mit elastischen Bauteilen”. In: *Zeitschrift für Angewandte Mathematik und Mechanik (ZAMM)* **65**, 12 (1985), pp. 613–621. DOI: 10 . 1002 / zamm . 19850651208.



- [25] BREMER, H. *Elastic Multibody Dynamics – A Direct Ritz Approach*. Ed. by TZAFESTAS, S. G. Vol. 35. Intelligent Systems, Control, and Automation: Science and Engineering. Springer, 2008. DOI: 10.1007/978-1-4020-8680-9.
- [26] BREMER, H. “Subsystem Computation of Large Mechanical Systems”. In: *Proc. 7th World Congress on Theory of Machines and Mechanisms*. Mar. 1987, pp. 413–16.
- [27] BREMER, H. and PFEIFFER, F. *Elastische Mehrkörpersysteme*. (German). Teubner Verlag, 1992.
- [28] BUSCHMANN, T. “Simulation and Control of Biped Walking Robots”. Dissertation. Technische Universität München, 2010. URL: <http://nbn-resolving.de/urn/resolver.pl?urn:nbn:de:bvb:91-diss-20101201-997204-1-6>.
- [29] BUSCHMANN, T., WITTMANN, R., SCHWIENBACHER, M., and ULBRICH, H. “A Method for Real-Time Kineto-Dynamic Trajectory Generation”. In: *Proc. IEEE Int. Conf. Humanoid Robots (Humanoids)*. 2012.
- [30] BUSCHMANN, T., FAVOT, V., LOHMEIER, S., SCHWIENBACHER, M., and ULBRICH, H. “Experiments in Fast Biped Walking”. In: *Proc. IEEE Int. Conf. Mechatronics (ICM)*. Apr. 2011.
- [31] BUSCHMANN, T., LOHMEIER, S., SCHWIENBACHER, M., FAVOT, V., ULBRICH, H., HUNDELSHAUSEN, F. von, ROHE, G., and WÜNSCHE, H. “Walking in Unknown Environments—a Step Towards More Autonomy”. In: *Proc. IEEE Int. Conf. Humanoid Robots (Humanoids)*. IEEE. 2010, pp. 237–244.
- [32] CHEN, X.-D., YONG, J.-H., ZHENG, G.-Q., PAUL, J.-C., and SUN, J.-G. “Computing minimum distance between two implicit algebraic surfaces”. In: *Computer-Aided Design* **38**, 10 (2006), pp. 1053–1061. DOI: 10.1016/j.cad.2006.04.012.
- [33] CHENG, F.-T., CHEN, T.-H., and SUN, Y.-Y. “Resolving manipulator redundancy under inequality constraints”. In: *IEEE Trans. Robot. Autom.* **10**, 1 (1994), pp. 65–71. ISSN: 1042-296X. DOI: 10.1109/70.285587.
- [34] CHENG, H. and GUPTA, K. “A Study of Robot Inverse Kinematics Based upon the Solution of Differential Equations”. In: *J. Robot. Systems* **8**, 2 (1991), pp. 159–175.
- [35] CHIAVERINI, S., SICILIANO, B., and EGELAND, O. “Review of the damped least-squares inverse kinematics with experiments on an industrial robot manipulator”. In: *Trans. IEEE Control Systems Technology* **2**, 2 (1994), pp. 123–134. ISSN: 1063-6536. DOI: 10.1109/87.294335.
- [36] CHIAVERINI, S., EGELAND, O., and KANESTROM, R. K. “Weighted damped least-squares in kinematic control of robotic manipulators”. In: *Advanced Robotics* **7**, 3 (1992), pp. 201–218. DOI: 10.1163/156855393X00122.

- [37] CHIU, S. L. “Control of redundant manipulators for task compatibility”. In: *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*. Vol. 4. 1987, pp. 1718–1724. DOI: 10.1109/ROBOT.1987.1087795.
- [38] CLAUBERG, P. J. “Methoden zur parallelen Berechnung von nicht-glatten dynamischen Systemen”. (German). Dissertation. München: Technische Universität München, 2013.
- [39] COHEN, J. D., LIN, M. C., MANOCHA, D., and PONAMGI, M. “I-COLLIDE: An interactive and exact collision detection system for large-scale environments”. In: *Proc. Symposium on Interactive 3D graphics*. ACM. 1995, 189–ff.
- [40] CRAIG, J. J. *Introduction to Robotics – Mechanics and Control*. Pearson Prentice Hall, 1986.
- [41] *DARPA Robotics Challenge*. U.S. Defense Advanced Research Project Agency. Oct. 2013. URL: <http://www.theroboticschallenge.org>.
- [42] DEO, A. S. and WALKER, I. D. “Robot subtask performance with singularity robustness using optimal damped least-squares”. In: *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*. 1992, 434–441 vol.1. DOI: 10.1109/ROBOT.1992.220301.
- [43] DIETRICH, A., WIMBÖCK, T., ALBU-SCHAFFER, A., and HIRZINGER, G. “Integration of Reactive, Torque-Based Self-Collision Avoidance Into a Task Hierarchy”. In: *IEEE Trans. Robot.* **28**, 6 (2012), pp. 1278–1293. ISSN: 1552-3098. DOI: 10.1109/TR0.2012.2208667.
- [44] EHMANN, S. A. and LIN, M. C. “Accurate and fast proximity queries between polyhedra using convex surface decomposition”. In: *Computer Graphics Forum (Proc. of Eurographics)*. Vol. 20. 3. Wiley Online Library. 2001, pp. 500–511.
- [45] ENGLISH, K., HAYES, M. J. D., LEITNER, M., and SALLINGER, C. “Kinematic Calibration of Six-Axis Robots”. In: *Proc. of the CSME Forum*. Kingston (Canada), 2002.
- [46] *Enhanced Intel SpeedStep Technology for the Intel Pentium M Processor*. Edition: Order Number: 301170-001. Note: Order Number: 301170-001. Intel Corporation. Mar. 2004.
- [47] ESCANDE, A., MIOSSEC, S., and KHEDDAR, A. “Continuous gradient proximity distance for humanoids free-collision optimized-postures”. In: *Proc. IEEE Int. Conf. Humanoid Robots (Humanoids)*. 2007, pp. 188–195. DOI: 10.1109/ICHR.2007.4813867.
- [48] ESCANDE, A., MANSARD, N., and WIEBER, P.-B. “Fast resolution of hierarchized inverse kinematics with inequality constraints”. In: *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*. 2010, pp. 3733–3738.

- [49] FALCO, P. and NATALE, C. “On the Stability of Closed-Loop Inverse Kinematics Algorithms for Redundant Robots”. In: *Robotics, IEEE Transactions on* **27**, 4 (2011), pp. 780–784. ISSN: 1552-3098. DOI: 10.1109/TR0.2011.2135210.
- [50] FAVOT, V., BUSCHMANN, T., SCHWIENBACHER, M., EWALD, A., and ULBRICH, H. “The Sensor-Controller Network of the Humanoid Robot LOLA”. In: *Proc. IEEE Int. Conf. Humanoid Robots (Humanoids)*. Osaka, Japan, 2012.
- [51] FEATHERSTONE, R. “A divide-and-conquer articulated-body algorithm for parallel  $O(\log(n))$  calculation of rigid-body dynamics. Part 2: Trees, loops, and accuracy”. In: *Int. J. Robot. Research* **18**, 9 (1999), pp. 876–892.
- [52] FEATHERSTONE, R. “Efficient Factorization of the Joint Space Inertia Matrix for Branched Kinematic Trees”. In: *Int. J. Robot. Research* **24**, 6 (2005), pp. 487–500.
- [53] FEATHERSTONE, R. *Rigid Body Dynamics Algorithms*. Springer Berlin, 2008.
- [54] FEATHERSTONE, R. “The calculation of robot dynamics using articulated body inertias.” In: *Int. J. Robot. Research* **2**, 1 (1983), pp. 13–30.
- [55] FEATHERSTONE, R. and ORIN, D. E. “Robot dynamics: Equations and algorithms”. In: *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*. Vol. 1. 2000, pp. 826–834.
- [56] FÖRG, M. “Mehrkörpersysteme mit mengenwertigen Kraftgesetzen: Theorie und Numerik”. (German). PhD thesis. Technische Universität München, 2007.
- [57] FRIEDRICH, M. “Parallel Co-Simulation for Mechatronic Systems”. PhD thesis. Technische Universität München, 2011.
- [58] GATTRINGER, H. *Starr-elastische Robotersysteme: Theorie und Anwendungen*. (German). Springer Verlag Berlin Heidelberg, 2011. DOI: 10.1007/978-3-642-22828-5.
- [59] GATTRINGER, H., BREMER, H., and KASTNER, M. “Efficient dynamic modeling for rigid multi-body systems with contact and impact. An  $O(n)$  formulation”. In: *Acta Mechanica* **219**, 1 (2011), pp. 111–128.
- [60] GIENGER, M. “Entwurf und Realisierung einer zweibeinigen Laufmaschine”. (German). Dissertation. München: Technische Universität München, 2005.
- [61] GIENGER, M., TOUSSAINT, M., and GOERICK, C. “Task Maps in Humanoid Robot Manipulation”. In: *Proc. IEEE Int. Conf. Intell. Robots Syst. (IROS)*. 2008, pp. 2758–2764. DOI: 10.1109/IROS.2008.4651027.
- [62] GIENGER, M., JANSSEN, H., and GOERICK, C. “Task-Oriented Whole Body Motion for Humanoid Robots”. In: *Proc. IEEE Int. Conf. Humanoid Robots (Humanoids)*. 2005, pp. 238–244. DOI: 10.1109/ICHR.2005.1573574.
- [63] GILBERT, E. G., JOHNSON, D. W., and KEERTHI, S. S. “A fast procedure for computing the distance between complex objects in three-dimensional space”. In: *IEEE J. Robot. Autom.* **4**, 2 (1988), pp. 193–203.

- [64] GOSWAMI, A. and KALLEM, V. “Rate of change of angular momentum and balance maintenance of biped robots”. In: *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*. 2004.
- [65] HAGER, W. W. “Updating the Inverse of a Matrix”. In: *SIAM Review* **31**, 2 (1989), pp. 221–239.
- [66] HAYATI, S. and MIRMIRANI, M. “Improving the absolute positioning accuracy of robot manipulators”. In: *Journal of Robotic Systems* **2**, 4 (1985), pp. 397–413. DOI: 10.1002/rob.4620020406.
- [67] HOLLERBACH, J., KHALIL, W., and GAUTIER, M. “Springer Handbook of Robotics”. In: ed. by SICILIANO, B. and KHATIB, O. Springer, 2008. Chap. Model Identification, pp. 321–344.
- [68] HUSTON, R. L. “Useful Procedures in Multibody Dynamics”. In: *Dynamics of Multibody Systems*. Ed. by G., B. and W., S. IUTAM/IFTToMM Symposium. Udine, Italy: Springer, Sept. 1985, pp. 69–77.
- [69] *Jack and Process Simulate Human*. Siemens PLM Software. Sept. 2013. URL: [http://www.plm.automation.siemens.com/en\\_us/products/tecnomatix/assembly\\_planning/jack/](http://www.plm.automation.siemens.com/en_us/products/tecnomatix/assembly_planning/jack/).
- [70] JIMENEZ, P., THOMAS, F., and TORRAS, C. “3D collision detection: a survey”. In: *Computers & Graphics* **25** (2001), pp. 269–285.
- [71] KAJITA, S., NAGASAKI, T., KANEKO, K., YOKOI, K., and TANIE, K. “A Running Controller of Humanoid Biped HRP-2LR”. In: *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*. 2005, pp. 616–622. DOI: 10.1109/ROBOT.2005.1570186.
- [72] KAJITA, S., KANEHIRO, F., KANEKO, K., FUJIWARA, K., HARADA, K., YOKOI, K., and HIRUKAWA, H. “Resolved Momentum Control: Humanoid Motion Planning based on the Linear and Angular Momentum”. In: *Proc. IEEE Int. Conf. Intell. Robots Syst. (IROS)*. 2003, pp. 1644–1650.
- [73] KANEHIRO, F., SULEIMAN, W., MIURA, K., MORISAWA, M., and YOSHIDA, E. “Feasible Pattern Generation Method for Humanoid Robots”. In: *Proc. IEEE Int. Conf. Humanoid Robots (Humanoids)*. 2009, pp. 542–548.
- [74] KANEKO, K., KANEHIRO, F., MORISAWA, M., AKACHI, K., MIYAMORI, G., HAYASHI, A., and KANEHIRA, N. “Humanoid robot HRP-4-humanoid robotics platform with lightweight and slim body”. In: *Proc. IEEE Int. Conf. Intell. Robots Syst. (IROS)*. IEEE. 2011, pp. 4400–4407.
- [75] KANOUN, O. “Real-time prioritized kinematic control under inequality constraints for redundant manipulators”. In: *Robotics: Science and Systems VII* (2012), pp. 145–152.

- [76] KETTNER, L., MEYER, A., and ZOMORODIAN, A. “Intersecting Sequences of dD Iso-oriented Boxes”. In: *CGAL User and Reference Manual*. 4.2. CGAL Editorial Board, 2013. URL: <http://doc.cgal.org/4.2/CGAL.CGAL/html/packages.html#PkgBoxIntersectionDSummary>.
- [77] KHATIB, O. “Real-Time Obstacle Avoidance for Manipulators and Mobile Robots”. In: *Int. J. Robot. Research* **5**, 1 (1986), pp. 90–98. DOI: 10.1177/027836498600500106.
- [78] KUFFNER, J., NISHIWAKI, K., KAGAMI, S., KUNIYOSHI, Y., INABA, M., and INOUE, H. “Self-Collision Detection and Prevention for Humanoid Robots”. In: *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*. 2002, pp. 2265–2270. DOI: 10.1109/ROBOT.2002.1013569.
- [79] LARSEN, E., GOTTSCHALK, S., LIN, M. C., and MANOCHA, D. “Fast distance queries with rectangular swept sphere volumes”. In: *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*. Vol. 4. 2000, pp. 3719–3726. DOI: 10.1109/ROBOT.2000.845311.
- [80] LENARČIČ, J. “An efficient numerical approach for calculating the inverse kinematics for robot manipulators”. In: *Robotica* **3** (1 Jan. 1985), pp. 21–26. DOI: 10.1017/S0263574700001430.
- [81] LIÉGEOIS, A. “Automatic Supervisory Control of the Configuration and Behavior of Multibody Mechanisms”. In: *IEEE Trans. Syst., Man, Cybern.* **7**, 12 (1977), pp. 868–871. DOI: 10.1109/TSMC.1977.4309644.
- [82] LIN, M. C. and GOTTSCHALK, S. “Collision Detection Between Geometric Models: A Survey”. In: *Proc. of IMA Conference on Mathematics of Surfaces* **7** (1998), pp. 1–20.
- [83] LIN, M. C. and MANOCHA, D. “Collision and Proximity Queries”. In: (2003).
- [84] LÖFFLER, K. “Dynamik und Regelung einer zweibeinigen Laufmaschine”. (German). Dissertation. München: Technische Universität München, 2006.
- [85] LOHMEIER, S. “Design and Realization of a Humanoid Robot for Fast and Autonomous Bipedal Locomotion”. Dissertation. München: Technische Universität München, 2010. URL: <http://nbn-resolving.de/urn/resolver.pl?urn:nbn:de:bvb:91-diss-20101126-980754-1-4>.
- [86] LOHMEIER, S., BUSCHMANN, T., and ULBRICH, H. “Humanoid robot LOLA”. In: *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*. 2009, pp. 775–780. DOI: 10.1109/ROBOT.2009.5152578.
- [87] LOHMEIER, S., BUSCHMANN, T., ULBRICH, H., and PFEIFFER, F. “Modular Joint Design for Performance Enhanced Humanoid Robot LOLA”. In: *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*. 2006, pp. 88–93. DOI: 10.1109/ROBOT.2006.1641166.

- [88] LOHMEIER, S., BUSCHMANN, T., and ULBRICH, H. “System Design and Control of Anthropomorphic Walking Robot LOLA”. In: *IEEE/ASME Trans. on Mechatronics* **14**, 6 (2009), pp. 658–666. ISSN: 1083-4435. DOI: 10.1109/TMECH.2009.2032079.
- [89] LUH, J. Y. S., WALKER, M. W., and PAUL, R. P. C. “Resolved-acceleration control of mechanical manipulators”. In: *IEEE Trans. Autom. Control* **25**, 3 (June 1980), pp. 468–474. ISSN: 0018-9286. DOI: 10.1109/TAC.1980.1102367.
- [90] MACIEJEWSKI, A. A. and KLEIN, C. A. “Numerical filtering for the operation of robotic manipulators through kinematically singular configurations”. In: *Journal of Robotic Systems* **5**, 6 (1988), pp. 527–552.
- [91] MACIEJEWSKI, A. A. and KLEIN, C. A. “The Singular Value Decomposition: Computation and Applications to Robotics”. In: *Int. J. Robot. Research* **8**, 6 (1989), pp. 63–79.
- [92] MANSARD, N., STASSE, O., EVRARD, P., and KHEDDAR, A. “A versatile Generalized Inverted Kinematics implementation for collaborative working humanoid robots: The Stack Of Tasks”. In: *Advanced Robotics, 2009. ICAR 2009. International Conference on*. 2009, pp. 1–6.
- [93] MEAGHER, D. “Geometric modeling using octree encoding”. In: *Computer Graphics and Image Processing* **19**, 2 (1982), pp. 129–147. DOI: 10.1016/0146-664X(82)90104-6.
- [94] MIRTICH, B. “V-Clip: Fast and robust polyhedral collision detection”. In: *ACM Transactions on Graphics (TOG)* **17**, 3 (1998), pp. 177–208.
- [95] MOORING, B. W., ROTH, Z. S., and DRIELS, M. R. *Fundamentals of Manipulator Calibration*. John Wiley & Sons, 1991.
- [96] MUJA, M. and LOWE, D. G. “Fast approximate nearest neighbors with automatic algorithm configuration”. In: *International Conference on Computer Vision Theory and Applications (VISSAPP'09)*. 2009, pp. 331–340.
- [97] NAKAMURA, Y. *Advanced Robotics: Redundancy and Optimization*. 1st ed. Boston, MA, USA: Addison-Wesley Pub. Co., 1991.
- [98] NAKAMURA, Y. and HANAFUSA, H. “Inverse Kinematic Solutions With Singularity Robustness for Robot Manipulator Control”. In: *J. Dyn. Sys., Meas., Control* **108** (3 1986), pp. 163–171. DOI: doi:10.1115/1.3143764.
- [99] NAKAMURA, Y. and HANAFUSA, H. “Optimal Redundancy Control of Robot Manipulators”. In: *Int. J. Robot. Research* **6** (1987), pp. 32–42. DOI: 10.1177/027836498700600103.
- [100] NAKAMURA, Y., HANAFUSA, H., and YOSHIKAWA, T. “Task-Priority Based Redundancy Control of Robot Manipulators”. In: *Int. J. Robot. Research* **6**, 2 (1987), pp. 3–15. DOI: 10.1177/027836498700600201.

- [101] NAKANISHI, Y., NAMIKI, Y., HONGO, K., URATA, J., MIZUUCHI, I., and INABA, M. “Design of the musculoskeletal trunk and realization of powerful motions using spines”. In: *Proc. IEEE Int. Conf. Humanoid Robots (Humanoids)*. IEEE. 2007, pp. 96–101.
- [102] NISHIWAKI, K., KAGAMI, S., KUNYOSHI, Y., INABA, M., and INOUE, H. “Online generation of humanoid walking motion based on a fast generation method of motion pattern that follows desired zmp”. In: *Proc. IEEE Int. Conf. Intell. Robots Syst. (IROS)*. Vol. 3. IEEE. 2002, pp. 2684–2689.
- [103] NISHIWAKI, K., KUFFNER, J., KAGAMI, S., INABA, M., and INOUE, H. “The experimental humanoid robot H7: a research platform for autonomous behaviour”. In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* **365**, 1850 (2007), pp. 79–107.
- [104] NISHIWAKI, K., KAGAMI, S., KUNYOSHI, Y., INABA, M., and INOUE, H. “Toe joints that enhance bipedal and fullbody motion of humanoid robots”. In: *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*. Vol. 3. IEEE. 2002, pp. 3105–3110.
- [105] NOCEDAL, J. and WRIGHT, S. J. *Numerical optimization*. Springer Science+ Business Media, 2006.
- [106] OGURA, Y., AIKAWA, H., LIM, H.-O., and TAKANISHI, A. “Development of a human-like walking robot having two 7-DOF legs and a 2-DOF waist”. In: *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*. Vol. 1. IEEE. 2004, pp. 134–139.
- [107] OKADA, K., INABA, M., and INOUE, H. “Real-time and Precise Self Collision Detection System for Humanoid Robots”. In: *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*. 2005, pp. 1060–1065.
- [108] OMER, A. M. M., OGURA, Y., KONDO, H., MORISHIMA, A., CARBONE, G., CECCARELLI, M., LIM, H.-o., and TAKANISHI, A. “Development of a humanoid robot having 2-DOF waist and 2-DOF trunk”. In: *Proc. IEEE Int. Conf. Humanoid Robots (Humanoids)*. IEEE. 2005, pp. 333–338.
- [109] ONG, C. J. and GILBERT, E. G. “Fast Versions of the Gilbert-Johnson-Keerthi Distance Algorithm: Additional Results and Comparisons”. In: *IEEE Trans. Robot. Autom.* **17**, 4 (Aug. 2001), pp. 531–539. DOI: 10.1109/70.954768.
- [110] ORIN, D. E. and SCHRADER, W. W. “Efficient Computation of the Jacobian for Robot Manipulators”. In: *Int. J. Robot. Research* **3**, 4 (1984), pp. 66–75. DOI: 10.1177/027836498400300404.
- [111] OTT, C., BAUMGARTNER, C., MAYR, J., FUCHS, M., BURGER, R., LEE, D., EIBERGER, O., ALBU-SCHAFFER, A., GREBENSTEIN, M., and HIRZINGER, G. “Development of a biped robot with torque controlled joints”. In: *Proc. IEEE Int. Conf. Humanoid Robots (Humanoids)*. IEEE. 2010, pp. 167–173.

- [112] PABST, S., KOCH, A., and STRASSER, W. “Fast and scalable cpu/gpu collision detection for rigid and deformable surfaces”. In: *Computer Graphics Forum*. Vol. 29. 5. Wiley Online Library. 2010, pp. 1605–1612.
- [113] PARK, J., CHUNG, W., and YOUM, Y. “Computation of Gradient of Manipulability for Kinematically Redundant Manipulators Including Dual Manipulators System”. In: *Trans. Control, Automation and Systems Engineering* 1, 1 (1999), pp. 8–15. URL: <http://www.ijcas.com/admin/paper/files/7641.pdf>.
- [114] PFEIFFER, F. *Mechanical System Dynamics*. Vol. 40. Lecture Notes in Applied and Computational Mechanics. Springer-Verlag Heidelberg, 2008. DOI: 10.1007/978-3-540-79436-3.
- [115] PIEPER, D. L. “The kinematics of manipulators under computer control”. PhD thesis. Stanford University, 1968.
- [116] PRESS, W. H., TEUKOLSKY, S. A., VETTERLING, W. T., and FLANNERY, B. P. *Numerical Recipes in C: The Art of Scientific Computing*. 2nd ed. Press Syndicate of the University of Cambridge, 2002.
- [117] QUINLAN, S. “Efficient distance computation between non-convex objects”. In: *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*. Vol. 4. 1994, pp. 3324–3329. DOI: 10.1109/ROBOT.1994.351059.
- [118] RAIBERT, M. H. *Legged Robots that Balance (Artificial Intelligence)*. Cambridge: MIT Press, 1986.
- [119] RAIBERT, M. H. “Legged robots”. In: *Communications of the ACM* 29, 6 (1986), pp. 499–514.
- [120] ROBERSON, R. E. and SCHWERTASSEK, R. *Dynamics of multibody systems*. Springer-Verlag Berlin, 1988.
- [121] RUSU, R. B. and COUSINS, S. “3D is here: Point Cloud Library (PCL)”. In: *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*. 2011, pp. 1–4. DOI: 10.1109/ICRA.2011.5980567.
- [122] SCHIEHLEN, W. “Multibody System Dynamics: Roots and Perspectives”. In: *Multibody System Dynamics* 1 (1997), pp. 149–188.
- [123] SCHIEHLEN, W. and EBERHARD, P. *Technische Dynamik*. (German). Teubner Verlag, 2004.
- [124] SCHIELA, A. and OLSSON, H. “Mixed-mode Integration for Real-Time Simulation”. In: *Modelica Workshop*. Lund, Sweden, 2000, pp. 69–75.
- [125] SCHWIENBACHER, M., BUSCHMANN, T., LOHMEIER, S., FAVOT, V., and ULBRICH, H. “Self-Collision Avoidance and Angular Momentum Compensation for a Biped Humanoid Robot”. In: *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*. Shanghai, China, May 2011, pp. 581–586. DOI: 10.1109/ICRA.2011.5980350.



- [126] SCHWIENBACHER, M., BUSCHMANN, T., and ULBRICH, H. “Vertical Angular Momentum Minimization for Biped Robots with Kinematically Redundant Joints”. In: *The 23rd International Congress of Theoretical and Applied Mechanics*. IUTAM. Aug. 2012.
- [127] SICILIANO, B. and SLOTINE, J.-J. E. “A general framework for managing multiple tasks in highly redundant robotic systems”. In: *Advanced Robotics, 1991. 'Robots in Unstructured Environments', 91 ICAR., Fifth International Conference on*. 1991, 1211–1216 vol.2. DOI: 10.1109/ICAR.1991.240390.
- [128] SICILIANO, B., SCIavicco, L., VILLANI, L., and ORIOLO, G. *Robotics: Modelling, Planning and Control*. Springer, 2009.
- [129] SICILIANO, B. and KHATIB, O. *Springer Handbook of Robotics*. Springer, 2008.
- [130] SORGE, K. “Mehrkörpersysteme mit starr-elastischen Subsystemen”. (German). PhD thesis. TU München, July 1992.
- [131] STASSE, O., ESCANDE, A., MANSARD, N., MIOSSEC, S., EVRARD, P., and KHEDDAR, A. “Real-Time (Self)-Collision Avoidance Task on a HRP-2 Humanoid Robot”. In: *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*. 2008.
- [132] SUD, A., GOVINDARAJU, N., GAYLE, R., KABUL, I., and MANOCHA, D. “Fast proximity computation among deformable models using discrete Voronoi diagrams”. In: *ACM Trans. Graph.* **25**, 3 (July 2006), pp. 1144–1153. DOI: 10.1145/1141911.1142006.
- [133] SUGIURA, H., GIENGER, M., JANSSEN, H., and GOERICK, C. “Real-Time Collision Avoidance with Whole Body Motion Control for Humanoid Robots”. In: *Proc. IEEE Int. Conf. Intell. Robots Syst. (IROS)*. 2007, pp. 2053–2058. DOI: 10.1109/IROS.2007.4399062.
- [134] SUGIURA, H., GIENGER, M., JANSSEN, H., and GOERICK, C. “Real-Time Self Collision Avoidance for Humanoids by means of Nullspace Criteria and Task Intervals”. In: *Proc. IEEE Int. Conf. Intell. Robots Syst. (IROS)*. 2006, pp. 575–580.
- [135] TAJIMA, R., HONDA, D., and SUGA, K. “Fast running experiments involving a humanoid robot”. In: *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*. 2009, pp. 1571–1576. DOI: 10.1109/ROBOT.2009.5152404.
- [136] TAKENAKA, T., MATSUMOTO, T., and YOSHIKE, T. “Real time motion generation and control for biped robot -1<sup>st</sup> report: Walking gait pattern generation-”. In: *Proc. IEEE Int. Conf. Intell. Robots Syst. (IROS)*. 2009, pp. 1084–1091. DOI: 10.1109/IROS.2009.5354662.

- [137] TAKENAKA, T., MATSUMOTO, T., YOSHIIKE, T., and SHIROKURA, S. “Real time motion generation and control for biped robot -2<sup>nd</sup> report: Running gait pattern generation-”. In: *Proc. IEEE Int. Conf. Intell. Robots Syst. (IROS)*. 2009, pp. 1092–1099. DOI: 10.1109/IROS.2009.5354654.
- [138] TAKENAKA, T., MATSUMOTO, T., and YOSHIIKE, T. “Real time motion generation and control for biped robot -3<sup>rd</sup> report: Dynamics error compensation-”. In: *Proc. IEEE Int. Conf. Intell. Robots Syst. (IROS)*. 2009, pp. 1594–1600. DOI: 10.1109/IROS.2009.5354542.
- [139] TAKENAKA, T., MATSUMOTO, T., YOSHIIKE, T., HASEGAWA, T., SHIROKURA, S., KANEKO, H., and ORITA, A. “Real time motion generation and control for biped robot -4<sup>th</sup> report: Integrated balance control-”. In: *Proc. IEEE Int. Conf. Intell. Robots Syst. (IROS)*. 2009, pp. 1601–1608. DOI: 10.1109/IROS.2009.5354522.
- [140] TANG, M., MANOCHA, D., LIN, J., and TONG, R. “Collision-Streams: Fast GPU-based collision detection for deformable models”. In: *I3D '11: Proc. of the 2011 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*. San Francisco, CA, 2011, pp. 63–70.
- [141] TÄUBIG, H., BÄUML, B., and FRESE, U. “Real-time swept volume and distance computation for self collision detection”. In: *Proc. IEEE Int. Conf. Intell. Robots Syst. (IROS)*. IEEE. 2011, pp. 1585–1592.
- [142] TESCHNER, M., KIMMERLE, S., HEIDELBERGER, B., ZACHMANN, G., RAGHUPATHI, L., FUHRMANN, A., CANI, M.-P., FAURE, F., MAGNENAT-THALMANN, N., STRASSER, W., et al. “Collision Detection for Deformable Objects”. In: *Computer Graphics Forum*. Vol. 24. 1. Wiley Online Library. 2005, pp. 61–81.
- [143] THE CGAL PROJECT. *CGAL User and Reference Manual*. 4.2. CGAL Editorial Board, 2013. URL: <http://doc.cgal.org/4.2/CGAL.CGAL/html/packages.html>.
- [144] THIBAUT, W. C. and NAYLOR, B. F. “Set operations on polyhedra using binary space partitioning trees”. In: *SIGGRAPH Comput. Graph.* **21**, 4 (Aug. 1987), pp. 153–162. DOI: 10.1145/37402.37421.
- [145] TOLANI, D., GOSWAMI, A., and BADLER, N. I. “Real-Time Inverse Kinematics Techniques for Anthropomorphic Limbs”. In: *Graphical Models* **62**, 5 (2000), pp. 353–388.
- [146] TOUSSAINT, M., GIENGER, M., and GOERICK, C. “Optimization of sequential attractor-based movement for compact behaviour generation”. In: *Proc. IEEE Int. Conf. Humanoid Robots (Humanoids)*. 2007, pp. 122–129. DOI: 10.1109/ICHR.2007.4813858.
- [147] TRÄGER, M. *Echtzeitfähige Kollisionserkennung mit Swept-Sphere-Volumes in einer Robotikanwendung*. Term Project. May 2010.

- [148] ULBRICH, H. *Maschinendynamik*. (German). Teubner Verlag, 1996.
- [149] URATA, J., NAKANISHI, Y., OKADA, K., and INABA, M. “Design of high torque and high speed leg module for high power humanoid”. In: *Proc. IEEE Int. Conf. Intell. Robots Syst. (IROS)*. 2010, pp. 4497–4502. DOI: 10.1109/IROS.2010.5649683.
- [150] URATA, J., NISHIWAKI, K., NAKANISHI, Y., OKADA, K., KAGAMI, S., and INABA, M. “Online decision of foot placement using singular LQ preview regulation”. In: *Proc. IEEE Int. Conf. Humanoid Robots (Humanoids)*. 2011, pp. 13–18. DOI: 10.1109/Humanoids.2011.6100894.
- [151] VRANEK, D. “Fast and accurate circle-circle and circle-line 3d distance computation”. In: *Journal of graphics tools* **7**, 1 (2002), pp. 23–31.
- [152] WALKER, M. W. and ORIN, D. E. “Efficient Dynamic Computer Simulation of Robotic Mechanisms”. In: *Journal of Dynamic Systems, Measurement, and Control* **104**, 3 (1982), pp. 205–211. DOI: 10.1115/1.3139699.
- [153] WAMPLER, C. W. “Manipulator Inverse Kinematic Solutions Based on Vector Formulations and Damped Least-Squares Methods”. In: *IEEE Trans. Syst., Man, Cybern.* **16**, 1 (1986), pp. 93–101. ISSN: 0018-9472. DOI: 10.1109/TSMC.1986.289285.
- [154] WHITNEY, D. E. “Resolved Motion Rate Control of Manipulators and Human Prostheses”. In: *IEEE Trans. Man-Mach. Syst.* **10**, 2 (1969), pp. 47–53. DOI: 10.1109/TMMS.1969.299896.
- [155] WILLIAMS, S., OLIKER, L., VUDUC, R., SHALF, J., YELICK, K., and DEMMEL, J. “Optimization of sparse matrix–vector multiplication on emerging multicore platforms”. In: *Parallel Computing* **35**, 3 (2009), pp. 178–194.
- [156] YAMANE, K. *Simulating and Generating Motions of Human Figures*. Vol. 9. Springer Tracts in Advanced Robotics. Springer, 2004.
- [157] YOON, S.-E. and MANOCHA, D. “Cache-Efficient Layouts of Bounding Volume Hierarchies”. In: *Computer Graphics Forum (Proc. of Eurographics)*. 2006.
- [158] YOSHIKAWA, T. “Manipulability of Robotic Mechanisms”. In: *The International Journal of Robotics Research* **4**, 2 (1985), pp. 3–9. DOI: 10.1177/027836498500400201.
- [159] ZEGHLOUL, S., RAMBEAUD, P., and LALLEMAND, J. P. “A fast distance calculation between convex objects by optimization approach”. In: *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*. Vol. 3. 1992, pp. 2520–2525. DOI: 10.1109/ROBOT.1992.220062.
- [160] ZHAO, J. and BADLER, N. I. “Inverse kinematics positioning using nonlinear programming for highly articulated figures”. In: *ACM Trans. Graph.* **13**, 4 (Oct. 1994), pp. 313–336. DOI: 10.1145/195826.195827.