Fakultät für Maschinenwesen
Lehrstuhl für Angewandte Mechanik

Autonomous Robots in Unknown and Dynamic Scenarios
Biped Navigation, Real-Time Motion Generation and Collision Avoidance

Arne-Christoph Hildebrandt

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

Doktor-Ingenieurs (Dr.-Ing.)

genehmigten Dissertation.

Vorsitzender: Prof. Dr.-Ing. habil. Boris Lohmann

Prüfer der Dissertation:

1. Prof. dr. ir. Daniel J. Rixen
2. Directeur de Recherche Olivier Stasse, Ph.D.

Die Dissertation wurde am 30.05.2018 bei der Technischen Universität München eingereicht und durch die Fakultät für Maschinenwesen am 01.10.2018 angenommen.

Abstract

Walking in real-world scenarios entails strong requirements on legged robotic systems. Navigation in complex environments featuring obstacles, varying ground compositions and external disturbances requires autonomous operation involving real-time motion generation, perception and stabilization. Finding a combined solution for real-time motion planning in unknown environments with external disturbances is one of the challenges of current robotics research.

In this thesis, methods for motion generation together with their integration in an overall real-time framework are presented. Special focus lies on the tight integration of methods for motion generation with a module for perception and disturbance rejection. Bipedal walking represents long motion sequences connecting successive foothold positions. This thesis provides a hierarchical approach to solve the resulting motion generation problem. Discrete optimization techniques are combined with a mobile platform planner to navigate safely in unknown environments. A parameter representation is applied to represent the full motion in a compact way. Parameter and trajectory optimization methods are developed to exploit the full capabilities of humanoid robots while respecting dynamic feasibility.

Solving the motion planning problem for long sequences is an often posed question in many robotic applications not only in bipedal walking. The applicability of the presented methods to industrial and agricultural manipulators is discussed.

The methods presented in this thesis are validated with experimental results on the robot LOLA. The experimental results show that real-time motion planning, perception and disturbance rejection can be combined to improve the autonomy of legged robots in unknown and dynamic environments. The robustness of the developed approach has been demonstrated in several public presentations.

Zusammenfassung

Gehen in unbekannten Umgebungen stellt zweibeinige Roboter vor große Herausforderungen. Sie müssen in unbekannten Umgebungen navigieren, komplexe Bewegungen planen und gleichzeitig große Störungen ausgleichen. Diese Anforderungen erforden eine Echtzeit-Bewegungsplanung, eine Umgebungswahrnehmung und eine Stabilisierung. Bisher liegt noch kein Ansatz vor, welcher Methoden der drei Forschungsgebieten kombiniert.

In der vorliegenden Dissertation werden Methoden zur Echtzeit-Bewegungsplanung vorgestellt. Besonderer Schwerpunkt wird auf ihre Integration in ein Gesamtsystem in der Kombination mit Methoden zur Umgebungswahrnehmung und Stabilisierung gelegt. Zweibeiniges Gehen besteht aus langen Bewegungssequenzen, welche diskrete Standfußpositionen verbinden. Diese Arbeit bietet einen hierarchischen Ansatz um das resultierende Bewegungsplanungsproblem zu lösen. Diskrete Optimierungsmethoden werden mit Navigationsalgortihmen kombiniert, um sicheres Bewegen in unbekannten Umgebungen zu ermöglichen. Es wird eine Parameterdarstellung genutzt um die komplexe Bewegung des humanoiden Roboters kompakt darzustellen. Entwickelte Parameter- und Trajektorienoptimierungsverfahren können so in Echtzeit die Bewegungen planen und die physikalischen Fähigkeiten zweibeiniger Roboter unter Berücksichtigung der dynamischen Randbedingungen ausnutzen.

Die Bewegungsplanung von langen Sequenzen ist nicht nur eine Herausforderung beim zweibeinigen Gehen, sondern stellt viele Robotikanwendungen vor Herausforderungen. Daher wird in dieser Arbeit die Übertragbarkeit des vorgestellten Ansatzes auf industrielle und landwirtschaftliche Anwendungen diskutiert. Die vorgestellten Methoden werden in Experimenten mit dem zweibeinigen Roboter LOLA untersucht. Die experimentellen Ergebnisse zeigen, wie zweibeiniges Gehen in unbekannten und dynamischen Umgebungen möglich ist. Mehrere öffentliche Vorführungen stellten zudem die Robustheit des Ansatzes unter Beweis.

Danksagung

Die vorliegende Dissertation entstand im Rahmen meiner Tätigkeit am Lehrstuhl für Angewandte Mechanik der Technischen Universität München in der Zeit zwischen Mai 2013 und Mai 2017. Die meisten Ergebnisse der Arbeit basieren auf meiner Forschung im DFG geförderten Projekt „Flexibles und Robustes Gehen in unbekannten Umgebungen". Ohne die Unterstützung zahlreicher Personen wäre die vorliegende Arbeit nicht möglich gewesen.
Mein besonderer Dank gilt meinem Doktorvater Prof. dr. ir. Daniel Rixen. Durch sein stetes Interesse, seine Unterstützung und sein Vertrauen hat er mich motiviert, auch neue Wege zu gehen. Hervorzuheben ist die hervorragende Arbeitsatmosphäre am Lehrstuhl, die zu einem großen Teil sein Verdienst ist. Die Korrektur der Dissertation und die Durchführung der zugehörigen Prüfung bedeuten einen hohen Arbeitsaufwand. Daher bedanke ich mich bei Prof. Dr.-Ing. habil. Boris Lohmann für die Übernahme des Prüfungsvorsitzes und bei Directeur de Recherche Olivier Stasse, Ph.D. für die Korrektur der Dissertation als Zweitprüfer.
Das Projekt „Flexibles und Robustes Gehen in unbekannten Umgebungen" wäre ohne die finanzielle Unterstützung der DFG nicht möglich gewesen. Für die Beantragung der Fördergelder und die Ausarbeitung des spannenden Forschungsvorhabens gilt Dr.-Ing. habil. Thomas Buschmann besonderer Dank. Er hat meine Begeisterung für das faszinierende Feld der Robotik geweckt. Selbst nach dem Verlassen des Lehrstuhls hat er das Projekt und meine Dissertation immer tatkräftig unterstützt.
Die erfolgreiche Arbeit an einer so hoch komplexen Maschine wie der eines humanoiden Roboters ist alleine unmöglich. Während meiner gesamten Zeit am Lehrstuhl - und darüber hinaus - konnte ich immer auf die Unterstützung des Lola-Teams zählen. Vielen Dank Dr.-Ing. Robert Wittmann, Dr.-Ing. Daniel Wahrmann, Philipp Seiwald und Felix Sygulla für die vielen aufschlussreichen Diskussionen, das gemeinsame Umsetzen von neuen Ideen und die zahlreichen im Labor verbrachten Tage und Nächte. Vielen Dank auch an das Lola-Team und Dr.-Ing. habil. Thomas Buschmann für das Korrekturlesen meiner Dissertation.
Dass die Robotik nicht nur aus zweibeinigem Laufen besteht, hat mir die gute Zusammenarbeit mit Felix Ellensohn in der Robotik-Gruppe und Dr.-Ing. Christoph Schütz in der Robotik-Gruppe, der *European Robotics Challenge* und dem Landwirtschaftsroboter *CROPS* vor Augen geführt. Auch die interessante Zusammenarbeit mit Dr.-Ing. Michael Gienger zum kooperativen Arbeiten von Robotern war eine Bereicherung.
Da die Robotik immer auf gut funktionierende Hardware angewiesen ist, bin ich Georg Mayr zu Dank verpflichtet. Mit seiner Erfahrung und seiner angenehmen Arbeitsweise ist er ein Garant dafür, dass die mechatronischen Projekte am Lehrstuhl erfolgreich bearbeitet werden können. Verlassen konnte ich mich zudem immer auf die Kollegen in der Werkstatt Simon Gerer und Georg König, die dafür sorgten, dass die Maschinen auch in wichtigen Momenten immer funktionierten.
Dass am Lehrstuhl effizient gearbeitet werden kann, ist auch einer reibungslos funktionierenden Infrastruktur zu verdanken, um die sich Dr.-Ing. habil. Thomas Thümmel immer hervorragend kümmert und dabei von Frau Müller-Philipp und Rita Schneider unterstützt wird.
Abschließend möchte ich mich bei allen KollegInnen des Lehrstuhls und allen StudentInnen, die ich betreut habe, bedanken. Der Erfolg meines Dissertationsvorhabens und unseres Forschungsprojekts ist zum großen Teil auch ihrer Motivation und ihrem Einsatz zu verdanken.

München, 28. Oktober 2018

Table of Contents

List of Abbreviations

AIST	National Institute of Advanced Industrial Science and Technology
AM	Chair of Applied Mechanics
BVP	Boundary Value Problem
CoG	Center of Gravity
CoP	Center of Pressure
CROPS	Clever Robots for Crops
DARPA	Defense Advanced Research Projects Agency
DFG	Deutsche Forschungsgemeinschaft
DLR	German Aerospace Center
DoF	Degree of Freedom
EoM	Equation of Motion
EU	European Union
EuRoC	European Robotics Challenge
FoR	Frame of Reference
FoV	Field of View
GMM	Gaussian Mixture Models
IK	Inverse Kinematics
IMU	Inertial Measurement Unit
KAIST	Korea Advanced Institute of Science and Technology
SSV	Swept-Sphere-Volumes
TCP	tool center point
ZMP	Zero Moment Point

Chapter 1

Introduction

Until recent years, robot industrial applications have been limited to static and known scenarios. In large-volume manufacturing lines, robots execute the same tasks, often their whole lifetime. Their hardware design and the environment are perfectly optimized to allow for fast and reliable production. These setups are rarely changed. Unforeseen changes, like humans entering the setup, are excluded by cages separating them. Exceptions lead to emergency stops of the whole line. Since these scenarios are set up only once for a long time, the planning of the robots' motion is not time-critical and does not have to consider input from sensors perceiving the environment. That way, human operators can design the robots' motion off-line, checking for their feasibility and optimality.

Robots working in dynamic and unknown scenarios are mainly limited to wheeled mobile platforms. Due to constantly changing scenarios, motion planning has to react to sensor feedback and becomes time-critical. State-of-the-art mobile platforms present some degree of autonomy, but in most applications they are still mainly tele-operated by a human controller. The human controller commands the tasks, plans the motions and reacts to changes in the scenarios. Although prototypes show impressive results in autonomous driving, even cars are still mostly operated by humans. The human controls the car and commands directions and velocity of its motions.

In recent years, requirements on robots have been increasing. Instead of executing tasks that are either repetitive or strongly guided by a human operator, robots should be able to autonomously adapt to changing scenarios. That way, robots could be able to interact and cooperate with humans, move in changing environments or allow for automation of production lines with small unit numbers and complex products. The robots' limited versatility seems to be one of the major bottlenecks to meet these requirements.

On the hardware side, one way to increase the mobility of robots in different kind of environments is a redundant kinematic design. Especially, in cluttered environments redundant robots are able to reach areas by using their additional degrees of freedom (DoFs). The "king", or in the case of this thesis the "queen", of redundant robots is the humanoid robot. Inspired by the human anthromorphique structure, its kinematic is highly redundant. Thanks to this structure, they are suited to be employed in diverse and complex scenarios. Similarly to humans, humanoid robots are in most cases not the best solution for one particular task. Nevertheless, their design allows to serve as a human replacement in a wide range of applications from standard industrial tasks in factories to service robotic application in houses. This variety of possible applications is rarely met by any other robot design.

The objective of this thesis is to exploit the potential of bipedal robots. Methods are developed which use their intrinsic versatility and which allow for application of humanoid robots not only in static environments but also in dynamic and unknown scenarios.

1.1 Problem Statement

Although companies are starting to show increasing interest in the topic, bipedal robots are still the subject of fundamental research. From a research point of view, humanoid locomotion is especially interesting. It combines challenges of different robotic research fields.

Autonomous Navigation

In an abstract way, humanoids are mobile platforms. They are able to move in the same environments as wheeled mobile platforms and they are restricted by the same constraints: they have to perceive and model the environment in real-time, they have to exploit unknown environments and they have to avoid collisions with the environment which can be static but also dynamic.

Real-Time Motion Generation

In contrast to autonomous navigation of wheeled vehicles, autonomous navigation of humanoids is not solved via a velocity and the corresponding direction. Bipedal locomotion consists of sequences of discrete foothold positions. The swing foot movement, which connects the discrete foothold positions, can be interpreted as the motion of a robotic manipulator. This characteristic and their kinematic structure allow for stepping up and down platforms or stepping over obstacles. Thus, on the one hand, humanoids can be applied in much more complicated environments than wheeled vehicles. On the other hand, the motion planning becomes more complex. Instead of searching a trajectory in a three-dimensional space, trajectories of all DoFs have to be determined.

Regarding the motion planning problem, humanoid robots differ from manipulators in that way that their base is not fixed. It changes its location with each discrete step. Compared with conventional motion planning for manipulators, walking consists not of one single motion from point A to point B. A couple of steps already represents a long motion sequence. Furthermore, the robot's motion has to cope with the dynamic constraints of bipedal locomotion.

Stability & Sensor Feedback

Bipedal locomotion is not inherently stable. Real-time motion generation requires reduced models of the complex robotic system, humanoid robot, and its interaction with the environment. Thus, even in perfectly known environments without any perturbations, sensor feedback is necessary to compensate for model inaccuracies and prevent humanoids from falling down. Therefore, motion generation for bipedal walking always has to respect modified trajectories and has to react fast to changes.

1.2 Related Work

In recent years, the interest in human-like machines has greatly increased in academia as well as in industry. Their structure is adapted to tools and environments designed for humans. Therefore, they are able to fulfill tasks which are too dangerous, too dirty or too dull for human workers. While stable walking in flat terrain is achieved by various research groups [29, 43, 53, 87, 103], locomotion in real environments is still limited. A thorough review of humanoid robotic research groups is done by Buschmann [12] and Wittmann [114]. Until recently, bipedal robots were not capable to reliably navigate through complex scenarios. Neither were they not capable to handle large disturbances nor were they able to generate motions in complex environments.

These limitations question the assumed superiority of bipedal walking to wheeled or tracked vehicles in rough terrain and limit their use in real application. To overcome these limitations is the motivation for multiple research projects:

The IEEE RAS (Robotics and Automation Society) founded the Technical Comittee *Model-Based Optimization for Robotics* [74]. The main intention behind this comittee is to close the gap between the research comunities in the field of optimization theory and robotics. Robots represent extremely complicated dynamical systems while optimization techniques offer one way to generate automatically movements derived from basic principles as, for example, the minimization of cost functions. Thus, the researchers taking part in the comittee try to establish model-based optimization approaches for motion generation for robotic systems. Their explicit objective is to implement and demonstrate these on real robots in complex scenarios.

In 2013, in the course of these activities, the project *Koroibot* was initiated[1]. It is a European Union (EU) founded project with partners from all over Europe. The focus of the project lies on the developement of methods for motion generation for humanoids in complex environments. The project partners follow the idea to transfer motion primitives observed by human walking to control strategies for humanoids. The transfer concepts are based on inverse optimization. Due to computationally heavy models, the derived control strategies are often not suitable for real-time control and adapted to the high number of different walking situation in real application [58].

The 2011 earthquake in Japan and the following partial meltdown in the Fukushima Dai-Ichi nuclear motivated the Defense Advanced Research Projects Agency (DARPA) to organize a robotic competition[2]. In this competition, teams from all over the world developed robots which had to solve tasks similar to tasks expected in disaster response applications. The robots had to solve these tasks semi autonomously. Due to the challenging time restrictions the teams mostly developed methods to solve the specific tasks. The focus lied mainly on tasks as navigation, manipulation and vision with help of tele-operation rather than on autonomous walking control. The team from the Korea Advanced Institute of Science and Technology (KAIST) won the competition [5].

In addition to these mainly university dominated initatives, publications and videos show that also national research institutes and companies are cunducting research activities in this direction.

The German Aerospace Center (DLR) presented their humanoid TORO in 2014 [29]. In contrast to most humanoids, TORO is equipped with torque sensors in each joint. Their publications focus on multi-contact motions [112] and basic walking controllers [28]. The mechanical design and the actuators of the legs, prevent TORO from making large and fast strides which restricts its walking abilities in complex scenarios.

Since many years, the National Institute of Advanced Industrial Science and Technology (AIST) publishes impressive results of humanoids walking in uneven terrain. Their research focus lies on the control of stiff position controlled robots. Nishiwaki et al. [81] gives an overview of recent achievements.

In addition to the scientific community, companies have always been interested in humanoid robotics. Among others, *Honda*, *Toyota* or *Sony* pursue research in humanoid robotics and barely publish their results [39, 103, 104]. In recent years, the world of robotics has undergone erruptive changes. Many start up companies were founded which focus on autonomous working robots. In addition to the old players in robotics such as *Toyota* and *Honda*, new players as *Alphabet* and *UbiSoft* aggressively entered the marked buying

[1]`www.koroibot.eu`

[2]DARPA Robotics Challenge `www.theroboticschallenge.org`

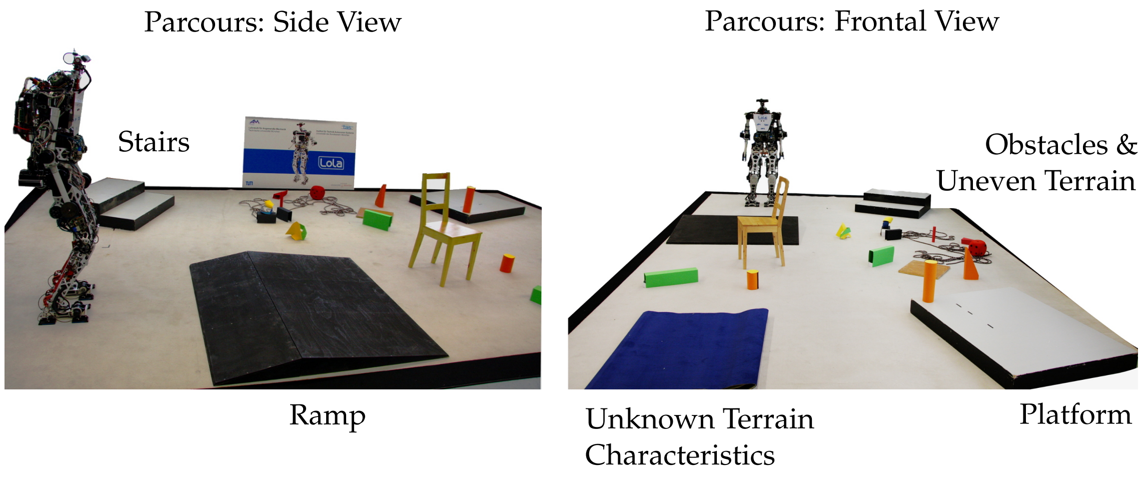


Figure 1.1: Motivational parcour for Deutsche Forschungsgemeinschaft (DFG)-project. It includes challenges for robust and versatile walking.

start ups and hiring a lot of experts from academia. In the context of this thesis, especially important are the companies *Boston Dynamics* and *Schaft*. Both, companies have long experiences in the technology of bipedal walking: the company *Boston Dynamics* exists since more than 20 years. *Schaft* is a spin-off of the University of Tokyo, which is active in the research area of bipedal locomotion since decades. Videos show their biped robots walking impressively robustly outside the laboratory in complex terrain. Unfortunately, beside the videos[3] and few basic publications [78, 107], no further information has been published.

This thesis was written as part of the DFG-Project *Walking in Uneven Terrain* (BU 2736/1-1). The project started in 2013 at the Chair of Applied Mechanics (AM) of the Technical University Munich. The objective of this project was to develop methods to achieve robust and versatile walking in complex scenarios. A scenario which represents a benchmark environment is depicted in Figure 1.1. The environment is previously unknown to the robot. It consists of a combination of obstacles and platforms and of unknown disturbances as, for example, terrain with unknown characteristics or obstacles which are too small to be detectable.

In the context of this project, *versatility* is used as a term to describe the robot's characteristic to exploit its physical capabilities. Methods are developed which allow for large strides, step over obstacles and step onto platforms and stairs. In addition to *versatility*, the project assumes *robustness* as an important characteristic in complex scenarios in real-world applications. The robot should be able to recover from very large disturbances, due to external forces or errors in the environment model. The project set special effort in developing not methods for particular tasks, but a framework which allows for application in scenarios which demand *versatile* and *robust* walking simultaneously. The methods were verified in simulations and validated in experiments with the robot LOLA which has been developed in the predecessor project *Biological and Technical Aspects of Intelligent Locomotion* [67]. It is depicted in Figure 1.2. In January 2017, the project's results were publicly presented live in front of researchers from all over Germany[4].

[3]Videos showing bipeds of the company *Schaft* - `https://youtu.be/diaZFIUBMBQ` - and *Boston Dynamics* - `https://youtu.be/oK9SfUJg1_Y`.

[4]A video showing the presented experiments is available at `https://youtu.be/g6UACMHgt20`. The presented slides are available as well at `goo.gl/GuReQu` on the project side.

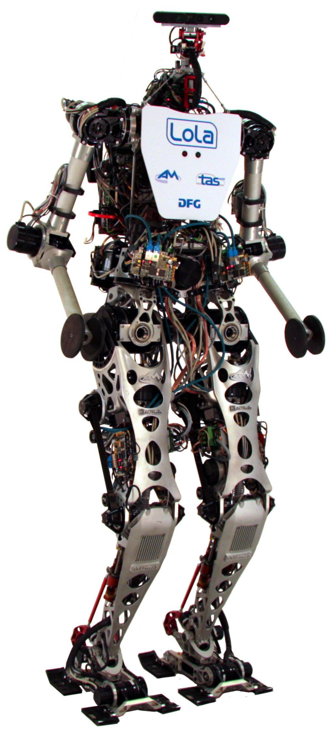


Figure 1.2: Photo of the humanoid robot LOLA.

1.3 Contributions and Outline of this Thesis

In the context of the previously mentioned DFG-Project *Walking in Uneven Terrain*, the objective of the work presented in this thesis is to develop methods which allow for versatile bipedal walking in complex scenarios. The particular contributions of this thesis are the following:

- an approach is developed to generate motions for serial, redundant robots over long sequences. The approach combines discrete optimization, parameter optimization and methods for trajectory optimization. Applications of this method to industrial and agricultural robots are discussed. In addition to bipedal locomotion, the approach has been partly validated with an agricultural robot and has been extended in collaboration with the *Honda Research Institute Europe* to be applicable on an industrial robot.

- methods are proposed for autonomous navigation of bipeds in dynamic and unknown environments. The methods use discrete optimization techniques to search for footholds in cluttered and dynamic environments. Local gradient based optimization methods and a mobile platform planner are integrated and analyzed to accelerate computational times. The approach is highly applicable in parallel software design.

- parameter and trajectory optimization methods are applied to a highly redundant robotic system, the humanoid robot. The proposed methods are real-time methods which respect dynamic constraints and exploit the physical capabilities of bipedal robots to allow for versatile walking in cluttered environments.

- all methods are integrated in a framework for versatile and robust walking. It combines methods for perception and stabilization. It has to be emphasize that all methods work simultanously allowing for walking in complex scenarios including external disturbances and unknown terrain.

- the methods presented in this thesis have been validated in various experiments with the bipedal robot LOLA. The robustness of the framework has been proven in multiple public demonstrations.

The contributions of this thesis are presented as follows: Chapter 2 describes the problem analyzed in this thesis. It presents the solution approach using a motivational example and discusses application to a wide range of serial redundant robotic systems. Chapter 3 presents how the methods discussed in Chapter 2 can be applied to the humanoid robot LOLA and how they can be integrated in an overall framework for versatile and robust walking. Chapter 4 and Chapter 5 presents the methods in more detail with respect to their application on bipedal walking. Furthermore, the methods are analyzed in simulation. Chapter 4 presents various methods for autonomous navigation in unknown and dynamic environments while Chapter 5 presents parameter and trajectory optimization methods. In Chapter 6 selected experiments validate the performance of the methods. Finally, Chapter 7 is devoted to a summary, a discussion and recommendations for future work.

Chapter 2

Motion Planning for Redundant Robots

Motion planning for robotic systems is a wide and active research field. Nevertheless, most developed motion planning methods are developed for specific hardware configurations and are therefore not easily adaptable to other designs or applications.

This section presents the methods developed within this thesis in the context of motion planning research. It defines the application and its challenges for motion planning algorithms. While this thesis focuses on motion planning methods for bipedal walking of humanoid robots as the most relevant application, an abstract example is used to emphasize the general characteristics and applications to further robotic systems.

Section 2.1 describes the overall strategy for solving motion planning problems applied in this thesis and puts it in the context of current research. In Section 2.2, the methods are presented and analyzed using a minimal model. When implementing planning algorithms for real systems, sensor feedback is used to modify the ideal planned motions. In this case, the executed motion does not coincide with the planned one. Subsection 2.2.3 presents methods to reactivly adapt ideal planned motions. Finally, Section 2.3 is devoted to a conclusion.

2.1 Problem Description

In this section, the problem class to which this thesis provides an approach is presented. The problem class can be visualized as depicted in Figure 2.1:

Based on a command from a human operator or a high-level unit, a robot has to reach subsequently a series of way points with its tool center point (TCP) (see Figure 2.1 for a motivational example). The way points themself are not known in advance, but have to be determined based on the task the robot has to solve. The motions of the robot are constrained by the environment which is cluttered with obstacles. These obstacles are not fully known in advance and their approximation may change during the execution of the motions due to sensor inaccuracies or obstacle movements. The combination of the environment and the task, the robot has to carry out, is named "motion situation" in this thesis. Additionally to the discrete way points and the environment, other constraints may be imposed during the whole motion.

One of the problems introduced by this example is how the robot's motion can be planned and executed to fulfill the asked goals. For this thesis, the motion planning problem is narrowed to the calculation of desired trajectories on joint level excluding the tracking control. This motivational example poses the following requirements on the motion planning: the way points, which the robot has to reach, have to be calculated based on the robot's specific task. The final motion of the robot connecting the way points represents long trajectories on joint level. Since the environment model changes during execution of the ideal planned motion, the motion planning has to be able to quickly react

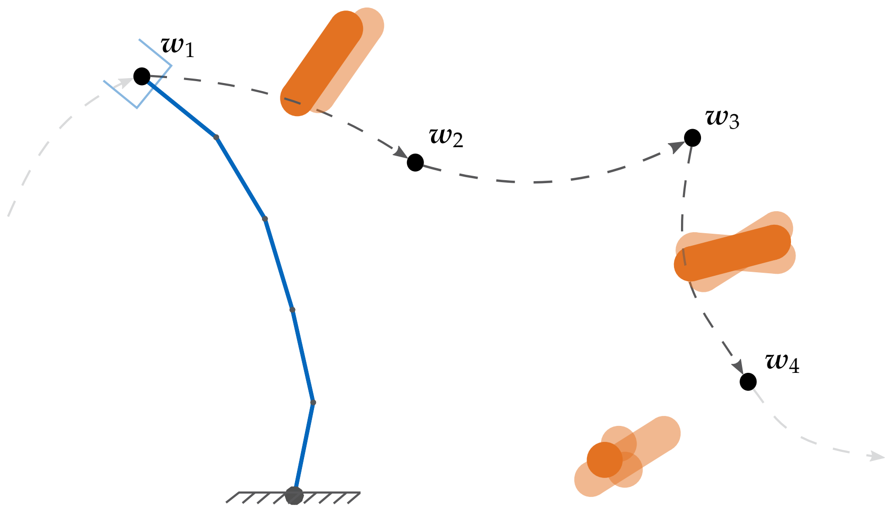


Figure 2.1: Redundant robot in blue, parameterized task-space trajectory in grey with unknown way points w_i, obstacle in orange. Blurry obstacles are not fully known in advance.

to these changes. Finally, the redundancy of the robot, given by a number of DoF which is higher than the imposed constraints, provides an optimization potential. This potential has to be exploited by the motion planning to allow for complex movements which are especially necessary in challenging environments.

This abstract problem description is true for motion planning for bipedal walking in unknown environments.

- Humanoid robots represent a highly redundant type of robot.
- Humanoid robots have to set their feet subsequently on the ground, while the foothold positions have to respect geometrical constraints. These foothold positions are equivalent with the previously introduced way points the redundant robot has to reach with its TCP.
- Walking a distance of a couple of meters represents already a long motion sequence on joint level.
- The restriction of stable walking ("not falling down") can be expressed as a task space constraint on the upper body and the CoG.
- Since the humanoid moves continuously, it approaches objects which haven't been known in advance due to the robot's limited Field of View (FoV) (see Figure 2.3). The objects approximation with respect to the robot changes during movement out of several reasons: the odometry of the robot is not exactly known, because of accumulated measuring errors, sliding on the ground over long walking sequences or unknown contact states with the ground. The visual sensors' accuracy highly depends on the distance between sensor and object. Therefore, approaching objects will result in a better and changed approximation. Additionally, not only static objects exist in real-world environment, but also dynamic moving objects.
- Furthermore, humanoid robots are expected to interact with humans in real applications. These interactions may result in constantly changing commands for the motion planning. The combination of environment and commands is refered to walking situation in the style of motion situation. Similar to the motivational example, a fast re-planning of the robot's motions has to be possible due to the constantly changing walking situations.

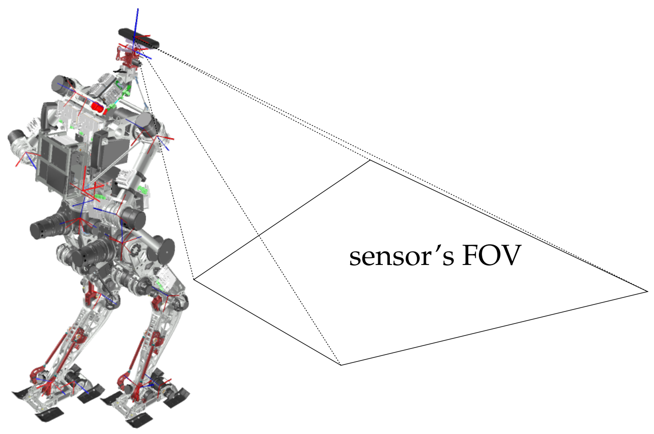


Figure 2.2: Lola's FoV [110].

Humanoid robots are still mainly a research topic, but the motivational example is also applicable to industrial scenarios.

On the example of the EuRoC, Hildebrandt et al. [121] and Wahrmann et al. [127] discuss an industrial application. In the challenge, a manipulator fixed on an actuated platform had to perform a sequence of pick and place tasks in an environment shared with humans. This corresponds to typical tasks in *Industrie 4.0* applications (see Figure 2.3). From a motion planning point of view, the tasks of the challenge can be reduced to the motivational example. First a sequence of way points has to be determined at which the robot can pick and place objects with its TCP. Then, the robot has to follow the sequence to pick and place the objects. While executing the motion, the environment may change, for example, by the presence of humans[1].

A very similar application is the harvesting of fruits as investigated at the AM within the Clever Robots for Crops (CROPS) project [93]. Figure 2.3 shows a human tele-operating the harvesting robot. A redundant manipulator has to detect and pick fruits. This represents a sequence of way points, which are connected by the movements of the robot. The environment in which the manipulator works is highly cluttered due to leafs and stems. Therefore, parts of the environment can only be reached if the robot is interacting with the environment.

2.1.1 Related Work

The research field of motion planning for robotic systems offers multiple approaches for the described problem class. Choset et al. [24], LaValle [63], Nakamura [75], and Siciliano et al. [100] provides a thorough overview. Out of the multitude of different methods, two basic approaches can be identified for generating the motion of redundant manipulators:

(1) The search of collision-free paths directly in the joint space, often called configuration space, of the robot via *sampling-based* methods. These methods search for executable paths by discretizing the joint space. The discrete configurations are checked for their feasibility. Once a path is found, the joint trajectories are determined by applying trajectory optimization techniques. Only in the second step, the time-dependence of the trajectories is taken into account. Since the dimension of the joint space depends on the number of joints, it represents a high dimensional search space for redundant manipulators. Therefore, mainly methods based on probabilistic-sampling approaches are suitable for application. Examples are the Probabilistic Roadmap or the Rapidly-exploring Random Trees presented by LaValle [62] resp. by Kavraki et al. [55], among others. Ioan A. Sucan,

[1]A video showing scenarios at the EuRoC is published at `https://youtu.be/OTWEZd6BMk8`.

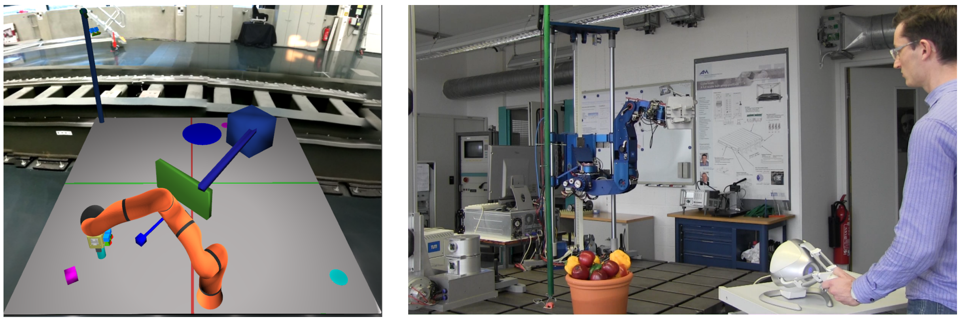


Figure 2.3: Left: A manipulator is performing a pick and place task at the European Robotics Challenge (EuRoC) (Adapted from [121]). Right: Human operator is tele-operating a harvesting robot [141].

Mark Moll [48] provides an open source library (OMPL) with a wide range of probabilistic planners. A framework for motion planning was presented by Chitta et al. [23]. It uses the OMPL and connects it with further open source libraries, for example for collision checking [85]. The probabilistic character of the sampling-based motion planning methods limits their usage in real-time applications. In general, the goal path is not optimal and the search can not be aborted while providing a goal directed sub path.

(2) In contrast to the searching directly in configuration space, a second possibility is to first plan the robot's motion in task space. In a subsequent step the task space trajectories have to be mapped into configuration space by solving the Inverse Kinematics (IK) and solve the redundancy problem. The main difficulty in this approach seems to be the design of appropriate task space trajectories. They have to respect the robot's kinematic and dynamic capacities while enabling collision-free motions. Siciliano et al. [100] and Nakamura [75] give an overview over trajectory design and methods for mapping task space trajectories in configuration space and vice versa. This approach is advantageous for most real-time applications. The search for a collision-free task space trajectory can be done in a lower dimensional task space. Additionally, not only the path but the whole trajectory is determined. The IK can be calculated using efficient numerical methods with low computational effort, as for example presented by Klein et al. [57]. In many applications the IK do not have to be solved before execution of the motion, but can be calculated during run-time. The approach is limited in complex environments. Determining task space trajectories without knowledge about the robot's kinematics may lead to non-executable motions in complex environments. Nevertheless, in many applications simple heuristics are sufficient to find executable task space trajectories.

Once executable trajectories, in configuration space or in task space, have been found, optimization methods can be applied to successively optimize the motions. An overview of optimization methods is given in J. Nocedal and S. J. Wright [49].

The discussed approaches represent possibilities to solve the motion sequence as presented in Section 2.1. Nevertheless, calculating long sequences requires high computational effort. Instead, hierarchical methods are proposed. For example, Kaelbling et al. [50] and Sacerdoti [91] propose strict separating of sequence planning and the robot's motion planning. The work presented in this thesis follows a similar idea as outlined in the following.

2.1.2 Solution Approach

In this thesis, a hierarchical approach is applied to solve the motion planning problem described in Section 2.1. The strategy is based on the application of hierarchical ordered motion planning modules. They all use model-based approaches to predict and plan the

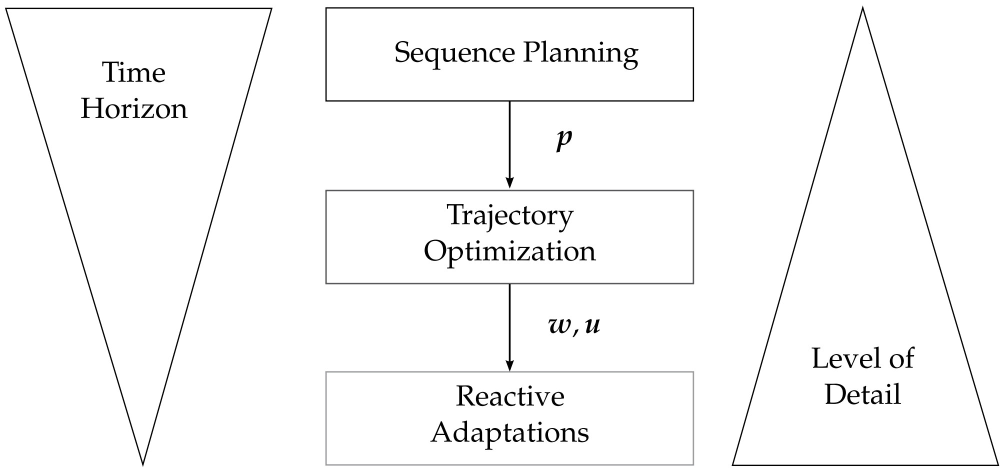


Figure 2.4: Hierarchical Approach for Motion Generation: the three modules *Sequence Planning*, *Trajectory Optimization* and *Reactive Adaption* and their output resp. input, the parameter sets $\boldsymbol{p}$, the task space trajectories $\boldsymbol{w}$ and the nullspace input $\boldsymbol{u}$. The output of the *Reactive Adaption* is the reference motion, which the robot tries to follow.

robot's motion. Figure 2.4 gives an overview of the modules. The different layers use different levels of detail for the robot model as well as for the motion representation to describe the motion planning problem. Due to the calculation time, the higher levels are therefore able to take a longer time horizon into account than the subsequent layers, but with less accurate models.

In a first step, the way points of the motion are calculated. The corresponding motion planning layer is called *Sequence Planning*. It uses simplified models of the robot and approximates the robot's motion. That way, large parts of the environment and long sequences can be evaluated. The result of the *Sequence Planning* is a set of parameters $\boldsymbol{p}$ describing the way points and, roughly, the motion in between.

In a second step, the actual robot's motion on joint level are calculated by a layer named *Trajectory Optimization*. The way points $\boldsymbol{p}$ are connected via splines $\boldsymbol{w}(\boldsymbol{p})$. The robot's motion on joint level are calculated by using $\boldsymbol{w}(\boldsymbol{p})$ as task space trajectories and exploiting the robot's redundancy. By applying $\boldsymbol{w}(\boldsymbol{p})$ as task space trajectories the search space for the joint motions is reduced. Both, task space trajectories as well as the robot's nullspace motion can be modified to optimize the robot's motion in respect to a user chosen criteria. Also the way points may be further optimized by taking the full robot motion into account as long as the specific tasks can be full-filled. Depending on the available calculation time, the time horizon of the *Trajectory Optimization* can be chosen. The result of the *Trajectory Optimization* is an ideal planned motion over a time horizon of parts of the sequence.

In a third step, a layer called *Reactive Adaption* modifies the robot's motion at each control cycle of the robot. On the one hand, this is necessary to be able to react to sensor feedback. On the other hand, it can be interpreted as an additional more detailed layer. Taking the *Reactive Adaption* into account already in the *Trajectory Optimization* layer provides further optimization potential in addition to the discrete parameters. The result of the *Reactive Adaption* is the reference motion, which the robot tries to follow.

The main motivation behind this approach is the assumption that the necessary level of detail for long motion sequences in terms of modeling and motion planning strongly depends on the time horizon. If it is assumed that the environment (or more generally the current motion situation) may change during execution, a complete calculated motion plan for the robot over the whole sequence may provide at advanced time instances only an imprecise motion. Furthermore, it may even not be executable in dynamic environments.

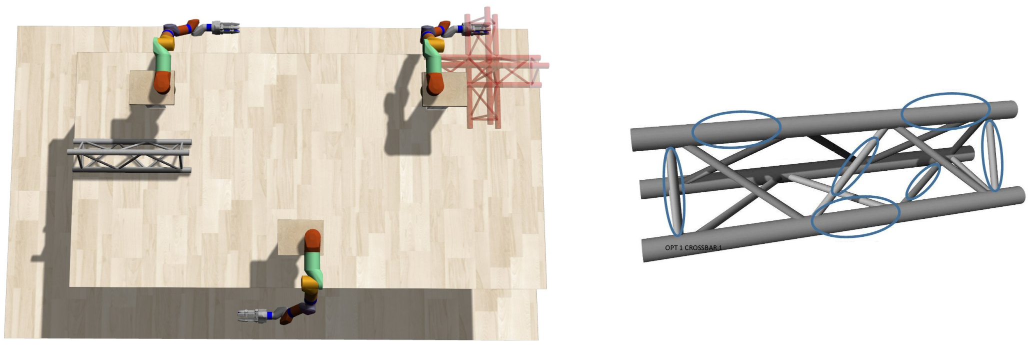


Figure 2.5: Example of sequence planning for collaborative working robots (left) with discretized truss (right). Adapted from [142].

Consequently, the optimality of a motion plan on joint level describing the robot's motion with high levels of detail is at least questionable. A drawback of the applied approach is inherent to the hierarchical structure. By determining constraints on successive layers the optimization potential of the subsequent layers is limited. Thus, this hierarchical approach will never find the overall optimal solution.

2.2 Motivational Example - Predictive Kinematics

In this section, the solution approach discussed above is presented using the motivational example as depicted in Figure 2.1. Following the order of the hierarchical layers (see Figure 2.4), first the application of the *Sequence Planning* to the motivational example is discussed. This thesis focuses on the sequence planning for humanoid walking. A generalization to a wider range of robotic systems has been developed in cooperation with Dr.-Ing. Michael Gienger and Benjamin Kammermeier [117, 142] (see Figure 2.5).

Second, the presented methods for *Trajectory Optimization* are reviewed and compared. Methods for parameter optimization and redundancy exploitation are presented. Furthermore, these methods, *Sequence Planning* and *Trajectory Optimization*, are combined to a novel method for motion optimization. Third, different possibilities are discussed for reactive adaptions of the ideal planned motions.

2.2.1 Sequence Planing

The *Sequence Planing* will be discussed in more detail in Chapter 4 with the application of bipedal walking. Here, only an overview of the used approach is given. Although, a generic approach was developed for redundant robots [142], the *Sequence Planing* still strongly depends on the particular application[2]. The presented approach is based on the idea of reducing the planning problem to the search of multiple discrete configurations. Depending on the application, the environment is segmented into different kind of objects. The objects are described by characteristics, which are, for example, their DoFs. The objects itself are discretized into areas at which the manipulator could get in contact with the objects. In Figure 2.5 a truss is visualized which has to be picked and placed by cooperative working robots. The struts of the truss are the areas at which the robots could carry it. They are discretized into a set of possibilities which are used in the sequence planning. Using the same idea, the surface where the truss can be placed would be discretized into a set of areas. The manipulator is described by its kinematic limits. They limit the number of contact possibilities the robot can reach from its current configuration. Its motions,

[2]Additionally, the algorithm [142] in its current implementation can not be applied to real-time applications.

which connect the discrete configurations, are approximated by heuristics. Having a set of all discrete configurations, a search tree with the discrete configurations as states and the approximated robot motions as actions can be set up. This representation allows to apply effective search algorithms to search for desired sequences. At this point, the same algorithms can be used for all applications. The difference in the applications lies in the identification of an abstract problem description which is suitable for the particular application. To simplify the subsequent planning, additional configurations as way points can be introduced to the robot's motion providing start solutions. Referring to Figure 2.1, the result of the *Sequence Planning* are the parameter sets $\boldsymbol{w}_1, \boldsymbol{w}_2, \boldsymbol{w}_3$ and $\boldsymbol{w}_4$. In this example, the discrete configurations are reduced to points in task space that the robot has to reach with its TCP.

2.2.2 Trajectory Optimization

The *Trajectory Optimization* calculates the motion of the robot necessary to reach the points calculated by the *Sequence Planning*. As discussed in Section 2.1, this is a task often posed for serial robots to reach with their end-effector a sequence of points in space. The methods' application to bipedal walking is presented in Section 5.3. Without the loss of generality, the robot's motion is only considered from some point A to another point B in the following. One way to generate the robot's motion is to describe the path of the end-effector from A to B via trajectory primitives, for example polynomials, in task space. The trajectories $\boldsymbol{w} = \boldsymbol{w}(\boldsymbol{p})$ are configurable with the parameters $\boldsymbol{p}$, which can be adapted according to the overall desired motion. For redundant robots, the joint space's dimension n is larger than the task space dimension m ($n > m$). This kinematic characteristic needs to be resolved during motion generation. Common approaches such as the *Resolved Motion Rate Control* [113] or *Automatic Supervisory Control* [1] can be used to solve the redundancy on velocity level at each discrete time step t_i. Using these approaches, the inverse kinematics problem can be expressed as

$$\dot{\boldsymbol{q}}(t_i) = \boldsymbol{f}(\boldsymbol{q}(t_i), \boldsymbol{u}(t_i), \boldsymbol{p}, t_i) \tag{2.1}$$

$$= \boldsymbol{J}_w^{\#} \dot{\boldsymbol{w}}(\boldsymbol{p}, t_i) - \alpha \boldsymbol{N} \boldsymbol{u}(t_i) \tag{2.2}$$

$$\boldsymbol{q}(t_b) = \boldsymbol{q}_0,\ t_i \in [t_b, t_e] \tag{2.3}$$

with the weighted Moore-Penrose-Pseudo-Inverse of the Jacobian matrix $\boldsymbol{J}_w$ of $\dot{\boldsymbol{w}}$ with respect to $\dot{\boldsymbol{q}}$ defined as

$$\boldsymbol{J}_w^{\#} := \boldsymbol{W}^{-1} \boldsymbol{J}_w^T (\boldsymbol{J}_w \boldsymbol{W}^{-1} \boldsymbol{J}_w^T)^{-1} \tag{2.4}$$

with the weighting matrix $\boldsymbol{W}$. The nullspace projection matrix $\boldsymbol{N}$ is defined as $\boldsymbol{N} = \boldsymbol{I} - \boldsymbol{J}_w^{\#} \boldsymbol{J}_w$. For the sake of simplicity, the time dependency is omitted in the following. The joint angles resp. angular velocities of the robot are denoted with $\boldsymbol{q}, \dot{\boldsymbol{q}}$. The nullspace input, which can be freely chosen, is $\boldsymbol{u}$ and $\alpha \in \Re$ is a weighting factor. The advantage of this approach compared to a complete generation of motion directly in configuration space is a reduction of complexity. First, the parameterized motion of an end-effector can be generated in a more meaningful way in task space and second, redundancy can be exploited for local optimality without interfering with the end-effector motion. Due to the reduced dimension of the search space the computational complexity can also be reduced, which makes this approach applicable in real-time applications such as, for example, dynamic bipedal walking of humanoids with their high number of DoF or redundant serial manipulators. In the following, several methods are discussed to determine the free variables $\boldsymbol{u}$ and $\boldsymbol{p}$.

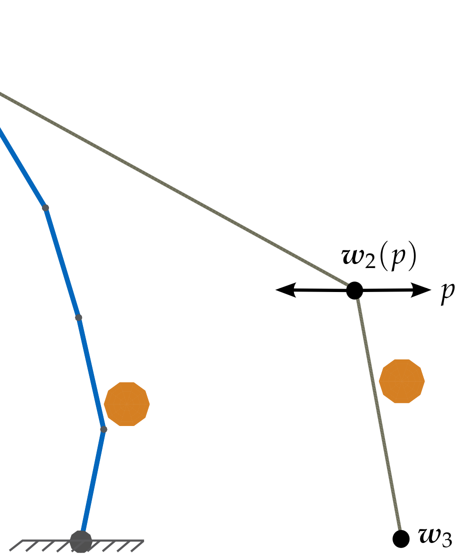


Figure 2.6: Model: redundant robot in blue, parameterized task space trajectory in grey, obstacles in orange.

Application Scenario

For the sake of simplicity, the motion sequence of the 5-DoF manipulator of the illustrative example is reduced to a planar motion. It is described by two straight trajectories which connect three points w_1, w_2 and w_3 in an environment with two obstacles (see Figure 2.6). Thereby, w_2 can be interpreted as a path point whose horizontal coordinate can be configured by a parameter p. Here, the parameter dependency of w_1 and w_3 is neglected. In this case, the desired task space trajectory $w_d(t)$, defined by w_1, $w_2(p)$ and w_3, is supposed to be collision free. Additionally the robot's links may collide with the obstacle, which has to be prevented. The timing of $w_d(t)$ is defined via the fixed time instances t_1, t_2, t_3 at the points w_1, $w_2(p)$ and w_3.

In the following, the motion of the robot is calculated by taking different characteristics of the robot and the environment into account. First, an initial solution is calculated for a planned motion (*Local Kinematics*). Then, the path is modified to take characteristics of the environment into account (*Task Space Optimization*). Finally, the redundancy of the robot is exploited to increase maneuverability (*Redundancy Exploitation*). Furthermore, these methods are all combined (*Task Space Optimization & Redundancy Exploitation*).

Local Kinematics

A. Liegeois [1] proposed to choose the input $\boldsymbol{u}$ as the gradient of a cost function to exploit the robot's redundancy. The inverse kinematics of (2.1) lead to

$$\dot{\boldsymbol{q}} = \boldsymbol{f}(\boldsymbol{q}, \boldsymbol{u}) = \boldsymbol{J}_w^{\#}\dot{\boldsymbol{w}} - \alpha \boldsymbol{N}\frac{\partial L}{\partial \boldsymbol{q}} \tag{2.5}$$

$$\boldsymbol{q}(t_b) = \boldsymbol{q}_0. \tag{2.6}$$

$\frac{\partial L}{\partial \boldsymbol{q}}$ is the gradient of a cost function defined as

$$L = c_{coll,self}L_{coll,self} + c_{coll,obst}L_{coll,obst} + c_{vel}L_{vel} + c_{cmf}L_{cmf}. \tag{2.7}$$

L is the sum of cost functions which penalizes collisions of the robot links with themselves $(c_{coll,self}L_{coll,self})$ and obstacles $(c_{coll,obst}L_{coll,obst})$ as well as high angular velocities $(c_{vel}L_{vel})$.

The cost function $c_{cmf}L_{cmf}$ is used to define a comfort pose [98]. This formulation also applies to the inverse kinematic module of the walking control system of the humanoid robot LOLA. In the following, (2.5) and (2.6) are referred to as *Local Control*.[3]

Task Space Optimization

In most robotic applications, it is not important that the end-effector exactly follows a pre-defined task space trajectory, but that the end-effector moves from point A to point B or, in the motivational example, from the point $\boldsymbol{w}_1$ to $\boldsymbol{w}_3$. Following this line of thought, the horizontal coordinate of the second point $w_{2,x}(p)$ is chosen to be dependent on a single parameter p (see Figure 2.6). Any parametric movement representation $\boldsymbol{w}(t, \boldsymbol{p})$ could be applied to the presented method, also with more than a single parameter, but this is considered to be one of the most illustrative ones. Thus, the task space trajectory depends on p. The parameter p can be optimized to improve the robot's motion. The optimization problem is formulated as follows

$$\min_{\boldsymbol{p}} \phi(\boldsymbol{q}) = g(\boldsymbol{q})|_{t_e} + \int_{t_b}^{t_e} L(\boldsymbol{q})\mathrm{d}t \tag{2.8}$$

$$\dot{\boldsymbol{q}} = \boldsymbol{f}(\boldsymbol{q}, \boldsymbol{p}) = \boldsymbol{J}_w^{\#}\dot{\boldsymbol{w}}(\boldsymbol{p}) - \alpha \boldsymbol{N}\frac{\partial L}{\partial \boldsymbol{q}} \tag{2.9}$$

$$\boldsymbol{q}(t_b) = \boldsymbol{q}_0 \tag{2.10}$$

with $g(\boldsymbol{q})|_{t_e}$ denoting a cost term at $t = t_e$. As described in Hildebrandt et al. [118] the optimization problem can be solved using a gradient method for the parameter $\boldsymbol{p}$. The value of the cost function $\phi(\boldsymbol{q})$ is evaluated by numerically integrating the robot's motion, described by (2.9) and (2.10) for $\boldsymbol{p} = \boldsymbol{p}_k$ over the whole sequence. The forward Euler integration scheme is used including a drift control. The updating rule

$$\boldsymbol{p}_{k+1} = \boldsymbol{p}_k - h_{opt}\nabla_{\boldsymbol{p}}\phi(\boldsymbol{q}) \tag{2.11}$$

with the updating step-size h_{opt} is used to optimize $\boldsymbol{p}$ with respect to $\phi(\boldsymbol{q})$ after each evaluation k. In this context $\nabla_x(\cdot)$ is the gradient of $(\cdot)$ with respect to a variable x. It is calculated by a finite difference using a small delta around the current x. A back-tracking algorithm determines h_{opt} to improve the performance of the optimization process [49]. As long as the Armijo condition

$$\phi_{k+1}(\boldsymbol{q}) \leq \phi_k(\boldsymbol{q}) - c\,\|\nabla_{\boldsymbol{p}}\phi_k(\boldsymbol{q})\|^2\,h_{opt} \tag{2.12}$$

is violated, the updating step-size is decreased by

$$h_{opt} \leftarrow \rho\, h_{opt} \tag{2.13}$$

and $\boldsymbol{p}_{k+1}$ is recalculated using (2.11). The parameters c and $\rho \in (0, 1)$ are freely chosen. Figure 2.7 shows three snapshots of the resulting task space trajectory. The parameter p is optimized such, that the links of the robot are shifted away from the obstacle which is equivalent to a reduction of collision costs. Table 2.1 lists the optimized values.

Redundancy Exploitation

In Schuetz et al. [94], a model-predictive approach based on an indirect optimization is presented to solve the inverse kinematics (see (2.9)) and to exploit the nullspace of

[3]The terms *local* and *global* are used with respect to the considered time horizon of the motion.

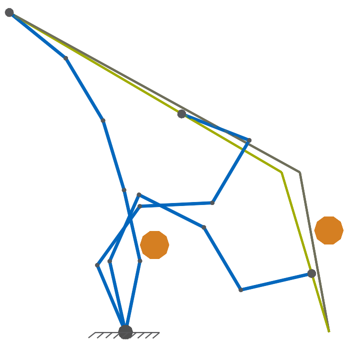


Figure 2.7: Task Space trajectory before (gray) and after parameter optimization (green)

redundant manipulators. In contrast to direct optimization methods, it respects the run-time constraints of the application as discussed in [93].

Instead of using a discrete number of parameters as optimization variable, a continuous input $\boldsymbol{u}_{opt}$ is searched as proposed in Nakamura et al. [76]. It replaces the local-acting gradient $\frac{\partial L}{\partial \boldsymbol{q}}$. It follows

$$\min_{\boldsymbol{u}} \phi(\boldsymbol{q}(\boldsymbol{u})) = g(\boldsymbol{q}(\boldsymbol{u}))|_{t_e} + \int_{t_b}^{t_e} L(\boldsymbol{q}(\boldsymbol{u}))\mathrm{d}t \tag{2.14}$$

$$\dot{\boldsymbol{q}} = \boldsymbol{f}(\boldsymbol{q}, \boldsymbol{u}) = \boldsymbol{J}^{\#}\dot{\boldsymbol{w}} + \alpha \boldsymbol{N}\boldsymbol{u} \tag{2.15}$$

$$\boldsymbol{q}(t_b) = \boldsymbol{q}_0 \tag{2.16}$$

As proposed in Schuetz et al. [94], the optimization problem is solved applying *Pontryagin's Minimum Principle*. It leads to an optimization problem for $\boldsymbol{u}$ which is solvable with gradient methods. Furthermore, the formulation is extended from velocity to acceleration level. That way, discontinuous angular velocities are avoided. Figure 2.8 illustrates the resulting motion. Treating the nullspace control as a global optimization problem leads to a more predictable motion, which is superior in avoiding the obstacle located near the kinematic links of the robot.

Task Space Optimization & Redundancy Exploitation

Methods for parameter optimization of task space trajectories and methods for optimization of a continuous input vector to exploit the robot's nullspace have been presented. Having a parameterized task space trajectory and the continuous input vector $\boldsymbol{u}$ the combined optimization problem results in

$$\min_{\boldsymbol{u},\, \boldsymbol{p}} \phi(\boldsymbol{q}(\boldsymbol{u})) = g(\boldsymbol{q}(\boldsymbol{u}))|_{t_e} + \int_{t_b}^{t_e} L(\boldsymbol{q}(\boldsymbol{u}))\mathrm{d}t \tag{2.17}$$

$$\dot{\boldsymbol{q}} = \boldsymbol{f}(\boldsymbol{q}, \boldsymbol{u}, \boldsymbol{p}) = \boldsymbol{J}^{\#}\dot{\boldsymbol{w}}(\boldsymbol{p}) + \alpha \boldsymbol{N}\boldsymbol{u} \tag{2.18}$$

$$\boldsymbol{q}(t_b) = \boldsymbol{q}_0 \tag{2.19}$$

Bocek [10] presented an optimization formulation to simultaneously optimize a discrete parameter describing the system and a continuous system input. This formulation is applied as follows to combine the previously presented methods: by approaching the computation of the inverse kinematics as the combined optimization problem stated in equation (2.17) - (2.19), the Hamilton function is defined as:

$$H(\boldsymbol{q}, \boldsymbol{\lambda}, \boldsymbol{u}, \boldsymbol{p}) = L(\boldsymbol{q}(\boldsymbol{u})) + \boldsymbol{\lambda}^T \boldsymbol{f}(\boldsymbol{q}, \boldsymbol{u}, \boldsymbol{p}) \tag{2.20}$$

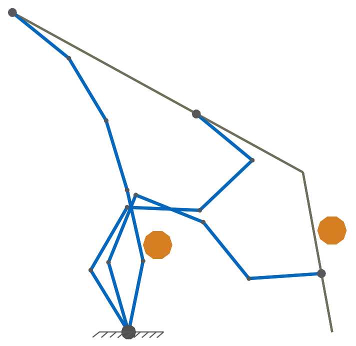


Figure 2.8: Motion with global exploitation of redundancy

with the adjoint variable $\boldsymbol{\lambda}$. In the application of the optimal kinematic planning approach, the nullspace control vector as well as the task space parameterization are constrained variables. Therefore, *Pontryagin's Minimum Principle* is applied to the combined optimization problem. The following optimality conditions are obtained:

$$\dot{\boldsymbol{q}} = \frac{\partial H}{\partial \boldsymbol{\lambda}} \tag{2.21}$$

$$\dot{\boldsymbol{\lambda}} = -\frac{\partial H}{\partial \boldsymbol{q}} = -\frac{\partial L}{\partial \boldsymbol{q}} - \left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{q}}\right)^T \boldsymbol{\lambda} \tag{2.22}$$

$$\boldsymbol{q}(t_b) = \boldsymbol{q}_0 \tag{2.23}$$

$$\boldsymbol{\lambda}(t_e) = 0 \tag{2.24}$$

$$\boldsymbol{u}_{opt} = \arg\min_{\boldsymbol{u}} H(\boldsymbol{q}_{opt}, \boldsymbol{\lambda}_{opt}, \boldsymbol{u}, \boldsymbol{p}_{opt}) \tag{2.25}$$

$$\left[\left(\frac{\partial g}{\partial \boldsymbol{p}}\right)_{t_e} + \int_{t_b}^{t_e} \frac{\partial H}{\partial \boldsymbol{p}} \mathrm{d}t\right]^T \delta p \geq 0 \tag{2.26}$$

The boundary condition (2.23) comes from the given initial position of the robot and those in (2.24) stem from its free final form regarding the nullspace and the parameter. With the conditions (2.21) - (2.24), $\boldsymbol{\lambda}$ and $\boldsymbol{q}$ are calculated for any given nullspace control $\boldsymbol{u}$ and parameter $\boldsymbol{p}$. This allows us to solve the optimization problem by applying a projected conjugate gradient algorithm.

Conjugate Gradient Method

It is well known that the descent gradient algorithm shows slow convergence[4], therefore a nonlinear conjugate gradient method, described in Bocek [10], is applied for the constrained nonlinear optimal control with parameterization. This algorithm provides a computationally efficient solution of the optimal control problem. Furthermore, it has the advantage for real-time applications that the optimization can be aborted at any given time and partial results can be obtained. Thus, the robot can execute an intermediate, but current best, solution. To speed up convergence, the solution of $\boldsymbol{u}(t)$ calculated by the *Local Control* for the nominal parameter $\boldsymbol{p}$ can be used as a good initial solution $\boldsymbol{u}_0$ for the algorithm. The basic algorithm integrates the differential equation of the inverse kinematics (2.18) (or equivalent to (2.21)) forward and the adjoint variable (2.22) backward in time. The input $\boldsymbol{u}(t)$ and the parameter $\boldsymbol{p}$ will be modified according to the gradient with a stepsize β. The correction stepsize may be chosen as either fixed or adaptable by line-search algorithms. A detailed description is shown in Algorithm 1.

[4]See, for example, Schuetz [93].

Algorithm 1 Conjugate Gradient method for combined optimization

```
$i \leftarrow 0$
$\boldsymbol{p}^0 \leftarrow \boldsymbol{p}_{initial}$
Converged $\leftarrow$ false
$\boldsymbol{q}^0 \leftarrow \int_{t_b}^{t_e} \dot{\boldsymbol{q}}(\boldsymbol{u}^0, p^0)\mathrm{d}t$ (using *local control* and (2.23))
$J^0 \leftarrow \int_{t_b}^{t_e} L(\boldsymbol{q}^0(\boldsymbol{u}^0))\mathrm{d}t + g(\boldsymbol{q}(\boldsymbol{u}))|_{t_e}$
**while** Converged $\neq$ true **do**
  $\boldsymbol{\lambda}^i \leftarrow \int_{t_e}^{t_b} \dot{\boldsymbol{\lambda}}(\boldsymbol{q}^i, \boldsymbol{u}^i, p^i)\mathrm{d}t$ (from (2.22) with (2.24), backward in time: $t_e \rightarrow t_b$)
  $\boldsymbol{g}_u^i \leftarrow \frac{\partial H}{\partial \boldsymbol{u}}^i$
  $\boldsymbol{g}_p^i \leftarrow (\frac{\partial g}{\partial \boldsymbol{p}}|_{t_e})^i \; + \; \int_{t_b}^{t_e} (\frac{\partial H}{\partial \boldsymbol{p}})^i \mathrm{d}t$
  **if** $i \neq 0$ **then**
    $\beta^i \leftarrow \frac{\int_{t_b}^{t_e} \boldsymbol{g}_u^i \boldsymbol{g}_u^i \mathrm{d}t}{\int_{t_b}^{t_e} \boldsymbol{g}_u^{i-1} \boldsymbol{g}_u^{i-1} \mathrm{d}t}$
    $\gamma \leftarrow \frac{\boldsymbol{g}_p^i \boldsymbol{g}_p^i}{\boldsymbol{g}_p^{i-1} \boldsymbol{g}_p^{i-1}}$
    $\boldsymbol{s}^i \leftarrow -\boldsymbol{g}_u^i + \beta^i \boldsymbol{s}^{i-1}$
    $\boldsymbol{c}^i \leftarrow -\boldsymbol{g}_p^i + \gamma^i \boldsymbol{c}^{i-1}$
  **else**
    $\boldsymbol{s}^i \leftarrow -\boldsymbol{g}_u^i$
    $\boldsymbol{c}^i \leftarrow -\boldsymbol{g}_p^i$
  $\boldsymbol{u}^{i+1} \leftarrow \boldsymbol{u}^i + \alpha^i \boldsymbol{s}^i$
  $\boldsymbol{p}^{i+1} \leftarrow \boldsymbol{p}^i + \alpha^i \boldsymbol{c}^i$
  $i \leftarrow i + 1$
  $\boldsymbol{q}^i \leftarrow \int_{t_b}^{t_e} \dot{\boldsymbol{q}}(\boldsymbol{u}^i, p^i)\mathrm{d}t$
  $J^i \leftarrow \int_{t_b}^{t_e} L(\boldsymbol{q}^i(\boldsymbol{u}^i))\mathrm{d}t + g(\boldsymbol{q}(\boldsymbol{u}))|_{t_e}$
  Converged $\leftarrow |\frac{J^i - J^{i-1}}{J^{i-1}}| <$ *tolerance*
```

Results

The presented procedures are analyzed in a test case with the model presented before and two obstacles. The task space trajectories are collision-free. Figure 2.9 shows snapshots of the resulting motion using the *local control* and the combined optimization method. The influence of the methods on the parameter and the nullspace is clearly visible. It can be observed that the local character of the *local control* leads to a disadvantageous situation during the second half of the motion. It almost leads to a collision. Contrary to that, the global optimization of both, task space parameter and nullspace control, shows a more *anticipatory* behavior, as expected. Figure 2.10 shows the resulting costs over time for all four method combinations. Table 2.1 shows the resultant parameter and values of the cost functions. Although parameter and nullspace optimization show separately impressive results, the combined optimization is able to reduce the costs even more.

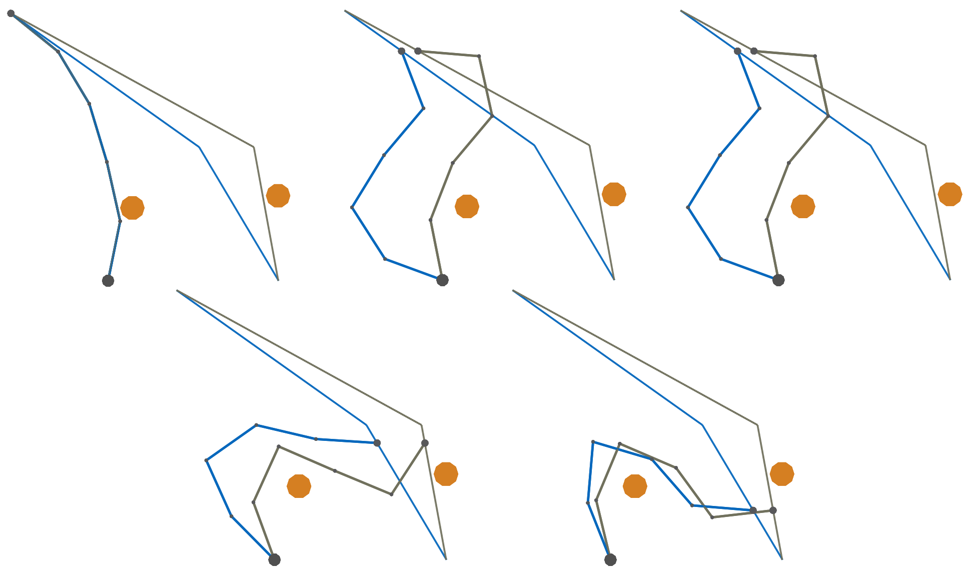


Figure 2.9: Comparison of initial motion calculated with the *Local Control* (grey) and motion with combined optimization (blue). Order is from top left to bottom right.

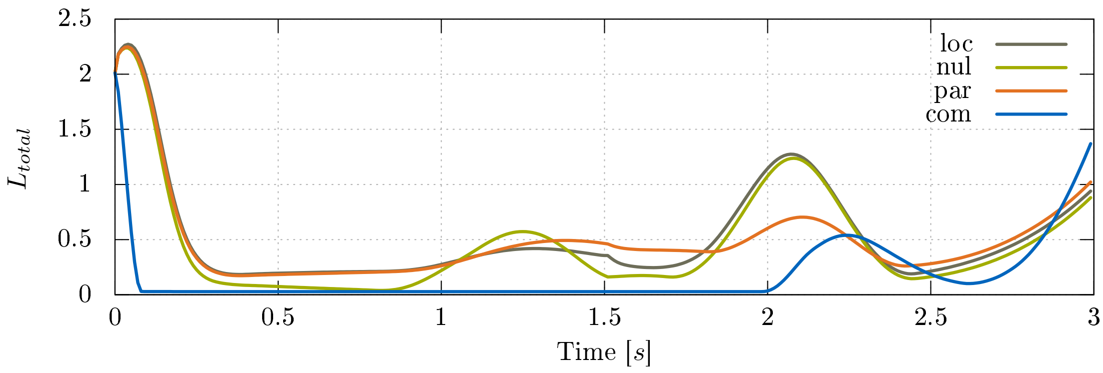


Figure 2.10: Total cost over time for local (*loc*), nullspace (*nul*), parameter (*par*) and combined optimization (*com*).

Table 2.1: Detailed results of optimization process showing integrated cost parts over time. For total cost over time, see Figure 2.10. Reference and initial parameter $p = 60.0$.

	p	L_{vel}	L_{cmf}	$L_{coll,self}$	$L_{coll,obs}$
loc	const.	6.1	9.0	84.7	146.24
par	53.72	6.3	9.3	85.4	136.88
nul	const.	6.5	9.4	84.5	126.27
com	37.49	14.2	12.9	85.2	42.38

2.2.3 Reactive Adaptions

Following the hierarchical approach, the methods presented above first determine a sequence of way points. Then, the robot's motion over the whole sequence is calculated. It is crucial for real application to be able to modify the ideal planned trajectories locally out of two reasons: (1) for long motion sequence, changes in the perceived environment likely occur during execution. Changes in the environment, which was used to generate the reference motion, could result in non-executable trajectories. (2) the motion of the robot is governed by (2.1) as

$$\dot{\boldsymbol{q}}(t_i) = \boldsymbol{J}_w^{\#}\dot{\boldsymbol{w}}(\boldsymbol{p}, t_i) - \alpha \boldsymbol{N}\boldsymbol{u}(t_i) \tag{2.27}$$

$$\boldsymbol{q}(t_b) = \boldsymbol{q}_0, t_i \in [t_b, t_e] \tag{2.28}$$

In this representation, the robot's motion is described by the input vector $\boldsymbol{u}$ and the parameter $\boldsymbol{p}$. The parameter $\boldsymbol{p}$ influences the task space motion $\boldsymbol{w}(\boldsymbol{p}, t_i)$. Thus, the shape of the trajectory strongly depends on the number of parameters. Since the calculation time increases with each additional p_j, the influence of the *Trajectory Optimization* is limited. For this reason, the *Trajectory Optimization* may fail in complex scenarios, because it is not possible to find executable task space trajectories.

In the following, two possibilities are presented to adapt the ideal planned motions. Both, may be interpreted as extensions to the *Local Control* as presented in Subsection 2.2.2. They modify the robot's motion at each control cycle.

Task Space Adaption

One possibility to continuously influence the task space trajectories is to add a term $\boldsymbol{w}_{mod}$ to $\boldsymbol{w}(\boldsymbol{p}, t_i)$ as

$$\dot{\boldsymbol{q}}(t_i) = \boldsymbol{J}_w^{\#}(\dot{\boldsymbol{w}}(\boldsymbol{p}, t_i) + \dot{\boldsymbol{w}}_{mod}) - \alpha \boldsymbol{N}\boldsymbol{u}(t_i) \tag{2.29}$$

$$\boldsymbol{q}(t_b) = \boldsymbol{q}_0, t_i \in [t_b, t_e] \tag{2.30}$$

This term can be designed in such a way that changes in the environment can be taken into account and the shape of the trajectories is modified. Depending on the application, the modified trajectory may have to converge to the original reference trajectory.

Nullspace Adaption

Another possibility is to change the definition of the task space for those sections of the trajectories, where following the exact task space trajectories is not necessary for the success of the motion. By reducing the dimension m of the task space, the dimension n of the nullspace gets larger. Consequently, the input to the nullspace of the robot's motion can change the motion without the necessity of a re-planning of the task space trajectories.

Since the weighted Moore-Penrose-Pseudo-Inverse is defined as in (2.4), simply changing the rank of $\boldsymbol{J}_w$ would result in undesired discontinuous angular velocities (see [70]). Defining $\boldsymbol{J}_{w,k}$ as the column k of the Jacobian related to dimension k of the task space and setting $\boldsymbol{W}$ to the identity matrix, $\boldsymbol{J}^{\#}$ can be written as

$$\boldsymbol{J}_w^{\#} := [\boldsymbol{J}_{w,1}^{\#}, \ldots, \boldsymbol{J}_{w,m}^{\#}] + \boldsymbol{X}_{nl}. \tag{2.31}$$

where the non-linear parts of the pseudo inverse resulting from the coupling of the $\boldsymbol{J}_{w,k}$ are summarized in $\boldsymbol{X}_{nl}$. In order to change the dimension, a trigger vector $\boldsymbol{h} \in \Re^m$ with $\boldsymbol{h} = [h_1, \ldots, h_m]$ is introduced. It enables or disables specific coordinates of the task space. To avoid discontinuities, it is applied as follows

$$\boldsymbol{J}_w^{\#} := [h_1\boldsymbol{J}_{w,1}^{\#}, \ldots, h_m\boldsymbol{J}_{w,m}^{\#}] + \prod_{i=1}^{m} h_i \boldsymbol{X}_{nl}. \tag{2.32}$$

The non-linear parts $\boldsymbol{X}_{nl}$ are calculated for exampple as

$$\boldsymbol{X}_{nl,op} = \boldsymbol{J}^{\#}_{w,op} - \boldsymbol{J}^{\#}_{w,o} - \boldsymbol{J}^{\#}_{w,p}. \tag{2.33}$$

Furthermore, for turning on or off each coordinate i of the task space trajectories, the corresponding trigger value h_i has to be a continuous function from 1 to 0 with continuous boundary values.[5] A similar approach was presented by Mansard et al. [70].

Discussion

Of both presented methods, only the task space adaption has been successfully applied to bipedal walking in simulation as well as in experiments. It is discussed in further detail in Section 5.5. The nullspace adaption was only evaluated in simulation (see Kunze [147]). The task space adaption has the advantage of less computational effort in difference to the nullspace adaption. For the nullspace adaption, at every control cycle step multiple Moore-Penrose-Inverses have to be calculated (see (2.32)). Furthermore, for the transition between two task space dimensions the behavior is not clearly defined.

In contrast to the task space adaption, the nullspace adaption has been implemented in simulation and experiments with the CROPS robot (see the student thesis Dünhuber [139], Jonas Wittmann [141], and Kissel [144])[6]. Both methods do not interfere with the optimization since they modify only the system equations. Therefore, they are consistently be taken into account in the optimization.

2.3 Summary

In this chapter, the problem class to which this thesis tries to provide a solution approach is presented. For the sake of simplification it is reduced to a theoretical example. Nevertheless, application to academia as well as industrial applications are discussed. An outline of the solution approach used in this thesis is given. It is based on a hierarchical planning consisting of three modules, *Sequence Planning*, *Trajectory Optimization* and *Reactive Adaptions*. All three modules are presented and analyzed using a simple motivational example. In the following chapters, in particular in Chapter 5, these methods are applied to biped locomotion.

[5] Further explanation and analysis can be found in Kissel [144], Kunze [147], and Smith et al. [153].

[6] A video of the experiments is available at `https://youtu.be/uDiXij2O5Y4`

Chapter 3

Framework for Versatile & Robust Walking

The methods presented in this thesis were developed in colloboration with Dr.-Ing. habil. Thomas Buschmann, Dipl.-Ing. Robert Wittmann and Dipl.-Ing. Daniel Wahrmann within the DFG-founded project *Walking in Uneven Terrain*. Although application of the methods to different robots are shown throughout this thesis, the main motivation behind it corresponds to the objective of the DFG-project to realize autonomous bipedal walking in unknown terrain. Walking autonomously in unknown environments does not only require real-time motion planning, but also environment perception and stabilization. Developing methods separately from one another will not result in a high-performance overall system. In the worst case, the methods are incompatible and the whole system becomes not suitable for application in complex environment. In the following, the resulting research question of bipedal walking in unknown environments is refered to as *versatile and robust walking*.

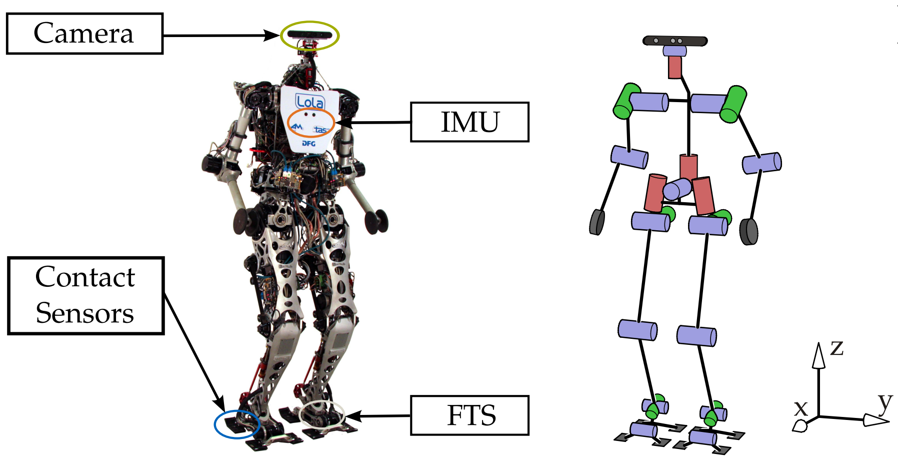
In this chapter, the hardware setup of the experimental platform, the robot LOLA, is presented. Section 3.1 gives an overview of its mechanic and electronic hardware design. Furthermore, the framework for tightly integrating methods for real-time motion planning, environment perception and stabilization is summarized. In Section 3.2, the basic framework based on Buschmann [12] is presented. Three extensions to this common platform are introduced.

A vision system as presented in Wahrmann [110] (Subsection 3.3.3), methods for real-time motion generation as introduced in this thesis (Subsection 3.3.4) and methods for robust walking (Subsection 3.3.5) as presented in Wittmann [114] are summarized. A tight integration of perception, real-time motion generation and stabilization relies on a common environment modeling and the integration of methods for real-time motion generation in the generation of stabilizing motions as presented in Subsection 3.3.2 and Section 3.4. While the basic framework for stable and fast walking is applied on the humanoid robots JOHNNIE and LOLA [12], the extensions for versatile and robust walking are experimentally validated only with the robot LOLA.

3.1 Hardware Overview

The experimental platform used in this work, the humanoid robot LOLA, was developed between 2004 and 2010 within the DFG-funded project-cluster *Natur und Technik intelligenten Laufens*. It is the successor of the humanoid robot JOHNNIE [35]. LOLA's principle design objective was to allow for fast, human-like and autonomous walking.

Its physical dimensions are based on anthropometric data. LOLA weights approximately 60 kg and it is 180 cm tall. Figure 3.1 depicts LOLA and shows its joint distribution. Within this thesis, the redundant kinematic design has to be emphasized. In total, LOLA has 24 actuated joints with high power brushless DC motors: the 7 DoF of both legs and



Joint	DoF
Head	2
Shoulder	2
Elbow	1
Pelvis	2
Hip	3
Knee	1
Ankle	2
Toe	1
Total	**24**

Figure 3.1: Left side: Photo of the humanoid robot LOLA. Sensors for measuring LOLA's state with respect to the environment are highlighted. Right side: kinematic approximation, joint distribution and used world coordinate system. Adapted from Schwienbacher et al. [98].

the 2 DoF in the pelvis gives LOLA a large action radius to execute kinematically complex motions. The 3 DoFs of the arm are mainly used for angular momentum compensation and CoG tracking, which is especially beneficial during fast walking. Actuated toes allow for larger strides. The head is equipped with 2 additional DoFs. These can be used for exploring the environment when walking autonomously.

The overall design of the system follows the paradigm of extremely light-weighted and stiff design to achieve good dynamic performance. Since the legs represent the parts of the robot which execute fast motions while walking, special focus is set on their design: all heavy parts of the legs, in particular the motors, are shifted upwards to reduce inertia effects. Furthermore, the mechanic parts are constructed using prototype casting processes and topology optimization. That way, the stiffness requirements could be met while the weight of the parts could be extremely reduced.

3.1.1 Sensors & Communication System

In Figure 3.1, the sensors measuring LOLA's state with respect to the world and its interaction with the environment are highlighted. The Inertial Measurement Unit (IMU) located in the upper body consists of three fiber-optic gyroscopes and three MEMS accelerometers. The system includes internal sensor fusion algorithms that provide accurate, drift-free measurements for the absolute orientation and rotation rate. Each foot is equipped with a 6-axis force-torque sensors (FTS) to measure the external forces and torques acting on the robot. Furthermore, four contact sensors in each foot are used to detect ground contact.

An Asus Xtion PRO LIVE RGB-D camera[1] is mounted on the head for environment perception. It has a frame rate of approx. 30 Hz. Its FoV reaches approx. 4 m with decreasing precision depending on the distance (see Figure 2.3).

Two encoders, an absolute on the link side and an incremental encoder on the motor side, measure the joint positions. These signals are directly used for independent joint control with servo controllers from Elmo Motion Control[2].

The distributed joint controllers communicate with the central control unit using an Ethercat-bus protocol[3] [126].

The two on-board computers for the vision processing software and LOLA's real-time control are located on its back. Each of the computers has an Intel Core i7-4770S@3.1GHz

[1] ASUS Xtion PRO LIVE, see `http://www.asus.com/Multimedia/Xtion_PRO_LIVE/`
[2] www.elmomc.com
[3] www.beckhoff.de

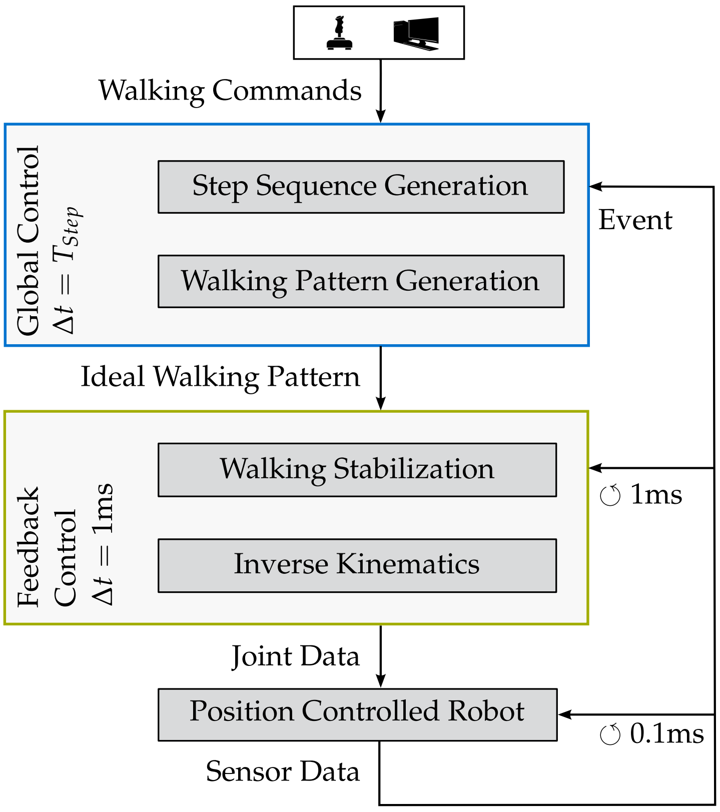


Figure 3.2: Control overview of the framework for stable and fast walking (adapted from Buschmann [12]).

(4x) processor and 8GB RAM. The computer with the vision processing software runs under a Linux OS and the other, on which the walking control is executed, runs under a QNX-RTOS. Both computers use TCP to communicate with each other.

3.1.2 Related Work

The previous section give an overview of the hardware and the sensors of the experimental platform used in this thesis. Dr.-Ing. habil. Thomas Buschmann and Dr.-Ing. Sebastian Lohmeier mainly defined the design objectives and set up the first prototypes of LOLA. Their theses, Buschmann [12] and Lohmeier [67] provide a more detailed description of design and simulation software of LOLA. Dr.-Ing. Valerio Favot did important work in setting up sensors, low-level communication and low-level control. His thesis gives a thorough insight in sensor integration and low-level control [32]. Robert Wittmann describes in his thesis [114] the adjustment of the low-level communication and control on state-of-the-art commercial available hardware.

3.2 Stable & Fast Walking

The framework introduced by Buschmann [12] is the common basis for the methods for *versatile and robust walking*. Its main design objective was to provide a software architecture which allows for fast and stable walking of fully actuated bipedal robots. Figure 3.2 depicts the overall framework. The user's high-level input to the framework is a 2D velocity vector or parameter describing the desired walking pattern. The framework's output are ideal joint data which are tracked by a low level joint control.

The framework itself relies on a hierarchical approach. The *Global Control* determines the *Ideal Walking Pattern* (consisting of the ideal task-space trajectories of the robot) for a time horizon of multiple steps. The subsequent *Feedback Control* locally modifies the *Ideal Walking Pattern* to take into account disturbances and modeling errors. In the following, an insight on the *Global Control* is given, followed by the *Feedback Control* and the *Joint Control* are presented.

Global Control

Before each step, the *Global Control* is executed. Based on the user's command, summarized as *Walking Commands*, the *Global Control* generates the *Ideal Walking Pattern* for the next n_{Steps} steps.

The *Walking Commands* can be expressed as a parameter set $\boldsymbol{p}_{wp}$ for each step which describe the overall walking behavior. It contains, among others, the robot's foothold for each step, parameters describing the robot's movements connecting the footholds (e.g. height of CoG or swing-foot height) and the step time T_{Step}.

The module *Step Sequence Generation* determines, based on the desired p_{wp}, the next n_{steps} foothold positions and parameterizes the stepping motions. This is the input to the *Walking Pattern Generation*-module, which generates the *Ideal Walking Pattern*. In contrast to the step sequence, the *Ideal Walking Pattern* contains time-continuous trajectories. They include the Center of Pressure (CoP) reference trajectories and the ideal task-space trajectories $\boldsymbol{w}_{id}(t) \in \mathbb{R}^m$ which are composed of

- the CoG position,
- the torso orientation,
- the toe angles,
- the foot positions,
- the foot orientations,
- and the pan and tilt angles for the alignment of the camera.

The task-space trajectories are represented as splines which are configurable with p_{wp}.

The horizontal CoG movement is calculated to meet the constraints of dynamic stable walking as follows: Buschmann et al. [14] proposed to approximate the robot's multi-body-system by a three-mass-model. One of the three masses approximates the upper body dynamics and the others represent the legs' dynamics. The mass quantities are a result of an optimization problem to reduce inclination errors of the torso during fast walking. The movements of the masses are defined via the foot trajectories and the vertical CoG trajectories included in the *Ideal Walking Pattern*. Further, the robot's desired CoP trajectory is given as input. Thus, a Boundary Value Problem (BVP) for the desired horizontal CoG trajectories with boundary values at the end of $n_{Steps,COG}$ steps can be stated. Buschmann et al. [14] propose to approximate the horizontal CoG trajectories via cubic splines and solve the BVP using a collocation method. The simplified model enables for model-predictive planning of the whole *Ideal Walking Pattern* within approx. 10ms. The resulting *Ideal Walking Pattern* is the input to the *Feedback Control*.

Feedback Control

The *Feedback Control* uses the *Ideal Walking Pattern* as set points to calculate the desired joint target data $\boldsymbol{q}_d \in \mathbb{R}^n$ and $\dot{\boldsymbol{q}}_d \in \mathbb{R}^n$ for the low-level control. The desired motion and the desired total forces and moments acting on the feet at time t_k, $\boldsymbol{w}_{id,k} = \boldsymbol{w}_{id}(t_k)$ and $\boldsymbol{\lambda}_{id,k} = [\boldsymbol{F}_{id}(t_k), \boldsymbol{M}_{id}(t_k)]$, are modified locally taking sensor input into account. The objective of the *Walking Stabilization* is to stabilize the robot by keeping the upper body in an upright position. Therefore, the measured absolute inclination and inclination rate errors are mapped via a PD-controller to stabilization moments and forces. Using an explicit contact model for ground-foot contact, modifications of $\boldsymbol{w}_{id,k}$ are calculated to follow the stabilization moments and forces. Further details are presented in Buschmann et al. [16]. The modified task-space trajectories $\boldsymbol{w}_k$ resp. $\dot{\boldsymbol{w}}_k$ are input to the *Inverse Kinematics* on velocity-level. The methods presented in [1, 113] are used to solve for the joint space velocities $\dot{\boldsymbol{q}}_{d,k} \in \mathbb{R}^n$ from $\dot{\boldsymbol{w}}_{d,k}$. The robot's redundancies ($m < n$) are exploited at each control cycle to minimize a cost function H_y. Thereby the motion in the nullspace of the robot is used to take into account constraints as self-collision and joint limit avoidance as well as minimization of angular momentum (compare Schwienbacher [96] and Schwienbacher et al. [98]). The calculated $\boldsymbol{q}_{d,k}, \dot{\boldsymbol{q}}_{d,k}$ are then processed by the distributed joint controllers.

Results & Limitations

Buschmann [12] summarizes the performance of the framework. The implementation on the humanoid LOLA allows to achieve fast walking speeds up to 1 $\frac{m}{s}$ and to reliably follow arbitrary Joystick input in real-time. Ewald et al. [31] improved the stability for early ground contacts by introducing an event-based phase switching strategy. Furthermore, Buschmann et al. [15] combined a vision system based on stereo cameras with the presented framework. The vision system does not build up a global map but analyzes the viability of a set of 2D trajectories with respect to the perceived environment. The interface to the walking control consists of, similar to the interface for Joystick control, 2D velocity vectors. This system allows for autonomous walking in unknown environment. Since only 2D velocity vectors guide the robot, it does not exploit the robot's capabilities to traverse obstacles or step onto platforms. Buschmann [12] presents reactive step adaption to allow the robot also to step over obstacles or onto platforms. The methods are restricted to synthetic environments with one obstacle or platform at a time lying horizontal in front of the robot. Furthermore, collision-free trajectories are chosen heuristically and the methods are not coupled with the vision system. The *Walking Stabilization* performs very well for flat terrain and compensates for small disturbances due to modeling errors or irregularities in the assumed flat ground. However, large disturbances which can no longer counteract with the contact forces lead to a falling of the robot.

3.3 Versatile & Robust Walking

The following subsections give an overview of the extended framework for versatile and robust walking and presents the methods discussed in this thesis in the context of a system for bipedal walking in unknown and uneven terrain.

3.3.1 Overview

A control overview of the extended framework is shown in Figure 3.3. The framework as described in Section 3.2 was extended by the following modules:

- A *Navigation-* and *Kinematic Optimization*-module. It plans the kinematic motion of the robot over a time horizon of multiple steps.
- *Trajectory Adaptation*-module. It takes sensor feedback into account to adapt the *Ideal Walking Pattern* to account for disturbances at the planning level.
- The *Feedback Control* is extended by a *Collision Avoidance*-module. It modifies the robot's motion locally to reactively avoid collisions and joint limits.

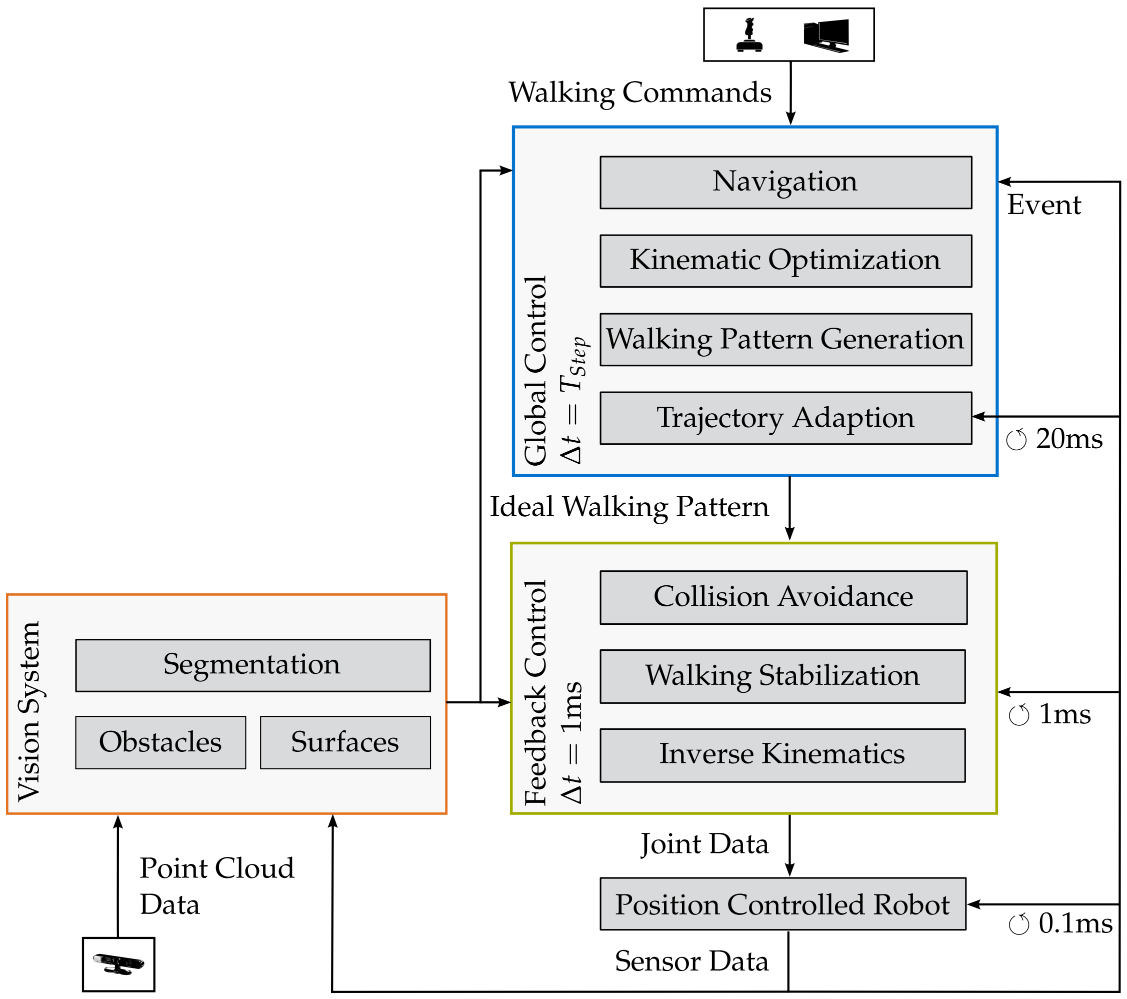


Figure 3.3: Control overview of the framework developed during the DFG-founded project *Walking in Uneven Terrain*.

- To allow for environment perception, a new *Vision System*-module is set up. It approximates the environment as 3D objects.

The developed modules may serve as standalone modules and can also be potentially employed to other robotic systems (see for example Chapter 2). With regards to their application on bipedal walking, their strengths rely on their mutual compatibility and their tight integration in the overall framework for bipedal walking.

To achieve bipedal walking in unknown environments, a hierarchical approach is followed. The robot's control is divided into modules. The subsequent modules take the result of the previous one into account and improve it. That way, the overall demand of reactive dynamic walking is divided into smaller parts. These sub-problems can be solved efficiently in real-time. Furthermore, the whole system becomes more robust, since disturbances and errors are balanced by each subsystem.

The same approach is valid for the *Vision System*. Although it is a standalone vision system, which is successfully applied to further robotic systems, its key value is the suitability to bipedal navigation in unknown and dynamic environments. The environment representation, which strongly affects the algorithms for vision processing, were developed with respect to its application in navigation and motion planning for bipedal walking and not for e.g. manipulation tasks. This allows for fast calculation times on both sides, robot control and environment perception. It is presented in the following.

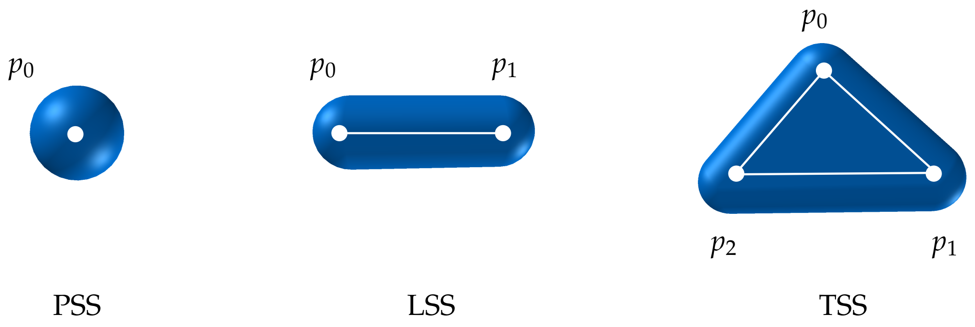


Figure 3.4: SSV elements used for distance calculations.

3.3.2 Environment Modeling

Motion planning for robot movements in general and bipedal locomotion in particular needs fast calculation times to enable for fast movements in unknown environments. In terms of computational time, collision detection is the most time-consuming part [98]. Therefore, a trade-off between a high-level of detail object representations with expensive distance calculations and a poor approximation but simple distance calculation is necessary. In contrast to e.g. manipulation tasks, in most cases it is not important for collision avoidance how to approximate the exact shape of objects. However, it is important to approximate possible collision objects conservatively enough to avoid contacts between collision pairs even while having sensor or model inaccuracies.

An object representation based on simple 3D volumes is chosen, the Swept-Sphere-Volumes (SSV), throughout the framework. It is not suitable for exact representation of object shapes but it is able to approximate objects with bounding volumes as conservatively as desired and it is efficient for distance calculation.

Distance Calculation Library

Schwienbacher et al. [98] developed a software library for fast distance calculations between SSV-objects. The SSV-objects are simple 3D objects, which can be described as geometric 2D elements with a radius. Schwienbacher et al. [98] used three different elements: the Point-SSV (PSS), a point p_0 and a radius r, the Line-SSV (LSS), two points p_0, p_1 connected by a line and a radius r, and the Triangle-SSV (TSS), three points p_0, p_1, p_3 connected by lines in a shape of a triangle and a radius. Figure 3.4 depicts the three elements. The representation of the elements as 2D elements based on line and points and their convex shape allows for fast distance calculations. The SSV-objects can be arbitrary combined to approximate objects. The variety of different element types, PSS, LSS and TSS, and their combinations enable to model each object in any desired level of detail. Further, Schwienbacher [97] presents approaches based on bounding box hierarchies to accelerate the calculations. The library is applied to reactive self-collision-avoidance for humanoid walking and to collision avoidance of the agricultural robot *CROPS*.

Collision World Modeling

The robot approximation proposed for self-collision avoidance was extended to a more general collision world representation. In addition to the robot, surfaces onto which the robot could step and obstacles are included. As published in Hildebrandt et al. [42], the geometric world is approximated by a collision world model W. It consists of a robot model R_m and an environment model E: $W = \{E, R_m\}$. Depending on the desired level of detail in the planning modules different robot models are chosen. The environment model

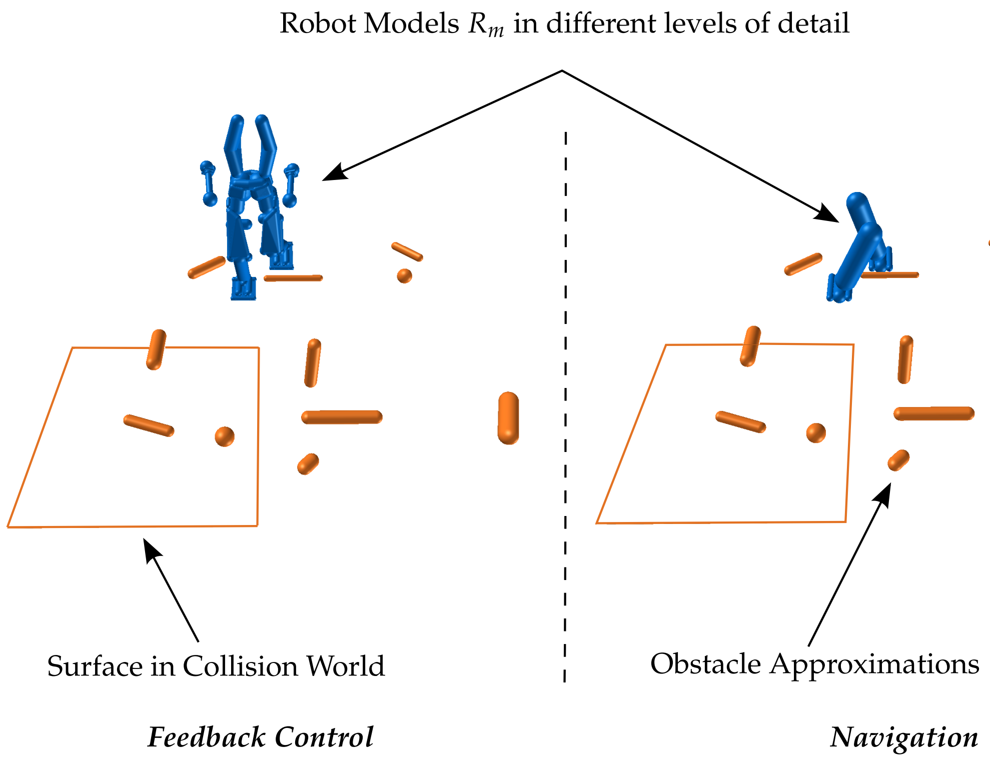

Robot Models R_m in different levels of detail
Surface in Collision World
Obstacle Approximations
Feedback Control
Navigation

Figure 3.5: Cluttered environment represented in collision world: Consistent environment approximation E in the *Local Control* and in the *Navigation*-module. E consist of elements modeling surface edges and approximating obstacles. Robot R is approximated in different level of detail for *Local Control* and the *Navigation*-module.

E is made up of n_O obstacles [4]. To avoid collisions with surface edges, these edges are included as obstacles and modeled as LSS. That way, they are consistently handled in the methods for collision avoidance. R_m is made up of n_S segments, depending on the chosen robot representation. In contrast to the robot model R_m, obstacles in E are dynamically added, removed or modified during run-time. In order to model the robot as well as the environment accurately, each robot segment S_i resp. obstacle O_k is approximated by n_{SV} resp. n_{OV} SSV objects V_j. This is summarized in the following definition

$$\begin{aligned} R_m &:= \{S_i | 0 \le n_i < n_S\}, && (3.1)\\ E &:= \{O_k | 0 \le n_k < n_O\}, && (3.2)\\ S_i &:= \{V_{ij} | 0 \le n_j < n_{SV,i}\}, && (3.3)\\ O_k &:= \{V_{kj} | 0 \le n_j < n_{OV,k}\}. && (3.4)\end{aligned}$$

In Figure 3.5, two different levels of detail for robot modeling and the environment approximation are depicted.

To avoid unnecessary distance calculations, the distance calculations are performed only for n_C defined collision pairs P_l. Each pair consists of either two different segments (S_F, S_T) or a segment and an obstacle (S_F, O_T). Between each collision pair the shortest distance d_l as well as the closest pair of points on the related SSV objects are calculated. To sum up the collision pairs, the collision environments C_m is defined as

$$\begin{aligned} C_m :=& \{P_l | 0 \le l < n_C\}, && (3.5)\\ P_l :=& \{(S_F, S_T) \,|\, S_F \in R_m,\, S_T \in \{R_m \setminus S_F \cup E\}\}. && (3.6)\end{aligned}$$

[4]So far, E has been chosen to be always the same in all control modules. In more complex environments, it might be advantageous also to chose E task dependent.

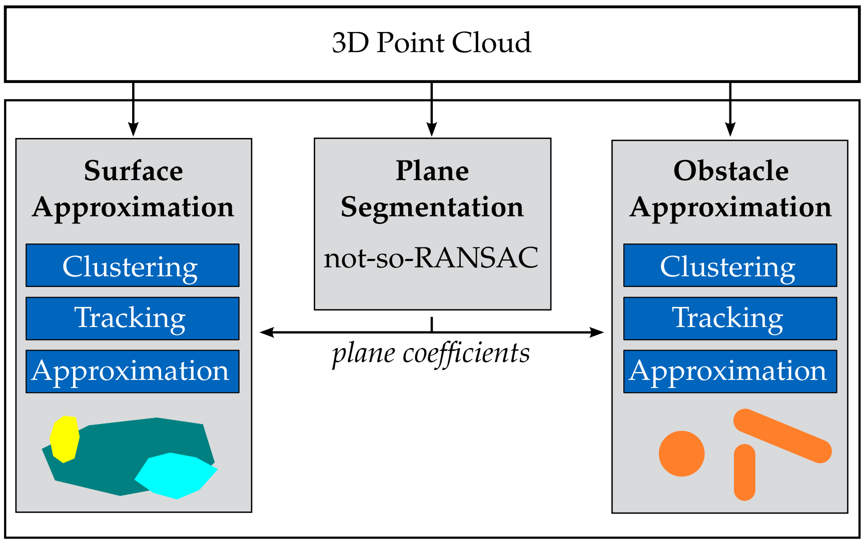


Figure 3.6: Architecture of Vision System [110].

Robot-robot collision pairs in P_l are predefined, while possible environment collision pairs are dynamically generated at run-time.

Application in Software Framework

The presented object representation is an efficient and dense way to represent 3D bodies. SSV elements are defined only by points, lines and corresponding radii. Thus, it does not only allow for fast distance calculations, but also to approximate complex environments without the need of high memory storage. Furthermore, the compact representation simplifies the communication between the software modules. The whole environment can be send with a reduced network traffic compared to more detailed representation as for example *OctoMaps* [47]. The collision world representation used in the planning modules is already clustered, which is inherent to the SSV approximation due to the hierarchy of SSV objects, segments and robots resp. the environment. Thus, it is computationally inexpensive to introduce bounding box hierarchies for distance calculations. This concept was used not only as part of the Distance Calculation Library, but on the planning level to omit unnecessary calculations. Furthermore, occupancy grids were analyzed to further cluster E and accelerate calculations. In the analyzed environments, this has not shown performance improvements. In more complex environments, it may be advantageous.

3.3.3 Vision System

For autonomous walking in unknown environments, perception is necessary. The following section gives a short overview over the developed *Vision System*. It was mainly developed by Dipl.-Ing. Daniel Wahrmann. More details are presented in Wahrmann et al. [128] and in his PhD thesis [110].

In contrast to current research in vision data processing, it was developed with its application to bipedal locomotion in mind. It represents a link between motion planing and vision data processing. The developed *Vision System* applies state-of-the-art algorithms, which are improved with respect to the specific application.

The *Vision System* approximates the environment via SSV-objects as introduced in Subsection 3.3.2. The input to the *Vision System* is a 3D point cloud. Instead of building up a 2.5D grid based map as presented, for example, by Nishiwaki et al. [80] and Sabe et al. [90] the 3D point cloud data are processed directly. First, all points belonging to surfaces which are steppable are separated by a segmentation algorithm. Thus, the original point cloud is divided in two - one point cloud containing all points belonging to the surfaces and one including the remaining points. In parallel running processes, these two point cloud data sets are simplified and approximated by simple geometries. To allow for stepping onto platforms, surfaces are approximated by polygons. As discussed in Subsection 3.3.2, the surface edges are approximated with obstacles using SSV objects to avoid collisions.

For both, surface and obstacle approximation, this process consists of four consecutive steps. Figure 3.6 shows an architecture overview:

1. in *Segmentation*, the point cloud data is filtered for points belonging to surfaces and obstacles.
2. in *Clustering*, the points belonging to the same objects are grouped.
3. in *Approximation*, all point groups are approximated with simple geometries.
4. the geometries are tracked and filtered in *Tracking* to improve their approximation with respect to sensor noise.

The module-based software architecture and the parallel running processes allow for high update rates and, therefore, modeling of dynamic environments. The surface approximation runs with a frequency of $5 - 10$Hz. Surfaces that move too fast, are classified as not steppable and considered as obstacles. The obstacle approximation is only limited by the performance of the sensor and it is able to approximate moving obstacles with a frequency of 30Hz. The software of the *Vision System* is published open-source[5]. In the following, the main steps are described.

Segmentation

The *Segmentation* is based on a modified *Random Sample Consensus* algorithm (RANSAC). It uses the surface model

$$ax + by + cz + d = 0 \tag{3.7}$$

with the unknown variables a, b, c and d. The basic RANSAC algorithm randomly chooses three points and generates a surface. If enough points are lying in this surface, it is classified as valid. This implementation has two extensions:

- In each frame, the surfaces of the previous frame are used as initial guess. Only then, the basic RANSAC algorithm is started. That way, the run-time can significantly be reduced (it is up to seven times faster).
- Surfaces are classified based on their inclination. Thus, only surfaces which are steppable are further taken into account.

The modified RANSAC algorithm is named *not-so-RANSAC* (see Figure 3.6).

Surface Approximation

The *Surface Approximation* uses the points filtered in the *Segmentation*. First, the *Clustering* (see left figure of Figure 3.7) groups the points depending on their distances in clusters. Here, the inclination classification of the *Segmentation* is used to differentiate, for example, the ground and ramps. Second, the *Approximation* (see center figure of Figure 3.7) filters the

[5]The software of the *Vision System* is available at `http://github.com/am-lola/lepp3`.

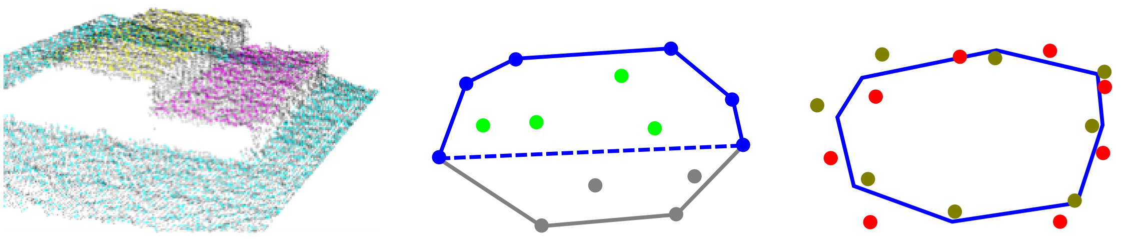


Figure 3.7: Surface approximation: *Euclidean Clustering* (left), *Quickhull Approximation* (center) and *geometric-first order filter Tracking* (right).

outliers of each cluster using a *Quickhull*–Algorithm. The resultant polygon is iteratively reduced by eliminating all points which modify least the full area of the polygon. The algorithm stops when a desired number of corner points is reached. Third, the *Tracking* filters the resulting surfaces. It compares and filters each new polygon A with the former polygon B. For this step, a *geometric PT1 Filter* was developed. For each point in A, A_i, a projection on B, A_{i_B}, and for each point in B, B_j, a projection on A, B_{j_A}, is generated. The new polygon C is calculated as

$$C = \left(\alpha A_i + (1-\alpha)\, A_{i_B} \cup (1-\alpha)\, B_j + \alpha B_{j_A}\right) \forall i, j \tag{3.8}$$

with $0 \leq \alpha \leq 1$. Consequently, the number of corner points of C is the sum of points of A and B. Thus, the reduction algorithm is executed again. The filter output is the reduced polygon C. Using the developed filter, an accurate and stable approximation of the surfaces is achieved.

Obstacle Approximation

The *Obstacle Approximation* models obstacles as SSV segments as introduced in Subsection 3.3.2. It follows the same procedure as Subsection 3.3.3. The input to the *Obstacle Approximation* are all points, which do not belong to surfaces, as calculated in the *Segmentation*.

Similar to the *Surface Approximation*, first, a *Clustering* groups all points in separated clusters, using a version of the Gaussian Mixture Models (GMM). This is an approach commonly used in *Machine Learning* to cluster non-supervised data. Typically, it consists of an iterative approach that converges to a classification of points. In this application, each iteration is executed on new point data and the covariance is filtered with a PT1-filter. That way, different obstacles stay separated although their points lie next to each other (see left picture of Figure 3.8).

Second, the *Approximation* (see picture in the center of Figure 3.8) calculates, motivated by inertia tensors, geometric invariants of each cluster. Based on the principal moments of inertia I_{min}, I_{mid} und I_{max} the geometric invariants are determined as $\xi_1 = \frac{I_{min}}{I_{max}}$ und $\xi_1 = \frac{I_{mid}}{I_{max}}$. They are independent of the obstacle size and give a clue how the points are distributed in the space: Obstacles, that resemble a PSS, are described by $\xi_1 \simeq \xi_2 \simeq 1$ and obstacles, that resemble a LSS, are described by $\xi_1 < \xi_2 \simeq 1$. Further, obstacles are divided until the resultant objects can be approximated with LSS or PSS elements.

Third, for the *Tracking* (see right picture of Figure 3.8) of obstacles a Kalman filter is used. Constant obstacle velocity is assumed and Gaussian noise is applied on the expectation value of position and velocity of the GMM.

The approximation processes run in parallel at maximum speed updating an internal world map, which consists of a list of active surfaces and obstacles with velocities. This set is sent to the motion planning with an update rate of 10 Hz.

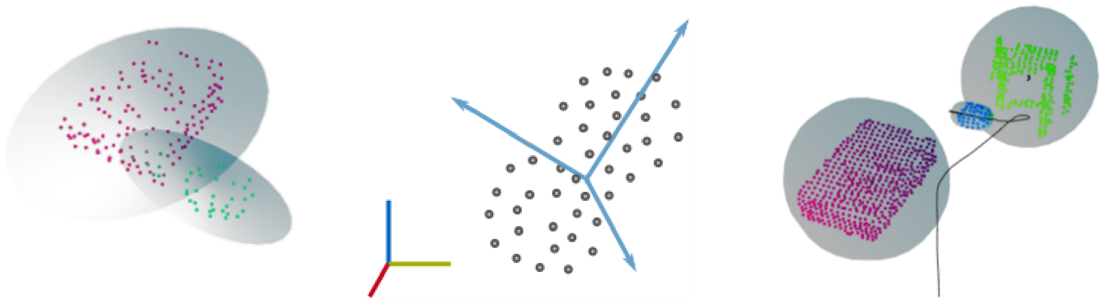


Figure 3.8: Obstacle approximation: *Gaussian Clustering* (left), *Inertia Tensor Approximation* (center) and *Kalman Tracking* (right).

3.3.4 Motion Planning

Section 3.2 summarizes the limitations of the basic control framework for bipedal locomotion. To achieve a locomotion strategy that exploits the capabilities of bipedal walking, the basic framework is extended by mainly three components for motion planning (see Figure 3.3): (1) a new *Navigation* module explained in more detail in Chapter 4, (2) a *Kinematic Optimization* module detailed in Chapter 5, and (3) a *Reactive Collision Avoidance* module explained in more detail in Chapter 5 as well. In the following, a brief description is made.

Navigation

The new *Navigation*-module replaces the *Step Sequence Generation*-module. The input to the *Navigation*-module are the user's *Walking Commands* represented as the parameter set $p_{wp,d}$, which includes a desired step sequence as well as desired goal positions and a snapshot of the current environment model as it is introduced in Subsection 3.3.2. Based on its input, it calculates a sequence of foothold positions guiding the robot as close as possible to the user's input taking into account the environment such as obstacles and platforms. Furthermore, it determines the height and the width of the swing-foot movements necessary to step over obstacles, the height of the CoG trajectory to step onto platforms and it adapts the step time T_{Step} according to the stride. The resultant parameter set can be summarized as the modified set p_{wp}.

The *Navigation*-module is located on a high abstract level within the control framework. It approximates the motions of the full robot by a simple model (see Figure 3.5), to be able to solve the navigation problem taking a large area into account and long distance step sequences. As part of the *Global Control* it is executed each step. This corresponds to a cycle time of approx. 0.8s. On the one hand, it has to fulfill hard real-time constraints, since an executable solution has to be provided for each step. On the other hand, the short computational times allow for re-planning with high sample times. That way, the system becomes robust to changes in the environment, highly responsive to user input and able to handle dynamic environments.

Kinematic Optimization

A main deficit of the basic control framework (see Section 3.2) is the planning of the ideal walking pattern without knowledge of the robot's future motions. The *Walking Pattern Generation* (see Figure 3.2) calculates based on $\boldsymbol{p}_{wp}$ ideal reference trajectories. The model used for generating dynamically stable motions is a three-mass model without information on the robot's kinematics.

Planning trajectories without taking the future kinematics of the robot's motion into account requires high safety margins to avoid collisions and joint limits. Thus, well-chosen heuristics are necessary to adapt the trajectories to the terrain. In real world scenarios, walking situations are always different and also different from simulation cases. Motions based on heuristics may lead to non-executable solutions, which would lead to a system fail. To overcome these limitations, the *Kinematic Optimization*-module is introdued [118]. It is sorted sequentially after the *Navigation*-module in the *Global Control*. Based on p_{wp} and the *Ideal Walking Pattern*, it integrates a full kinematic model over one step while taking the local methods for collision avoidance of the *Feedback Control* (the full kinematic model is depicted in Figure 3.1 and its collision approximation in Figure 3.5) into account. Consequently, p_{wp} as well as the nullspace input, can be evaluated and optimized. Here, a trade-off between calculation time and parameterization level of detail has to be done. Based on the *Kinematic Optimization*'s output the *Walking Pattern Generation*-module calculates the *Ideal Walking Pattern*.

Reactive Collision Avoidance

Both the *Navigation*- and the *Kinematic Optimization*-module are part of the *Global Control*. They take the environment approximated by the vision system into account. As planning modules, they influence the motion before it is executed. However, in real world scenarios, the robot's motions have to be adapted during execution to account for disturbances. Furthermore, the environment may change, which cannot be accounted for during the motion planning.

Therefore, a *Collision Avoidance*-module is introduced [124]. It optimizes the desired motion locally taking kinematic constraints each control cycle into account. The main motivation behind such an adaption of the desired motion lies in the idea that in many applications, as also in bipedal locomotion, the exact execution of the planned trajectories is not necessary. The restriction in bipedal locomotion on the executed motion is that the CoG has to track its desired trajectory to guarantee a dynamically balanced motion and that the swing leg hits the ground at the planned position and time. A local adaption of the planned trajectories has an important influence on the trajectory optimization in the *Global Control*. The real-time constraints limit the available processing time for optimization. Consequently, the optimization is only able to influence the robot's motion through few optimization parameters. Without a local adaption of the planned motion, many kinematic movements would not be possible due to the coarse description of the motions via the parameter p_{wp} and the, therefore, limited influence of the optimization.

3.3.5 Robust Walking

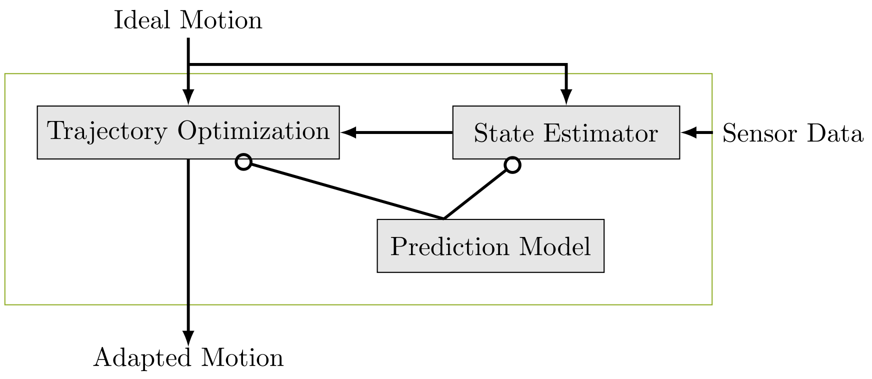


Figure 3.9: Overview of methods for robust walking. They are included in the *Trajectory Adaption*-module (see Figure 3.3). Adapted from [114].

In real world scenarios, the robot is expected to be exposed to disturbances which cannot be anticipated by the planning modules. Possible reasons include model inaccuracies of the *Vision System* due to, for example, sensor noise, environment conditions which prevent vision perception or, just, badly motivated persons pushing the robot. These disturbances may be too large to be balanced by local adaptions in the *Feedback Control*. Therefore, a model-predictive control was developed which adapt the desired motion of the robot over an extended time horizon based on current sensor data. For stiff position controlled robots, their absolute inclinations w.r.t. the ground are considered to be the DoF with the main significance to the stability of the robot [132]. Consequently, the main control objective of these methods is to hold the robot upright. They consist basically of three components:

- a prediction model, which is used as common base for the state estimator and trajectory optimization. It is introduced in Wittmann et al. [132].
- A state estimator to monitor the inclinations of the robot is introduced in Wittmann et al. [133];
- a trajectory optimization to correct swing foot and CoG trajectories online, which is introduced in Wittmann et al. [133, 135].

In Figure 3.3, the components are included in the block *Trajectory Adaption*. Figure 3.9 gives a more detailed overview of the methods within the *Trajectory Adaption*. The input to the *Trajectory Adaption* are the *Ideal Walking Pattern* and the IMU measurements, namely the robot's absolute inclination $\boldsymbol{\varphi}_m$ and inclination rate $\dot{\boldsymbol{\varphi}}_m$. The output are modified task space trajectories $\boldsymbol{w}_{mod}$. The *Trajectory Adaption* adapts via the final values of the swing-foot trajectories the foothold position in x-, y- and z-direction and its horizontal orientation. Furthermore, the CoG trajectory is continuously modified. The following subsections are largely inspired by the results published in Hildebrandt et al. [123].

Prediction Model

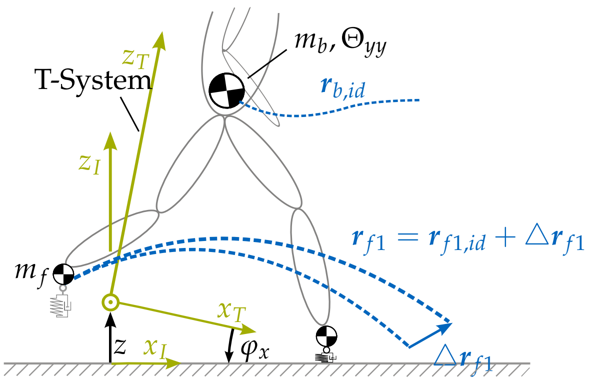

m_b, Θ_{yy}
z_T
T-System
$\boldsymbol{r}_{b,id}$
z_I
$\boldsymbol{r}_{f1} = \boldsymbol{r}_{f1,id} + \triangle\boldsymbol{r}_{f1}$
m_f
x_T
z
x_I
φ_x
$\triangle\boldsymbol{r}_{f1}$

Figure 3.10: Prediction model with three point masses, m_b for the upper body and m_f for each foot. The swing foot trajectories $r_{f,id}$ and their modifications Δr_{f1} and the trajectories of the upper body $r_{b,id}$ [123].

For real-time application, the robot's multi-body system is approximated by a planar dynamic model (see Figure 3.10). The same planar model is used for the x- and y-directions (the motion in the sagittal and frontal plane, respectively) with different geometry and input trajectories. In the following, the model predicting the motion in x direction is described. The model in y direction can be analogously derived.

Inspired by the model used in *Walking Pattern Generation*, inertia effects of the full robot model are approximated via three point masses approximating the robot's upper body and its feet, respectively. The relative position of the three masses is constrained. They are assumed to perfectly track the ideal planned motions of the upper-body ($_T\mathbf{r}_b(t)$) and the feet trajectories ($_T\mathbf{r}_{f1}(t)$, $_T\mathbf{r}_{f2}(t)$) as part of $\boldsymbol{w}_{id}$ in the planning Frame of Reference (FoR). It rotates with the robot's upper body (named T-system in Figure 3.10, index T, x_T, z_T)). The model's unactuated DoF are the inclination φ_x and the vertical translation z, while the translation in x-direction, which corresponds to sliding of the robot, is neglected. They describe the transformation from an inertial FoR (index I, x_I, z_I) to the T-system. The contact between feet and ground is approximated as one point contact located at the middle of the foot. The contacts are modeled as linear spring-damper pairs (stiffness k_c, damping d_c) with the experimentally identified values from the real robot's rubber sole and the carpet in the laboratory. They act only in z_I-direction and are considered to be unilateral, since slipping is neglected. The Equation of Motion (EoM) for the model with $\boldsymbol{q} = [z, \varphi]^T$ can be stated as

$$\boldsymbol{M}_p(\boldsymbol{q}, t)\ddot{\boldsymbol{q}} + \boldsymbol{h}_p(\boldsymbol{q}, \dot{\boldsymbol{q}}, t) = \boldsymbol{\lambda}_p(\boldsymbol{q}, \dot{\boldsymbol{q}}, t) + \boldsymbol{T}_s \tag{3.9}$$

with $\boldsymbol{M}_p$ as the mass matrix of the prediction model, $\boldsymbol{h}_p$ containing all nonlinear terms of the EoM and $\boldsymbol{\lambda}_p$ representing the resultant force and moment of all active contacts N_c projected on the model's DoF. The robot's local stabilization controller is approximated via a torque $\boldsymbol{T}_s$ calculated as a saturated PD-controller.

State Estimation

The predictive-model approach is based on an accurate estimate of the current robot state. The estimate is calculated from the sensor feedback provided by IMU measurements of the absolute inclination $\boldsymbol{\varphi}_m$ and inclination rate $\dot{\boldsymbol{\varphi}}_m$. For large disturbances, the IMU measurements include undesired oscillations which have to be filtered. In Wittmann et al. [134] a state observer is proposed as filter. It is based on an extended Kalman filter which compensates model errors and external disturbances. The observer is implemented for both horizontal directions and provides an accurate initial state $\boldsymbol{z}_0$ for the x- and y-direction.

Trajectory Optimization

The objective of the trajectory optimization is to adapt the overall desired motion of the robot to stabalize the robot. It adapts the final values of the swing-foot trajectories, the foothold position in x-, y- and z-direction and its horizontal orientation according to the prediction of the robot's motion using the presented model. Furthermore, the CoG trajectory can be continuously modified as described in Wittmann et al. [135]. In the following, the modifications in x and y direction are considered decoupled. A parameter ΔL_x is introduced which describes the horizontal modification in x-direction of the final swing foot position. The swing foot trajectory is adapted with a quintic polynomial. The polynomial begins at the current position, velocity and acceleration and ends at ΔL_x with zero velocity and acceleration. Figure 3.10 depicts a polynomial $\Delta_T\boldsymbol{r}_{f1}(\Delta L_x)$ added to the ideal trajectory $_T\boldsymbol{r}_{f1,id}$ via

$$_T\boldsymbol{r}_{f1} = {_T\boldsymbol{r}_{f1,id}} + \Delta\, {_T\boldsymbol{r}_{f1}}\,(\Delta L_x). \tag{3.10}$$

To compensate for the upper body inclination between the robot and the ground and avoid early ground contacts, the polynomial includes a modification of the swing foot height. Consequently, the contact force's lever arm changes and, thereby, influences the model's state $\boldsymbol{z} = [\boldsymbol{q}, \dot{\boldsymbol{q}}]$. The overall first order differential equation with ΔL_x is modified to

$$\begin{aligned}\dot{\boldsymbol{z}} &= \begin{bmatrix} \dot{\boldsymbol{q}} \\ \boldsymbol{M}_p^{-1}(\boldsymbol{q}, t)[\boldsymbol{\lambda}_p(\boldsymbol{q}, \dot{\boldsymbol{q}}, \Delta L_x, t) + \boldsymbol{T}_s - \boldsymbol{h}_p(\boldsymbol{q}, \dot{\boldsymbol{q}}, t)] \end{bmatrix} \\ &= \boldsymbol{f}(\boldsymbol{z}, t, \Delta L_x). \end{aligned} \tag{3.11}$$

The objective of the *Trajectory Adaption*, to stabilize the robot, may be formulated mathematically as a minimization of the quadratic cost function

$$J = \Delta \boldsymbol{z}^T(t_f)\boldsymbol{S}_z \Delta \boldsymbol{z}(t_f) + s_p \Delta L_x^2 + \int_{t_0}^{t_f} \Delta \boldsymbol{z}^T \boldsymbol{Q} \Delta \boldsymbol{z} \, \mathrm{d}t \tag{3.12}$$

over a time horizon $t \in [t_0, t_f]$. The cost function weights, s_p, $\boldsymbol{S}_z$ and $\boldsymbol{Q}$, are manually parameterized to minimize the absolute inclination error, which is included in the state error $\Delta \boldsymbol{z} = \boldsymbol{z} - \boldsymbol{z}_{ref}$. A direct shooting method is used to solve the optimization problem. For a given inital value $\boldsymbol{z}(t_0) = \boldsymbol{z}_0$ and ΔL_x, the model's motions can be determined by integrating (3.11) numerically. The resultant trajectory $\boldsymbol{z}(t, \Delta L_x)$ is used to compute the cost function $J(\boldsymbol{z}(t, \Delta L_x), \Delta L_x)$. This way, the problem is converted into the static optimization problem

$$\min_{\Delta L_x \in \mathcal{A}_s} J(\Delta L_x). \tag{3.13}$$

It can be considered as unconstrained as long as the variable ΔL_x remains inside the allowable set $\mathcal{A}_s$. $\mathcal{A}_s$ can be interpreted geometrically as a *valid area*. It takes constraints due to the robot's kinematics as well as restrictions resulting from the environment into consideration. The constraint optimization problem can be solved applying nonlinear programming methods such as Newton's method [49]. The modification in y-direction is derived analogously.

3.3.6 Implementation Details

The following section gives more details about the implementation and real-time realization of the presented framework. Overall, the framework can be divided in control and vision modules. The control modules directly generate the robot's motion. Thus, they represent safety-critical units. Furthermore, the stabilization of the robot requires short and fixed control cycles. Therefore, the control modules are executed on a computer running with a real-time operating system *QNX Neutrino 6.6*, which guarantees hard timing.

The environment perception is executed on a second computer. It does not rely on fixed and short control cycles. Its control cycle mainly depends on the update rate of the camera. Since fixed timing is not required, a *Linux OS* was pragmatically chosen which already provides the required software for vision processing. The control as well as the vision modules are split into different main processes to accelerate calculations. Figure 3.11 gives an overview.

Control Processes

The robot control is divided into three main processes: (1) The *Global Control*, (2) the *Feedback Control* and (3) the *Low Level Communication*. They exchange data using shared memory. In the *Global Control*, a separated thread executes the *Navigation*- and the *Kinematic Optimization*-module during execution of step k for the following $k+1$ to $k+n_{Steps}$ steps.

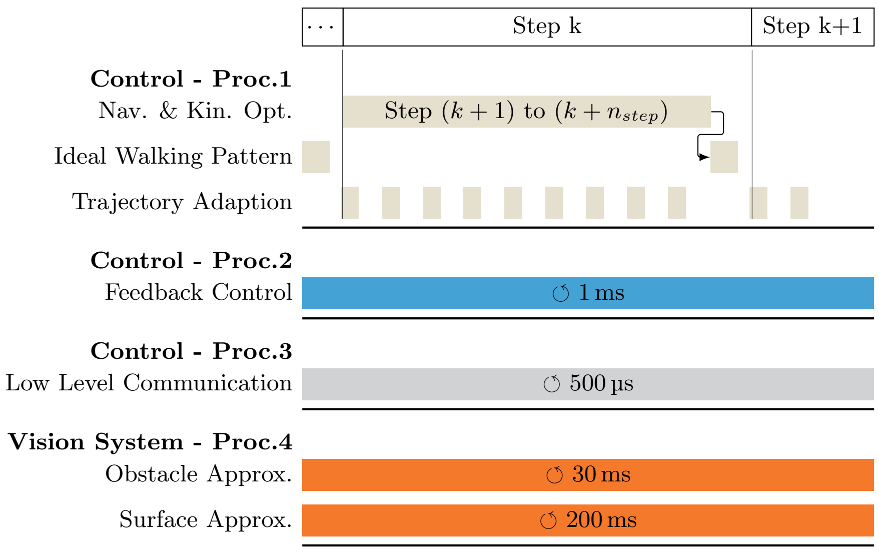


Figure 3.11: Multi-process and multi-thread software architecture of Lola's real-time walking control system. Adapted from [123].

The *Walking Pattern Generation*-module uses the result before each step $k+1$ to calculate its *Ideal Walking Pattern*. During execution of the current step, the *Trajectory Adaption* modifies the reference trajectories sequentially to the *Ideal Walking Pattern*. The second process executing the *Feedback Control* has a cycle time of 1ms. It communicates with the local joint controller and the sensor data of the robot via the third process, the *Low Level Communication*. This in turn runs with a cycle time of 500µs interpolating the data of the *Feedback Control*. Wittmann [114] gives more details about the joint control and the sensor data feedback.

Vision Process

The *Vision System* is divided in two main processes: (1) the *Obstacle Approximation* and (2) the *Surface Approximation*. Limited by the frame rate of the camera, the *Obstacle Approximation* runs with a cycle time of approximately 30ms, which varies according to the scene. The *Surface Approximation* has a cycle time of approximately 100-200ms. The *Vision System* sends updates to the robot control with a cycle time of 100ms.

3.4 Robust Walking with Geometrical Constraints

The methods presented in the following were developed in collaboration with Lisa Jeschek [140] and they have been published in Hildebrandt et al. [123]. In the previous section, methods for environment perception, for *versatile* walking and for *robust* walking are presented. However, when navigating in complex scenarios, all methods have to perform simultaneously. Figure 3.12 shows an exemplary situation based on this hierarchical approach: the *Vision System* detects and approximates an obstacle and sends it to the robot's control system. The *Navigation*- and the *Kinematic Optimization*-module plan the robot's ideal motions based on the valid foothold location of the swing foot $\boldsymbol{x}_{id}$, determined by the step planner. Assuming an unknown disturbance, the *Trajectory Adaptation* then modifies this position by $\Delta\boldsymbol{x} = [\Delta L_x, \Delta L_y]^T$ to stabilize the robot. The resulting final foothold position, however, would cause a collision. Thus, the desired output is a modified

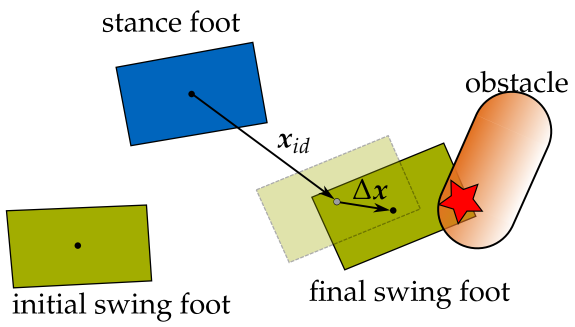


Figure 3.12: Example for problem description: ideal collision-free final swing foot position and the modified invalid position [123].

yet collision-free foot position

$$\boldsymbol{x}_m^* = \boldsymbol{x}_{id} + \Delta\boldsymbol{x}^*. \tag{3.14}$$

with feasible, collision-free modification $\Delta\boldsymbol{x}^*$ instead of the modification $\Delta\boldsymbol{x}$ which could lead to collisions with the environment.

Here, the main question is how constraints on the foothold positions can be described and accounted for in a real-time optimization procedure as described in Subsection 3.3.5. Since *Trajectory Adaptions* have to react instantaneously to unknown disturbances short computation time is crucial. Longer calculation times would result in higher latencies and which would degrade the performance of the feedback control. The requirements are summarized as follows:

- Short computational time ($\ll 1\,\text{ms}$);
- Several arbitrarily shaped obstacles may be present;
- Over-stepping of obstacles should be possible;
- Kinematic limits must be respected;
- A collision-free position has to be found reliably.

3.4.1 Related Work

In existing literature, scenarios as the presented above, are rarely investigated. Most works handle *versatile* walking and *robust* walking separately. Only few authors present approaches to allow for *versatile* and *robust* walking simultaneously. The authors of Chestnutt et al. [21] showed experiments of a robot walking on the spot and adjusting footstep locations to reject perturbations while taking obstacles into account. They extended their step planner to compute not only step sequences, but also safe regions around the target footholds. The walking controller uses those regions to calculate permissible adjustments of the target footholds to reject perturbations in the presence of obstacles. Following this approach, only the step planner takes the environment into account. When the walking controller adjusts the target footholds, it needs to recalculate the robot's trajectories. Collisions can thus only be avoided by using large safety margins, which do not prevent the possibility for the robot to step close to or over obstacles. Recently, Naveau et al. [77] published a method to extend their walking controller by taking convex obstacles as additional constraints into account. Using the walking controller, the robot *HRP2* can adjust footholds locally to avoid circular obstacles in experiments. They also proved their concept in simulation to reject disturbances. To the best of the author's knowledge the combination of perturbations and obstacles has not been shown. Integration into a whole planning framework including a footstep planner is pending. It is a very interesting approach, however, and in contrast to the method presented in this work, the model predictive control with nonlinear constraints is solved at a high frequency. A simplified template model of the robot thus has to be used to meet the real-time requirements. Approximation of the environment and of the robot is kept simple. Complex scenes and self-collisions are therefore difficult to consider in this framework.

In Kuindersma et al. [61], the authors present an optimization-based framework that treats all tasks for planning and control in complex scenarios. A footstep planner calculates valid footholds using mixed integer optimization. It is based on a height map segmented into convex allowable regions represented as polytopes. A human operator has to provide the algorithm seed points to find the polytopes. To include arbitrary static environments, they perform the dynamic motion planning with the robot's complete linear and angular momentum equations. This enables multi-contact problems as well as the full kinematics of the robot to be included. For feedback control, a QP is formulated that takes long-term stability into account using the inverted pendulum model. It provides motor commands via additional inverse dynamics for the current time-step. To the best of the author's knowledge, information about the environment is not used in the stabilization.

3.4.2 Method Overview

The hierarchical approach of the framework, presented in Section 3.3, has several advantages: it is very robust, since each module takes the results of those preceding it into account and improves them. Due to the breakdown of the motion generation problem to sub-problems that can be solved in real-time, it is also able to allow for reactive bipedal locomotion. Since the combination of the methods should not decrease the performance of the overall system, it was decided to preserve the hierarchical control architecture.

Referring to the example of Figure 3.12, it is decided to first determine a collision free motion that is then adapted for disturbance rejection based on sensor data. Furthermore, it is stipulate upfront that avoiding a collision has a higher priority than rejecting a disturbance. The decision is based on the assumption that disturbance rejection can take place during more than one step, whereas a collision will lead to a system failure. This means that the possible solutions of (3.13) are restricted to reachable and obstacle-free regions. However, other criteria may be defined without changing the logic of this system.

There are basically two approaches to handle such complex scenarios: the first is to determine an optimal step length modification without constraints and project the final solution onto the cone of $\mathcal{A}_s$. The second approach accounts for the constraints during optimization: however, this requires an optimization for the step length modifications in both directions since the feasible set is at least two-dimensional and the boundaries for ΔL_x and ΔL_y are coupled. For this reason, the decoupled planar models presented in Subsection 3.3.5 are combined to a single 3D model.

In the following, the approach accounting for the constraints is derived. The approach relying on projecting an invalid point onto a feasible region is discussed in the subsequent part. This projection method is then applied to the stated problem and two solution strategies are discussed. An overview of the control flow for the *Trajectory Adaption* taking constraints into account is depicted in Figure 3.13.

3.4.3 Geometrical Constraints

The key point when taking the geometrical constraints imposed on *robust* walking, namely kinematic constraints and obstacles, into account is their consistent and dense representation. To allow fast calculations, convex polytopes are chosen to represent both kinematic

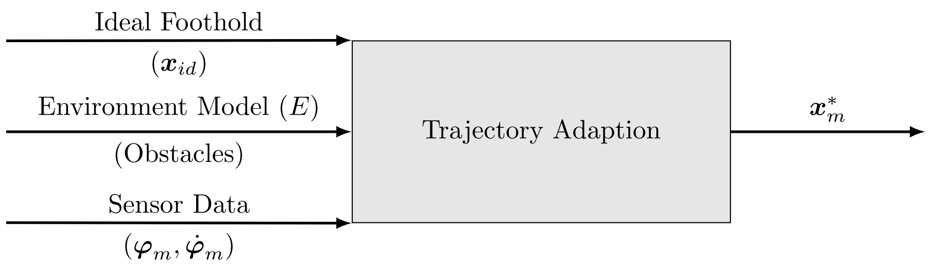


Figure 3.13: Control flow of *Trajectory Adaption* taking into account constraints. Adapted from Hildebrandt et al. [123].

constraints and obstacles.[6] This results in n_{eq} linear inequalities, c_{eq},

$$c_{eq,j} := \boldsymbol{a}_j^T \boldsymbol{x} > b_j \tag{3.15}$$

represented by the parameters $\boldsymbol{a}_j$ and b_j for $j = 1, ..., n_{eq}$. A set of $n_{\mathcal{C}_I}$ linear inequalities describes one convex polytope, $\mathcal{C}_I$, which is an *invalid region* (see for example Figure 3.18). In total, the *invalid area* $\bar{\mathcal{A}}_S$ consists of n_p polytopes and is defined as follows:

$$\bar{\mathcal{A}}_S := \bigcup_{i=1}^{n_p} \mathcal{C}_{I_i}. \tag{3.16}$$

The corresponding *valid area*, $\mathcal{A}_S$, is formally defined as

$$\mathcal{A}_S := \mathcal{A} - \bar{\mathcal{A}}_S, \tag{3.17}$$

based on the total search area $\mathcal{A}$. The determination of $\bar{\mathcal{A}}_S$ is described below.

Kinematic Limits

Starting with the current stance foot, the kinematically reachable area is heuristically approximated by the convex polytope as depicted in Figure 3.14. The reachable area results in n_{kin} convex polytopes (see Figure 3.14). The system of inequalities for one convex polytope results in

$$\mathcal{C}_{I,kin,j} = \{\boldsymbol{x} | \forall i \in \{1, ..., n_{kin,j}\} : \boldsymbol{a}_{kin,j,i}^T \boldsymbol{x} > b_{kin,j,i}\}. \tag{3.18}$$

Obstacles

As described in Subsection 3.3.2, 3D segments approximate obstacles and areas of the environment the robot can not step onto. The former consist of n_{SSV} convex SSV-objects to allow for a detailed approximation of the real objects. Like the representation of the kinematic limits, the n_{SSV} convex SSV-objects are reduced to 2D polytopes to comply with the hard timing constraints. The SSV-objects are first projected onto the ground. Then, the three types of SSV-objects – sphere, line and triangle – are represented as polytopes as shown in Figure 3.15. For each object j, we get a convex hull, which is described by

$$\begin{aligned}\mathcal{C}_{I,SSV,j} = \{\boldsymbol{x} | \forall i \in \{1, ..., n_{eq,SSV,j}\} :\\ \boldsymbol{a}_{SSV,j,i}^T \boldsymbol{x} > b_{SSV,j,i}\}.\end{aligned} \tag{3.19}$$

Kinematic limits already restrict the valid area for a step. Therefore, only obstacles within this kinematically reachable area are considered. This approach greatly reduces the computational costs for the inequality constraints.

[6]The representation as convex polytopes does not represent a limitation since concave polytopes can be decomposed in convex polytopes.

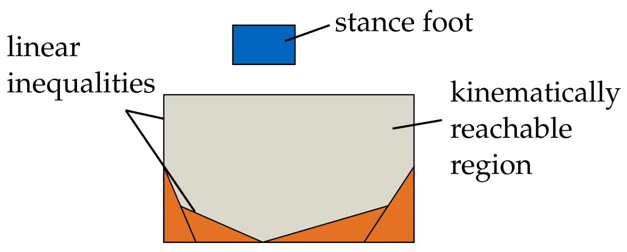


Figure 3.14: Kinematically reachable region (grey) and current stance foot (blue). Kinematically reachable region bounded by seven linear equations Hildebrandt et al. [123].

Large Obstacles

Considering the robot's whole kinematic movement, obstacle-free regions are not necessarily steppable or kinematically reachable. Large obstacles, which the robot can not over-step because of their height, make adjacent obstacle-free regions inaccessible due to kinematic constraints. These large obstacles are already considered in the environment representation to avoid repeated checks for obstacle-free regions and inaccessible regions. Instead of introducing a polytope for the inaccessible region, the large obstacle's representation is enlarged (refer to Figure 3.16). That way, the obstacle approximation as well as the representation of the inaccessible region stay convex.

Foot Geometry

The sophisticated 3D representation of the foot and lower leg geometry used in the step planner helps to give a better approximation of the robot's whole kinematic movement. Nevertheless the obstacles are enlarged by the foot geometry as shown in Figure 3.17. Thereby, the desired foot rotation α are taken into account. Thus the geometric constraints can be analyzed using only one point, which describes the foothold position.

3.4.4 Finding Safe Footholds

Below, the initially posed question is answered: given a modified but invalid foothold position, how can the closest valid one be determined (see Figure 3.12)? Several methods are described and compared in this subsection.

Sampling

One solution is to discretize the area around the modified foothold position. The discrete positions are checked to determine whether they are valid or not. Among all the valid positions, the one closest to the modified position is chosen. The advantage of this procedure is that calculation time only depends on the number of equations and on the discretization level. The calculation time is therefore deterministic making the method suitable for real-time application. Furthermore, all discretized points can be checked in parallel on multi-processor platforms. The disadvantage is the strong dependence between the solution's quality and the discretization grid. Fine discretization is computationally expensive, but is necessary to find a solution in complex scenarios.

Geometric Testing

Another solution is to search for the closest valid point on the boundaries, which are described by linear equations. This reduces the problem to a distance calculation between this valid point in $\mathcal{C}_{I_i}$ and lines describing the boundaries of $\mathcal{C}_{I_i}$. The closest point on the line can be calculated based on the smallest distance between point and line. To accelerate

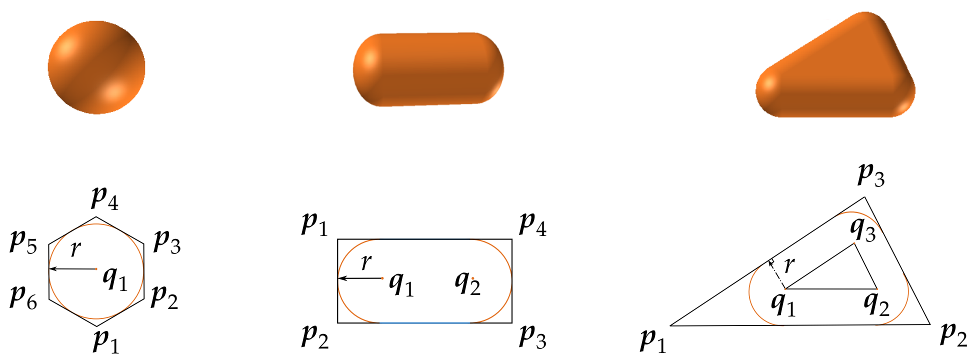


Figure 3.15: Approximation of SSV-Objects as polytopes [123].

the calculations, the residuum of the point and the inequalities is used: A small residuum indicates a small distance between the point and the boundaries and is used to reduce the number of distance calculations. However, this procedure is not possible for multiple invalid regions that intersect each other. The closest point on one of the boundaries is not necessarily valid since it may lie in another invalid region. This extended problem is solved as follows: (1) the closest point on all boundaries is found. (2) Each of these points is checked for validity. If one is invalid, the next nearest valid point on the same boundary is chosen. (3) Finally, all resulting points are compared to find the closest point. In contrast to the *sampling*-based approach, the quality of the solution does not depend on the environment or prior assumptions (e.g., a discretization grid). However, the closest point is searched for on each boundary and thus the computational costs increase significantly with the number of invalid regions.

Safe Regions

The previously presented solutions all begin with an invalid starting point, $\boldsymbol{x}_m$, and try to find a valid point, $\boldsymbol{x}_m^*$. Another possibility is to invert the problem: instead of finding the closest valid point, $\mathcal{A}_S$ is divided into a set of convex valid regions C_V. In each of these valid and convex regions, the closest point to the invalid starting point can be determined separately and, because of the regions' convexity, very efficiently. In contrast to the *Geometric Testing* method, intersections of the valid convex regions do not pose a problem, because the search is applied on a set of valid regions. The solutions are consequently independent of each other. The best one is chosen based on the calculated set of closest points. This procedure of searching valid convex regions instead of only considering the invalid regions is largely inspired by Chestnutt et al. [21], who presented a method to calculate a valid convex area around a valid starting point. The algorithm starts with a convex area. It iterates around the starting point and it removes invalid parts of the initial convex area. A drawback of the implementation is that only one convex area around the starting point is found. Deits et al. [27] presented a powerful open-source tool, called *IRIS*, which has already been applied to step planning [26]. *IRIS* uses the corner points of invalid convex regions as input and calculates the corresponding inequalities. It also determines the largest valid convex region which is closest to a starting point. Although it seems to be well suited for the present problem, it exhibits some shortcomings with respect to this application[7]:

- *IRIS* does not guarantee that the starting point lies in the convex region.

[7]The following statements are based on the authors experience with *IRIS* tool available as open source at `https://github.com/rdeits/iris-distro`.

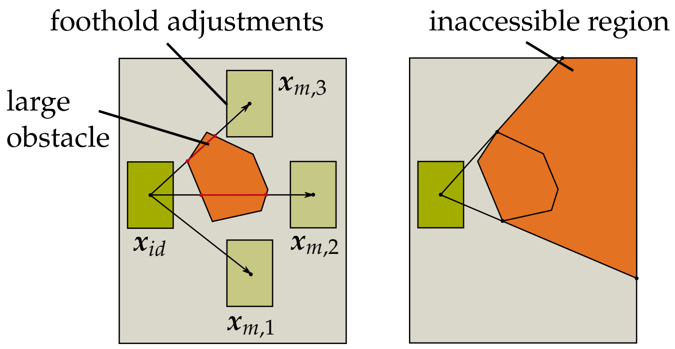


Figure 3.16: Large obstacle and its representation as inaccessible region. Ideal $\boldsymbol{x}_{id}$ and modified $\boldsymbol{x}_{m,i}$ foothold positions [123].

- *IRIS* needs a predetermined search area. The algorithms seem to be very sensitive to starting points close to the borders.
- *IRIS* only determines one convex area. When it is applied iteratively on the same search area by respectively removing the determined regions, it often fails to find additional valid regions.

The authors of Liu et al. [65] and Sarmiento et al. [92] presented methods to divide arbitrary areas in convex regions. On the one hand, neither method is restricted to convex invalid regions; however, neither benefits from reduced computational costs for convex problems. In this application, only convex invalid regions are used, to gain the benefit in calculation times. Like Liu et al. [65] and Sarmiento et al. [92] the objective here is to completely cover the valid area with multiple convex regions that might intersect. The characteristics of this mathematical problem are exploited to meet the timing limitation:

- $\mathcal{C}_{I,i}$ are all convex.
- The algorithm's starting point, which is the ideal step location, $\boldsymbol{x}_{id}$, calculated by the step planner, always lies in $\mathcal{A}_S$.
- The kinematic constraints limit $\mathcal{A}$.

The algorithm takes an iterative approach. A grid of seed points is set over $\mathcal{A}$. The seed points, which lie in $\mathcal{A}_S$, successively become the starting points for searching a convex region surrounding them. Once the convex region around a seed point is determined, it is removed from the remaining valid area to avoid repetitive searches. Since it is valid per definition, the ideal footstep location is used as a first seed point. This helps to reduce the computational costs. For one seed point, the search for a valid convex region can be summarized as follows: The closest boundaries to the seed point are determined iteratively. The boundaries are therefore considered to be line segments with start and end points. Once the closest boundary, l_j, is found, it is added as a linear inequality to the valid region $\mathcal{C}_{V_k}$. Here, the convexity of the invalid region is used to efficiently find the closest boundaries. All boundaries outside $\mathcal{C}_{V_k}$ are skipped and considered inactive for the following steps (see Figure 3.18). The search stops when there are no active boundaries left. Algorithm 2 summarizes the procedure. The calculation of $\mathcal{C}_V$ has to be done only once before each of the robot's physical steps. The kinematically reachable area lies outside the camera's FoV. Therefore, the representation of the environment does not change during execution of the step.

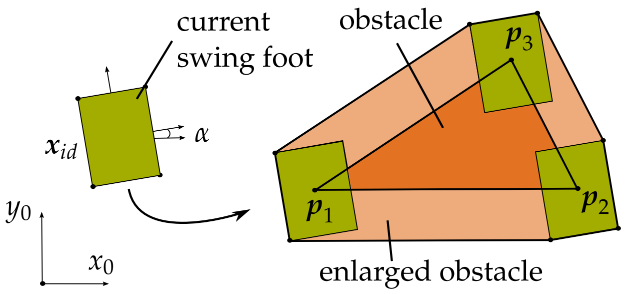


Figure 3.17: Obstacle (red) enlarged by foot geometry (light red). The ideal foothold position $\boldsymbol{x}_{id}$ and the foothold's rotation α [123].

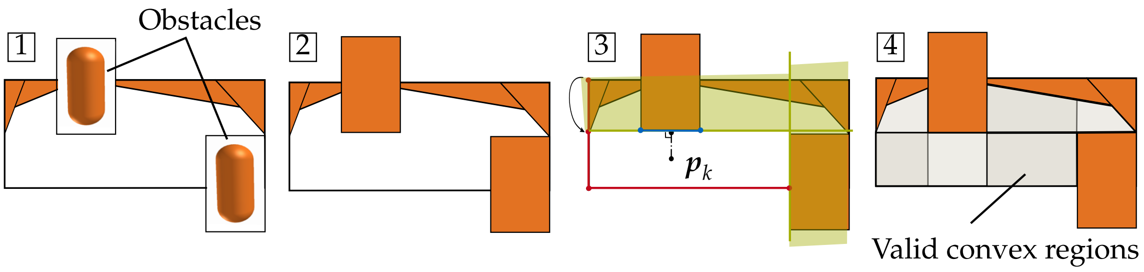


Figure 3.18: Finding convex valid regions. (1) and (2) obstacles and kinematic constraints. (3) Seed point p_k. Already found inequalities in green. Current closest line in blue. Remaining active boundaries in red. (4) Output of Algorithm; Multiple convex valid regions (different brightness level of ivory) [123].

Algorithm 2 Dividing A_S in convex regions [123]

```
 1: function FIND-CONVEX-REGIONS($\mathcal{A}_S$, $\bar{\mathcal{A}}_S$)
 2:     initialize set of seed points $\mathcal{P}_S$
 3:     $\mathcal{C}_V = \{\}$
 4:     for all $p_k \in \mathcal{P}_S$ do
 5:         if $p_k$ is valid then
 6:             j ← 0
 7:             repeat
 8:                 Calculate closest boundary line $l_j$ of $\bar{\mathcal{A}}_S$
 9:                 Add $l_j$ as inequality to boundaries of $\mathcal{C}_{V_k}$
10:                 Remove all inactive boundaries of $\bar{\mathcal{A}}_S$
11:                 $j \leftarrow j + 1$
12:             until no more active boundaries
13:             remove $\mathcal{C}_{V_k}$ from $\mathcal{A}_S$
14:             update $\mathcal{C}_V = \mathcal{C}_V \cup \mathcal{C}_{V_k}$
15:     Output: $\mathcal{C}_V$
```

Discussion

In summary, all of the methods presented can be used to find a valid point which is close to the desired invalid point. In the tested implementation the *sampling*-based method showed strong dependence on the grid size. The *Geometric Testing*-method and the *Safe regions*-method showed similar results in terms of computational costs and quality of the solution. However, the latter has the advantage that it calculates not only the closest point but also $\mathcal{C}_V$. These inequalities are suitable for optimization algorithms as shown below. Therefore the *Safe Regions*-method is chosen for the final implementation on the robot.

3.4.5 Footstep Modification with Geometrical Constraints

Finally, this section describes how the algorithm for finding a point in the safe regions can be combined with the optimization of the next foothold positions (compare Subsection 3.3.5) for stabilizing the robot.

Constrained Optimization Result

One straightforward solution for considering safe regions is to project the optimized quantities $\Delta\boldsymbol{x} = [\Delta L_x, \Delta L_y]^T$ onto the safe regions. This way optimization for the step length modifications can be done separately in the x- and y-directions. The optimization results are projected onto the cone of $\mathcal{A}_S$. Two different criteria were tried to find the point that is closest to the optimal solution: the geometric distance between $\Delta\boldsymbol{x}$ and $\Delta\boldsymbol{x}^*$ and the point with the best (lowest) costs determined using (3.12). For the second criteria several candidate points are generated all lying on the cone of $\mathcal{A}_S$. Nevertheless, this requires additional time consuming evaluations of (3.11) and (3.12). Both methods were tried with the result that they perform similarly and consequently the closest distance criterion was chosen.

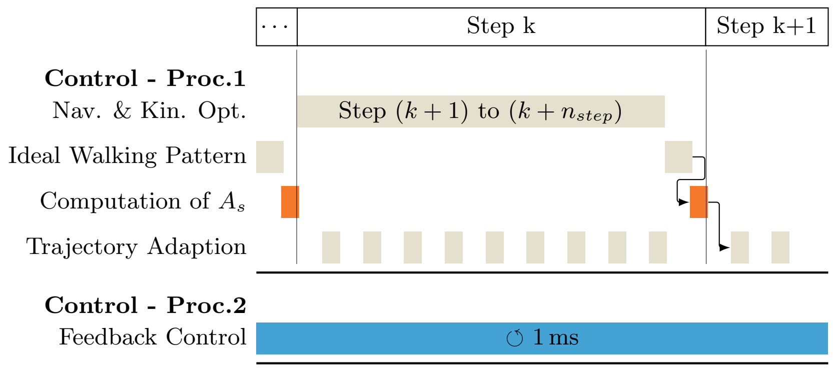


Figure 3.19: Integration of constraints calculation in multi-process and multi-thread software architecture of Lola's real-time walking control system (compare Figure 3.11). Adapted from Hildebrandt et al. [123].

Optimization with inequality constraints

It is mathematically more elegant to include the inequality constraints directly in the optimization. This can be realized with hard constraints or using a penalty function. The later has the advantage that the problem is again unrestricted and exhibits better computational time. The prediction model (3.11) is extended by an additional passive DoF,

Table 3.1: Computational time summary for maximal four obstacles. Runtimes are obtained from the real-time QNX computer [123]. See Section 3.1 for hardware specifications.

method	avg. [µs]	max. [µs]
Computation of A_s	600	2000
Find closest point	4	250
Trajectory Adaptation (total)	1000	2500

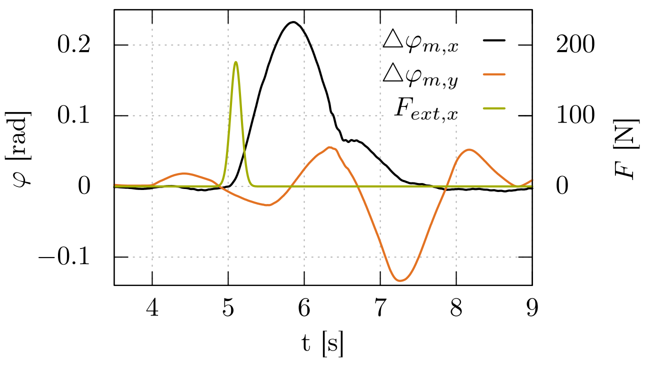


Figure 3.20: Synthetic simulation example: disturbance force and resulting inclination errors.

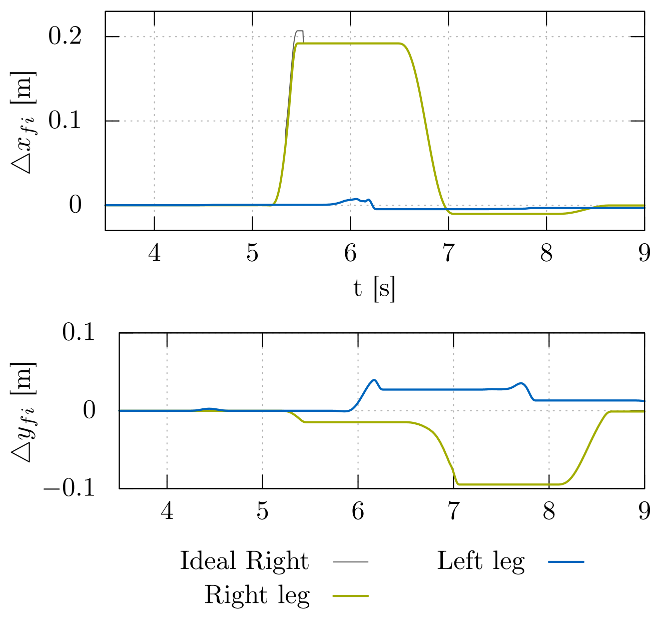


Figure 3.21: Synthetic simulation example: foot trajectory adaptations for stabilizing the robot. Ideal trajectory would be executed without the obstacle in front of the right foot (see Figure 3.22).

φ_y, which is an inclination in y-direction. The foot trajectory

$$_T\boldsymbol{r}_{f1} = {_T\boldsymbol{r}_{f1,id}} + \Delta\, {_T\boldsymbol{r}_{f1}}\left(\Delta L_x, \Delta L_y\right) \tag{3.20}$$

includes both final foot step modification quantities ΔL_x and ΔL_y. The EoM of the spatial model with $\boldsymbol{q}_s = [z, \varphi_x, \varphi_y]^T$ can be derived and written as first order differential equation

$$\dot{\boldsymbol{z}}_s = \boldsymbol{f}_s(\boldsymbol{z}_s, t, \Delta L_x, \Delta L_y) \quad , \boldsymbol{z}_s = [\boldsymbol{q}_s, \dot{\boldsymbol{q}}_s]^T. \tag{3.21}$$

The cost function for the optimization is rewritten for the spatial model (3.21) as

$$\begin{aligned} J_s = \boldsymbol{z}_s^T \boldsymbol{S}_z \boldsymbol{z}_s + \Delta\boldsymbol{x}^T \boldsymbol{S}_p \Delta\boldsymbol{x} + \int_{t_0}^{t_f} \boldsymbol{z}_s^T \boldsymbol{Q} \boldsymbol{z}_s \mathrm{d}t \\ + h(\Delta\boldsymbol{x}) \end{aligned} \tag{3.22}$$

and extended by an additional penalty term

$$h(\Delta\boldsymbol{x}) = \begin{cases} \beta\left(\boldsymbol{x}_m - \boldsymbol{x}_p\right)^2 & \boldsymbol{x}_m \in \bar{\mathcal{A}}_S \\ 0 & \boldsymbol{x}_m \in \mathcal{A}_S \end{cases} \tag{3.23}$$

that includes the distance from $\boldsymbol{x}_m = \boldsymbol{x}_{id} + \Delta\boldsymbol{x}$ to the closest valid point $\boldsymbol{x}_p$ at the cone of $\mathcal{A}_S$. The additional weight β is set to a value higher than all other weighting matrix entries. The optimization result from (3.21) and (3.22) is not necessarily valid since invalid solutions are penalized but not completely avoided. Nonetheless, the solution is at least close to $\mathcal{A}_S$. Consequently, the validity of the optimization result, $\boldsymbol{x}_m$, will be verified and if necessary projected onto $\mathcal{A}_S$ as described in Subsection 3.4.5. In simulation and experiments this strategy shows the ability to find solutions that require over-stepping of obstacles to maintain the robot's balance. This is possible because not only one safe region but a set of safe regions is used. Instead of using always the closest point as the starting point for the optimization, the starting points in each of the safe regions can be used as well. That way, multiple optimization processes can be run in parallel and the optimal result out of all optimization processes can be chosen as final result. So far, this approach has not shown superior results to just using the closest point as starting point. The same extension is applicable for the method *Footstep Modification with Geometrical Constraints*.

Implementation Details

Figure 3.19 depicts the integration of methods for versatile and robust walking in the overall software architecture. Before a new step k begins the ideal motion is planned (*Ideal Walking Pattern*) and the set of valid regions $\mathcal{A}_S$ is computed from the environment model. While the environment model is continuously updated by the *Vision System*, $\mathcal{A}_s$ is computed only once before a new physical step. This is sufficient since obstacles that are within the kinematic limit of one step lie outside the camera's FoV and are, therefore, no longer updated. The *Trajectory Adaption*, as part of the *Feedback Control* prevents collisions that a footstep adaptation between the swing foot and the obstacle might cause. This may otherwise occur since only the final position is checked and the swing foot height is adapted via a heuristic. The computational times of the presented algorithms are summarized in Table 3.1.

Results

The method is presented in an simple synthetic example using a multi-body simulation. The simulated LOLA is commanded to walk in place while it is subjected to a disturbance

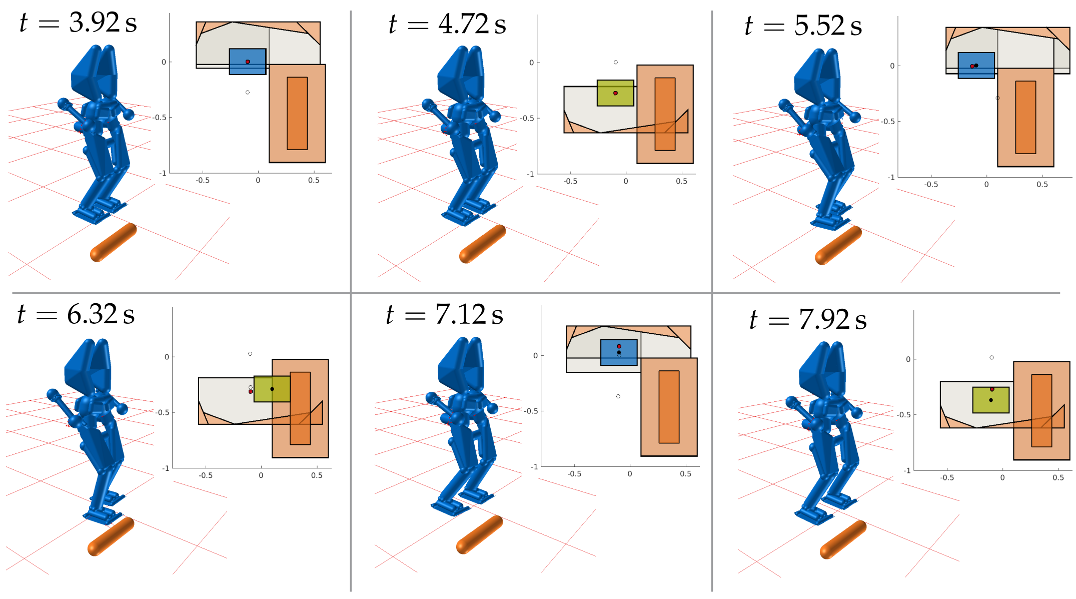


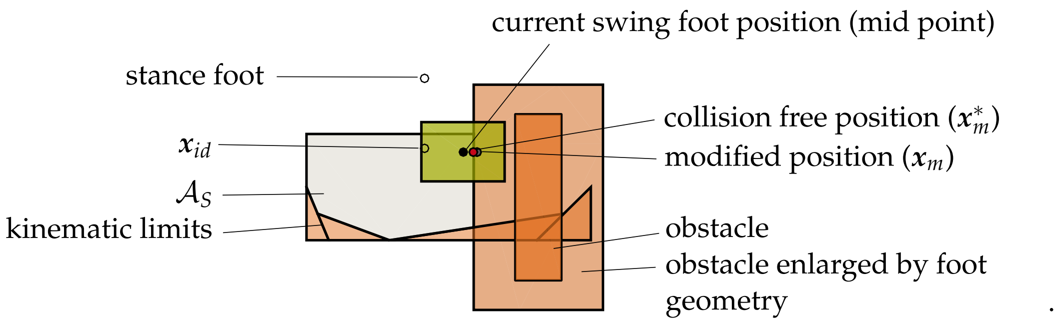
Figure 3.22: Synthetic simulation example: collision model of the robot and footstep adaptation with one obstacle at different time instants. The robot is stepping on the spot and a forward disturbance force is given around $t = 5\,\mathrm{s}$. See Figure 3.23 for the explanation of the 2D plots.

force in its walking direction (x-direction). One obstacle is placed close to the robot to limit feasible footstep modifications. Figure 3.20 shows the disturbance force and the resulting inclination errors for the constrained optimization result (see Subsection 3.4.5). The robot is still able to stabilize itself with the limited foot positions (Figure 3.21). A snapshot at each walking step of the 3D-collision model and the 2D-polytopes is shown in Figure 3.22. An explanation of the 2D plot is given in Figure 3.23. The same simulation experiment was conducted with the 3D-prediction model including the inequality constraints during optimization (see Subsection 3.4.5). Results are not presented separately since they closely resemble those shown above.

3.5 Summary

In this chapter, a control architecture to allow for versatile and robust walking in unknown environments is presented.

The common platform, the basic framework for stable and fast walking, is summarized. The basic framework is extended by three modules: a perception module, a motion generation module and a module for disturbance rejection. These modules are introduced.



current swing foot position (mid point)
stance foot
x_{id}
$\mathcal{A}_S$
kinematic limits
collision free position (x_m^*)
modified position (x_m)
obstacle
obstacle enlarged by foot geometry

Figure 3.23: Explanation of the 2D plots to visualize the overall "find safe foothold" algorithm.

Special focus is set on their integration in the overall concept and their integration with each other. In addition to the introduction of the concepts, implementation details of the software design are given.

Furthermore, the experimental platform, the humanoid robot LOLA, is presented. Its hardware setup, including available sensors and computers, is presented.

Chapter 4

Autonomous Navigation

In the previous chapter, the framework for versatile and robust walking is presented. This chapter focuses on the *Navigation*-module of the robot. The presented approach allows for reactive path planning in previously unknown and dynamically changing environments. In the scope of this thesis, it represents the *Sequence Planning* for humanoid locomotion (see Figure 4.1).

An application scenario of these methods would include a planner or a user with limited informations about the environment that roughly guides the robot. The developed methods would reactively navigate the robot through dynamically changing environments in short distances, as for example a distance of $1 - 10$m in a room, relying only on oboard sensing. Path planning methods for long distances are not investigated. However, possibilities are presented to combine the reactive methods with long distance paths calculated on a higher planning level. These methods are generalized to a concept for whole body pose planning in Kammermeier [142].

In Section 4.1, a short insight on the characteristics of bipedal navigation is given. Possible interfaces and basic requirements on the *Navigation*-module are defined. Furthermore, a literature overview is given. Section 4.3 presents the methods for bipedal navigation. It formally defines the planning problem and introduces an A*–based step planner. Specifics of the presented approach are emphasized. This part is based on Hildebrandt et al. [42]. The first implementation of the step planner were done in collaboration with Moritz Sattler, Wolfgang Wiedmeyer and Johannes Klotz [146, 150, 154]. Section 4.4 and Section 4.5 discuss different aspects of the methods presented in Section 4.3 and present further extensions. Section 4.4 introduces a novel method for using a mobile platform planner for bipedal locomotion. Different possibilities to beneficially couple the mobile platform planner and the A*–based step planner are developed and discussed. The methods as well as the results are published in Hildebrandt et al. [119]. Section 4.5 introduces methods for softening the discretization done in Section 4.3 and to exploit the continuous range of values. Finally, Section 4.6 is devoted to a summary.

4.1 Bipedal Navigation

In the following, a short overview of interfaces for human-robot interaction are given. They represent basic options which can be easily extended by more sophisticated solutions as proposed in Section 4.4. Furthermore, Subsection 4.1.2 summarizes the navigation problem of bipedal locomotion from a planning point of view.

4.1.1 Human-Machine Interfaces

According to its preferences and the application scenario of a bipedal robot, a user should have the possibility to guide the robot with a joystick, give it desired walking parameters

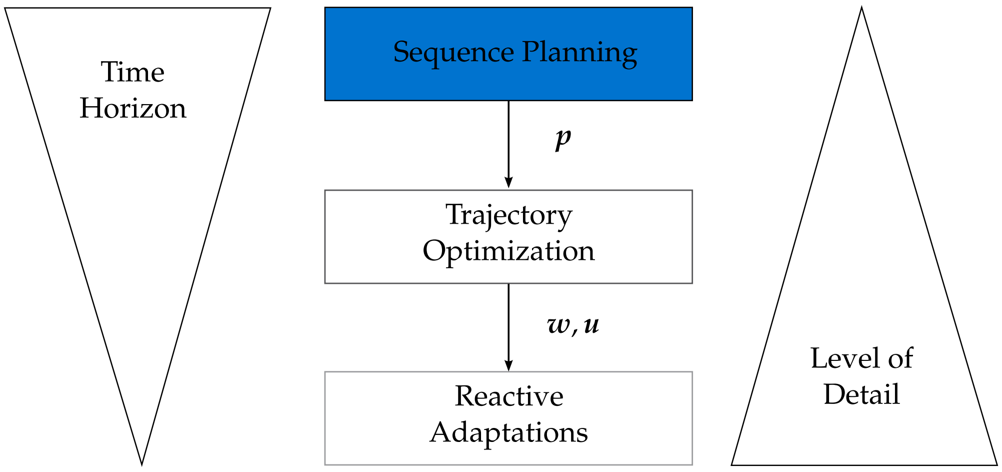


Figure 4.1: Hierarchical Approach for Motion Generation.

or set intermediate goal positions that the robot should reach. The robot should reactively follow these high-level commands and take care of finding a safe and optimal path. Different user interfaces pose different requirements on the navigation module. They are discussed in the following.

Joystick – Control

The joystick–control is a direct interface for controlling the robot. Similarly to a video game, the user gives the robot walking direction and speed as input via the joystick. To allow for direct control of the robot, Buschmann et al. [15] implemented the velocity vector to be relative to the current robot position.

It is assumed that the user has only limited information about the environment due to, for example, limited bandwidth of data connections, in real world scenarios. Therefore, the robot should be able to follow the user input as close as possible while taking care of traversable obstacles. This concept is inspired by Chestnutt et al. [17].

In the presence of non-traversable obstacles, the joystick–control reaches its limits. The robot has to differ largely from the desired input to find a valid path. Thus, the interpretation of the user input gets difficult. Does the user want the robot to follow still the original velocity vector? Should the robot interpret the velocity relative to its current position or relative to the position when it receives the command?

Since large deviations of the robot from the original input do not coincide with the user's intuition [2], the user is expected to adapt its commands to the robot's current position. As presented in Wang et al. [111], feedback from the robot to the user, as for example force feedback, would help to improve the interaction between user and robot.

For the *Navigation*-module, the following requirement is derived: the navigation module should have small calculation times to be as reactive as possible. It should be able to take changing user input at each step into account. Since the user input may change continuously, planning horizons of more than a couple of steps are not required.

Desired Step Parameters

The user should be able to command the robot directly via setting desired step parameters as rotation angle, step length or step width. The robot should stay as close as possible to the user chosen parameter while taking the environment into account. This is similar to the joystick–control, since the velocity vector can be transformed to desired step parameters. However, the transformation between velocity vector and desired step parameters is done by an algorithm.

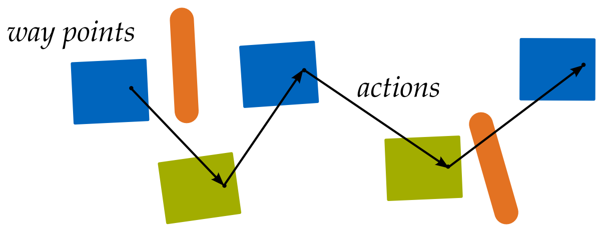


Figure 4.2: Hierarchical approach: simplification of robot motion via foothold positions (*way points*) and whole-body motions connecting the footholds (*actions*). Footholds in blue and green, obstacles in orange.

Goal Position

Additionally to the direct way of controlling the robot's motion via joystick or desired step parameters, the user should have the possibility to set goal positions. The robot should be able to walk autonomously to the goal positions taking the environment into account, while staying close to a desired walking pattern provided by the user. It should find the optimal path and may deviate from the direct way to the goal. In contrast to the joystick–control, the user is not expected to change its input continuously.
This chapter focuses on the input via *Goal Positions*. The input via *Desired Step Parameters* or joystick can be interpreted as the input via *Goal Position* by assuming virtual changing goals to accelerate the navigation module as explained in the following. In real-world scenarios, the robot is assumed to navigate in unknown and dynamic environments. Due to the moving and limited field of view and sensor noise, even static environments are perceived and modeled as dynamically changing. For this reason, regardless of the user interface, calculation time plays a crucial role for the navigation module. It is discussed in the following.

4.1.2 Problem Description

The proposed interfaces represent possibilities to interact with the robot and define requirements. Most current humanoid control frameworks solve the motion generation problem using a hierarchical approach to achieve real-time bipedal walking [12, 52, 80, 105]. In the following, this premise is followed.

The hierarchical approaches allow us to separate, up to a certain level, the navigation problem from the walking pattern generation. First, consecutive states (the footholds) are determined and then the full-body motion is calculated connecting the discrete footholds. From a planning point of view, path planing for humanoids is an interesting and challenging research topic: a walking path of a bipedal robot can be described via consecutive states. It is comparable with a set of end-effector poses a manipulator has to reach sequentially. Following this idea, the footholds are considered in the general concept of end-effector poses *way points* (see Chapter 2). Each *way points* is connected via a continuous motion to its predecessor. In the case of bipedal walking, the humanoid robot has to execute a swing-foot movement to walk. This whole-body motion is called an *action*. Thus, the position of each *way point* relative to its predecessor is limited by the capabilities of the robot. Figure 4.2 depicts a sequence of *way points* and *actions* for bipedal locomotion.

The partially discrete character differentiate the navigation problem from continuous 2D path planning for, e.g., cars. On the one hand, the discrete character of the *way points* complicates the motion planning problem. It is not possible to plan continuous trajectories and directly use a mobile platform planner. The search space consist of discrete areas which depend on each other and which are connected by the robot's motion. On the other hand, consequently, it gives the biped robot more possibilities to navigate in complex environments, for example by traversing obstacles. While most mobile platform planners can be applied to solve the navigation problem for bipedal robots by heuristically adapting foothold position to the 2D path, it prevents exploiting the characteristics of bipedal locomotion.

4.2 Related Work

In the following, the research on path planning for humanoid robots is divided in two approaches: most of the published methods propose to discretize the robot's kinematically reachable area and to apply implicit graph search algorithms to search for step sequences on the set of discrete areas. This body of literature is presented in Subsection 4.2.1. Subsection 4.2.2 gives an overview of approaches which are not based on graph search algorithms.

4.2.1 Graph-Search-Based Approaches

Early work on autonomous walking is presented by Kagami et al. [51] and Okada et al. [83]. They use stereo data from a vision system which is transformed into a discrete 2D height map. Based on the 2D height map and on a discrete set of statically-stable single-step motions, they propose to search footstep transition graphs for footstep sequences via heuristics. Although the system is limited to environments with sufficient structure for dense stereo algorithms, their experimental platform, the robot *H7*, is able to navigate through cluttered environments without colliding. Nishiwaki et al. [82] present an extension which allows the robot to follow a color-coded object.

A similar approach is presented by Gutmann et al. [39]. Using only on-board sensing, the proposed algorithm converts stereo data into a labeled grid with height information and a 3D occupancy grid. The algorithm is limited to environments with enough texture, as well. Based on the labeled grid, an A*-based algorithm computes a path assuming the cells as nodes [90]. This allows the robot to climb stairs or navigate through obstacles, but not to step over them. Additionally, the coarse grid limits the robot in its movements. This approach is successfully tested on Sony's robot *QRIO*.

Experiments on climbing stairs and avoiding obstacles is also conducted by Michel et al. [72]. Based on object models, the robot is able to localize itself with respect to recognized objects. Using an A*-based footstep planner, the robot avoids obstacles and climbs stairs.

Although the presented approaches render humanoid robots more autonomous, they are all restricted to environments which fit their particular assumptions. Chestnutt et al. [20] generalize these results. They set up a laser scanner to build a 2.5D map of the environment. The laser scanner has the advantage of providing quite accurate distance measurements. That way, the system does not longer strongly rely on specific textures. The required pivoting of the scanner to obtain a full 3D point cloud leads to a measurement time of 1s per scan as reported by Nishiwaki et al. [80]. This leads to a delay for taking new obstacles into account. Chestnutt [18] and Chestnutt et al. [19] present and compare local methods to adapt the fix discrete *action set* of an A*–Search to the environment. These methods improve largely quality and speed of the A*–Search applied to cluttered environments. Furthermore, Chestnutt [18] discusses the influence of the local step adaptations to the optimality of the search. They show very impressive experiments with their experimental platform, the robot *HRP2*. Using an external motion capture system for localization of the robot, it is able to move autonomously through a cluttered environment, while traversing obstacles and stepping onto platforms.

Hornung et al. [45] and Hornung et al. [46] present several methods to speed up the

A*–Search especially for long footstep plans: they evaluate the influence of the heuristics to improve the speed of the A*–Search based footstep planning. They introduce combinations of the A*–Search with locally acting, randomized components. This helps to avoid local minima and accelerates the planning process. Furthermore, it reduces the dependence of the A*–Search to well-designed heuristics. For further acceleration, Hornung et al. [45] propose to exploit the knowledge of the terrain by using 2D path planning algorithms in regions without obstacles and a more sophisticated footstep planner in cluttered regions.

Garimort et al. [33] and Hornung et al. [46] published the developed algorithms as an open source library. This library is used by Stumpf et al. [102] for the *DARPA Robotics Challenge* and adapted to the human-size robot *Atlas*.

The previously presented approaches have characteristics in common, namely that they do not explicitly take the stepping motion of the robot in the footstep planning into account. They use heuristics to decide whether a footstep location is reachable or not. Perrin et al. [88] presented an approach which directly takes the stepping motion into account. They divide each step in half-steps. At discrete configurations they check for collisions using precomputed body approximations. In a second step, they smooth the determined half step combinations to get an approximately dynamic motion of the robot. For a near real-time solution, Baudouin et al. [7] simplified the collision checks. Additionally, to handle the high dimensional search space in a reasonable time, they had to take a random solution using an RRT method instead of the A*–Search into account. This sped up the planning process, but it still needs 2.5 s to find a solution. This fact makes an application in changing environments and/or changing user input difficult. They have not integrated their navigation module in a framework with on-board-sensing. Instead, either a known environment is assumed or a motion capture system is employed.

Maier et al. [68, 69] present a framework which combines perception, path planning and collision checking for the robot *Nao* in a completely unknown and cluttered environment. They set up an RGB-D sensor to identify obstacles and patches the robot can step on. To analyze the stepping motions in an A*–search based step planner, an approach similar to Perrin et al. [88] is applied: all the robot's possible actions are approximated via inverse height maps. These height maps are further used for collision checks with the environment. The inverse height map represents conservative approximations of the kinematic capabilities of bipedal robots. They model the whole space traversed by the robot's parts with one collision object. Due to disturbances and timing problems between encoder reading and depth camera data, the robot has to stop between every second step in order to update the map. This makes fluent movements impossible. Although impressive results were shown using graph-based search algorithms, they have the disadvantage, that the quality of the resulting path always depends on the discretization level: a coarse discretization has the advantage of a very fast search, but the quality of the solution is suboptimal due to the limited discrete action set.

4.2.2 Not Graph-Search-Based Approaches

In contrast to the presented methods taking explicitly the discrete character of the consecutive footholds into account, methods are presented which simplify the navigation problem to the search of a continuous 2D path. Based on the 2D path, local methods are applied to adapt the footholds. In 2004, Cupec et al. [25] and Lohmeier et al. [66] conducted experiments with the robot *Johnnie*. The work is a break-through for vision-guided walking. The proposed system is able to detect geometries based on their edges. Heuristically, a step planner adapts the step length to step over or onto geometries lying horizontal or vertical relative to the robot.

Buschmann et al. [15] applied a mobile platform planner to allow for autonomous walking using only on-board sensing. Instead of building up a global map, they propose to test 2D trajectories and check whether or not they are viable. The interface between the step planner and the 2D path relies on the directed velocity. The step planner heuristically adapts the step parameters to follow the desired velocity vector. This enabled safe robot movements in an unknown environment. However, it does not allow the robot to step onto or over objects.

Karkowski et al. [54] follows a similar idea. They propose to search for sub-goals directing to the goal. The sub-goals are connected via line segments. The robot's foothold positions are geometrically calculated with respect to the 2D segments. The robot's kinematic capacities such as stepping over obstacles are not fully exploited, nor is the solution's optimality explicitly considered.

Within the *DARPA Robotics Challenge Trails*, Deits et al. [26] presented an interesting method different to the previous approaches. The presented vision system determines convex areas in which the robot is able to step. Starting with a fixed number of steps, the step planning problem can be solved using a mixed integer optimization. Due to the convex characteristic of the areas, the optimization problem converges fast and the time consuming collision checks can be omitted. Drawbacks of this approach are the beforehand defined total number of steps, the complex calculation of convex areas in the presence of changing environments and the need of a user to define seed points for the convex areas.

4.3 Step Planning

The objective of the navigation module follows ideas from Buschmann et al. [15] and Nishiwaki et al. [80]. In contrast to most related work, its purpose is not to find long distance paths but to give the user an reactive system which is able to safely navigate in cluttered environments. This is especially important, if no full map of the environment is available and the user as well as the robot depends only on the robot's limited field of view.

In the following, the navigation problem is formally defined. Subsequently, the approach developed within this thesis to solve the navigation problem is presented.

The methods as well as the results are partly published in Hildebrandt et al. [42].

4.3.1 Formal Definition

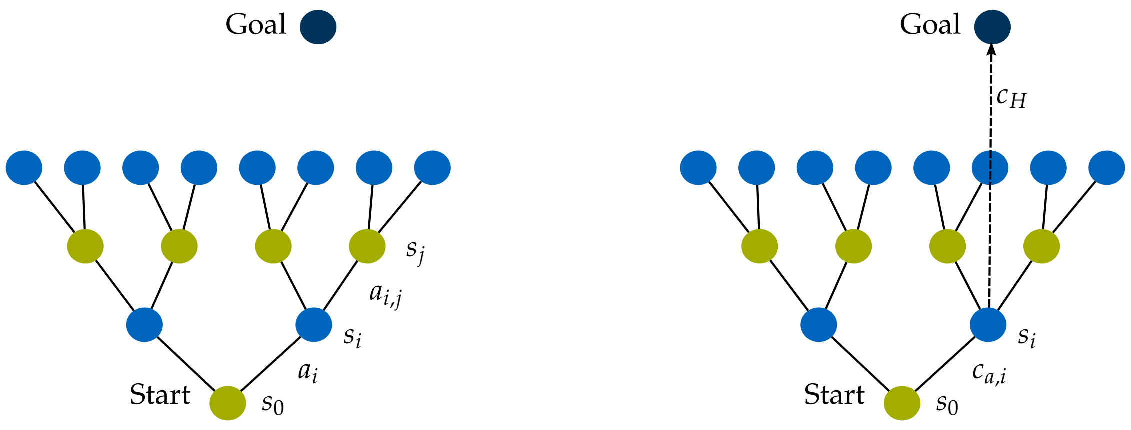
Chestnutt [18] presents a formal representation which is used in this work. As mentioned in Subsection 4.1.2, the robot's motion is approximated as a sequence of *way points* which are connected by *actions* to handle the search space. The *way points* represent the robot state reduced to the set of footholds S_F. Thus, one *way point s* consists of the current stance foot $stance = (right, left)$ and its global position $\boldsymbol{r} = (x, y, z)$ and orientations $\boldsymbol{\theta} = (\theta_x, \theta_y, \theta_z)$. Since θ_x, θ_y and z result directly from the position x, y and θ_z in the environment (the surface on which the foot stands), they do not represent DoFs, but they are needed for further calculation as for example costs and actions. The *way points* are defined as

$$s = (x, y, z, \theta_z, stance). \tag{4.1}$$

The stepping motion is approximated by the action model. It is defined as

$$a = (\Delta x, \Delta y, \Delta z, \Delta\theta_z, \boldsymbol{h}_{obst}, h_{Step}, c_a). \tag{4.2}$$

Since the possible footholds are symmetric for the left and the right stance foot, one action model is defined for left and right stance foot. Δx, Δy and $\Delta\theta_z$ represent the possible displacements and rotations relative to the current stance foot. In order to take the stepping motion into account, the action model is augmented by $\boldsymbol{h}_{obst}$ and h_{step}. The



(a) Search tree. **(b)**Costs.

Figure 4.3: A*–search: search tree with states (s_k), actions (a_k) and costs (c_k).

parameter $\boldsymbol{h}_{obst}$ denotes the dimensions of the obstacles the robot has to step over or to swing by to reach the next s. The parameter h_{step} denotes the height change the action has to make to allow for stepping up or down. The cost for each action is denoted by c_a. The costs are explained in Subsection 4.3.4 in more detail. Thus, the whole robot motion is approximated with a sequence of *way points* and *actions* as follows

$$s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \xrightarrow{a_3} \ldots \xrightarrow{a_n} s_{n+1}. \tag{4.3}$$

4.3.2 Discretization

Having a reduced state representation depending on three DoFs, it is possible to discretize the search space and still search for a sequence of *way points* in reasonable calculation time. Consequently, for each *way point* s_i a set of n_{disc} *way points* as possible successors s_{i+1} is obtained. These *way points* represent the kinematically reachable area relative to the current stance foot. As mentioned above, the objective is to give the user the possibility to directly influence the robot's behavior. Buschmann [12] introduces an intuitive step sequence representation using the three parameters, $\boldsymbol{p}_{Step} = (L_x, L_y, \phi)$ denoting the step lengths in sagittal and in lateral direction and the turning angle. It uses a sequence representation based on a *standard circular path*. $\boldsymbol{p}_{Step}$ can be transformed to the displacement $(\Delta x, \Delta y, \Delta\theta_z)$ as

$$(\Delta x, \Delta y, \Delta\theta_z) = f(\boldsymbol{p}_{Step}). \tag{4.4}$$

In contrast to most publications, $\boldsymbol{p}_{Step}$ is discretized symmetrically around the user chosen desired $\boldsymbol{p}_{Step,des}$ in between set limits. The displacement $(\Delta x, \Delta y, \Delta\theta_z)$ is not directly discretized. That way, the user is able to intuitively change the discretization and influence its walking behavior.

4.3.3 Graph-Search

Based on the discretized search space, a search tree can be set up. Starting with the current *way point* s_0 of the robot, n_{disc} successors s_i can be reached. Each successor $s_{0,i}$ has again n_{disc} successors s_j and further on. The transition between the *way points* s_i and s_j are defined by $a_{i,j}$. Consequently, s_i represent the search nodes of the tree and $a_{i,j}$ the edges with the costs $c_{a,i}$. Graph-search algorithms applicable to search trees can be used to search for a sequence s_t of n_{goal} steps

$$s_t = \bigcup_{i=1}^{n_{goal}} s_i \tag{4.5}$$

which guides the robot to the goal with the lowest total costs

$$c_t^* = \sum_{n=0}^{n_{goal}} c_a. \tag{4.6}$$

Figure 4.3 depicts a simplified search tree.

In this work, as most of the work presented in Subsection 4.2.1, an implicit A*–Search [63] is used to search for the optimal step sequence. The implicit A*–Search allows to construct the search tree during run-time. That way, only a small part of the whole search space has to be discretized at the start of the search. This would be memory and time expensive. Additionaly, the A*–Search allows to use a heuristic to guide the search to the goal avoiding unnecessary node expansion. The A*–Search always expands the search tree at the node which is currently the node with the lowest costs. The cost of each node c_t is a sum of the real cost of the sequence of n_{Steps} nodes guiding to the currently analyzed node $c_{S,seq} = \sum_{n=0}^{n_{Step}} c_a$ and the heuristic cost c_H estimating the remaining cost till the goal is reached. It follows

$$c_t = \sum_{n=0}^{n_{Step}} c_a + c_H. \tag{4.7}$$

That way, a well chosen heuristic can significantly accelerate a search, since only nodes which lead to the goal are analyzed and expended. In most published work on step planners for humanoid robots, heuristics based on the Euclidean distance to the goal are used (see Subsection 4.2.1). Section 4.4 proposes a new method for the choice of the heuristic.

4.3.4 Cost Function Design

One of the shortcomings in publications of A*–based step planners is the cost function design. Cost function design is especially important if a user should interact with the robot. If the cost function is not designed in an appropriate way, the sequence executed by the robot may not match the user's expectations. Using, for example, an A*–Search with a heuristic which overestimates the real costs would also lead to suboptimal *way point* sequences. This questions the stated optimal character of an A*–based step planner with respect to the chosen cost function.

The costs of the descritized *way point* suggestions have to be designed in a way that the sequence calculated by the step planner meets the following requirements:

- Requirement 1: The user should be able to set goal positions interactively for the robot to reach. The sequence of the robot has to be the optimal sequence with respect to the chosen cost function.

- Requirement 2: The user has to be able to set desired step parameters $\boldsymbol{p}_{step,des} = (L_{x,des}, L_{y,des}, \phi_{des})$ reactively to adapt the robot movement to the current situation (see Hildebrandt et al. [118] and Hildebrandt et al. [42]). The robot should execute the desired step parameters, if possible, or the executed parameters should stay as close as possible to the desired ones. In this context, the robot is explicitly not expected to reach the goal as fast as possible, but to reach the goal executing the desired step parameters. If the influence of minimizing the step number is too large in the A*–Search, the robot would always try to move as fast as possible. To the best of the author's knowledge, in other A*–based footstep planner (see Subsection 4.2.1) only goal positions or desired 2D paths are given to follow. This does not take the speed resp. the step parameters of the robot into account which seems to be of importance in case a user should interact with the robot.

- Requirement 3: The A*–Search has to be as fast as possible to meet the real-time requirements and enable walking guided by a joystick.

Based on the requirements, the cost function is derived as follows: comparable to other A*–based step planners, the total cost of each *way point* c_t is calculated based on (4.7) with the Euclidean distance as a heuristic for the search. To take rotations of the robot into account the Euclidean distance is extended by the rotation angle. It follows

$$c_H = \|\boldsymbol{W}(\boldsymbol{x}_{Current} - \boldsymbol{x}_{Goal})\| \tag{4.8}$$

with $\boldsymbol{x} = [x, y, \theta_z]$ and $\boldsymbol{W}$ a weighting matrix to eliminate dimension effects due to the comparison of rotations and length. For the optimality of the path (requirement 1), it has to be ensured that the heuristic always under-estimates the remaining costs of the path $c_{S,seq,n_{goal}-n_{Step}}$:

$$c_H \leq c_{S,seq,n_{goal}-n_{Step}}. \tag{4.9}$$

The cost for each *action* is defined as

$$c_a = c_{step,des} + k\|\boldsymbol{W}(\boldsymbol{x}(\boldsymbol{p}_{step}) - \boldsymbol{x}(\boldsymbol{p}_{step,des}))\|. \tag{4.10}$$

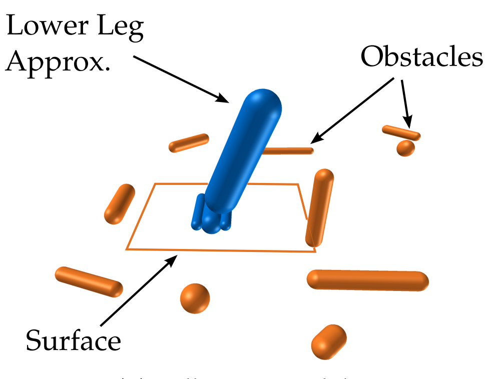
$c_{step,des}$ denotes the execution cost of the desired step parameters. To ensure requirement 1, $c_a > \Delta c_H$ is demanded, the difference in the heuristic should be smaller than the cost of the executed *way point*. The execution cost is set to

$$c_{step,des} = \|\boldsymbol{W}\Delta\boldsymbol{x}(\boldsymbol{p}_{step,des})\|, \tag{4.11}$$

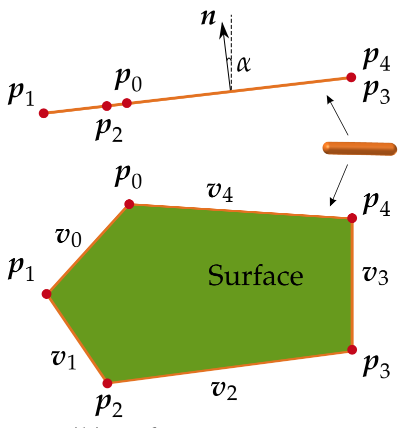
which results in a ratio of 1 between $\Delta\boldsymbol{x}(\boldsymbol{p}_{step,des})$ and cost for executing $\boldsymbol{p}_{step,des}$. In order to overestimate the heuristic and penalize deviation from the desired step parameters, the scaling factor k has to be larger than 1.

4.3.5 State & Transition Evaluation

When a node is expanded, the n_{discr} successors s_i are created. Each of them has to be evaluated for viability. Therefore the position as well as transition from the current node to its successor, the swing-foot movement, has to be analyzed. The node expansion and in particular the analysis of each *key pose* with respect to the world is the most time consuming part in the step planner. A disadvantageous choice in the world representation and the corresponding robot approximation may lead to long calculation times and limited robot movements. As stated in Subsection 4.2.1, most work on step planning for bipedal locomotion, as for example Nishiwaki et al. [80], use a collision world representation based on a binary 2.5D grid map. The corresponding robot model, the foot representation, consist of a planar rectangle. The step planner checks the rectangle against the discrete grid for the *way point* viability and its position in the world. Since the robot's kinematic movement is not taken into account, high safety margins are necessary.



(a)Collisions model.



(b)Surface representation.

Figure 4.4: Collision model of the step planner for LOLA stepping over an obstacle with movable leg segments. Surface representation: corner points with edges. Edges modeled as SSV-Objects for collision avoidance.

Collision Checking

A different approach is applied in this thesis. Instead of using a grid based environment representation, the SSV-representation presented in Subsection 3.3.2 is used in the step planner. That way, the step planner takes the same world approximation into account as all modules for motion generation. This avoids inconsistent motion planning and redundant information. Furthermore, the SSV-representation allows to check for viability in full 3D. The robot model is not a planar rectangle but a 3D model of the lower leg including the foot and a leg segment approximation. Instead of assuming a static approximation the lower leg is rotated with a constant angle for each step relative to the corresponding foot. The angle is calculated based on a heuristic taking the stance foot position relative to the desired center of mass in the current step into account. Thus it is possible to check more accurately for collisions in 3D already in the step planner. Therefore, the walking movement is taken into account avoiding high safety margins of the stance foot positions (see Figure 4.4a). The swing-foot movements are analyzed by approximating the motion with a sampled straight line on a reference height of the swing-foot. On condition that the robot's SSV approximation intersects with one or more obstacles, the parameter $\boldsymbol{h}_{obst}$ becomes the dimensions of the highest obstacle which overlaps with the robot's SSV approximation. If $\boldsymbol{h}_{obst}$ exceeds a threshold the successor node is not viable.

Spatial Walking

In addition to the viability of a created *way point* the step planner has to evaluate the 6D pose of the foothold. As introduced in Subsection 4.3.1, the rotation θ_y, θ_z and the height z depend on the environment and x, y and θ_z. Apart from areas the robot is not able to step on (obstacles), areas are introduced the robot can step on (surfaces) to the environment representation. As presented in Subsection 3.3.2, surfaces are represented by convex hulls descriped by polygons and a normal of the surface (see Figure 4.4b). The robot is not able to step onto the edges of the surfaces. Therefore, the edges are modeled as obstacles using line-SSVs. This representation has several advantages:

- Based on the current x and y value of the *way point* the step planner is able to determine the whole 6D pose of the foot just by checking in which polygon the current *way point* is lying. This can be implemented very efficiently in terms of computational time and memory space.

- Surfaces are completely defined by the corner points and the normal of the surface. This is a extremely dense representation which is memory efficient and simplifies communication between planning modules and vision system. The current implementation uses maximal eight corner points. Depending on the desired level of detail it can be easily extended to a higher (or lower) number of corner points.
- Additionally, surfaces are included consistently using SSV elements in the collision avoidance framework [118, 124] to avoid collisions of the robot with edges of platforms, stairs, etc..

4.3.6 Real-Time Application

Everybody who has played a video game knows how annoying it is when her or his character does not react fast to the desired input. In real applications, long delays between the users input and the robots reaction makes it difficult to command the robot. For this reason, when commanding the robot via a joystick, and in order to take changing environments into account, the step planner is expected to be as reactive as possible.

Error Handling

Due to the limited calculation time of $T_{calc} < T_{Step}$, the search may fail to find an executable path. For this reason, it is proposed to start a second search if the first fails. The second search uses an adapted discretization set preferring step length around zero to walk on the spot. The desired step parameters $L_{x,des}$, $L_{y,des}$ and ϕ_{des} are set correspondingly. That way, the discretization grid changes and influences the result of the search. Since both searches do not depend on each other, they can be parallelized and executed on the same time. Having only a limited number of processors, the current implementation uses still a serialized implementation of the first and the second search. In real world application, it is necessary to provide a fall back solution to avoid a failing of the system. One possibility to solve this issue might be a set of searches running in parallel with different parameter sets, similar to the proposed first and second search.

Joystick – Control

In the case the user guides the robot using a joystick, no goal position is explicitly set. In order to accelerate the A*–Search and guide the robot close to the desired input, a virtual goal is assumed. The virtual goal is calculated as the position the robot would reach executing the desired step parameters.

Receding Search Horizon

In real application, the step planner searches for a limited number of steps and not for the complete sequence connecting start and goal position for the following reasons: (1) The robot has only a limited FoV. Therefore, obstacles which are too far away can not be taken into account. (2) Calculating and then executing a long sequence of footholds requires an exact knowledge of the robot's odometry. It is assumed that the state of the robot diverges from its ideal odometry considered in the step planner. This assumptions are confirmed by observations in experiments. (3) The quality of the environment approximations of the vision system depends strongly on the distance of the robot relative to the objects. Therefore, the approximations' quality increases with decreasing distance. (4) Dynamic and unknown environments require a constant re-planning with small cycle times. Therefore, the number of steps the step planner should search for is limited. When a time limit is reached and not enough steps are found the search is aborted resp. the proposed error

Table 4.1: Results for simple and complex environment (see Figure 4.5). The significance of the quantitative search times for the real application is limited. It does not represent the search time on the real-time system of the robot and includes necessary logging of data.

Environment	simple	complex
number of node suggestions	85	85
number of steps	9	11
distance calculations	19699	293397
costs	5.02	5,5
search time step–planner [s]	0.33	9.64

handling is applied. Figure 4.6b depicts the result of a search which is aborted after a set limit of 7 steps was reached. The robot would execute the first step and search for the next steps when being closer to the goal. Naturally, the resulting sequence to the goal would not be optimal with respect to the defined costs and the robot may get stucked in local minima, but it represents a trade-off between reactivity and optimality. One possibility to solve this issue is presented in Section 4.4.

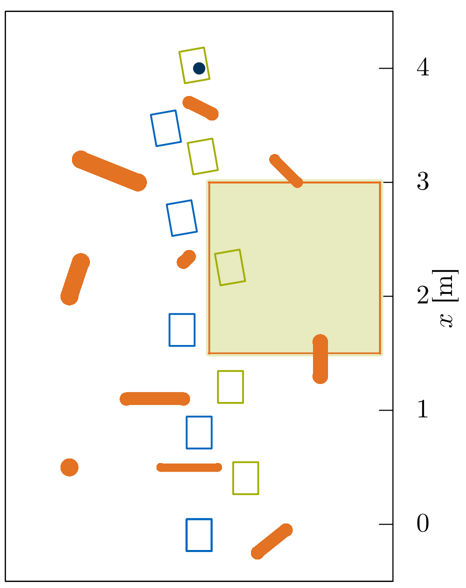
Environment Representation

The collision checks are not only binary checks for viability, but calculate a distance gradient. That way, for each node, the distances and the directions to the obstacles are known. This characteristic of the used world representation can be exploited. In order to save computational time the analysis of the swing-foot movement is omitted if no obstacle is inside a radius r_{obst} around the current node. Furthermore, an advanced handling of the distance calculation is implemented. Once the distances are calculated for one node to all obstacles, all successor nodes can use these distance informations. Depending on the distance of the current node to the last node having the distance information to all obstacles, distance calculations can be omitted for obstacles which are too far away. That way, the number of distance calculations can be reduced and therefore the search is accelerated. Furthermore, a grid based approach is analyzed. Based on a 2D grid of the environment, obstacles can be transferred to a quadtree representation widely used in informatics for data structures [40]. This clustering allows to analyze only currently relevant obstacles. In the analyzed environments, this approach has not shown an accelerated run-time. For larger search spaces with more obstacles, this approach might be beneficial.

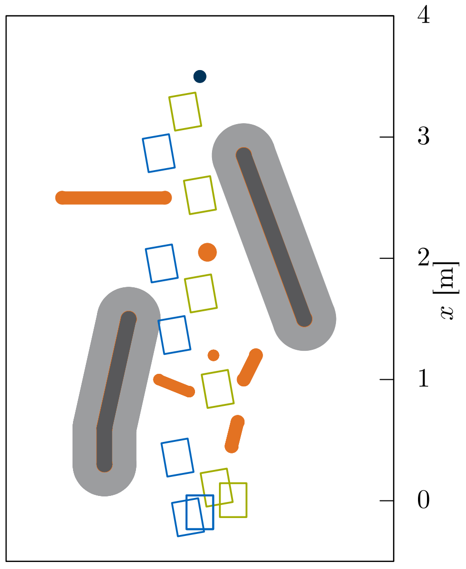
4.3.7 Results

The step planner is analyzed in multiple environments. Figure 4.5 depicts two exemplary environments.[1] Figure 4.5a represents an environment with few obstacles and one platform. It is chosen to be similar to the *simple environment* in Chestnutt [18]. Figure 4.5b includes non-traversable and traversable obstacles. It is similar to the *complex environment* in Chestnutt [18]. In both environments, the step planner is able to find a sequence which leads to the goal exploiting the characteristics of bipedal locomotion as stepping on a platform in the *simple environment*. Table 4.1 summarizes the results of the A*–Search. As expected, the A*–Search in the *complex environment* is more complex. It needs significantly more distance calculations which is directly coupled, as stated before, to the desired search time. However, the significance of the search time for the real application is limited. It does not represent the search time on the real-time system of the robot and includes logging of data. But it is a way to compare the search results in both environments.

[1]A video showing the results is available online at `https://youtu.be/ogZ9oBdCNU0`.



(a) Simple Environment.

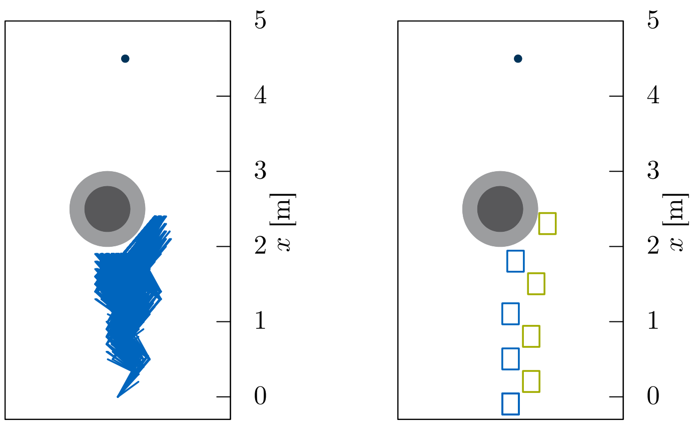


(b) Complex Environment.

Figure 4.5: Result of step planner in simple environment and complex environment. Search analysis is presented in Table 4.1. Simple environment includes surface represented with light green. Traversable obstacles in orange. Complex environment includes non-traversable obstacles represented in (dark) grey. Security margin (light) grey. Goal location in dark blue.

4.4 Reactive Navigation

In real-world scenarios with dynamically changing environments, computation time plays a crucial role. In the presence of non-traversable obstacles the search for discrete consecutive footholds gets complex and time-consuming [18, 42]. The heuristics used in the presented A*–search is one key to guide the search to the goal and, consequently, to reduce planning time. Figure 4.6 depicts a result of the step planner applied in a simple environment with one large obstacle. Figure 4.6a shows the evaluation of the analyzed nodes. The effect of the heuristic based on the Euclidean distance is clearly visible. The search is directed on a straight line to the goal, although the way is non-traversable. As a consequence, the remaining goal distance is greatly underestimated. The search tends to explore irrelevant areas and a large number of irrelevant step suggestions are evaluated. This requires a lengthy computation time and the step planners becomes impracticable for real-time applications. This example represents a typical application for conventional mobile platform planner. It seems reasonable to consider the environment at different detail levels and combine mobile platform planners with the step planner in order to improve the performance (see Figure 4.7). In the following, work is presented which analyzes the influence of the heuristic of the A*–search with focus on step planning. Furthermore, applications of mobile platform planners in step planning are discussed. The methods as well as results presented in this section are developed in collaboration with Moritz Klischat [145] and they are published in Hildebrandt et al. [119].



(a) Node expansion. **(b)**Calculated sequence.

Figure 4.6: Motivational example: large obstacle (grey) with security margin (light grey) in front of robot. Goal location in dark blue. Heuristic based on Euclidean distance guides search to goal. Search is aborted before goal is reached since desired number of steps is found.

4.4.1 Related Work

In Hornung et al. [46] the authors evaluate the influence of the heuristics to improve the speed of an A*–Search based step planner. They propose to dynamically adapt the weighting factor of the heuristic based on the Euclidean distance to the goal. Starting with a high weighting of the heuristic, the search is strongly guided in direction of the goal. That way, the calculation time can be reduced, but with the drawback of a resultant sub-optimal step sequence. By successively reducing the weighting of the heuristic and reusing results from the previous searches, the quality of the resultant step sequence is improved. They call the adapted search the ARA*–Search. For reducing the influence of well-designed heuristics, they introduce local-acting randomized components influencing the heuristics. This helps to avoid local minima and accelerates the planning process. This approach helps to find paths which follow predefined heuristics. Nevertheless, a poorly chosen heuristic could still lead the search in wrong directions. This would result in long calculation times. Chestnutt [18], Hornung et al. [45], and Karkowski et al. [54] propose hierarchical path planning methods. Their common idea is to combine a simplified global mobile platform planner with a detailed local step planner. The mobile platform planner calculates a 2D path in a simplified environment representation which is used to guide the local step planner search. Chestnutt [18] presents an hierarchical path planning approach for branched buildings, which uses three levels of detail: On the top-level, a global 2D path is divided into sub-goals. An A*-based mobile platform planner searches backwards from the sub-goal to provide the low-level step planner with a heuristic. This approach speeds up planning time significantly. Karkowski et al. [54], as shortly presented in Subsection 4.2.2, proposes to first plan a global 2D path using an A*–Search, as well. Then sub-goals are generated from the 2D path. The local step planner first constructs a path from line segments which is subsequently used for geometrically generating the foothold positions. This allows for real-time capability. However, the robot's kinematic capacities such as stepping over obstacles are not fully exploited, nor is the solution's optimality explicitly considered. Hornung et al. [45] proposes to use a combination of a mobile platform planner and a detailed step planner. In contrast to previous authors, they propose to switch between a mobile platform planner in regions without obstacles and to use a more sophisticated step planner in cluttered regions which allows for stepping over

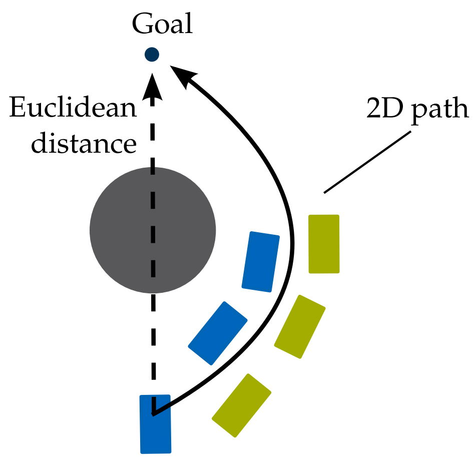


Figure 4.7: 2*D Pre-Planning* in A*−search.

obstacles. This method's performance relies heavily on the availability of obstacle-free space. Ayaz et al. [4] uses a simplified approach designed for finding multiple paths which resembles the principle of visibility graphs. The planner aims at finding the lowest-energy path. Sub-goals are placed next to the edges of obstacles and connected by straight step sequences. Since a relatively simple step planner is used, the resultant paths are not optimal. They are restricted to rather simple environments, even though small obstacles can be traversed. Unlike the work described above, Buschmann et al. [15] do not use a global map, but they propose to test a set of 2D trajectories whether or not they are viable. This enables safe and reactive robot movements in an unknown environment using only on-board sensing. This system, however, does not allow the robot to step onto or over objects. Furthermore, Chestnutt [18] discusses the benefit of a heuristic provided by a mobile platform planner. It is shown that the use of that heuristic, which is adapted to the environment, helps to significantly accelerate the A*-based step planner. Nevertheless, when using the mobile platform planner, the ability of the robot to step over obstacles or onto platforms is neglected.

4.4.2 Proposed Method

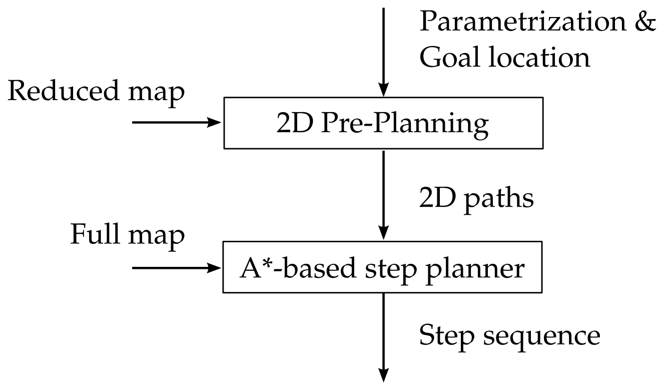


Figure 4.8: Control flow of 2*D Pre-Planning* integration.

The proposed approach combines ideas from most of the cited publications. The objective is to exploit the possibility to accelerate the step planner by providing a continuous guideline calculated by a mobile platform planner. However, the presented approach differs from the ones presented above (see Subsection 4.4.1): (1) different levels of environment details are used for both planners. For the mobile platform planner, a reduced map is used, which only takes non-traversable obstacles into account; the following A*-based step planner uses the full map. That way, the search for a step sequence is accelerated by using the 2*D Pre-Planning*, but still exploits the capacities of a bipedal robot. (2) sub-goals, which guide the A*–Search via heuristics based on Euclidean distances, are not used. Instead different strategies to use a continuous 2D path in the A*–Search are proposed and evaluated. Sub-goals as presented in Ayaz et al. [4], Chestnutt [18], and Karkowski et al. [54] could easily be integrated in this approach as intermediate goals. (3) In contrast to Chestnutt [18], Hornung et al. [45], and Karkowski et al. [54] not one final 2D path, but (similarly to Buschmann et al. [15]) different candidates are searched for. The set of 2D paths guides multiple searches of the detailed step planner. This approach is necessary as the mobile platform planner only uses a reduced map. Therefore, the quality of the provided 2D paths greatly depends on the presence of traversable obstacles or stairs which are not considered in the pre-planning. This approach is mainly suitable for real-time applications such as navigation in unknown areas - the robot has to react quickly to the newly discovered environment without global knowledge about the environment (in contrast to Ayaz et al. [4] and Karkowski et al. [54]). Figure 4.9 shows the basic approach of the method. A 2*D Pre-Planning* is first executed on a reduced map and provides the step planner a set of 2D paths.

4.4.3 2D Pre-Planning

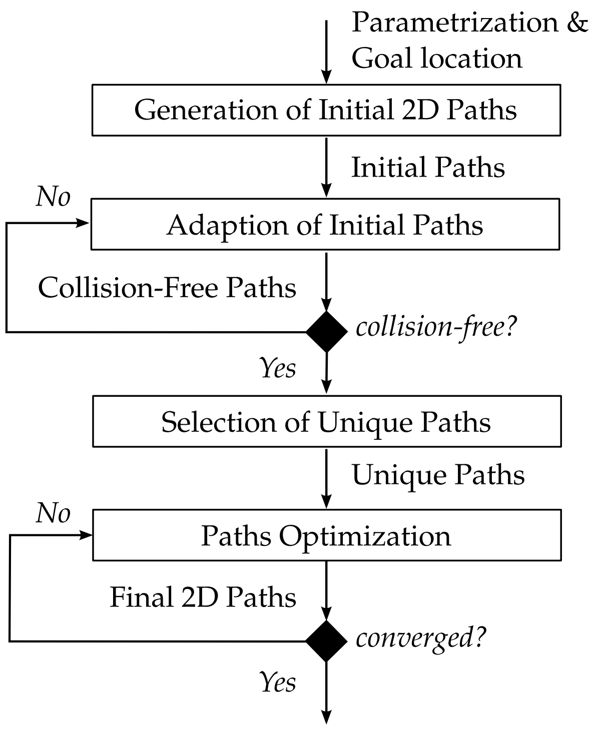


Figure 4.9: Flow chart for 2*D Pre-Planning*.

Mobile platform planners plan continuous paths and, therefore, do not take the robot's capability to step over obstacles into account. The bipedal robot could traverse obstacles which the continuous mobile platform planner would avoid. Following a continuous path with a bipedal robot could unnecessarily lead the robot walk a long way around. An overview of the method, which is presented in the following, is given in Figure 4.9. Based on a reduced map, a set of inital 2D paths is generated. These 2D paths are adapted to get collision-free paths. Out of all collision-free paths, subsets are identified which are not separated by obstacles. For each subset, one unique path is selected and optimized. The optimized paths are the input to the subsequent step planner.

Environment Modeling

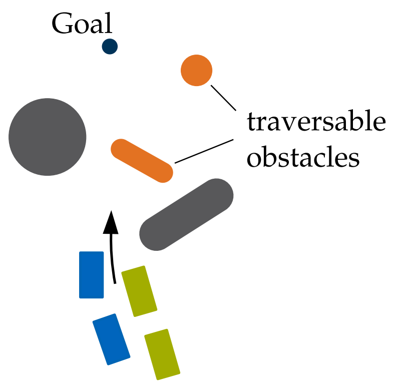
In order to preserve the robot's ability to traverse obstacles, only non-traversable obstacles are taken into account during 2D *Pre-Planning*. Since obstacles are clustered and approximated by combinations of SSV objects, this is efficiently implemented by checking for the obstacle's dimensions. In the subsequent detailed step planning, the full map is used in turn.

Robot Modeling

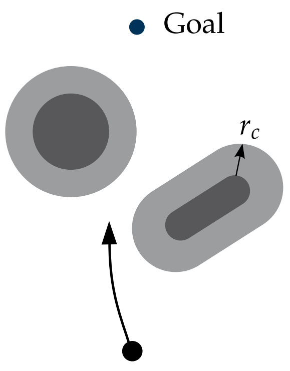
The robot's approximation is simplified for 2D *Pre-Planning*. The mobile platform planner searches for continuous paths. The robot is represented by a point model with an additional safety margin. Consequently, no rotations of the robot are considered. Therefore, the search space dimension is reduced by one DoF. Furthermore, collision checking and detection are significantly simplified, since the complex foot approximation is omitted.

Reduced Map – Limitations

Standard path planning algorithms for mobile platforms [30, 34] search for one continuous 2D path. This path is the optimal path out of a set of multiple solutions with respect to the cost definition used. The 2D *Pre-Planning* presented in this work uses a reduced



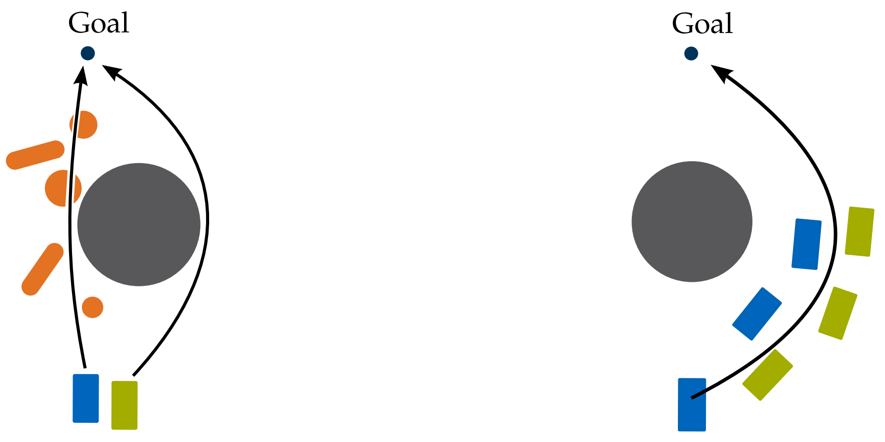
(a) Full map



(b) Reduced map

Figure 4.10: Reduction of the environment map.

environment map, since small obstacles are neglected (see Figure 4.10). This induces an uncertainty regarding the resultant costs of the subsequent step sequence. Therefore, the optimal path which is found by the mobile platform planner could turn out to be sub-optimal when planning the step sequence and using the full map (see Figure 4.11). In the presence of multiple small objects on the 2D–path, the optimal path could even not be traversable. For this reason, the proposed mobile platform planner has to be designed to find various path variants. Multiple 2D path options offer the opportunity to parallelize planning of several step sequences and to choose the best variant.

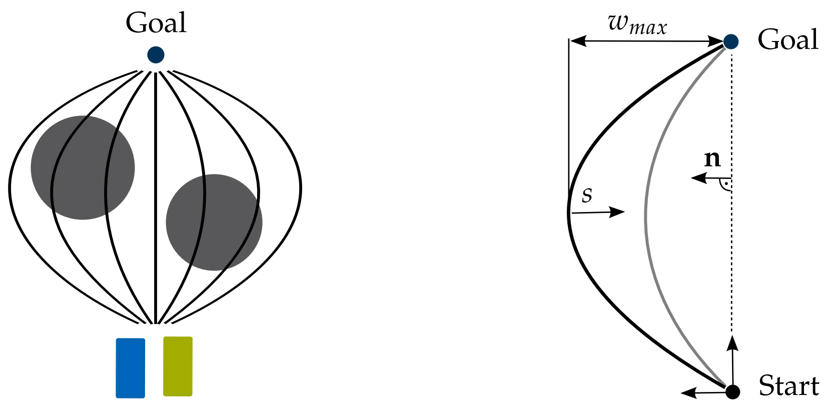


(a) Path options **(b)**Chosen path

Figure 4.11: Several continuous 2D–paths in presence of obstacles. Grey obstacle is non-traversable by bipedal robot. Orange obstacles are traversable.

Generation of Initial 2D Paths

The initial paths are generated by a set of n_{init} parabolas which connect start and goal position (see Figure 4.12). The parabolas are parameterized with the parameter s which defines the distance perpendicular to the direct connection between start and goal position. The direction is defined by the unit vector $\boldsymbol{n}$. The n_{init} parabolas and s are predefined by the user and determine the discretization of the area. Using parabolas at this stage is not paramount for the success of the procedure. The key point is that curves are used which cover the area in which the robot has to navigate with a set of initial and different solutions. Each parabola g is discretized with $n_{S,g}$ supporting points $\mathbf{x}_{s,g,i}$ lying on the initial parabolas. The supporting points are connected via linear splines.



(a) Initial set of 2D paths **(b)**Parametrization of initial paths

Figure 4.12: Discretization - Generating initial solutions.

Adaptation of Initial Paths

Up to this point, $\mathbf{x}_{s,g,i}$ are created based on the set of parabolas without knowledge of the obstacles. In the next step, $\mathbf{x}_{s,g,i}$ lying inside in obstacles are iteratively shifted to become collision-free. To generate collision-free paths, a potential approach widely applied in path planning [56] is used. This approach is presented for one exemplary path, but the method is used simultaneously for all paths generated from the initial set of parabolas. Each obstacle j is modeled by an artificial potential ϕ_j as

$$\phi_j(\mathbf{x}^k_{s,g,i}) = \ln\left(r_j(\mathbf{x}^k_{s,g,i})\right) \tag{4.12}$$

with the shortest distance $r_j(\mathbf{x}^k_{s,g,i})$ between an obstacle j and a given point $\mathbf{x}^k_{s,g,i}$ in iteration step k. Note: the same library for distance calculation as used in our methods for collision avoidance [98] is used. Superposition of source terms of all n_{obs} obstacles results in the potential equation

$$\phi(\mathbf{x}^k_{s,g,i}) = \sum_{j=1}^{n_{obs}} \ln\left(r_j(\mathbf{x}^k_{s,g,i})\right) \tag{4.13}$$

For shifting each supporting point i, the derivative $\nabla_{\mathbf{n}}\phi(\mathbf{x}^k_{s,g,i})$ is numerically calculated with respect to $\boldsymbol{n}$ using the perturbation ε:

$$\nabla_{\mathbf{n}}\phi(\mathbf{x}^k_{s,g,i}) = \frac{\phi(\mathbf{x}^k_{s,g,i} + \varepsilon \cdot \mathbf{n}) - \phi(\mathbf{x}^k_{s,g,i})}{\varepsilon} \tag{4.14}$$

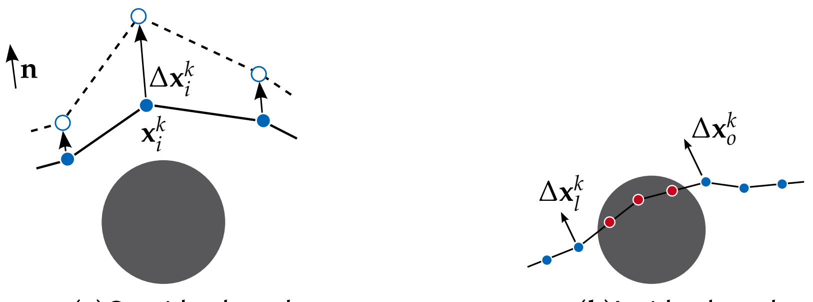
The translation increment is computed using a scaling parameter c_{shift}

$$\Delta\mathbf{x}^k_{s,g,i} = \mathbf{n} \cdot c_{shift} \cdot \nabla_{\mathbf{n}}\phi(\mathbf{x}^k_{s,g,i}). \tag{4.15}$$

$\mathbf{x}_{s,g,i}$ is updated as follows (see Figure 4.13 (a))

$$\mathbf{x}^{k+1}_{s,g,i} = \mathbf{x}^k_{s,g,i} + \Delta\mathbf{x}^k_{s,g,i}. \tag{4.16}$$

If a supporting point is located inside an obstacle, no potential is defined. In this case, the increment $\Delta\mathbf{x}^k_{s,g,i}$ is set to the maximal $\Delta\mathbf{x}_{s,g,j}$ in direction $\boldsymbol{n}$ from both neighboring supporting points l and o which are still collision-free (see Figure 4.13 (b)). The iterative



(a) Outside obstacles **(b)**Inside obstacles

Figure 4.13: Generating collision-free paths – updating of supporting points.

translation is terminated when either all paths are collision-free or a maximal number of iterations has been reached.

Selection of Unique Paths

In a subsequent step, the paths are optimized to obtain not only collision-free, but optimal paths with respect to the path length. To reduce computational effort, only collision-free paths which tend to converge to different final paths are considered. This happens when there is at least one obstacle between two paths (see Figure 4.11). First, each subset of collision-free paths are identified which are not separated by an obstacle. Then, the shortest path of each subset is selected for further optimization.

Path Optimization

In order to ensure smooth paths, the selected collision-free paths are optimized with respect to the path length. The *elastic band method* is used, which simulates a contracting force acting in an elastic band. It is described in Quinlan et al. [89].

Ordering - Cost Estimation

Costs for all available 2D paths are estimated in order to omit irrelevant path options. This is necessary to reduce computational cost with and without parallelization. Additionally to the 2D path length $l_{2D,s}$, path costs are mostly related to small obstacles which have to be traversed. Consequently, estimation of path costs is conducted as following: first, all small obstacles in a distance $d_{relevant}$ of the respective 2D path and the distance free space $d_{free,i}$ between two adjacent obstacles are identified. Then, the path cost estimates are calculated with

$$\hat{c}_{p,s} = l_{2D,s} + \sum_{i}^{n_{to}} \left(k_{base} + \begin{cases} a_1 \cdot \|2l_d - d_{free,i}\|, & \text{for } d_{free,i} < a_2 \\ 0, & \text{for } d_{free,i} > a_2 \end{cases} \right) \tag{4.17}$$

The first term k_{base} adds constant costs for each small obstacle. Its value amounts to the average additional step costs caused by traversing an obstacle. A second term weighted by constant factor a_1 is added, since step costs tend to increase when obstacles are located close to each other and therefore force the robot to deviate from the preferred step length l_d. Those additional costs are only added for $d_{free,i}$ below a certain value a_2, because for larger distances this effect vanishes. Furthermore, with small values of $d_{free,i}$ the probability increases that the region is not traversable and needs to be detoured. Based on the estimated costs, the paths are ordered relatively to their costs. In Table 4.2 and Table 4.3 the estimated path costs and the real costs for different test cases are compared. The real costs are accurately estimated by the cost estimation. Furthermore, the estimated costs are much more precise than just using the path length as estimation for ordering.

Alternative Methods

Furthermore, two different possibilities are considered to search for 2D paths. Generalized Voronoi-Diagrams divide the reduced map in separate regions. Each region belongs to one non-traversable obstacle. The border of the regions represent a network of edges which always have the maximal distance to the obstacles. Thus, graph-search algorithms can be used to search on this network for paths [6]. Advantages of the Voronoi-Diagrams are its compact environment representation and its network of separated collision-free edges. Since all paths are always separated by obstacles, it is not necessary to separate them in a consequent step as it is proposed to do in the presented method. However, the construction of generalized Voronoi-Diagrams is time-consuming and the resulting paths are not optimal. Thus, a consequent optimization step is still necessary. Furthermore, another considered option is to discretize the map and search on the grid for a path. The advantage over the step planner is the largely reduced search space. Nevertheless, the search has to be artificially limited to find more than one path option. Therefore, regions on the map have to be determined which have to be blocked to guide the planner to another path options. The implementation of such an algorithm turned out to be not suitable for this problem.

4.4.4 Coupling with Step Planner

The 2D paths are passed to the A*-based step planner as described in Section 4.3. Although, they can be combined with any node-based search algorithm for bipedal locomotion. The integration of the 2D path takes place in the A*-based step planner when a *key pose* is expanded and new step suggestions are generated. Overall, three different methods are proposed to use the 2D path for accelerating the search. These can be applied independently from each other as well as in combination:

Heuristic

In this approach, the Euclidean distance to the goal used in the cost evaluation is replaced by the distance along the 2D path. That way, the heuristic can better estimate the remaining path costs. This accelerates the A*–search significantly (see Chestnutt [18]). The heuristic costs c_h are computed by orthogonally projecting the position $\boldsymbol{x}_{sug}$ of a step suggestion onto the 2D path. Integrating the arc length from the projection $\boldsymbol{x}_{sug,\perp}$ to the goal yields the remaining path length l_g. To prevent the A*–search from deviating too much from the 2D path, the lateral distance from the path weighted by a factor $w_\perp$ is incorporated in the heuristic as well:

$$c_h = l_g + w_\perp \|\boldsymbol{x}_{sug} - \boldsymbol{x}_{sug,\perp}\|. \tag{4.18}$$

For computing the projection and the remaining length a linear spline representation is used for connecting the supporting points and approximating the 2D path.

Restriction of Search Area

It is assumed that the optimal step sequence is located within the immediate vicinity of the 2D path. Therefore, the search area S of the A*–search can be restricted to a maximal distance d_{max} from the 2D path. All step suggestions which lie outside the search area are omitted, since they are considered to lead to sub-optimal step sequences:

$$\|\mathbf{x}_{sug} - \mathbf{x}_{sug,\perp}\| < d_{max}. \tag{4.19}$$

Consequently, the number of investigated step suggestions will be reduced. Choosing a sufficiently large value d_{max} still allows the step planner to react to small obstacles which were ignored during 2*D Pre-Planning*.

Reduction of Search Space Dimension

Due to the tree structure of the A*–search graph, a reduction of the search space dimension tends to decrease the number of investigated step suggestions exponentially. As described in Section 4.3, the state of each node consists of the relative displacement $\boldsymbol{x}_{sug}$ and the orientation φ of each foot. By orienting the angle φ relative to the 2D path, φ is no longer considered in the search space S of the A*–search. That way, the search space is reduced by one dimension. This greatly accelerates the search. φ is computed as the tangent's angle φ_t of the projection $\boldsymbol{x}_{sug,\perp}$ on the 2D path of $\boldsymbol{x}_{sug}$.

4.4.5 Real-Time Implementation

The proposed methods are implemented for real-time applications. Like the step planner, the 2*D Pre-Planning* is executed before each new step. If no feasible step sequence can be found during the set time limit (approx. 400ms) and using the shortest 2D path, the error handling introduced in Subsection 4.3.6 becomes active. The step planner searches for a valid sequence to walk on the spot. Since multiple 2D paths after 2*D Pre-Planning* are obtained the step planner could ideally be parallelized on a multi-core processor in order to compare step sequences for different 2D paths. However, the calculations have not been parallelized yet, since it has not been necessary so far in the current implementation.

4.4.6 Results

The proposed methods are evaluated in simulation and validated the real-time character in experiments.[2]

Simulation

In the following, the step planner is compared with and without 2*D Pre-Planning* and the different methods presented in Subsection 4.4.4. The test cases presented here are chosen to discuss and illustrate different aspects of the methods. Longer step sequences than in real experiments are shown, which may result in exceptionally-long calculation times.

Table 4.2: Results for different methods: (none) Euclidean distance as heuristic, (1) 2D path heuristic, (2) restriction of search area, (3) reduced search space, (1 & 2) (1) & (2) combined, (1 & 3) (1) & (3) combined, (2 & 3) (2) & (3) combined, (1,2,3) all methods combined.

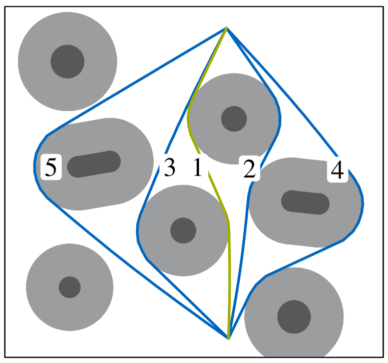
Coupling methods	none	1	2	3	1 & 2	1 & 3	2 & 3	1,2,3
goal reached in time	no	yes	no	yes	yes	no	yes	yes
2D–path length [m]	-	5.62	5.62	5.62	5.62	5.62	5.62	5.62
costs	-	5.990	-	7.151	5.990	6.900	6.841	6.900
estimated costs	5.927	5.927	5.927	5.927	5.927	5.927	5.927	5.927
number of steps	12	14	14	18	14	18	18	18
distance calculations	324428	403710	401942	131446	403710	13596	41230	13596
search time [s]	60.016	16.607	60.001	7.151	16.607	0.169	4.347	0.169

The first test case represents a simple environment similar to Figure 4.15. The results are summarized in Table 4.2. Using the method denomination mentioned in the legend of Table 4.2, the following can be stated: When using only (none) or (2) the path planning was not able to find a result in the set time limit. The results for the combination of (1) and (2) are nearly the same as for (1) alone, since the heuristic already guides the search close to the 2D path in this particular example. Results for (2) combined with (3) range in between those for (3) alone and all methods activated. Step planning with all methods combined yields the shortest search time even though the costs of the resulting step sequence are slightly higher. This is mainly due to the combination of (1) and (3) as confirmed by the result of this combination. All analyzed environments show similar results. For navigation in unknown environments, slightly raised costs are accepted in favor of a significantly reduced search time. Therefore, in the following, all methods are activated. The next test environment is depicted in Figure 4.14b. Even in this complex environment the 2*D Pre-Planning* was able to find multiple path variants (see Figure 4.14a). The final step sequence for the shortest path is depicted in Figure 4.14b. To investigate the benefit of finding multiple path variants, another test case with small obstacles is presented. The 2*D Pre-Planning* finds two possible path variants and corresponding step sequences as shown in Figure 4.15. The results are summarized in Table 4.3. When comparing the path costs the right solution yields higher costs even though its 2D path is shorter than the left one. This increase is caused by the presence of small obstacles, which result in more expensive steps.

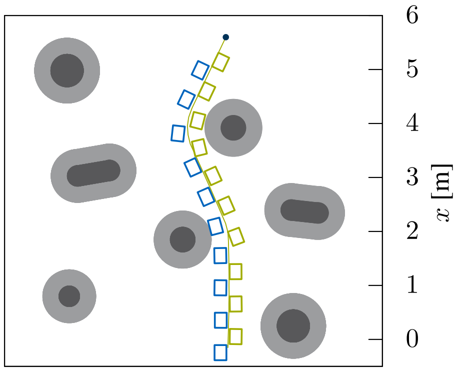
4.4.7 Discussion on the Optimality of the A*–Search

The presented methods influence the A*–Search and may jeopardize the optimality character of the A*–Search with respect to the predefined costs. The optimality of the 2*D Pre-Planning* depends strongly on the discretization level of the search space by the initial

[2]Videos of the results are available online at `https://youtu.be/-VvxzFg9ATU`.

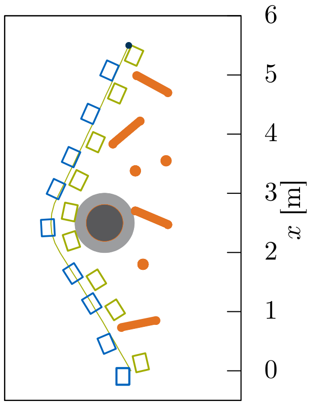


(a) Results of 2*D Pre-Planning*.

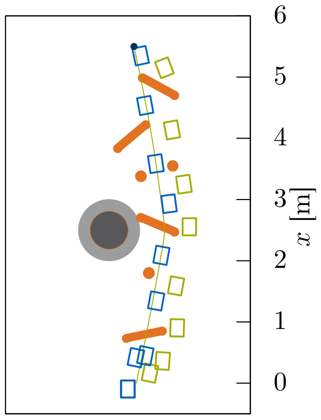


(b) Step sequence for shortest path.

Figure 4.14: Results of pre-planning and final step sequence for complex environment. Large obstacles in dark grey, safety zones (different for step planner and mobile platform planner) in light grey, shortest path in green, and other paths in blue.



(a) Longer 2D path (Solution 1)



(b) Shorter 2D path (Solution 2)

Figure 4.15: Step sequences for two different 2D–paths.

Table 4.3: Results of a test case with two alternative solutions for step sequences (see Figure 4.15).

	Solution 1 (left)	Solution 2 (right)
length of 2D–path [m]	6.15	5.59
costs	6.42	7.29
estimated costs	6.26	6.33
number of steps	17	16
distance calculations	2220	130725
search time step–planner[s]	0.04	14.2
search time 2D planner[s]	0.004	0.004

set of parabolas. A trade-off between calculation time and coverage of the search space has to be made. The *2D Pre-Planning* presents a possibility to find multiple solutions, which is important in the context of this work for the real-time application. Furthermore, it is important for considering the presence of traversable obstacles as shown in Subsection 4.4.6.

However, a different mobile platform planner may find a better 2D path, which could be combined with the methods presented in Subsection 4.4.4 as well. If it is assumed that the *2D Pre-Planning* finds all possible 2D paths, including the shortest path, in the reduced map: (1) Using the shortest 2D path as initial estimate will always underestimate the distance of the path subsequently optimized by the step planner and will therefore not influence the optimality of the result of the A*–search. Further, the shortest path approximates the remaining path costs better than a Euclidean distance. Therefore, it accelerates the search, but will always underestimate the remaining costs. (2) Limiting the search area of the A*–search and reducing the analyzed step suggestions influences the optimality of the A*–search. It will still find an optimal solution, but only in the reduced search area. Since, the search space is limited, it is important that the *2D Pre-Planning* determines more than one solution. In the presence of multiple obstacles not considered in the reduced map, a 2D path may not be walkable. Searches following different 2D paths significantly increase the probability to find an executable step sequence. (3) Calculating φ based on the 2D paths reduces the dimension of the search space by one. Therefore, the quality of the calculated step sequence will always be inferior to the solution of the search in a higher dimensional search space.

In conclusion, only the integration of the 2D path as a heuristic in the A*–search preserves the solution's optimality. For real-time application, the advantage of significantly faster calculation times is gained in the case of sub-optimal solutions when limiting the search space or reducing the dimension of the search space. With the application of legged robots in interaction with human users in mind, the method's evaluation has another facet. A robot trying to follow a 2D path instead of executing an internally calculated and optimal step sequence may help a user to predict the robot's behavior and interact with it. Applying the methods presented in Subsection 4.4.4, the search for an optimal step sequence could not only be improved, but a user has another option to guide a robot, for example by a 2D path as input.

4.5 Adaptive Discretization

In the previous sections, an A*–based step planner was presented. A new method was introduced which accelerate significantly calculation times by searching for a heuristic. It guides the A*–Search taking into account non-traversable obstacles. In addition to the heuristic, the capacities of the A*–Search depends largely on the action set. In the previous section, a static action model was used.

Chestnutt et al. [19] showed how an adaptive action model can be used to find footstep plans through cluttered environments without increasing the size of the action set. They propose to alternate step suggestions around bad terrain locations to find valid positions. In the following, a local adaptation method is presented, which makes the discrete A*–search character "more continuous". It is based on the ideas of Chestnutt et al. [19], but exploits the characteristics of the presented collision world representation.

4.5.1 Local Adaptation

The key of the presented method is the world representation. The terrain is modeled as obstacles or walkable surfaces. The collision check for each step suggestion checks not

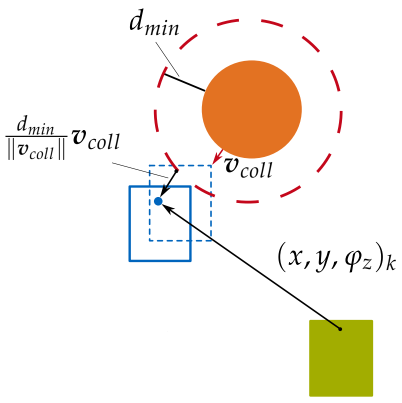


Figure 4.16: Local adaptation of step suggestion with invalid position.

only the node suggestion's viability but calculates a collision gradient, v_{coll}, as well. The collision gradient is used to optimize the node suggestion locally in the direction the node suggestion would be valid. That way, the action model is adjusted to the terrain. The modified position results in

$$\begin{bmatrix} \Delta x_{mod} \\ \Delta y_{mod} \end{bmatrix} = \frac{d_{min}}{\|v_{coll}\|} v_{coll} + \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \tag{4.20}$$

with d_{min} the minimal required distance between obstacle and stance foot. The big advantage of this procedure is, that the A*–Search uses a gradient information to find valid node suggestions and does not have to alternate the step positions without further informations. In order to avoid multiple node suggestions next to each other, the modifications are limited to half of the minimal distance between two step suggestions of the originial action set. In the case of more than one active v_{coll}, it is proposed to iteratively modify the position. In the worst case, the local adaptation does not find a valid position in the limited number of iterations and the analyzed node suggestion is invalid.

Results

Figure 4.17 depicts the resulting step sequence for the simple and complex environment introduced in Subsection 4.3.7. The step planner uses the *local adaptation* to adjust invalid node suggestions. In Table 4.4 the corresponding analysis of the search is presented. Results of different discretization levels are compared. The step planner with activated *local adaptation* outperforms the step planner without *local adaptation* in terms of costs for both environments. An explanation is the adjustment of invalid nodes to valid nodes. That way, the search tree becomes larger and the step planner is able to find a better sequence out of a larger pool of possibilities. The search time resp. the performed distance calculations of the step planner with *local adaptation* is for the simple environment significantly longer than for the conventional step planner. Chestnutt [18] observed a similar tendency. In environments with few obstacles the step planner is able to find a way to the goal also without adjusting invalid node suggestions. Therefore, adjustment of invalid node suggestions provoke an augmented number of distance calculations and consequently a longer planning time. In the complex environment, the step planner with *local adaptation* outperforms the step planner without *local adaptation*. In this complex environment, adjustment of invalid node suggestions is necessary to find a goal path in a reasonable amount of time. So far, the benefit of the *local adaptation* is questionable. Although, it performs better in terms of cost in the analyzed environments, it needs significantly longer planning times in simple environments, because of the augmented number of valid node suggestions. The local adaption can reduce the planning time only

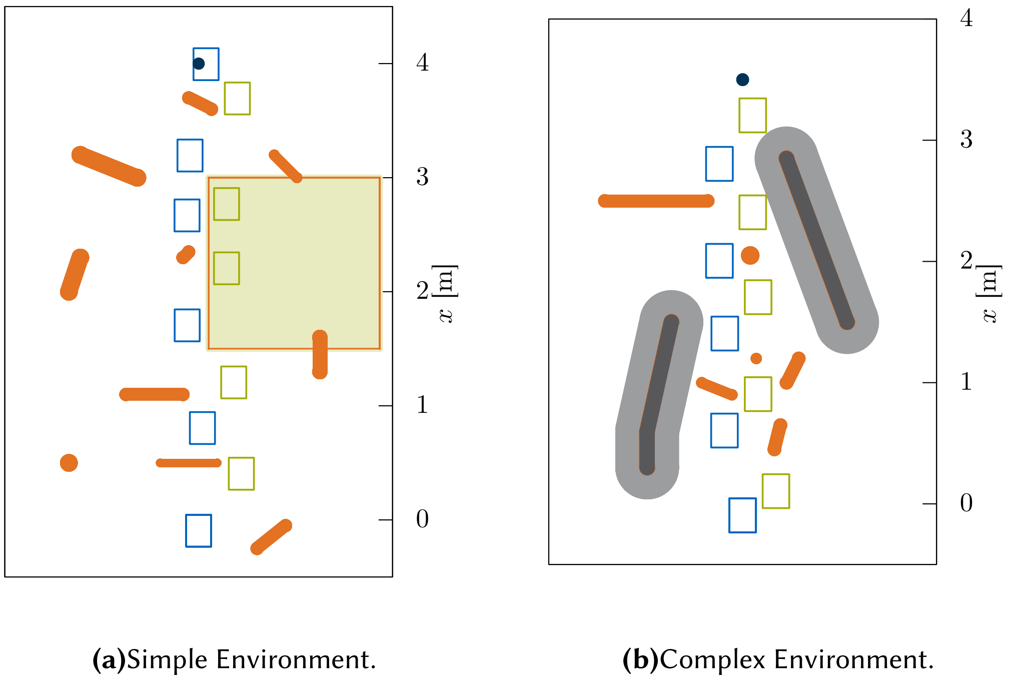


Figure 4.17: Result of step planner with local adaptation in simple and complex environment (compare Figure 4.5). Search analysis is presented in Table 4.4. Simple environment includes surface represented with light green. Traversable obstacle in orange. Simple environment includes non-traversable obstacles represented in (dark) grey. Security margin (light) grey. Goal location in dark blue.

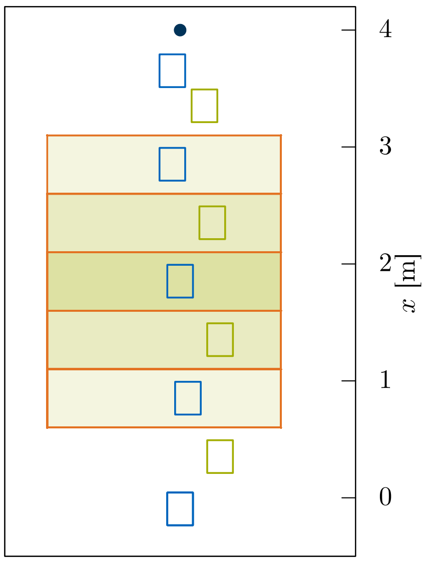


Figure 4.18: Result of step planner with local adaptation for stair testcase. Search analysis is presented in Table 4.4. Stairs include five platforms with different height level. Step planner without local adaptation does not find a goal path.

Table 4.4: Results for simple and complex environment (see Figure 4.5) and stair testcase (see Figure 4.18). Results produced with activated local adaption. step planner without local adaption was not able to find goal in a limit of 50s with the reduced number of node suggestions. See Table 4.1 for results with unactivated local adaption.

Environment	simple	simple	complex	complex	stairs
number of node suggestions	85	34	85	34	85
number of steps	10	11	9	13	8
distance calculations	129412	5305	2079	14641	1632
costs	4.8191	5.53	3.85	6.28	4.24
search time step–planner[s]	16.43	0.1	0.02	0.17	0.07

for complex environments. Reducing the discretization level and consequently the number of node suggestions makes the advantage of the adaptation clearly visible. Whereas the step planner without *local adaptation* is not able to find valid goal paths, the step planner with *local adaptation* finds sequences for both environments. Because of the reduced number of node suggestions the cost of the final sequence is higher for both environments. Nevertheless, the planning time for the simple environment is largely reduced and it remains similar for the complex environment. Through analyses of multiple scenarios, a similar tendency has always been observed. The *local adaptation* helps to reduce the dependency of the search's success on the chosen discritization level. Although it performs worse in environments with only a few obstacles in terms of planning time, it also finds valid paths in cluttered environments within the set time limits. This is not true for the conventional step planner. An example is depicted in Figure 4.18. The environment consist of a stair with five platforms with different heights. The step planner without *local adaptation* is not able to find a goal path because of the limited step suggestions. Furthermore, a step planner with the *local adaptation* allows for reduced discretization levels. This does not improve the solution's quality but further decreases planning time.

4.6 Summary

In this chapter, the module for high-level planning, the *Navigation*-module of the robot, is presented. It allows for reactive path planning in previously unknown and dynamically changing environments and users' input.

The navigation problem of bipedal locomotion is transferred to a discrete optimization problem. An A*–based step planner is applied to solve the problem. Instead of using the foothold areas for feasibility checks, a 3D dynamic approximation of the feet and the lower legs is applied. Thus, it is able to plan not only feasible foot-steps but also to compute initial swing-foot trajectories. For further acceleration of calculation times two extensions are presented:

(1) a novel method for using mobile platform planner for bipedal locomotion is introduced. Multiple approaches are discussed and analyzed to beneficially couple the mobile platform planner and the discrete optimization. (2) Furthermore, methods are introduced to soften the discretization used in the discrete optimization and to exploit the continuous range of values the footholds are able to reach.

Chapter 5

Real-Time Motion Generation

In the previous chapter, navigation methods for bipedal locomotion in unknown and dynamic scenario are presented. For real-time applications, the long planning sequences of multiple physical steps require an approximation of the robot's motions. For this reason, the robot's motions are approximated via sequences of discrete *key poses* connected with *actions*. This approximation allows for a condense motion representation with parameter sets and, thus, a real-time motion planning.

In this chapter, methods are presented to generate the whole-body motion of bipedal robots based on sequences of parameter sets. They are inspired by the methods presented in Chapter 2 and they are integrated in the overall framework for versatile and robust walking as presented in Chapter 3. This chapter focuses on the modules *Kinematic Optimization* and *Collision Avoidance* (see Figure 3.3). In the overall concept of the motion generation approach, this chapter details the trajectory optimization and reactive adaptions (see Figure 5.1).

In Section 5.1, an overview of current related research is given. Section 5.2 presents the motion generation of LOLA and summarizes the current limitations. After that, approaches are discussed which extend the current motion generation: in Section 5.3 model-predictive methods are presented which optimize the robot's motion of the next physical step. Based on the compact task space representation with parameter sets, not only the redundancy can be exploited but the robot's motion can be optimized in task space as well. Section 5.4 focuses on the task space representation. It introduces a new CoG trajectory representation which allows for more versatile motions. Using a simplified model, the CoG trajectory can be optimized for avoiding future kinematic limits in real-time while taking the constraints of dynamical feasible walking into account. Disturbance rejection and non-static environments requires adaptions of ideal planned motions. Therefore, reactive methods for collision avoidance are presented in Section 5.5.

This section describes work published in Hildebrandt et al. [118, 122, 124].

5.1 Literature

Most current humanoid control frameworks follow a similar approach as the one presented in this thesis. They hierarchically divide the motion generation problem. Typically, the navigation problem is reduced to a search of foothold sequences to take into account geometric constraints. The navigation systems have the provision of foothold positions in common, which can be summed up as a set of parameters. This set and, additionally, user chosen parameters or parameters explicitly calculated by the step planner [42, 80], as for example torso height or footstep height, roughly describe the robot's desired motion. It separates the handling of geometric constraints imposed by the environment from the motion generation.

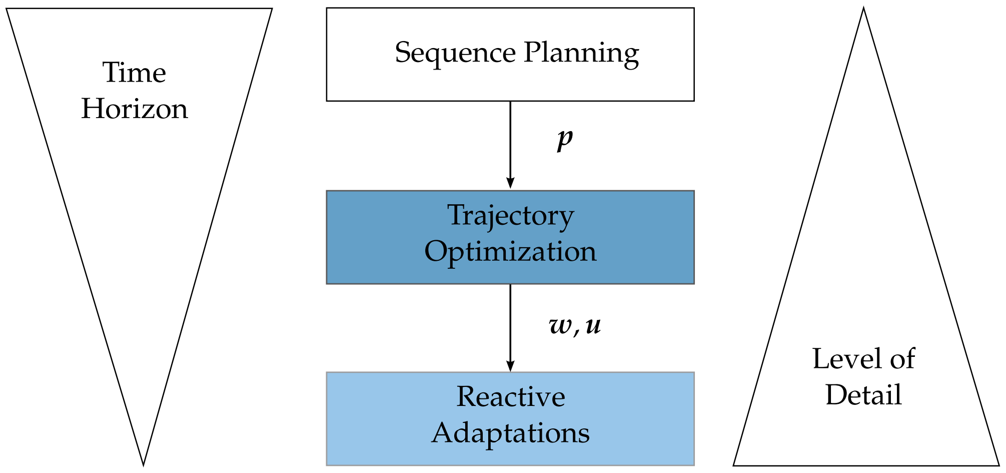


Figure 5.1: Hierarchical Approach for Motion Generation.

The robot's multi-body-behavior is approximated by simple point-mass models. Using these models and the parameter sets, reference trajectories allowing for dynamically feasible walking are calculated without long latencies. The reference trajectories are set points to generate the motion on joint level. Buschmann [12], Englsberger et al. [28], Kajita et al. [52], Nishiwaki et al. [80], and Takenaka et al. [105], among others, are prominent proponents of this approach for real-time motion generation.

By adapting the desired motion according to sensor feedback, Urata et al. [108] and Wittmann et al. [132], among others, showed impressive results in rejecting unknown disturbances even in the presence of obstacles [123].

All the above mentioned authors apply different methods for motion generation and different models to approximate the robot's dynamic behavior. However, they all have the generation of reference trajectories governing the robot's motion based on simplified models and heuristically chosen parameters in common.

An approach for optimization of walking pattern parameters was presented by Buschmann et al. [11]. Due to its offline character, it does not take the environment into account.

The environment was taken into account by Nishiwaki et al. [80]. They proposed planning collision-free foot trajectories in the task space connecting the desired footholds. However, the foot trajectories are calculated taking the kinematics of the robot only via heuristics into account.

5.1.1 Bipedal Locomotion

Most of the previous approaches take the robot's full kinematics only locally into account. Thus, a large optimization space of the robot's redundancy is neglected. Furthermore, the robot's ability to step over or onto obstacles can only be exploited by using heuristic approaches to respect kinematic limitations.

Work on kinematic planning for humanoid robots considering a whole step movement focuses mainly on narrow tasks such as stepping over [3, 38, 101, 109, 115] or onto obstacles. Guan et al. [38] and Yisheng Guan et al. [115] investigate the feasibility of humanoid stepping-over-motions. They propose a quasi-static trajectory planner for the task of stepping over a rectangular obstacle.

Arbulu et al. [3], Stasse et al. [101], and Verrelst et al. [109] additionally take the Zero Moment Point (ZMP) feasibility criteria via the preview control [52] into account. That way, they are able to shift the result for stepping over an obstacle from quasi-static to dynamic robot motions. Stasse et al. [101] and Verrelst et al. [109] extend the step planning process. First, they calculate required step length and waist height to obtain a collision-free double support phase. Second, smooth swing foot trajectories are generated based on collision checks in key configurations. Furthermore, they propose to adapt the horizontal foot trajectory on-line during step execution. Nevertheless, to the best of the author's knowledge, this has not been validated experimentally yet.

As opposed to Stasse et al. [101] and Verrelst et al. [109], Arbulu et al. [3] uses more sophisticated body approximations for collision checks. Instead of line segments the obstacle and the lower part of the swing leg are modeled as boxes. Arbulu et al. [3] propose, similar to Stasse et al. [101] and Verrelst et al. [109], that collisions between robot and obstacle be checked only for several key configurations. Smooth swing foot trajectories are generated using clamped splines by interpolating the key configurations. The robot's full motion is generated based on the methods presented in Kajita et al. [52]. The desired motions are checked for feasibility by calculating the inverse dynamics of the multi-body model.

The presented methods allow humanoid robots to step over one large obstacle at a time. However, for the work's purposes they have the following disadvantages:

- They all use very simplified geometric models. That way, complex 3D parts of the robot or the environment cannot be adequately represented.
- They only consider collisions between the lower legs and one obstacle. Neither potential self-collisions, nor collisions with several obstacles at the same time are taken into account.
- They limit the foot movements in a plane. More general movements which exploit all the robot's DoFs are not considered.
- The methods are presented as stand-alone: they are not integrated in frameworks involving perception and navigation. Thus, their compatibility with a whole motion-generation framework has yet to be proven.
- They concentrate on the stepping-over motion over one obstacle. The robot's whole kinematic is not optimized and it is not considered in more general walking scenarios.

The method presented by Koch et al. [58] is a more general approach. They generate the stepping-over-motion from a whole-body-motion-optimization. The method is applied to the test case of a stepping-over motion over one obstacle. However, this is an off-line method and not yet applicable to real-time applications.

5.1.2 Redundant Robots

Looking at the stepping motion of humanoid robots in cluttered environments from a point of view of whole-body-motion optimization seems important. Whole-body-motion optimization allows for a more complete and more general exploitation of the robot's capabilities without searching for solutions for specific tasks.

In recent years, many frameworks for motion planning were developed [23, 95, 116]. In their objective to develop a motion planning framework for any kind of robotic system lies the limitations for our purposes. They neither exploit nor take the characteristics of humanoid walking into account. This makes it difficult to satisfy the hard constraints of dynamic walking. Motion planning for humanoid robots has to satisfy hard real-time constraints, it has to take the dynamics of bipedal walking into account and it has to react to unknown perturbations.

Schulman et al. [95] and Zucker et al. [116], among others, present powerful gradient-based frameworks: Zucker et al. [116] introduces the algorithm *CHOMP*. It minimizes a cost function by covariant gradient descent taking into account the path smoothness and penalizing collisions. Distance gradients are calculated based on pre-computed distance fields to accelerate the computations.

Schulman et al. [95] name their presented algorithm *TRAJOPT*. It is an approach based on sequential convex optimization, which takes the system's boundary conditions as cost functions into account. Both algorithms have been applied to legged locomotion: Schulman et al. [95] present results for planning foot placements while maintaining static stability under environmental constraints for the humanoid *ATLAS*. Zucker et al. [116] apply *CHOMP* on the four-legged robot *LittleDog* walking on uneven terrain.

Gienger et al. [36] follow an approach similar to the one presented in this work. Trajectories are presented using linear attractor dynamics with control points. The control points are optimized by a gradient-based optimization algorithm. Depending on the number of control points and the search space, the computation time can be adapted on the problem. Since the methods were developed for grasping motions, they do no respect the hard timing and geometric constraints of humanoid locomotion, e.g. foot-ground contact. This is critical in order to maintain balance.

5.2 Motion Planning

This section summarizes parts of Chapter 3 and gives a detailed view on the motion generation. As described in Chapter 4, the *Navigation*-module calculates a sequence s_t of n_S *key poses* s_i

$$s_t = (s_0, s_1, ..., s_{n_s}) \tag{5.1}$$

with the corresponding actions a_i

$$a_t = (a_0, a_1, ..., a_{n_s}) \tag{5.2}$$

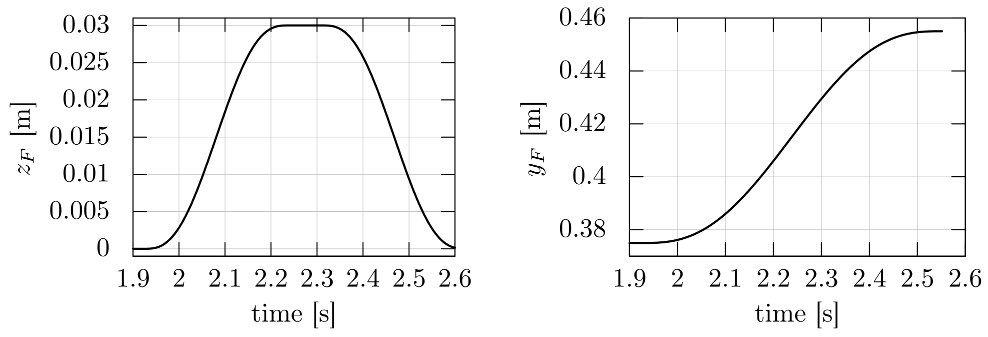
before each step. As defined in (4.2) an *action* a_i contains the walking parameters $(\Delta x, \Delta y, \Delta\theta_z, \boldsymbol{h}_{obst}, h_{Step}, c_a)_i$. Combined with user-chosen parameters, the input of the motion generation is generalized to a sequence of parameter sets $\boldsymbol{p}_{wp,i}$:

$$\boldsymbol{p}_{wp,t} = (\boldsymbol{p}_{wp,0}, \boldsymbol{p}_{wp,1}, ..., \boldsymbol{p}_{wp,n_s}). \tag{5.3}$$

These parameter sets determine the overall motion of the robot. Based on $\boldsymbol{p}_{wp,i}$ the *Ideal Walking Pattern* is generated. It includes the CoP reference trajectory resp. the desired contact force and moment trajectories $\boldsymbol{\lambda}_{id}(t) = \left[\boldsymbol{F}_{id}^T, \boldsymbol{T}_{id}^T\right]^T(t)$ and the ideal task-space trajectories $\boldsymbol{w}_{id}(t) \in \mathbb{R}^m$ which are composed of

- the CoG position,
- the torso orientation,
- the toe angles,
- the foot positions,
- the foot rotations,
- and the pan and tilt angles for positioning of the camera.

They are time continuous trajectories based on spline representation. All but the lateral CoG trajectories are fully described by $\boldsymbol{p}_{wp,i}$ and the imposed $\mathcal{C}^2$-continuity condition. In this work, special focus is set on the sagittal CoG trajectory and the foot trajectories.



(a) Sagittal. **(b)**Frontal.

Figure 5.2: Sagittal and frontal foot trajectories. Both are represented via piecewise polynomials fifth order. Vertical foot trajectory is parameterized with one free parameter denoting the maximum height (dz_{step}).

They are represented by quintic splines connecting start and end position of each step successively. The splines are fully defined by setting velocity and acceleration at the set points to zero. The foot trajectories are connecting foothold positions and are, therefore, defined by the step parameter $\boldsymbol{p}_{Step}$ as described in Chapter 4. The sagittal foot trajectory has an additional set point defining the height dz_{Step} of the swing foot movement. This is influenced heuristically by $\boldsymbol{h}_{obst}$. Figure 5.2 depicts the trajectories. The generation of the CoG trajectories is explained in the following.

5.2.1 CoG Trajectory Generation

The lateral and frontal CoG trajectories are generated to respect dynamic constraints and to allow for dynamically feasible bipedal locomotion. In contrast to most of the approaches in current research, the robot's multi-body system is approximated by a three-mass model instead of an inverse pendulum for real-time application. That way, dynamic effects caused by fast leg movements can be taken into account. Wittmann [114] gives a detailed overview of robot models, their characteristics and their application in bipedal locomotion.

The model is depicted in Figure 5.3. It has one mass m_b representing the upper body and two masses m_{fi} ($i = 1, 2$) approximating the legs dynamic characteristics. The input to the lateral and frontal CoG trajectory generation are the trajectories defining the motion of the three masses: foot trajectories $\boldsymbol{r}_{fi}(\boldsymbol{p}_{wp}) = [x_{fi}(\boldsymbol{p}_{wp}), y_{fi}(\boldsymbol{p}_{wp}), z_{fi}(\boldsymbol{p}_{wp})]$[1], the desired torso height trajectory $z_b = z_b(\boldsymbol{p}_{wp})$ describing the movement of the upper body mass m_b, and the desired contact moments trajectories $\boldsymbol{T}_{id} = \left[T_x, T_y, T_z\right]$. The torso height $z_b = z_b(\boldsymbol{p}_{wp}) = z_b(H_{CoG})$ is a fifth order polynomial with a configurable height H_{CoG} at the end of each step.

Thus, the equation of motion results in a linear and time variant differential equation for the lateral and frontal upper body mass trajectories x_b and y_b. The equation for the frontal direction can be stated as

$$\begin{aligned} m_b z_b \ddot{y}_b - m_b y_b(\ddot{z}_b + g) = &-T_x \\ &+ m_f y_{f1}(\ddot{z}_{f1} + g) - m_f z_{f1}\ddot{y}_{f1} \\ &+ m_f y_{f2}(\ddot{z}_{f2} + g) - m_f z_{f2}\ddot{y}_{f2} \end{aligned} \tag{5.4}$$

The equation for the lateral upper body mass movement can be derived analogously. The method based on spline collocation proposed in Buschmann et al. [14] is used to solve a

[1]For the sake of simplicity, the explicit dependency of $\boldsymbol{p}_{wp}$ is ommited in the following notation.

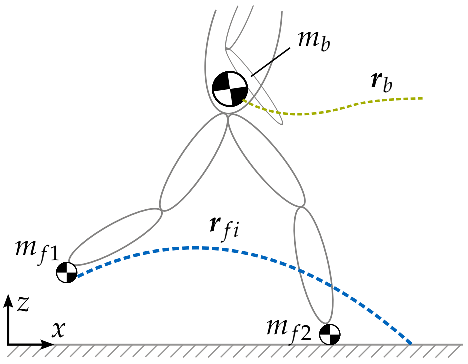


Figure 5.3: Three-mass model used for CoM trajectory calculation.

BVP for x_b and y_b over two steps. The CoG trajectories x_{CoG} and y_{CoG} can be computed by superposition based on the motions of the three masses m_b, m_{f1} and m_{f2}. It is important, that the real CoG is included in $\boldsymbol{w}_{id}$ and therefore tracked in the *Feedback Control* and not the trajectories $r_b(t)$ describing the upper body movement. That way, the chance for reaching knee singularities, as reported in Nishiwaki et al. [81], can be reduced, since the nullspace can be used to track the ideal CoG motion.

5.2.2 Feedback Control & Inverse Kinematics

In contrast to the *Global Control*, the *Feedback Control* is executed each control cycle of $\Delta t =$ 1ms. It does not take the robot's motion over a time horizon of multiple Δt into account, but adapts the desired contact forces $\boldsymbol{\lambda}_{id,k} = \boldsymbol{\lambda}_{id}(t_k)$ and the ideal work-space trajectories $\boldsymbol{w}_{id,k} = \boldsymbol{w}_{id}(t_k)$ at each control instant t_k to stabilize the robot according to sensor feedback. The target joint data $\dot{\boldsymbol{q}}_d \in \mathbb{R}^n$ are calculated based on the adapted work-space trajectories $\boldsymbol{w}_k$ (including drift compensation) with the Jacobian $J_w = \partial \mathbf{w} / \partial \mathbf{q} \in \mathbb{R}^{mxn}$ as

$$\dot{\boldsymbol{q}}_k = \boldsymbol{J}_w^{\#} \dot{\boldsymbol{w}}_k - (\boldsymbol{E} - \boldsymbol{J}_w^{\#} \boldsymbol{J}_w) \boldsymbol{u}_k \tag{5.5}$$

$$\boldsymbol{u}_k = \nabla_{\boldsymbol{q}} L_y. \tag{5.6}$$

They are executed by the robot. Here, E represents the identity matrix and

$$J_w^{\#} = W^{-1} J_w^T (J_w W^{-1} J_w^T) \tag{5.7}$$

represents the weighted pseudoinverse with a user-chosen diagonal weighting matrix W. The vector $\boldsymbol{u}$ is a gradient to an optimization criterion L_y which is designed as follows.

Cost Function

The cost function design is explained in more detail in Hildebrandt et al. [124] and Schwienbacher et al. [98]. It is a weighted sum of costs which can be summarized as

$$\begin{aligned} L_y = & c_{jl} L_{jl} + c_{coll} L_{coll} + \\ & c_{cmf} L_{cmf} + c_{vel} L_{vel} + c_{ang} L_{ang} \end{aligned} \tag{5.8}$$

The costs $c_{jl} L_{jl}$ and $c_{coll} L_{coll}$ are devoted to penalize violations of constraints as joint-limits resp. self-collision and collisions with the environment. $c_{cmf} L_{cmf}$ is a cost to penalize the deviation of the robot's motion from a desired comfort pose. While $c_{jl} L_{jl}$ contributes only to L_y, when joint angles reach a joint-limit, $c_{cmf} L_{cmf}$ keeps the joint angles close to desired positions. $c_{vel} L_{vel}$ is devoted to reduce angular velocities. The weights c_i are chosen taking dimensions of the costs into account to ensure that the constraints are met. The kinematic chain of humanoid robots represents a particularity compared to conventional

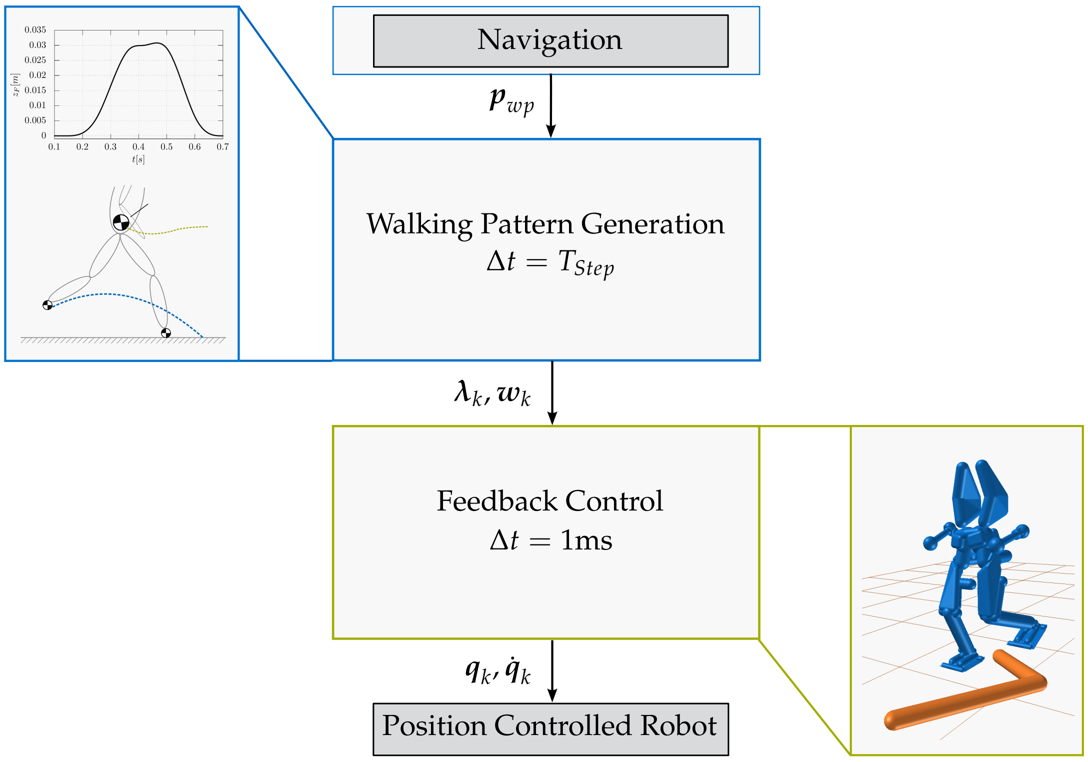


Figure 5.4: Models used for LOLA's motion generation.

manipulators: its arms are open kinematic chains without defined task space motions of their end effectors. Schwienbacher et al. [98] presented a method to use the arms DoFs to compensate for vertical angular momentum. $c_{ang}L_{ang}$ represents the corresponding cost.

5.2.3 Limitations

In a nutshell, ideal motions are generated by hierarchically dividing the search problem: first, work-space trajectories are generated to enable dynamically feasible motions based on a simplified model over a long time horizon. The system's motion is not taken into account. The work-space trajectories are represented as splines described with few parameters which further reduces the number of free variables. Second, the robot's inverse kinematic is solved locally while exploiting the redundancy. LOLA's motion framework with the corresponding robot models is depicted in Figure 5.4. This hierarchical procedure allows for real-time application, but implies several limitations:

- Since $\boldsymbol{u}$ is only projected into null-space, it doesn't affect the reference trajectories given in task-space. Consequently, ill-chosen $\boldsymbol{w}_{id}$ can cause violations of kinematic constraints or collisions.
- The robot's kinematics are not taken into account in a predictive way. The motion-governing task-space trajectories are calculated without feedback about the resulting kinematic motion. Therefore, high-safety margins are necessary to generate feasible work-space trajectories. This limits artificially the robot's motion. In the worst case, this leads to task-space trajectories which are kinematically not executable.
- The current work-space trajectory representation by quintic spline is only configurable by few parameters. A higher number of parameters would increase its versatility.

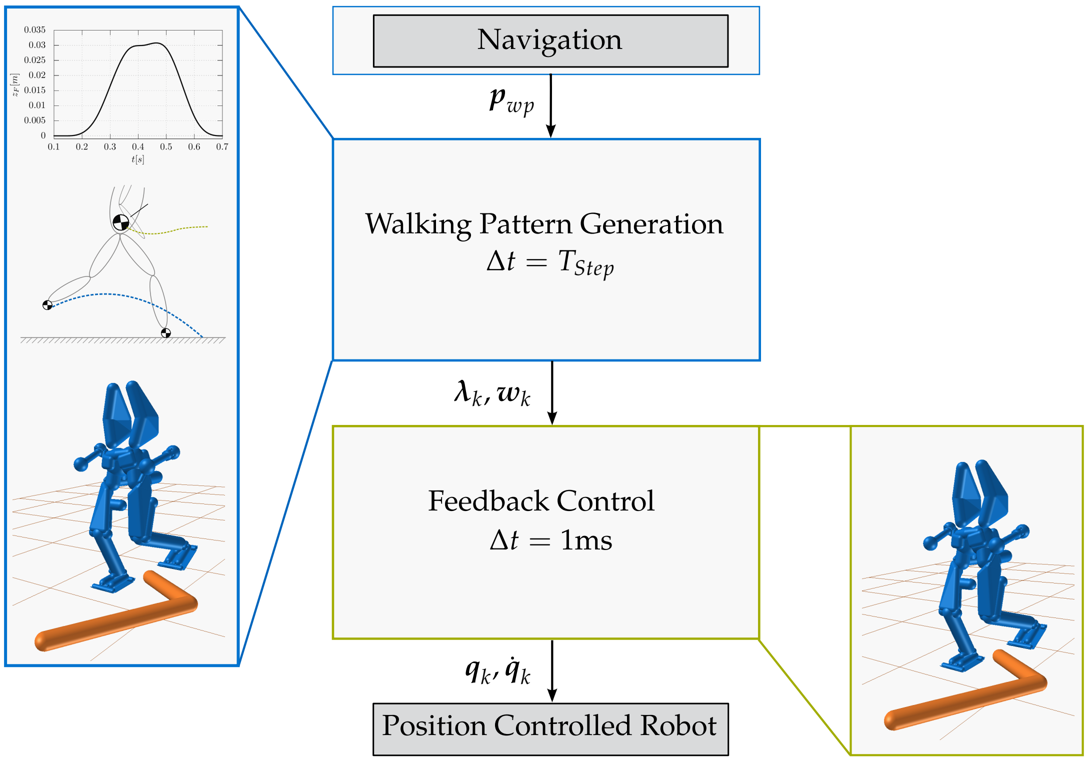


Figure 5.5: Extended motion generation for *Model-Predictive Kinematic Planning*. The future motions of the whole robot are taken already in the *Ideal Walking Pattern*-module into account.

- The null-space of the robot is only exploited locally without knowledge about the future motion. As shown in Chapter 2, this neglect a large optimization potential.

In the following, methods are proposed to reduce these limitations and to exploit the kinematic capabilities of humanoid robots.

5.3 Model-Predictive Kinematic Planning

The method and results presented in this section were developed in collaboration with Tobias Scheuermann, Tobias Blume, Manuel Demmeler and Simon Schwerd [136, 151, 152] and are published in Hildebrandt et al. [118, 122]. This section presents a model-predictive approach to optimize the robot's future motions.

The limitations described above are true for most of the current frameworks for real-time control of bipedal walking (see Section 5.1).

For this reason, it is proposed to extend the *Global Control* by a model-predictive kinematic evaluation and optimization. The underlying methods are based on the methods presented in Chapter 2. The approach is based on LOLA's kinematic model. Instead of only generating the *Ideal Walking Pattern*, the robot's next physical step is simulated using the kinematic model as well as the environment approximation. Figure 5.5 shows the extended *Global Control*. In the following, methods are presented to evaluate the motion's kinematics and to optimize them.

5.3.1 Model

The model-predictive approach uses the robot's kinematic model as depicted in Figure 5.5. It takes into account the *Feedback Control* without external influences or sensor feedback, but including the *Collision Avoidance*. The equations describing the robot's kinematic movement (5.5) can be summarized as a first order differential equation of the form

$$\begin{pmatrix} \dot{\boldsymbol{q}} \\ \dot{\boldsymbol{w}} \end{pmatrix} = f(\boldsymbol{q}, \boldsymbol{w}, \boldsymbol{u}, \boldsymbol{w}_{id}, \dot{\boldsymbol{w}}_{id}). \tag{5.9}$$

Due to real-time constraints, a time horizon of one physical step of the robot is used for time integration. The method could be extended to take multiple steps just by integrating over a longer time horizon into account. The model's initial conditions are determined by the models's state at the end of the previous step.

Contrary to the model described in Chapter 2, here, the work-space trajectories are not only describing desired geometric motions but they are also used to meet the dynamic constraints of bipedal locomotion. The trajectories describing the horizontal CoG movements are determined by the foot and torso height trajectories using the method presented in Buschmann et al. [14] and cannot be changed without influencing the dynamic feasibility of the bipedal walking.

5.3.2 Optimization

The different methods introduced in Section 2.2 on the example of a 5-DOF manipulator can be applied directly to the prediction model. It represents a redundant robot as explained in Section 3.1. The system equation depends on task space trajectories $\boldsymbol{w}_{id}$, which are fully described by a set of parameters $\boldsymbol{p}_{wp}$ and which are calculated before each step. The redundancy of the system is exploited to minimize a cost function. The input vector $\boldsymbol{u}$ to the nullspace motion is a continous input to the system. Currently, $\boldsymbol{u}$ is calculated each control cycle step as the gradient to L_y (see (2.6)). Nevertheless, it can also be used to represent a result of a global optimization taking a longer time horizon than one control cycle step as for example one physical robot step into account (see Subsection 2.2.2 or Subsection 2.2.2).

5.3.3 Initial Solution - Kinematic Evaluation

The *Model-Predictive Kinematic Planning* depends on an initial kinematic movement which is executable. The parameter $\boldsymbol{p}_{wp}$ is set approximating the full kinematic movement heuristically by the *Navigation*-module or by the user. For this reason, initially chosen $\boldsymbol{p}_{wp,init}$ may lead to kinematically non-feasible movements. In Hildebrandt et al. [118] the *Kinematic Evaluation* is introduced. By integrating the kinematic model parametrized by $\boldsymbol{p}_{wp}$, which would lead to kinematically non-feasible movements, can be identified. Additionally, it is proposed to establish an advanced error handling. Using another set of initial parameters including different footholds, the robot's movement is analyzed again and can be corrected. One obtains

$$\boldsymbol{p}_{wp,k+1} = \text{re-init}(\boldsymbol{p}_{wp}) \tag{5.10}$$

with the re-initialisation of $\boldsymbol{p}_{wp}$. The re-initialisation of $\boldsymbol{p}_{wp}$ is chosen based on the projection of the gradient of the local collision and joint limits avoidance on the task-space as described in Chapter 3. Therefore, it is an indicator of which task space trajectory, and thus, which parameter, is limiting the movement. In this context, the challenge is to establish an interaction of the planning modules, step planning and trajectory planning which works reliably and fast enough to meet the real-time requirements.[2] Up to now, the initial solutions $\boldsymbol{p}_{wp,init}$ are evaluated sequentially. Since, the motion of the model only depends on the robot's initial state, parallelization of the model integration to evaluate different $\boldsymbol{p}_{wp,init}$ at the same time could be beneficial. Different initial solutions could be generated

[2]The whole planning process has to be done in less than T_{Step}.

with an approach similar to the one presented in Section 4.4 using an approximation of the robot's motion kinematics.

5.3.4 Parameter Set

The robot's kinematic motion is governed by $\boldsymbol{w}_{id}$ parametrized by $\boldsymbol{p}_{wp}$ as explained in Subsection 5.2.2. For real-time application, it is focused on the subset $\boldsymbol{p}_{opt}$ of $\boldsymbol{p}_{wp}$. The subset $\boldsymbol{p}_{opt}$ consists of the parameter H_{CoG} and the set of supporting points $\boldsymbol{p}_{SF}$ describing the height of the CoG resp. the swing-foot's sagital and lateral movements. It is assumed that they have the biggest influence on the overall motion.

Trajectory Design - Foot Trajectories

The task space representation via quintic splines limits the configuration possibilities with $\boldsymbol{p}_{SF}$. The frontal trajectory is represented with one quintic polynomial connecting start and end position of the swing foot movement and is therefore not configurable. The sagittal trajectory is only configurable with a parameter denoting the height of the swing foot movement. Introducing additional supporting points in the trajectory representation with quintic polynomial requires not only parameters defining the position, but also the velocity and acceleration at this point. Since reasonable values on velocity and acceleration level are missing and collision avoidance only depends on the robot's configuration in task space on position level, adding additional supporting points in the trajectory representation via quintic polynomial seems not reasonable. Furthermore, quintic polynomials cause undesired oscillations, which may provoke undesired motions.

Therefore, it is decided to use a trajectory representation $x(t)$ based on cubic polynomials $c_k, k = 1, ..., N$:

$$x(t) = \begin{cases} c_1(t),\ if\ t_0 \leq t < t_1 \\ c_2(t),\ if\ t_1 \leq t < t_2 \\ \vdots \\ c_N(t),\ if\ t_{N-1} \leq t < t_N \end{cases} \tag{5.11}$$

with

$$c_k(t) = a_k(t - t_k)^3 + b_k(t - t_k)^2 + c_k(t - t_k) + d_k. \tag{5.12}$$

a_k, b_k, c_k and d_k are the spline parameters. Cubic splines have the advantage of reducing oscilations, which are more likely with higher dimensional polynomials. For stable humanoid walking the boundary conditions

$$\begin{aligned} x(t_0) &= s_0 & x(t_N) &= s_N \\ \dot{x}(t_0) &= 0 & \dot{x}(t_N) &= 0 \\ \ddot{x}(t_0) &= 0 & \ddot{x}(t_N) &= 0 \end{aligned}$$

and the continuity on acceleration level of the trajectories are critical. To sum up, these requirements result in $3 \times N$ constraints and $x(t)$ is described by $(N - 1) \times 4$ free parameters. Consequently, it is possible to impose for $N - 4$ supporting points $n_{F,opt}$ additional position constraints and design them according to the desired walking pattern. In the current implementation the timing of the supporting points is fixed. It is choosen heuristically based on simulations and experiments. Figure 5.6 shows an example of a trajectory of the sagital foot displacement with $n_{F,opt} = 3$ ($N = 7$) before and after an optimization. The parameter $\boldsymbol{p}_{opt}$ are chosen to consist of $n_{F,opt}$ parameters for the vertical and the horizontal

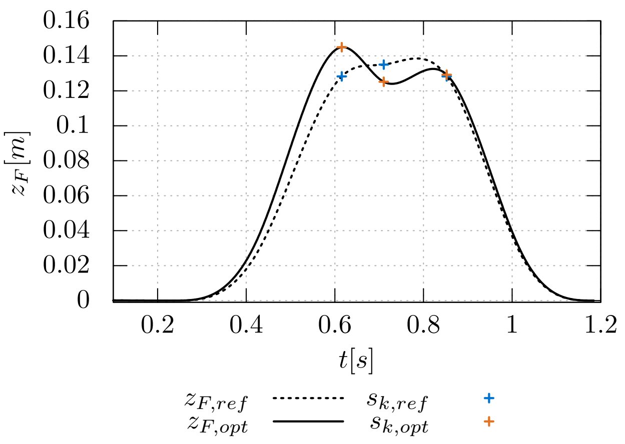


Figure 5.6: Vertical foot trajectory of step 8 for LOLA stepping over an obstacle before and after optimization. The parameters describing the initial foot trajectory are chosen based on a heuristic.

swing-foot displacement and the height of the CoM H_{CoG}. Consequently, the number of optimization parameters is $n_{opt} = 2n_{F,opt} + 1$ in total. This choice of the optimization parameters has proven to be detailed enough to improve the performance of the robot in our experiments. However, the method is open to choose additional parameters or different trajectory representations.[3]

5.3.5 Step-Time Adaption

The high level input of the control system may be a desired path, a joystick command or a desired step-parameter set, which can all be represented by a desired velocity v_{des} for each step. The desired walking command is modified by the A*-search to make navigation in cluttered environments possible. In order to guide the robot as close as possible to the desired velocity v_{des}, T_{step} is adapted according to the output of the A*-search, the relative displacement of the next foothold Δl, and v_{des} as follows:

$$T_{Step} = \begin{cases} T_{Step,min},\ if\ T_{Step,min} > \frac{\Delta l}{|v_{des}|} \\ \frac{\Delta l}{|v_{des}|},\ if\ T_{Step,min} < \frac{\Delta l}{|v_{des}|} < T_{Step,max} \\ T_{Step,max},\ if\ T_{Step,max} < \frac{\Delta l}{|v_{des}|} \end{cases} \tag{5.13}$$

That way, the time dependence can be excluded from the optimization. $T_{Step,min}$, $T_{Step,max}$ are manually set according to the robot's dynamics.

5.3.6 Gradients

A main difficulty in applying the methods presented in Chapter 2 to the problem of biped locomotion is the computation of the derivatives of the Hamilton function. In general, the gradients could be computed by deriving an analytically expression of the gradients which is solved in each time step or, alternatively, applying a finite-difference scheme. As a third option, the *Efficient Gradient Computation* was presented by Toussaint et al. [106], which is particulary suitable to parameter-dependent derivatives. In this approach, the gradients are re-formulated by the chain rule and then computed by forward integration from a known initial value.

[3]Further details to the trajectory design are presented in Blume [136].

In this case, gradients with respect to the nullspace control input $\boldsymbol{u}(t)$ can be derived analytically, as shown in Schuetz et al. [94]. Furthermore, a finite-difference scheme for a continuous trajectory would fail in real-time application due to its high computational expense. The formulae are presented in Appendix A.

Further, derivatives with respect to step parameters $\boldsymbol{p}_{wp}$ can be computed either by application of a finite-difference scheme or the *Efficient Gradient Computation*. The latter comes with very low computational expense since a limited set of parameters is used which are constant over the time of the motion. The gradients are derived in Appendix A.

5.3.7 Integration

Figure 5.8 gives an overview of the integration of the *Model-Predictive Kinematic Planning* in the *Global Control*. The output of the *Model-Predictive Kinematic Planning* is dependent

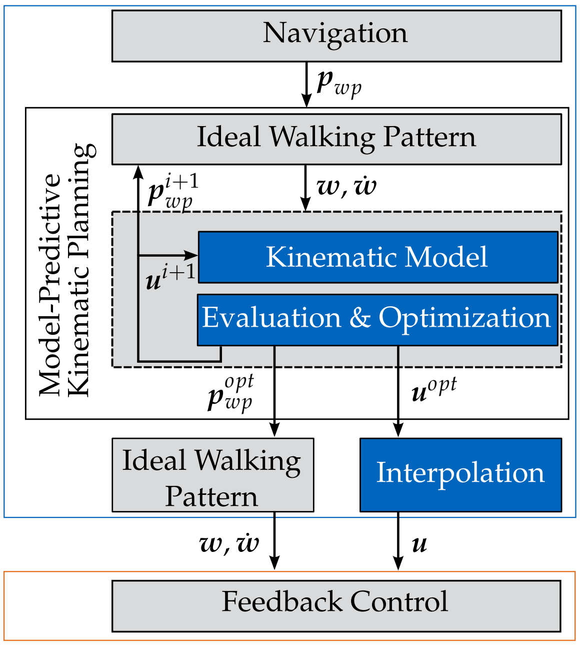


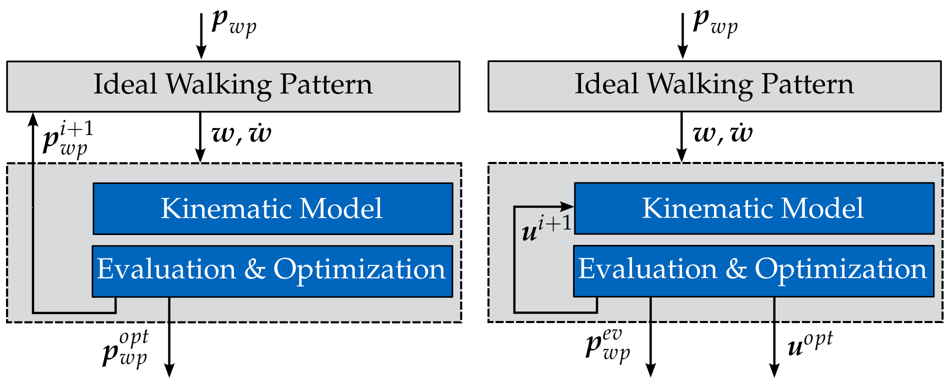
Figure 5.7: Schematic of predictive planning for combined parameter and nullspace optimization.

on the method applied - either an optimized parameter set $\boldsymbol{p}_{wp}^{opt}$ or an optimized continous input vector $\boldsymbol{u}^{opt}$ or both. Figure 5.8 shows the integration of parameter and nullspace optimization separately. Since the nullspace optimization separately does not influence the task space trajectories, $\boldsymbol{p}_{wp}$ still has to be evaluated for its feasibility. Applying the kinematic evaluation as presented in Subsection 5.3.3 an evaluated parameter set $\boldsymbol{p}_{wp}^{ev}$ is used in the following.

The $\boldsymbol{p}_{wp}^{opt}$ resp. $\boldsymbol{p}_{wp}^{ev}$ is directly used to calculate the desired walking pattern. This is the input as $\boldsymbol{w}, \dot{\boldsymbol{w}}$ to the *Feedback Control*. The optimized nullspace input $\boldsymbol{u}^{opt}$ is applied as a feed-forward term to (5.5). Since the model used in the *Model-Predictive Kinematic Planning* and the model used in *Feedback Control* do not perfectly correspond, a drift compensation is added. It is similar to the drift compensation used in the inverse kinematics for the end effector position to account for numeric drift. The resultant input vector is calculated as

$$\boldsymbol{u} = \boldsymbol{u}^{opt} + K_u(\boldsymbol{q}_M - \boldsymbol{q}_R) \tag{5.14}$$

with the vector $\boldsymbol{q}_M$ and $\boldsymbol{q}_R$ representing the joint position of the model resp. the *Feedback Control* and the constant matrix K_u, which is positive-definite. This control architecture is called *Model-Predictive Planning* in contrast to *Model-Predictive Feedback Control*. The basic idea of the hierarchical control framework is that the desired motion is planned in advance. Underlying control layers modify the desired motion to account for disturbances. Since each control layer takes the previous one into account and improves its result, the whole system becomes very robust.



(a) Parameter optimization. **(b)**Nullspace optimization

Figure 5.8: Schematic of predictive planning for parameter and nullspace optimization separately. Nullspace optimization is coupled with evaluation of p_{wp}.

Implementation Details

The multi-process and multi-thread software architecture of LOLA's real-time walking control system is presented in Chapter 3. Each stepping motion is analyzed and optimized before it is executed. In consequence, the planning time for the whole planning process of step k, including the calculation of the footholds and the planning of the desired walking pattern, is limited to the step time of the previous step $T_{Step,k-1}$ in the real-time application (see Chapter 3). $T_{Step,k-1}$ varies between 0.5...1.2s, since it is user dependent. Additionally, the step planner adapts T_{Step} according to the desired velocity of the robot and the environment. To handle the varying hard time constraints, an advanced time management is introduced. Based on $T_{Step,k-1}$ a maximal number of iterations is calculated before each step. It takes the time which is needed for one integration of the model depending on the current number of obstacles and depending on the necessity of the calculation of gradients into account. The calculation time for one integration with and without gradient calculation is summarized in Table 5.1.

Table 5.1: Summary of maximal computation time for integration of kinematic model over one physically step ($T_{Step} = 1.2$s). Comparison of gradient computation for the different methods and to integration of kinematic model without gradient computation. Runtimes are obtained from the real-time QNX computer.

method	max. [µs]
without gradient calculations	75
Numeric Gradient	75 n_{param}
Gradient of Nullspace	500

Since only a limited number of iterations is possible, in most cases it is sufficient to calculate once the descent of the cost function and apply a line search or an interval nesting method. That way, the time-consuming integration including gradient calculation has to be done only once.

The real-time application of the algorithm depends heavily on the possibility of aborting the optimization process at any given time. In the worst case, the optimization process is not able to converge to a solution superior to the initial one. In this case, only the kinematic feasibility is checked.

Furthermore, different integration step sizes are analyzed. The analysis showed that an integration step size which is six times higher than the control cycle time still seems to be an adequate trade-off between accuracy of the kinematic movement of the robot and speed of the integration. Adaptable step lengths depending on the collision gradients showed bad results, which corresponds to Betts [9].

A larger integration step size results in a coarse input vector $\boldsymbol{u}$. To be applicable to the *Feedback Control* it is interpolated between the discrete set points $\boldsymbol{u}_k$ at time step t_k using third order polynomials. That way, the input $\boldsymbol{u}$ to the *Feedback Control* is no longer optimal, but the methods become applicable to the real system. This is represented as *Interpolation* in Figure 5.7.

5.3.8 Results - Simulation

The proposed methods are analyzed in simulation and experiments with the robot LOLA. A video showing the simulations and the experiments can be found at `https://youtu.`

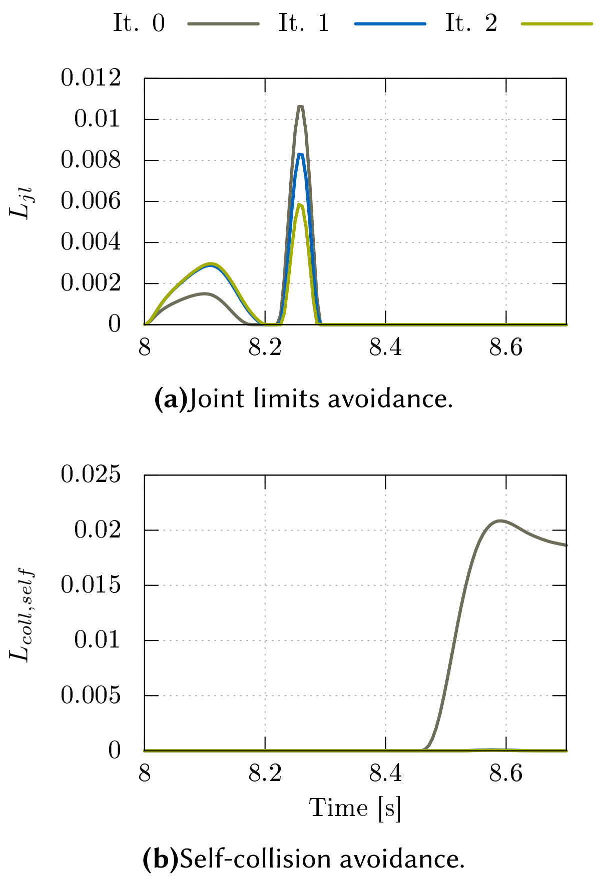


Figure 5.9: Optimization of joint limit avoidance & self-collision avoidance costs. The costs of joint limit avoidance and of self-collision avoidance for three iterations of the optimization algorithm are shown.

`be/twfDcsQvNBY`. The multi-body simulation used to verifiy the implemented methods includes unilateral and compliant contacts, motor dynamics as well as the joint control loops. For details see Buschmann [12]. Two different test cases are chosen to analyze the

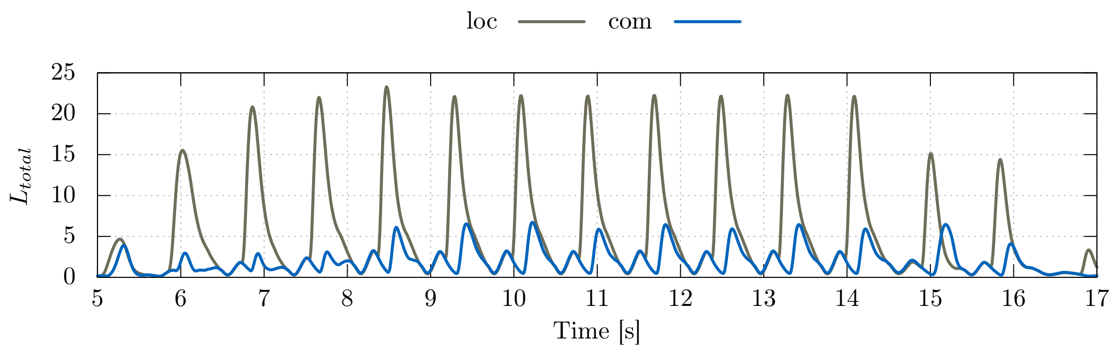


Figure 5.10: Simulation results for step sequence. Total costs for local (loc) and combined (com) optimization.

methods in this work - walking a step sequence and stepping over a complex obstacle.

Step Sequence

In the first test case LOLA executes a simple straight-forward step sequence in an obstacle-free environment. The step lengths are $L_x = 0.3...0.4$m. This test case is analyzed in experiments as well (compare Section 6.1). The resulting total costs of the combined optimization, the simultaneous optimization of $\boldsymbol{u}$ and the parameter $\boldsymbol{p}_{opt}$, and of the local method are depicted in Figure 5.10. A strong cost reduction is observed. Since the walking movement is a periodic and symmetric motion, results are very similar in each step. Figure 5.9 shows the costs L_{jl} and $L_{self,coll}$ over the iterations of the gradient method. The optimization is converged after two iterations for both, L_{jl} and $L_{self,coll}$. Since possible collisions may lead to a failure of the whole system, collision costs are strongly represented in the total costs. For this reason the reduction in the total costs stems mainly from the reduction of $L_{self,coll}$ which depends on the distance between arms and hip of the robot. Nevertheless, the other cost functions show moderate reductions as well.

Stepping Over a Complex Obstacle

In the second test case a scenario from Hildebrandt et al. [118] is adapted. It includes stepping over a complex shaped obstacle as depicted in Figure 5.12 in the collision world representation. This test case forces the robot to perform a kinematically complex stepping-over motion. It has been shown in Hildebrandt et al. [118] that a *model-predictive kinematic parameter evaluation & optimization* significantly improves the chance of success for the biped motion planning. Similar to the first test case, the combined optimization reduces the costs of $L_{self,coll}$ through nullspace optimization. Figure 5.13 shows the cost reduction during the three relevant steps of the overstepping motion over the iterations. Almost full cost reduction can be achieved with the first iteration of the gradient method, which includes backtracking line-search. Table 5.2 shows the resulting parameters. The modifications show, that the heuristic of the *Navigation*-module provides accurate initial values, but not optimal ones. In such complex scenarios, the predictive approach is superior to the heuristic approach.

5.4 Center of Mass Trajectory Planning

The method and results presented in this section were developed in collaboration with Konstantin Ritt and they are published in Hildebrandt et al. [120].

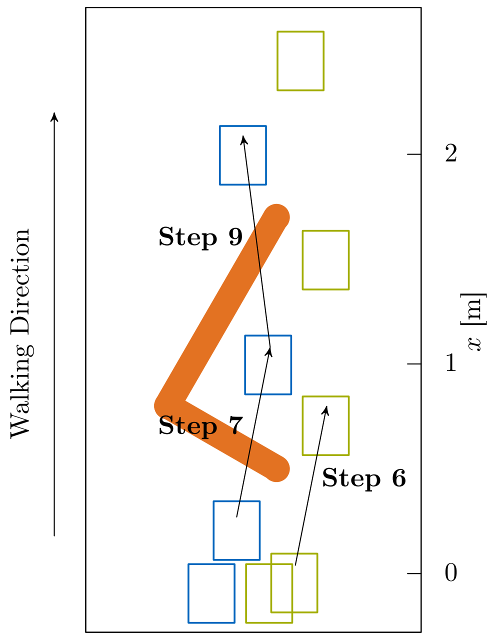
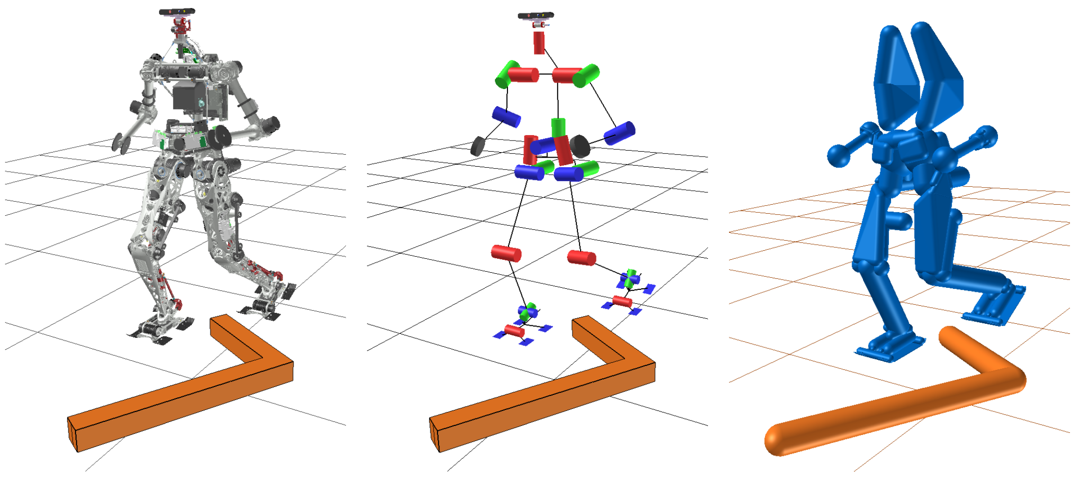


Figure 5.11: Step sequence for simulation test case with complex obstacle.



(a) Full Model. **(b)**Kinematic Model. **(c)**Collision Model.

Figure 5.12: Representation of complex scenario in full dynamic simulation. LOLA's full model, *Lola*'s kinematic model and the collision model, while stepping-over complex shaped obstacle (orange). Step corresponds to step 7.

Table 5.2: Optimized parameters for stepping-over complex shaped obstacle

	Initial		Optimized	
Step	H_{cog}	dz_{step}	H_{cog}	dz_{step}
6	0.880m	0.135m	0.879m	0.134m
7	0.880m	0.135m	0.883m	0.136m
9	0.880m	0.135m	0.882m	0.138m

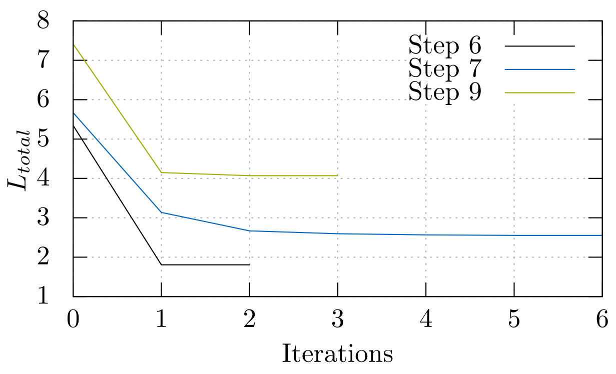


Figure 5.13: Cost over iterations for combined optimization. Gradient method includes backtracking line-search.

The capability to step up and down platforms or stairs makes legged locomotion an alternative to wheeled locomotion. Motion generation for stepping up and down is directly coupled with the open question of how the height of the robot's CoG has to be designed. A variable CoG height can result in several advantages for bipedal walking. Human walking extensively utilizes the advantage of a variable CoG height. Imitating a human-like walking may improve the energy efficiency [41].

Adapting the CoG to the current walking situation and terrain yields a greater kinematic versatility: larger strides are possible [64] and the maneuverability on stairs and on uneven terrain is improved [81]. Reaching kinematic constraints such as joint limits can be avoided. Furthermore, recent publications have even showed a self-stabilization influence applying a well designed CoG trajectory [22, 59].

Literature Review

Over the last decades, a large variety of models and methods have been presented to derive dynamically feasible CoG trajectories ([13, 52, 81, 104], among others). Often the robot is represented by an inverted pendulum where the height is considered as constant. Variable CoG height in motion generation is taken into account by Mayr et al. [71], Nishiwaki [79], and Takenaka et al. [104], among others.

However, in humanoid robotics, vertical CoG trajectories are often generated in a heuristic manner and not analyzed with respect to their influence on the overall motion. Chevallereau et al. [22], Li et al. [64], and Shafii et al. [99], for example, propose to use a cosine oscillation around an average height. The choice is motivated by observation of humans. In the implementations, the wavelength corresponds to the step length and the cosine is a function of the horizontal CoG position. Both works analyze the effects of vertical oscillations in simulation. Additionally, Shafii et al. [99] apply learning methods to optimize the function's parameter.

To allow climbing of stairs, Park et al. [86] generate the CoG height trajectory as a 6th order polynomial. They define boundary conditions and design parameters that are tuned in simulation to reduce the Zero Moment Point (ZMP) error. This yields an almost linear function, which the authors used for simplicity. Drawbacks of this simplification are not examined.

Hong et al. [44] deploy varying trajectories in dual and single stance. While a constant CoG height is assumed in single stance, cubic splines are used for the double support phase. This interpolation allows a transition to different CoG heights. The authors do not provide a rationale to motivate their choice but they demonstrate in simulations the ability to walk stably in uneven terrain.

Miura et al. [73] always set the robot's waist height as high as possible. When the legs are fully stretched, the waist height is lowered. The resultant trajectory suffers from discontinuities at the transitions between double and single stance phase. The authors therefore smooth the trajectory in an optimization with respect to a cost function constraining joint angles and velocities. The objective of the publication was to accurately imitate human-like motion, which was demonstrated on even terrain.

Nishiwaki [79] propose a similar approach. Keeping the torso height as high as possible while taking the solvability of the inverse kinematics and joint limits into account. First, they calculate the maximum feasible torso height based on the maximum joint velocities and maximum kinematically reachable torso height. Second, the torso trajectory is designed iteratively, taking into account future height limitations and vertical velocity and acceleration limits. In Nishiwaki et al. [81], the approach was modified using cubic splines which are defined by three heuristically defined support points per step. These points are chosen in such a way that the torso height stays close to its maximum, avoiding knee singularities.

Proposed Method

The *Model-Predictive Kinematic Planning* as presented in Section 5.3 directly influences z_b by determining an optimal support point value H_{CoG} at the end of the next step. z_b is generated as a quintic polynomial interpolation between the current height and the optimized support point. The parameterization of the torso height with only one parameter per step limits the possibilities to influence the overall motion. This results in violation of kinematic constraints in challenging scenarios:

joint limits define a maximum or minimum feasible torso height, which in turn determine indirectly the CoG height. Or, expressed the other way around, a wrongly chosen torso height or CoG height trajectory may result in joint limit violations. The minimum feasible torso height is commonly reached when the ankle joint cannot be further flexed. The maximum feasible torso height is mostly determined by the knee joint limits.

A typical scenario in which the current trajectory design reaches its limits is stepping up and down stairs or platforms[4]. Extending the sagital trajectory design of the CoG similar to the extended foot trajectory design as presented in Subsection 5.3.4 for the method presented in Section 5.3 seems not applicable for two reasons:

- In contrast to stepping over obstacles, motion kinematics when stepping up and down are assumed to be mainly influenced by the sagittal CoG trajectory. Stepping up and down is a complex motion, which is not completed after one step, but strongly depends on the previous and the future steps. These assumptions have been confirmed by the results presented in the following. Therefore, extending the sagital CoG trajectory by additional supporting points may improve the motion kinematics, but integrating the robot's motion over multiple steps as presented in Section 5.3 is not possible in the limited calculation time. For complex motions, as stepping up and down, the time horizon of one step seems to be too short.

- As discussed in Subsection 5.3.7, the gradient computation already reaches the timing limits. Additional parameters in the methods as described in Section 5.3 would significantly increase the run-time, and, therefore, real-time constraints would be violated.

The approach presented in this section extends the *Model-Predictive Kinematic Planning* taking the discussed limitations into account. The approach differs from the ones presented

[4]A video showing LOLA stepping up and down a platform with the trajectory design as presented in Section 5.2 is available at `https://youtu.be/ayj95PVvq0M`.

in Section 5.4 and extends the *Model-Predictive Kinematic Planning* taking the discussed limitations into account.

The collocation method presented in Buschmann et al. [14] is used to include an arbitrarily shaped trajectory for the CoG height. The method explicitly takes a predefined arbitrary CoG height trajectory in the calculation of dynamically feasible CoG motions into account. A new parametrization for the CoG height based on cubic spline representation is presented (see Subsection 5.4.1). It can be configured by set points. With this approach, the set of parameters used in the *Model-Predictive Kinematic Planning* is extended. Due to real-time constraints, the large number of free parameters is not manageable. Therefore, a simplified model approximating the kinematics is introduced in Subsection 5.4.2. This model is used to separately solve a parameter optimization for the CoG height trajectory's set points with respect to an objective function, taking kinematic limitations and joint velocities into account. The optimized torso height trajectory is used in the *Model-Predictive Kinematic Planning* to determine the remaining parameters describing the robot's motion as presented in Section 5.3. This is a trade-off between optimality of the solution and calculation time for real-time application. Figure 5.14 depicts the integration of the torso height optimization in the overall framework.

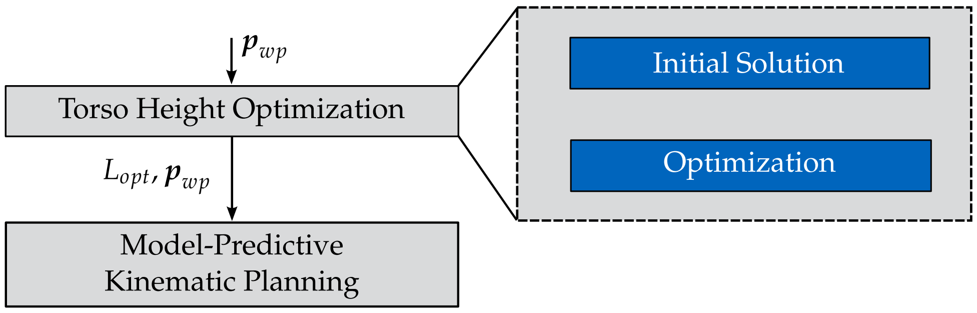

$\boldsymbol{p}_{wp}$
Torso Height Optimization
$L_{opt}, \boldsymbol{p}_{wp}$
Model-Predictive Kinematic Planning
Initial Solution
Optimization

Figure 5.14: Integration of CoG height optimization in control framework.

5.4.1 Trajectory Design - Torso Height Trajectory

In the methods presented so far, the torso height trajectory is composed of quintic polynomials with two support points at the start and end of each step period (see Figure 5.16). This representation provides C^2-smooth trajectories, which is important in order to avoid undesired discontinuities on acceleration level. A step period describes the interval between two consecutive double support phases. As explained above, additional support points per step period would allow for more variable trajectories. The drawback of the current representation with quintic polynomials is that the first and second derivative have to be set at each support point. Consequently, each support point introduces three new degrees of freedom (DoFs). Instead, the torso height trajectory is chosen to be represented using cubic splines. Each additional support point introduces one degree of freedom to the curve representation, while keeping the spline property of C^2-smoothness. The new torso height trajectory is represented by four support points per step period with the heights $z_i, i = 0..3$. Initial conditions reduce the DoFs to three parameters per step. In Subsection 5.4.3, the choice for the support points is discussed in more detail.

5.4.2 Simplified Kinematic Model

During the optimization, the underlying model is evaluated frequently. To maintain real-time capability, a low complexity of the model used is desirable. The robot's kinematics are approximated with a simple kinematic 2D model. Thus, the 2D kinematic chain of each leg is fully determined with the trajectories $\mathbf{r}_b = [x_b, y_b, z_b]$, $\mathbf{r}_{fi}$, φ, and the heuristically defined toe trajectory φ_{toe}. Figure 5.15 shows a 2D sketch of one leg depicting the variables. The trajectories of $\mathbf{r}_{fi}$, φ, and φ_{toe} are defined by $\boldsymbol{p}_{wp}$. The horizontal trajectory of the body

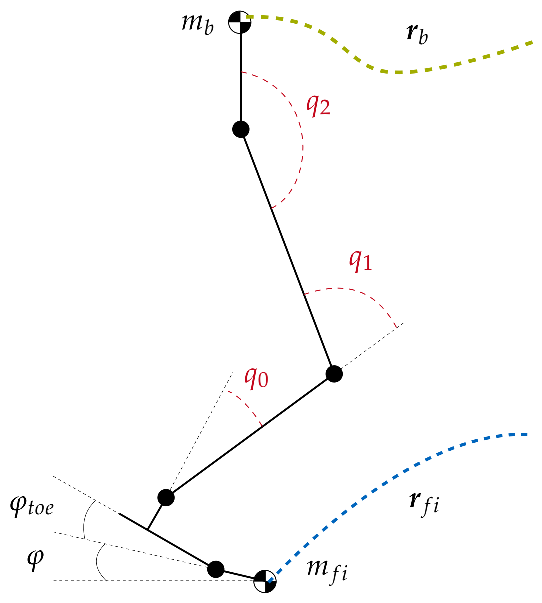


Figure 5.15: 2D kinematic model approximating robot's full kinematics. Input trajectories in black, joint angles in red. See Appendix B for the equations describing the model.

mass point of the three-mass model (x_b, y_b) result from solving the governing equations of motion (EoMs) (see (5.4)). As explained in Subsection 5.2.1, the torso height trajectory z_b itself is a necessary input to solve these equations. Providing an approximated initial torso height trajectory and solving the EoMs yields a sufficiently accurate initial horizontal trajectory. It is re-calculated after each iteration of the optimization. When $\mathbf{r}_b$, $\mathbf{r}_{fi}$, φ, and φ_{toe} are determined, the inverse kinematics can be solved analytically for the joint angles $\boldsymbol{q}$ (see Appendix B). The joint velocities $\dot{\boldsymbol{q}}$ are derived numerically. Thus, real-time capability can be achieved.

5.4.3 Initial Solution

The nonlinear nature of the optimization problem requires an accurate initial solution to avoid convergence to an undesired local minimum. Similar to Nishiwaki [79] and Nishiwaki et al. [81], a so called maximum kinematic feasible torso height z_{max} is used to derive the initial torso height trajectory (see Figure 5.16). The value of z_{max} is obtained from the geometric model with a fully stretched knee with a joint angle of $q_1 = 0$, while the horizontal torso position (described by x_b, y_b) is maintained. This approximation is conservative since the 2D model omits degrees of freedom in the lateral direction and does not include hip and pelvis movements. These additional degrees of freedom allow an even higher torso height in 3D. For calculation of an appropriate initial solution, timing of the support points and constraints have to be defined as follows.

Timing of Supporting Points

In the course of simulations the timing of the support points is found to be crucial for finding a feasible and optimal torso height trajectory. In this implementation, the torso height trajectory consists of four support points per step period. The first and last support point are at the beginning and end of each step (see Figure 5.16). The remaining two set points are chosen according to the current stepping scenario. Timing is not subject to a subsequent optimization due to limited calculation time. Therefore, fixing the timing limits the set of available torso height trajectories. Two characteristic time points inherent to each step period that achieved satisfying results are identified: The maximum height z_{max} for a step period reaches a local minimum at the time of lift-off of the swing foot or touchdown during downstairs movement (see Figure 5.16 for time t_1 and t_4). Choosing this characteristic point in time allows considerations of the kinematic significance in the torso height. The second intermediate support point considers the human ideal. In human walking, the torso is highest during the single stance phase. This maximum occurs when the swing foot approximately passes the CoG's hoizontal coordinate [64]. Figure 5.16 shows the resultant choice of support points for two step periods of moving up a platform.

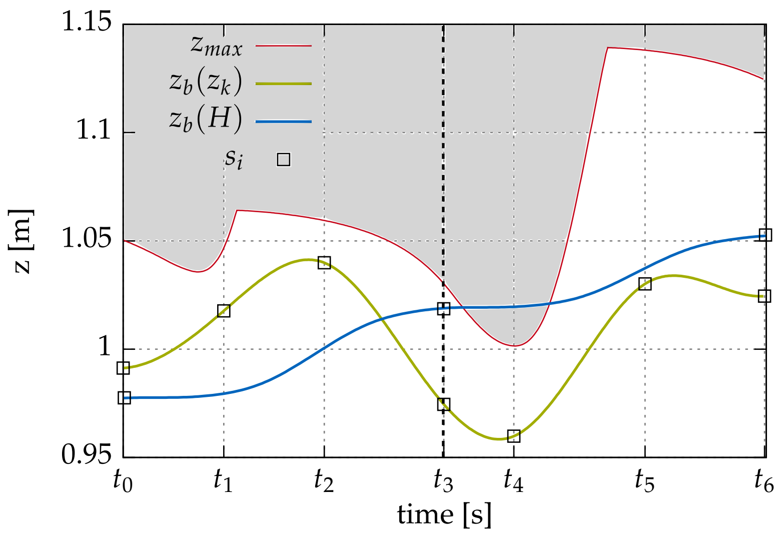


Figure 5.16: Support points of torso height trajectories for two steps periods (approx. 2s), walking up a platform with $\Delta z_{stair} = 7.5cm$. The dotted line seperates the two steps. Trajectories are shown with additional support points ($z_b(z_k)$) and with only one support point at the end of each step ($z_b(H)$).

Constraints for Initial Solution

The trajectory with N support points s_i is composed of $(N-1)$ cubic spline segments and has $4(N-1)$ parameters. There are necessary constraints to obtain a smooth trajectory: the first supporting point fulfills the boundary conditions to provide smooth C^2 connections from the previous step. To further enforce C^2 continuity, $3(N-2)$ continuity constraints are introduced on acceleration, velocity, and position level at the segment connections. For the initial solution three additional constraints per step period are empirically defined in order to avoid unfavorable local minima in the optimization as explained in the following. Figure 5.17 shows two step periods of stepping up a platform. The gray areas mark torso heights that violate joint limits according to the simplified model. Increasing platform heights Δz_{stair} leads to a smaller corridor of feasible solutions.

The constraints are imposed sequentially one step after another:

(1) In order to stay clear of the joint limits, a sufficient slope of the torso height trajectory is necessary. The minima in z_{max} marks the point in time where this occurs (see Figure 5.16: t_1 and t_4). Correspondingly, the slope is constrained at these intermediate support points to be proportional to the stepping height difference.

Two more constraints are imposed onto the supporting point at the end of each step period (i.e. at t_3 and t_6). Both, a change in step length and step height, affect the range of

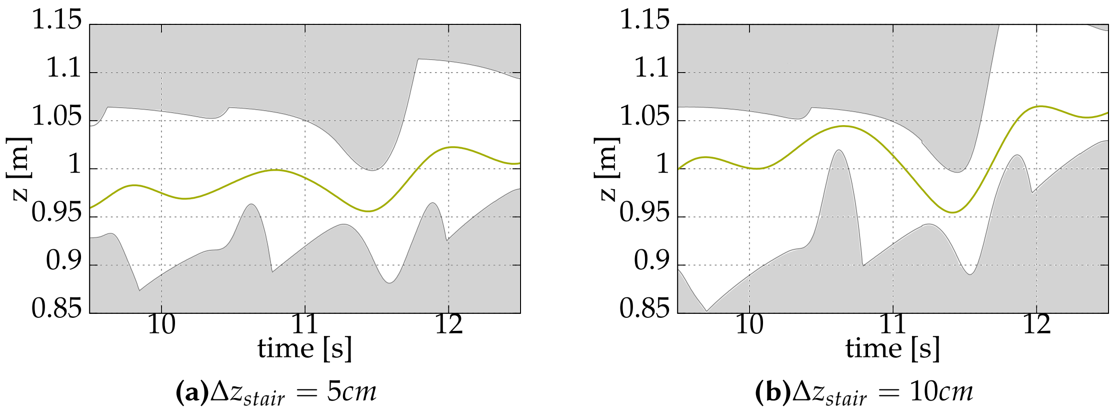


Figure 5.17: Range of feasible solutions for Δz_{stair}.

kinematically feasible torso height trajectories.

(2) Consequently, the height of the support points is constrained at t_3 and t_6 (see Figure 5.16). In the case of upstairs or downstairs movement, the support points at t_3 and t_6 as end points of a step are set resp. higher or resp. lower than the corresponding start point (at t_0 resp. t_3). An increase of the step length leads to a lower minimum in z_{max} since the feet are further apart. Accordingly, the height of the end points at t_3 and t_6 is adapted relative to the corresponding start point.

(3) The slope at the end points is constrained to achieve periodicity in the initial solution. In case neither step length nor height changed from the previous step, the slope at the end is constrained to the slope at the start of the step period; otherwise the slope is set to zero since no further insight about the slope is available.

The resultant cubic spline is uniquely defined with the complete set of $4(N-1)$ constraints.

5.4.4 Optimization

The previously defined initial trajectory serves as the starting point for a parameter optimization. Constraints imposed on the initial solution are omitted for the optimization. Only, the $3(N-2)$ conditions to ensure C^2–continuity as well as the three constraints providing smooth initial conditions are required. Therefore, three parameters per step period remain as degrees of freedom for optimization for $N = 4$. For each cubic spline segment, one of the parameters describing the trajectory is subjected to optimization. Consequently there are $N-1$ optimization parameters z_k. The parameters z_k are grouped in the parameter set $\boldsymbol{z}$. It is the minimal representation of the overall torso height trajectory z_b. The optimization $\boldsymbol{z}$ yields the local minimum with respect to a cost function $f(\boldsymbol{z})$. The optimization problem is defined as follows:

Cost Function

The cost function $f(\boldsymbol{z})$ is a scalar function of the optimization parameters $\boldsymbol{z}$ with

$$f(\boldsymbol{z}) = \int_0^{t_e} \Big(w_{\dot{q}} \dot{\boldsymbol{q}}(\boldsymbol{z})^T \dot{\boldsymbol{q}}(\boldsymbol{z}) + w_{jl} L_{jl}(\boldsymbol{z}) + w_{zm} L_{zm}(\boldsymbol{z}) \Big) dt. \tag{5.15}$$

L_{jl} and L_{zm} penalize kinematic constraints as the violation of joint limits or the violation of the maximum torso height with the weighting factors w_i. The kinematic constraints are not included in the optimization problem as hard constraints. Considering the kinematic limits as hard constraints would result in an increased run-time due to the nonlinear inequality constraints. Since the 2D model represents a conservative estimate of the real robot, violations of the kinematic constraints of the optimization model are acceptable for the full robot. Nevertheless, the optimization requires a well chosen initial solution, which can be provided as explained in Subsection 5.4.3. Furthermore, the weighting factor $w_{\dot{q}}$ for the joint velocities is considerably lower than the other two. The overall cost is obtained by numeric integration of the 2D geometric model over a time horizon t_e and summing up the discrete cost terms with a step size of $\Delta t = 0.006s$.

Time Horizon

The time horizon of the optimization problem is coupled to the robot's physical steps. By taking more than one step into account, future kinematic limits can be avoided. In this work, $n_{steps} \geq 1$ steps can be included in the trajectory optimization. Every additional step that is incorporated leads to an increased dimension of the optimization problem. As a trade-off between computational cost and prediction quality, $n_{steps} = 2$ steps are included in the optimization of the torso height trajectory.

Algorithm and Real-Time Constraints

The optimization problem is solved using the open-source library for nonlinear optimization NLopt[5]. The software package comes with a number of both global and local optimization routines and provides a C++ callable interface. Experiments with different available algorithms showed that the SLSQP algorithm [60], a sequential quadratic programming (SQP) gradient-based optimization algorithm, yields the best results. Since the torso height trajectory generation is part of the robot control, it has to satisfy real-time constraints. In the experiments, the optimization has to be aborted after $T_{opt} = 400\text{ms}$. The NLopt library allows the optimization process to be interrupted at any given time and returns the current best solution of the parameter set $\boldsymbol{z}$. This is especially important in terms of satisfying the hard timing constraints. Furthermore, the integration time step Δt_{exp} of the 2D kinematic model is increased in the experiments to $\Delta t_{exp} = 8\Delta t$ to satisfy real-time constraints. Its influence of the solution quality is analyzed in the following.

5.4.5 Results

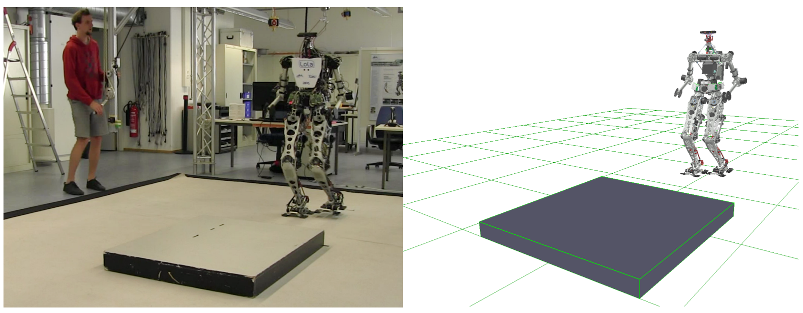


Figure 5.18: Platform test case: snapshots showing Lola in experiment and in simulation walking ahead to platform.

The performance of the proposed torso height trajectory generation has been assessed using the dynamics simulation framework and validated the presented approach in experiments (see Chapter 6)[6]. The approach shows improved performance in challenging

[5]http://ab-initio.mit.edu/nlopt – The library is not used in the previous chapters, since it seems non-compatible with the combined optimization process.

[6]A video of the experiments is available: `https://youtu.be/ayj95PVvq0M`.

stepping scenarios involving obstacles, platforms, or stairs. Even walking, especially fast walking, also yielded cost reductions. The presented simulation results compare the proposed method to generate $z_b(z_k)$ with optimized support points z_k ("on") and the torso height trajectory $z_b(H)$, where the final torso height position H is optimized as part of the *Model-Predictive Kinematik Planning* ("off").

Platform

In this scenario, a platform of height $\Delta z_{stair} = 12.5cm$ is placed in front of the robot. Figure 5.18 depicts LOLA in front of the platform in simulation and in experiment. LOLA stepping up and down the platform in experiments is shown in Figure 5.22.

Platform – Validation of Kinematic Model

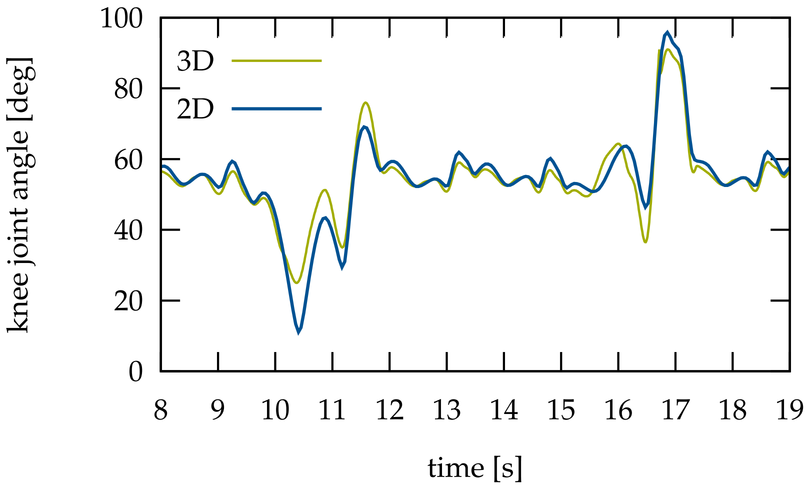


Figure 5.19: Validation between full 3D model and simplified 2D model: left knee joint model comparison.

The performance of the proposed trajectory optimization strongly depends on the validity of the presented 2D kinematic model. The ankle joint angles of the simple 2D model follow the full kinematics sufficiently well. Comparing the knee joint angles in Figure 5.19 shows that the 2D model differs by more than 15° compared to the 3D model when the robot steps onto the platform (compare time approx. 10.5s). During this movement, the DoFs in the hip and pelvis, which are not represented in the reduced model, play an important role. Regarding the knee joint angle, the 2D model approximation is conservative towards the lower joint limit. The simplified model angle could run into the limits while the real joint angle is still in the working range. To exploit this open optimization potential, the weight factor w_{jl} for the knee joint limits is reduced in the optimization.

Platform – Results of Simulation and Experiment

Figure 5.20 shows the resulting torso height trajectories using the proposed method on and off. Furthermore, the upper and lower limits of the torso height trajectory calculated with the 2D model are shown. It is clearly visible, how the torso trajectory with optimization of the support points outperforms the former torso height trajectory by meeting the kinematic constraints. When stepping down the platform, the ankle joint limit of the back leg is reached when only the final torso position is optimized (see Figure 5.21). The

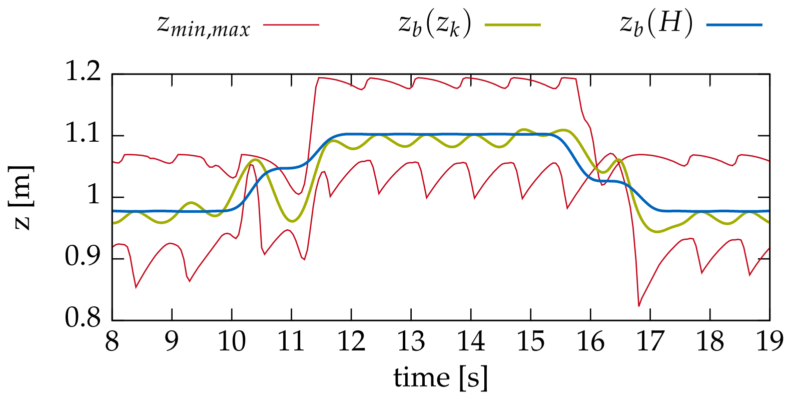


Figure 5.20: Platform test case: torso height trajectories. When stepping down (at time = 16s) there remains no valid corridor for the torso height to stay within joint limits based on the prediction using the conservative 2D kinematic model.

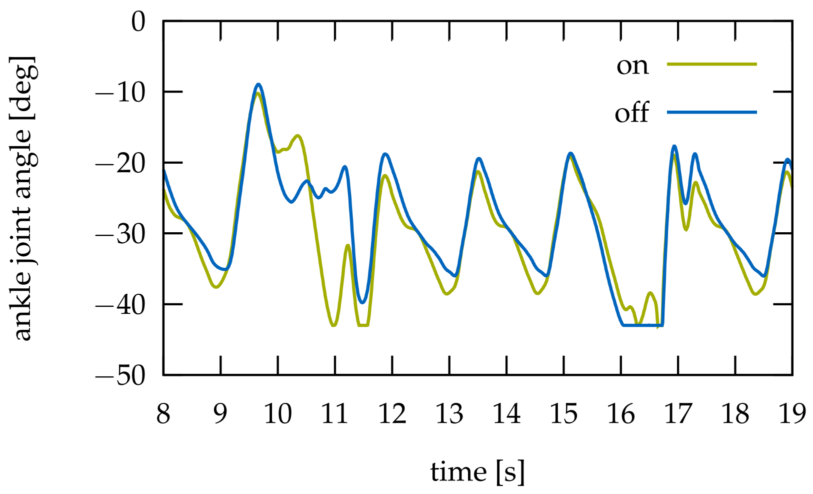


Figure 5.21: Platform test case: left ankle angle with optimization of final torso height position (off) and with proposed method (on). The joint limit is reached at $t = 16$s. Consequently, the joint angle stays constant at its limit.

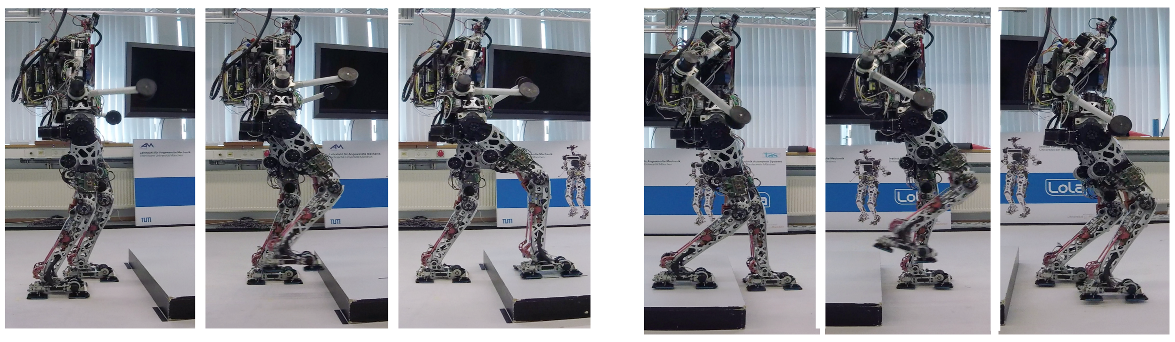


Figure 5.22: Experiment – stepping up and down: snapshots showing LOLA stepping dynamically up and down a plaform using the proposed method.

mechanical limit is avoided by limiting the executable joint angle resulting in the constant joint angle between $t = 16s$ and $t = 17s$ in Figure 5.21. This hard restriction affects the smoothness of moving up and especially down platforms. The optimized trajectory $z_b(z_k)$

avoids the ankle joint limit when stepping down. This is a major improvement over the former implementation and results in a much smoother stepping down movement. When stepping down, there remains no valid corridor for the torso height to stay within joint limits of the 2D kinematic model ($t = 16s$ in Figure 5.20). As it is learned from validating the 2D model, the model approximates the knee joint value conservatively. With the reduced weight factor w_{jl} for the knee joint limit the optimization yields a viable solution for the 3D case. In both approaches the knee joint limits are not violated.

Podium

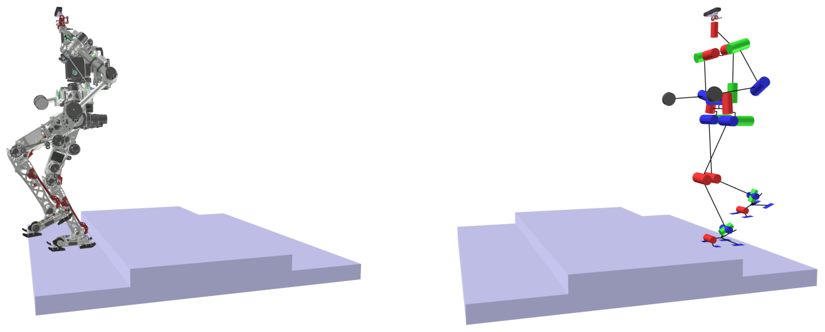


Figure 5.23: Podium test case: snapshots showing LOLA stepping dynamically up and down a podium with two different heigths. Left: LOLA fully modeled in dynamic simulation. Right: LOLA's kinematic model.

This test case consists of a podium with two consecutive stairs of height $\Delta z_{stair} = 10cm$ up and down. Figure 5.23 depicts LOLA stepping dynamically up and down the podium. This test case has not been validated in experiments yet. Nevertheless, according to the quality of the simulation and experimental proof for the platform test case, the author is confident, that this scenario is manageable in experiment as well. The resulting torso height trajectories are shown in Figure 5.24. Similar to the platform test case, it is clearly visible, that the torso height trajectory calculated using the proposed method performs best. The ankle joint angles in Figure 5.25 show that the new cubic spline trajectory avoids ankle limit violations where the quintic spline formulation does not (between 8-9s and 12-13s). Furthermore, the torso trajectories calculated using the 2D model are compared

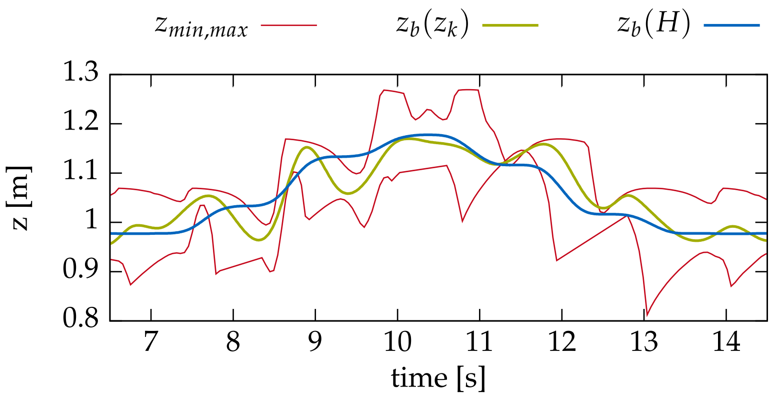


Figure 5.24: Podium test case: torso height trajectories.

with Δt and $\Delta t_{exp} = 8\Delta t$. Figure 5.26 shows that this decrease of accuracy leads to an only slightly different trajectory in this complex stepping scenario.

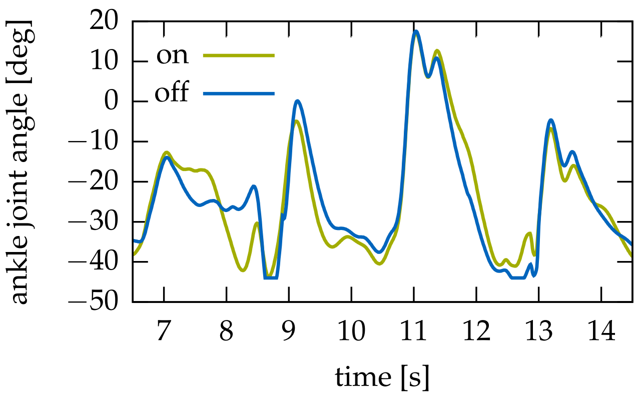


Figure 5.25: Podium test case: right ankle angle with optimization of final torso height position (off) and with proposed method (on).

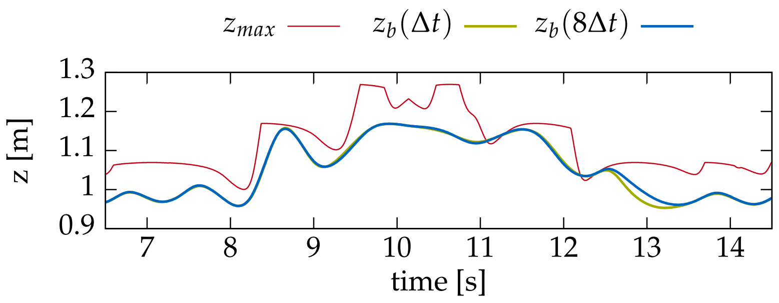


Figure 5.26: Comparison of integration step size of 2D kinematic model.

Complex Obstacle

In scenarios that involve stepping over an obstacle, kinematic joint limits are often violated due to the complex foot movements. Adapting the torso height z_b to stay within the joint limits yielded convincing results. Figure 5.27 shows the obtained torso height trajectory $z_b(z_k)$ in comparison to the trajectory with optimized endpoints $z_b(H)$ for the testcase including the complex shaped obstacle (see Figure 5.12). In particular, when stepping over the obstacle between times $t = 6s$ and $t = 9.5s$, the trajectory $z_b(z_k)$ differs from the formerly employed trajectory by more than $\Delta z = 3cm$. The torso inclination angle φ_y in

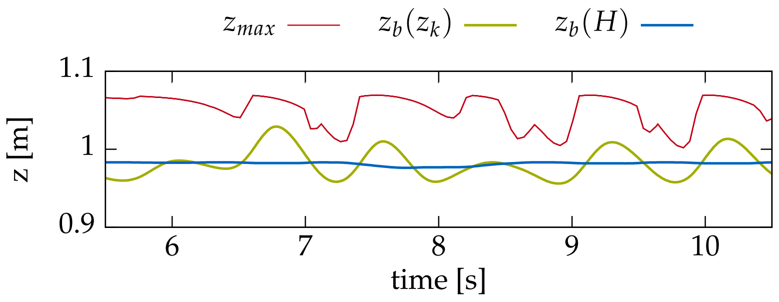


Figure 5.27: Obstacle test case (see Figure 5.12): torso height trajectories.

Figure 5.28 remains by more than 0.5° closer to the vertical orientation when stepping over the obstacle. After passing the obstacle during normal walking, the optimized trajectory yields still a slightly reduced torso inclination error. Holding the upper body upright is one of the main control objectives for stable walking in the used control approach [141, 134]. The kinematically improved motion also shows here a stability improvement.

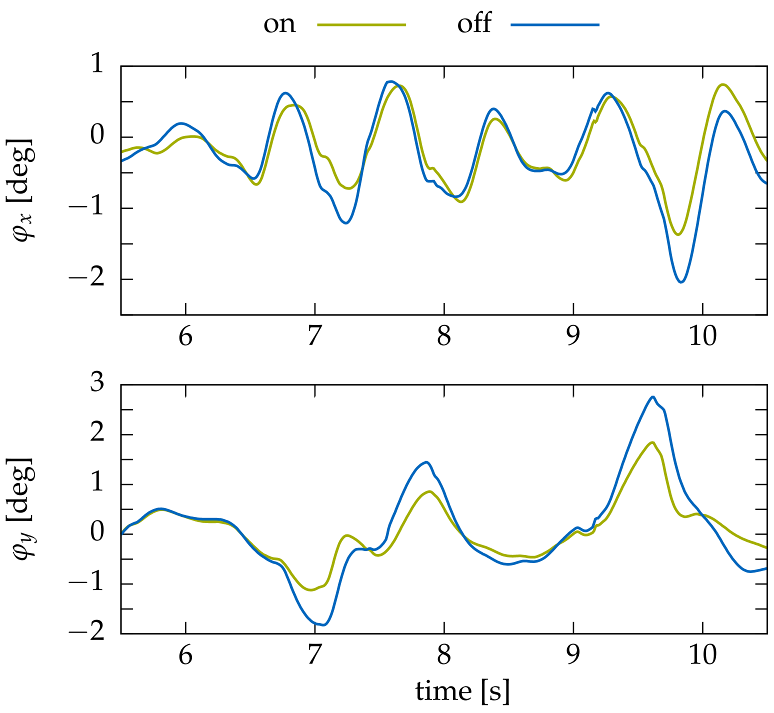


Figure 5.28: Obstacle test case: Inertial Measurement Unit (IMU) angles with optimization of final torso height position (off) and with proposed method (on).

5.5 Reactive Collision Avoidance

The ability to avoid collisions is crucial for locomotion in cluttered environments. It is not enough to plan collision-free movements in advance when the environment is dynamic and not precisely known. Especially when the robot has to modify its ideal planned motion to react to disturbances, local collision avoidance is necessary. In the following, a method is introduced which locally optimizes the task space trajectories in real-time. It was published in Hildebrandt et al. [124].

The publication of Behnisch et al. [8] has to be emphasized for the presented method. They combine a global sampling based search algorithm in task space with a local potential field based method which adapts the global solution and which will be used in a similar way in this work.

5.5.1 Task Space Trajectory Adaption

The method is integrated in the *Feedback Control* as *Collision Avoidance* module (see Figure 3.3).

As described in Section 5.2, the method for self-collision avoidance introduced in Schwienbacher et al. [98] is acting on the nullspace. Thus, it doesn't affect the reference trajectories given in task space. Consequently, ill-chosen reference trajectories can still cause collisions. In order to face this problem and to integrate reactive collision avoidance, it is suggested to modify the task space trajectories. Based on (5.5), the inverse kinematics is calculated as follows

$$\dot{\boldsymbol{q}}_k = J_w^{\#}(\dot{\boldsymbol{w}}_k + \dot{\boldsymbol{w}}_{rca}) - (I - J_w^{\#}J_w)\boldsymbol{u}_k \tag{5.16}$$
$$\boldsymbol{u}_k = \nabla_{\dot{q}} L_y. \tag{5.17}$$

$\boldsymbol{w}_{rca}$ resp. $\dot{\boldsymbol{w}}_{rca}$ represent the local modifications. This approach is largely inspired by Behnisch et al. [8].

Unlike Behnisch et al. [8], the influence of the cost function has to be limited to the six DoF of the swing-foot which are described by $\boldsymbol{w}_F \in \mathbb{R}^6$. Modifications of CoG trajectories could destabilize the robot. The modifications of the swing foot movement and their influence on the CoG trajectory can be balanced by the redundancy of the robot. $\boldsymbol{w}$ can be written as $\boldsymbol{w} = [\boldsymbol{w}_R^T, \boldsymbol{w}_F^T]^T$ with the remaining coordinates $\boldsymbol{w}_R \in \mathbb{R}^{(m-6)}$. A selection matrix $S \in \mathbb{R}^{6\times m}$ and a selection matrix $\bar{S} \in \mathbb{R}^{(m-6)\times m}$

$$\boldsymbol{w}_F = S\boldsymbol{w}, \tag{5.18}$$
$$\boldsymbol{w}_R = \bar{S}\boldsymbol{w}. \tag{5.19}$$

are defined. Hence, the modifications $\dot{\boldsymbol{w}}_{col}$ are determined as follows: The objective function for one time step Δt

$$\Phi = \frac{1}{2}\dot{\boldsymbol{w}}_{F,col}^2 + \Delta H_{RCA} \rightarrow \text{min!} \tag{5.20}$$

is introduced with the first-order change of H_{RCA}, $\Delta H_{RCA} = \frac{\partial H_{RCA}}{\partial \boldsymbol{q}}\dot{\boldsymbol{q}}\Delta t$, which is a potential related to collisions and $\dot{\boldsymbol{w}}_{F,col}$ as the swing foot modification. It is stated that $\dot{\boldsymbol{q}}$ is derived from the least squares solution of $\dot{\boldsymbol{w}}_{F,col} = SJ_w\dot{\boldsymbol{q}}$ yielding to the constraint

$$\dot{\boldsymbol{q}} = (SJ_w)^{\#}\dot{\boldsymbol{w}}_{F,col} \tag{5.21}$$

With the optimality condition

$$\frac{\partial \Phi}{\partial \dot{\boldsymbol{w}}_{F,col}} = \dot{\boldsymbol{w}}_{F,col} + \frac{\partial H_{RCA}}{\partial \boldsymbol{q}}(SJ_w)^{\#}\Delta t = 0 \tag{5.22}$$

it follows directly

$$\dot{\boldsymbol{w}}_{F,col} = -\Delta t\left[(SJ_w)^{\#}\right]^T\left(\frac{\partial H_{RCA}}{\partial \boldsymbol{q}}\right)^T \tag{5.23}$$

and

$$\dot{\boldsymbol{w}}_{rca} := [\boldsymbol{0}, \dot{\boldsymbol{w}}_F^T]^T \tag{5.24}$$
$$\dot{\boldsymbol{w}}_F = -[(SJ_w)^{\#}]^T\nabla_{\dot{\boldsymbol{q}}}H_{RCA}^T. \tag{5.25}$$

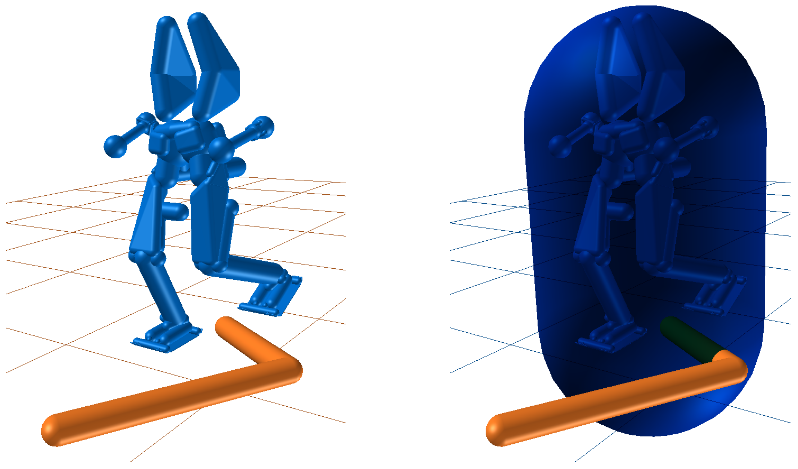
The pseudoinverse in (5.23)) is not directly calculated, but an algorithm similar to the algorithm presented in [57] is applied to solve the linear equation system. The cost function H_{RCA} used above is a combination of objective functions dedicated to collisions and joint limit avoidance. Starting with the collision avoidance the different parts of H_{RCA} are presented in the following.

Collision avoidance

Here, the author resorts to the framework presented in Subsection 3.3.2. The full collision model of the robot (see Figure 5.29) and the environment representation E are used. Not only collisions between robot and obstacles are taken into account, but also self-collisions as for example the two legs. According to Schwienbacher et al. [98], the cost function for collision avoidance is defined here as a piecewise cubic and quadratic function of d_l:

$$L_{coll}(d_l) = \begin{cases} \frac{s_0}{3(t_c^2-1)d_a^2}(d_a - d_l)^3 & : d_l < d_a \\ -\frac{s_0}{d_a(t_c+1)}d_l^2 + s_0 d_l & : d_l < d_a t_c \\ 0 & : d_l \geq d_a \end{cases} \tag{5.26}$$

where d_a is an activation distance for collision avoidance and s_0 and t_c are parameters that adjust the different objective functions to each other. In this work the parameters d_a, s_0 and t_c are chosen differently for collisions pairs including an obstacle or including only robot segments.



(a) Collision Model. **(b)**Boundingbox.

Figure 5.29: Collision world representation of complex scenario. LOLA's collision model and *Lola*'s collision model with the boundingbox, representing distance calculation hierarchy, while stepping-over complex shaped obstacle (orange). Step corresponds to step 7.

Notes

Because of the time consuming distance calculations in every control cycle the presented method is not applicable to environments with more than a few obstacles. To resolve this problem, a hierarchical approach is used:

Instead of distance calculations of each collision pair including an obstacle, a calculation hierarchy is introduced. The whole robot is approximated as a line-SSV called BBL (Bounding Box line-SSV) in the following and introduced in the distance calculation framework. That way it is possible to verify the distances between BBL and obstacles first, and only if the obstacles are inside BBL, the other collision pairs are taken into account. Figure 5.29 depicts the robot's approximation.

Joint Limit Avoidance

In order to avoid joint limits a quadratic objective function L_{limit} is added to take the joint limits into account [98]. Since not all joints (especially the joints of the arms) are supposed to not influence the swing foot movements, only the kinematic chain from the stance foot to the swing foot is taken into account.

Ideal Reference Trajectory Attractor and Re-Planning

For the walking process it is crucial that the robot reaches at the end of each step t_e the ideal reference trajectories $\boldsymbol{w}_d(t_e)$. Otherwise an ill-conditioned initial contact of the swing foot with the floor could result in a critical perturbation. A method composed of two approaches is proposed:

First, an additional objective function is added to H_{RCA}. It is a quadratic attractor function which relies on the error $\boldsymbol{e}_F = \boldsymbol{w}_{mod,F} - \boldsymbol{w}_{d,F}$. L_{att} reduces the influence of the rejecting objective functions defined previously. Therefore it is important to find a parameter configuration which ensures collisions and joint limits avoidance.

Second, the trajectories proposed in [31] are used to plan the error $\boldsymbol{e}_F$ back to zero. The new ideal reference trajectories of the swing foot $\boldsymbol{w}_{F,d,n}$ at $t = t_k$ results in

$$\boldsymbol{w}_{d,F,n} = \boldsymbol{w}_{d,F} + \boldsymbol{e}_F(t_k). \qquad (5.27)$$

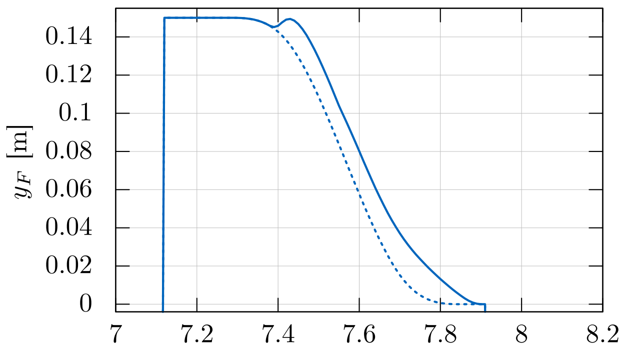
$e_F(t)$ denotes the trajectories which lead the error e_F back to zero. An important characteristic of the trajectories proposed in [31] is that they are overshoot-free. The re-planning process starts a fixed time period before the foot will reach the ground. At this time instance no further trajectory adaptions are allowed. In summary, the cost function H_{RCA} is denoted by

$$H_{RCA} = L_{coll} + L_{limit} + L_{att}. \tag{5.28}$$

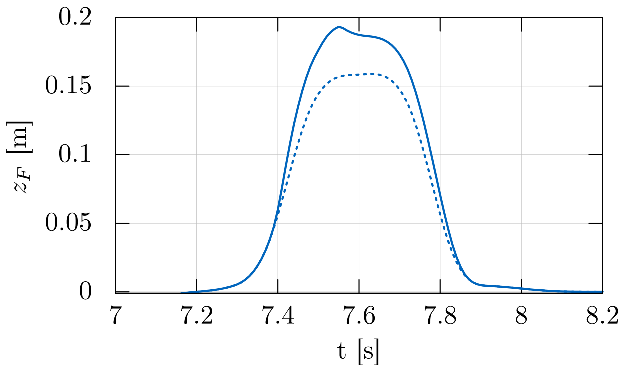
and the re-planning process ensures a correct initial contact of the swing foot with the floor.

Integration with Model-Predicitive Kinematic Planning

The *Model-Predictive Kinematic Planning* is able to provide collision-free, optimal whole-body motions. However, for real-time application, only a limted number of parameters can be included in the optimization problem. Thus, the influence on the shape of the task space trajectories is limited. In addition to the necessary adaptions, when the robot's motions do not longer match with the ideal planned motions, the reactive adaptions represent one possibility to continuously adapt the task space trajectories. Thus, much more complex motions become possible. In order to consistently approximate the influence of the reactive methods for collision avoidance, they are also integrated in the system equations of the *Model-Predictive Kinematic Planning* (5.9).



(a) Lateral.



(b) Vertical.

Figure 5.30: Lateral trajectories (denoted by y_F) resp. vertical movements (denoted by z_F) of the left foot's TCP for step 7 of testcase with complex shaped obstacle. Dashed lines show the ideal trajectories, solid lines the trajectories with modifications of the reactive adaptions.

5.5.2 Results

The reactive adaptions are active in all results presented in this chapter. Even in static scenarios they allow for much more complex motions. Figure 5.30 shows their influence on the task space trajectories in the testcase with the complex shaped obstacle for step 7. It only shows the most significant modifications, but all six DoFs of the swing foot are modified.

5.6 Summary

This chapter focuses on methods for real-time motion generation of bipedal robots. The algorithms presented in Chapter 2 are integrated within a model-predictive approach in the framework for bipedal walking. The algorithms are extended to meet the requirements of bipedal walking and real-time application.

Since the robot's kinematics are not simplified, but the whole kinematic model of the robot is used, whole-body collision avoidance including self-collisions as well as collisions with the environment can be achieved. That way, the high redundancy of humanoid robots can be exploited.

For further improving the versatility of bipedal walking, new task space parameterizations are discussed. A simplified kinematic model allows for optimizing the CoG trajectory taking the constraints of dynamical walking into account.

The methods integrated in a model-predictive approach are combined with a method which adapts online the robot's movements for avoidance of collision and of violation of kinematic constraints. This is especially important, when the ideal planned motions does not longer match with the real motions or the robot is walking in dynamic scenarios.

All methods are analyzed in simulation.

Chapter 6

Autonomous Walking Results

In the previous chapters, methods for navigation and whole-body motion generation with focus on bipedal locomotion are presented. All of the methods have been analyzed and compared in simulation with respect to each other. The simulation results verify their performance and effectiveness, but simulations can not prove their robustness and their real-time capacities. Compared to simulations, in experiments the whole system is exposed to various disturbances on all control layers due to sensor noise or model inaccuracies. These disturbances have to be balanced by each module separately and by their effective combination. Hard real-time constraints require an extended error handling in the communication of the control modules. Last but not least, even small errors, which are not visible in simulations would cause a system to fail in experiments.

In this chapter, experiments validating the method's applicability on real and complex robotic systems are presented. Different scenarios are presented to measure the performance and the real-time ability the presented methods and the software framework. First, the results of the robot executing a step sequence without vision input are presented in Section 6.1. Then, results of experiments with the robot moving in cluttered environments are shown. Section 6.2 sets special focus on the interaction of planning modules and environment perception. In addition to model inaccuracies, the robot is explicitly exposed to external disturbances by a human pushing it. The performance of the system is validated without and with vision input. The results are presented in Section 6.3. Section 6.4 presents results of the robot stepping up and down a platform with and without environment perception. This is a demanding task with respect to the motion kinematics and the robot's stability. Finally, the experiments presented in Section 6.5 validates the systems responsiveness. The robot has to traverse a dynamic environment.

The results presented in this chapter are partly published in Hildebrandt et al. [118, 119, 120, 122], Hildebrandt et al. [42], and Hildebrandt et al. [123, 124].

6.1 Step Sequence

The first experiment includes a simple step sequence in an environment without any obstacles. LOLA executing the sequence is depicted in Figure 6.1. The robot walks a distance of $l_{exp} = 5$m with a step length $L_x = 0.3 \dots 0.5$m, a step time of $T_S = 0.7 \dots 0.8$s and a step height of $dz_{step} = 0.04$m. In the following, first the methods presented in Section 5.3 are analyzed (see Subsection 6.1.1). Second, the influence of a variable torso height trajectory on the robot's motion as presented in Section 5.4 is shown in Subsection 6.1.2.

6.1.1 Model-Predictive Kinematic Planning

This experiment is similar to the simulation test case as presented in Subsection 5.3.8. In the experiments, the real-time requirement forced to limit the optimized parameters to $n_{opt} =$

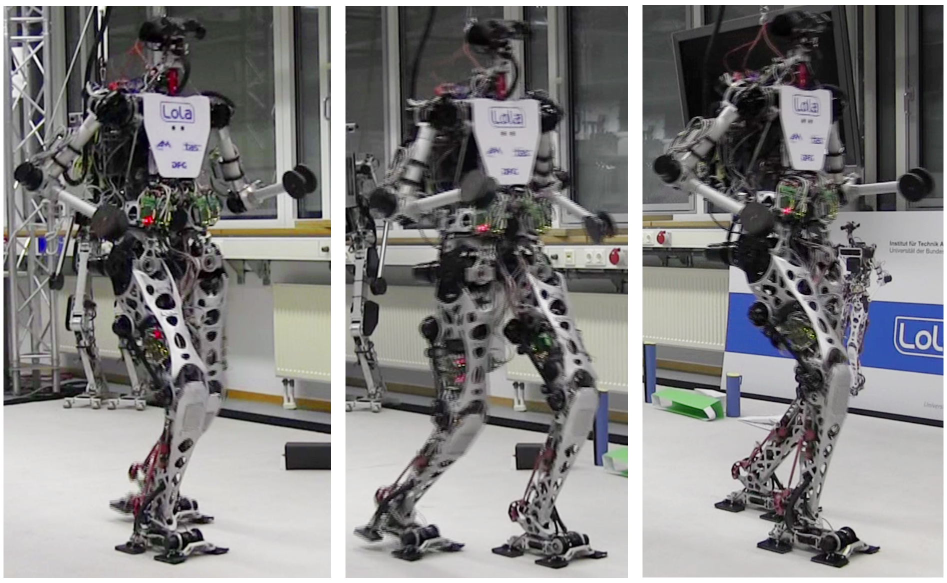


Figure 6.1: Experimental results: Lola executing step sequence with combined optimization [123].

2. Due to recent work by Hildebrandt et al. [118], dz_{step} and dy were chosen as optimization parameters. They seem to be important in kinematically challenging situations. Joint positions are directly measured to compute the cost functions. The measured costs in Figure 6.3 show a comparison of the local and the combined optimization. The results resemble the simulated case (for example the strong reduction of $L_{coll,self}$), yet they are quantitatively different. The reasons for this are the smaller number of iterations in the gradient optimization given by the real-time requirements of the walking control which are not completely reproducible in simulation and the presence of sensor feedback in the *Feedback Control*. Figure 6.2 shows a frontal view on Lola walking with and without combined optimization. It clearly shows the effect of the reduced $L_{coll,self}$ by the arm motions. As expected, in this step sequence experiment the lateral swing parameter dy does not change in optimization while the step height is decreased. This reduces the joint velocities. Figure 6.4 shows a comparison of L_{total} for the periodic walking motion of all presented methods - local method, parameter optimization, nullspace optimization and combined optimization. It is clearly visible, that the combined optimization method outperforms the others. The comparison between nullspace optimization and combined optimization illustrates that for kinematically simple motions the cost reduction can be mainly attributed to an optimized nullspace exploitation.

6.1.2 Torso Height Optimization

Furthermore, the results of the previous section are compared with a variable torso height trajectory (compare Section 5.4). Due to real-time constraints, only the optimization parameter H, describing the torso height at the end of the next step, is included in the *Parameter Optimization* without using the methods for nullspace exploitation. The methods for nullspace exploitation mainly modify the arm motions for straight walking, a combination of methods for nullspace exploitation and the variable torso height would further reduce the motion costs.

Figure 6.5 shows the torso height trajectory with optimized end points in comparison to the proposed cubic spline obtained via optimization. The resultant optimized trajectory strongly resembles the cosine torso height that can be observed in human walking. The corresponding joint velocity costs L_{vel} are depicted in Figure 6.6. A cost reduction is

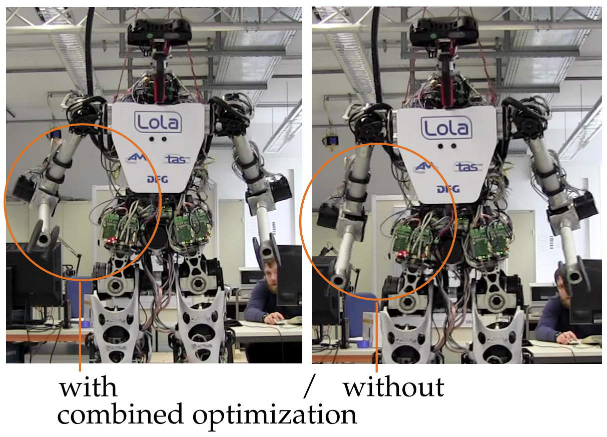


Figure 6.2: Nullspace exploitation: Lola walking with and without combined optimization. Increased distance between arms and hip for $L_{coll,self}$ reduction [123].

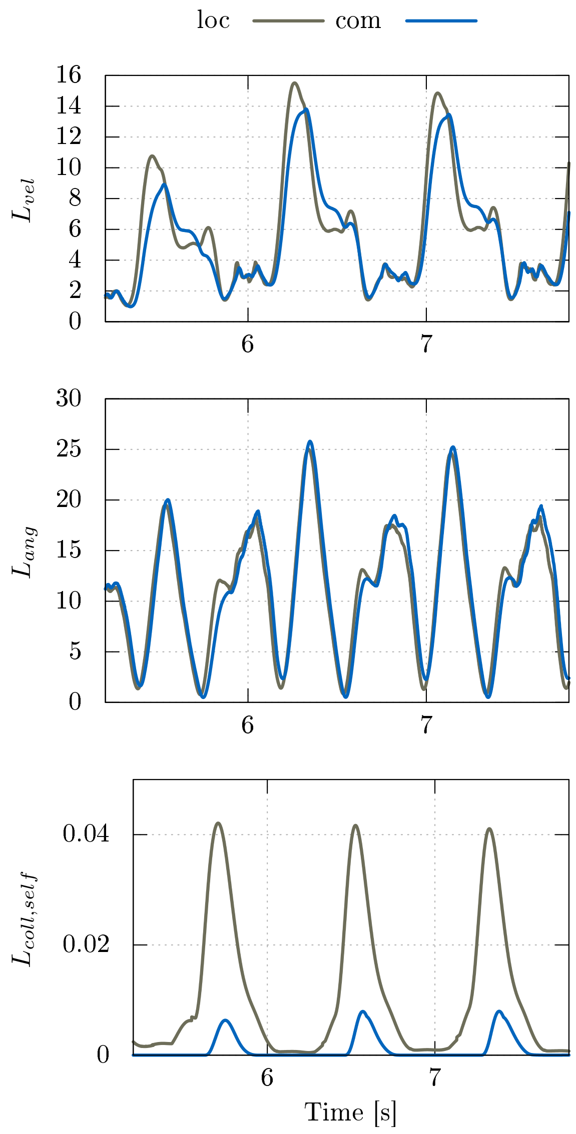


Figure 6.3: Data of step sequence experiment: costs (L_{vel}, L_{ang}, $L_{coll,self}$) over time [123].

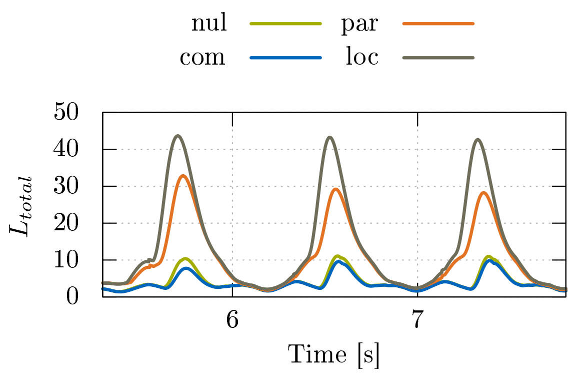


Figure 6.4: Data of step sequence experiment: total costs for local method, parameter optimization, nullspace optimization and combined optimization [123].

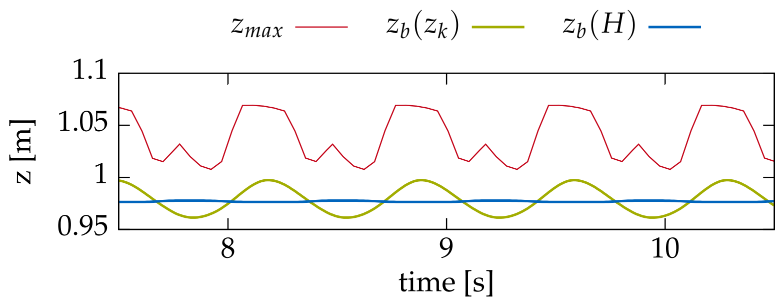


Figure 6.5: Fast walking test case: torso height trajectories.

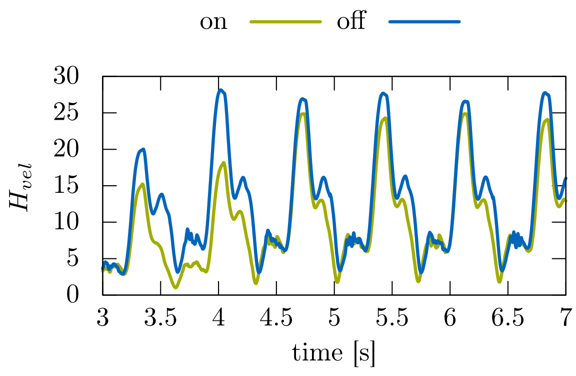


Figure 6.6: Experiment – fast walking test case: cost function L_{vel} with optimization of final torso height position (off) and with proposed method (on).

clearly visible. This confirms that the optimization performed using the reduced 2D model also yields reduced velocities considering the full kinematics and all robot joints. Figure 6.7 shows the joint velocity cost L_{vel} of the *Parameter Optimization*, including the final torso height in the optimization, and of the *Parameter Optimization*, using the torso height calculation based on the presented method for one step.

6.2 Cluttered Environment

In contrast to the experiments presented in the previous section, the experiments in this section include on-board environment perception (see Subsection 3.3.3).

In the following experiments, although the environment is static, it is perceived as

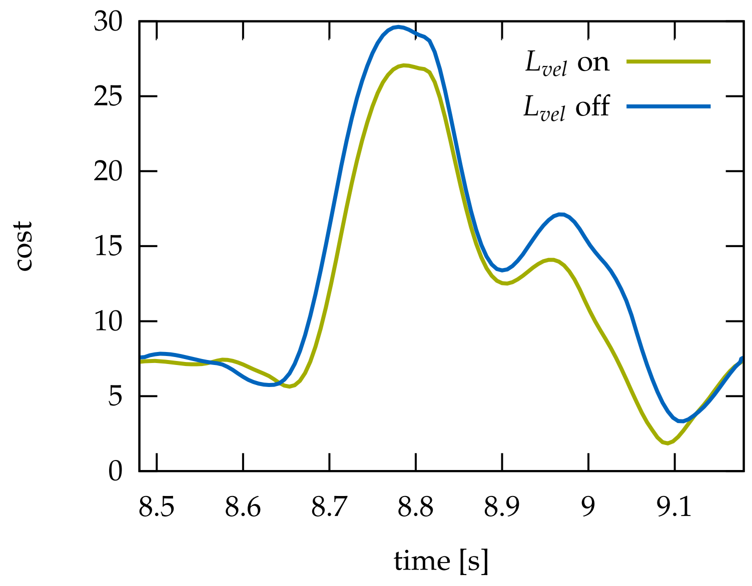


Figure 6.7: Fast walking test case: cost function terms with optimization of final torso height position (off) and with proposed method (on). Results obtained with full kinematic model.

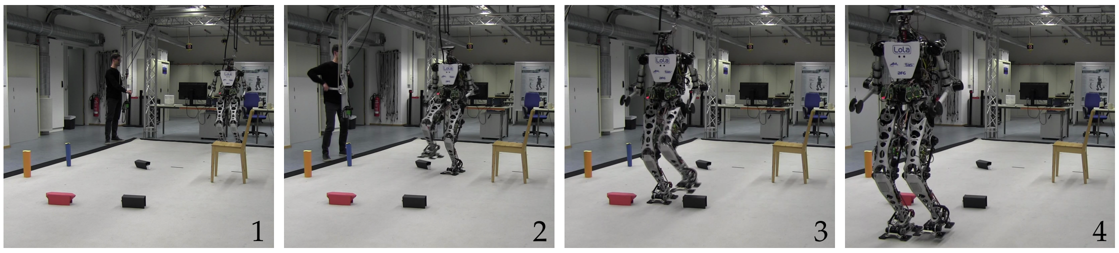
dynamic by the vision system: the robot's walking speed is relatively fast and its FoV is limited. Additionally, the vision sensor's signal quality largely depends on the distance between the robot and the objects [128]. Thus, the robot's environment approximation changes with every step.

Due to the complexity of the perception and planning modules of LOLA, it is hard to conduct experiments in cluttered environments which can be reproduced exactly for all different methods. Little changes in the environment setup lead to major changes in step planning and subsequently in motion planning. This makes the experiment highly demanding in terms of the method's real-time capacities. Therefore, the main objective of these experiments is not to compare the results of all methods against each other (that is done more effectively in simulation) but to validate the real-time capacities of the overall framework. It has to be possible to adapt planning time according to the current walking scenarios and to abort the optimization methods at any given time.

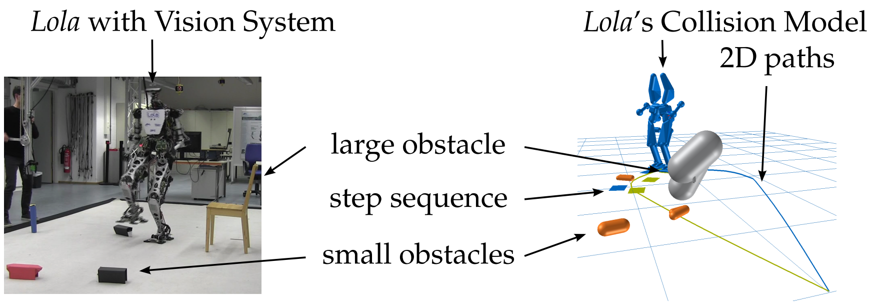
6.2.1 Reactive Navigation

The first setup includes one non-traversable obstacle in front of the robot (a chair) and multiple traversable obstacles at one side of the non-traversable chair similar to the simulation testcase depicted in Figure 4.15. The robot has to reach a goal position 5m in front of it. The desired user chosen step lengths are $0.2 \ldots 0.3$m. A sequence of snapshots showing LOLA traversing the room while stepping over obstacles and avoiding the non-traversable obstacle is shown in Figure 6.8a. Figure 6.8b depicts the internal motion planning results at the second time instance of the depicted sequence. It shows the current collision world approximation of the environment. Furthermore, the calculated 2D paths are visible. At this instance in time the mobile platform planner provides two path options guiding the robot to the goal. The calculated sequence based on the path option which is estimated as the shorter one takes the traversable obstacles into account.

As visible in Figure 6.8b, the navigation module only searches for a limited number of step positions. Since the collision world representation changes every step and new obstacles are detected as they enter the limited FoV, longer step sequences do not provide an advantage. To take the changing collision world approximation into account, the navi-



(a) Snapshots showing Lola at subsequent time instances.



(b) Collision world and 2D paths at time instance 2.

Figure 6.8: Reactive navigation: static environment with small and large obstacles, Lola with on-board vision system, SSV approximation of Lola and of obstacles at time instance 2 in the camera's field of view, calculated 2D paths and calculated step sequence.

gation module is executed each walking step and calculates a new step sequence. A video showing the whole experiment is available at `https://youtu.be/-VvxzFg9ATU`.

6.2.2 Model-Predictive Kinematic Planning

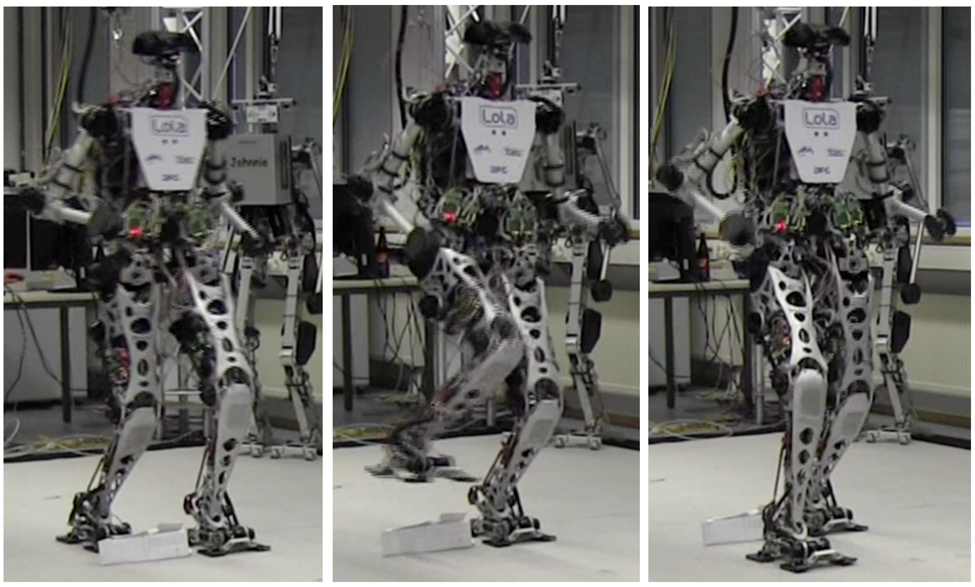


Figure 6.9: Model-predictive kinematic planning: Lola executing step sequence with combined optimization in environment with obstacles. Figure shows Lola stepping over an obstacle.

The second experimental setup includes mainly traversable obstacles. Lola has to traverse the room with desired, user chosen step lengths of 0.2 . . . 0.3m. The experimental setup and its approximation for the robot control is depicted in Figure 6.9 for one time instance. As discussed for the navigation module, the changing environment approxima-

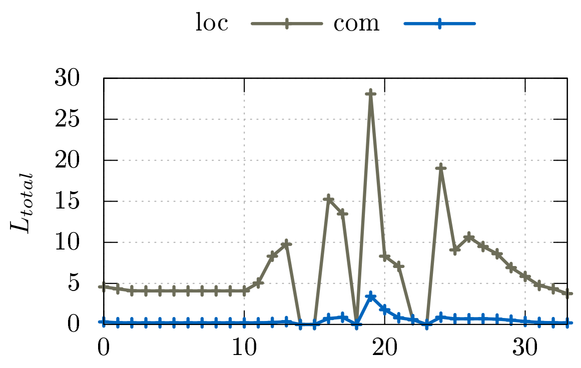


Figure 6.10: Model-predictive kinematic planning with combined optimization for cluttered environments: comparison between total costs resulting from combined optimization (com) and from *Local Control* (loc).

tion is highly demanding in terms of real-time capacity of the model-predictive kinematic planning. The navigation module adapts the step time of each step. Furthermore, the integration time of the model-predictive method is obstacle-dependent. For that reason, the optimization, e.g. the number of iterations, has to be adapted for each step to the current walking step, as presented in Subsection 5.3.7. Figure 6.10 shows L_{total} for each step for the combined optimization and the local method. For most steps, the combined optimization method could reduce the total costs. In steps 14, 15, 18, 22 and 23 the optimization had to be aborted because of the limited planning time. LOLA navigating among the previously unknown obstacles is shown in Figure 6.15.

6.3 Cluttered Environment with Disturbances

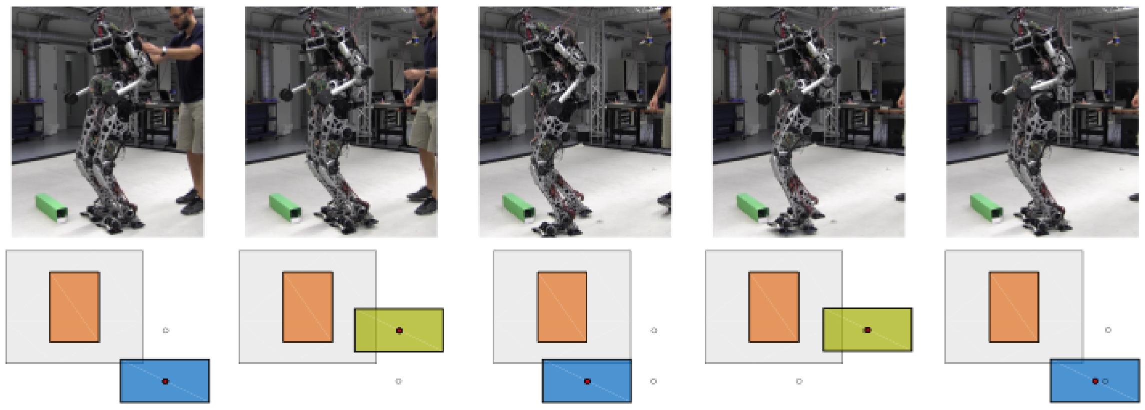


Figure 6.11: Cluttered environment with disturbances - Experiment 1: snapshots at different time instances of LOLA stepping in front of an obstacle and being simultaneously pushed. Corresponding obstacle approximation used in method presented in Section 3.4 and ideal and modified foothold positions. See Figure 3.23 for further explanations.

The previous section presents experiments with the robot moving in cluttered environments. In real world scenarios, the robot may be exposed to external disturbances in addition to model inaccuracies. This is highly demanding for the overall interaction of the different modules of the control framework. In this section, the robustness of the interaction of the methods for *flexible* and *robust* walking is validated. Its performance relies mainly on the method presented in Section 3.4. Due to its reduced computational time, it

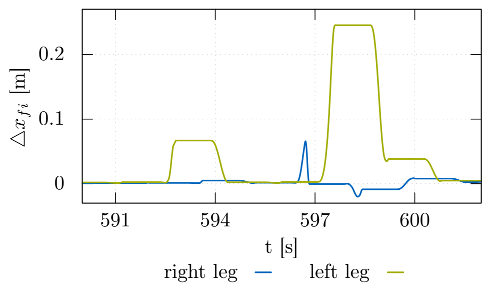


Figure 6.12: Cluttered environment with disturbances - Experiment 1: modification trajectories of the legs in x-direction.

is decided to test the footstep optimization only with the restriction of the optimization result on the real robot.

In the following, two experiments will be presented. The first experiment is a synthetic case resembling the simulation case presented in Subsection 3.4.5 whereas the second exhibits a more general setup. In experiment 1, the obstacle is put right in front of the robot. In contrast to experiment 2, the obstacle is manually approximated without input from the vision system according to the simulation setup (c.f. Figure 6.11). The vision system is not used for two reasons: (1) According to the simulation setup, the obstacle is not in the camera's field of view. (2) This experiment should examine only the procedure for the method to consider obstacles during disturbance rejection without having the uncertainties of a running vision system. The sequence in Figure 6.11 shows the robot at different time instances after it is pushed from behind. The corresponding modification trajectories for the feet and inclination errors of the upper body are depicted in Figures 6.12 and 6.13. The limitation of the footstep modification, ΔL_x, can be seen in Figure 6.14. The obstacle lying in front of the robot limits the right foot's step motion, since the robot shouldn't step on it (second picture of Figure 6.11). In the next step the robot performs a large step modification to avoid divergence of the inclination error.

Experiment 2 additionally includes the vision system. The setup is presented in Figure 6.16. The robot is commanded to walk forward with 30 cm step lengths. While walking the robot's walking control registers the obstacles detected online. The modules for *flexible* walking calculate an ideal step sequence and parameter set in real-time that ensures collision-free movements. The ideal motion is modified based on the robot's state. The robot is pushed several times during the experiment and recovers from the disturbances without colliding with the obstacles. The overall ideal step sequence and modified footholds calculated are shown in Figure 6.17.

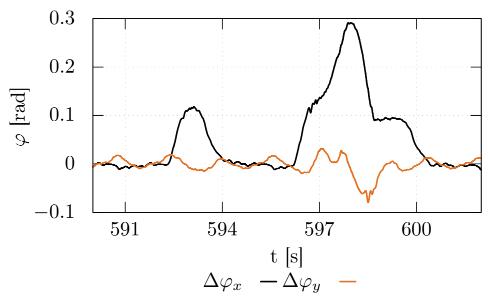


Figure 6.13: Cluttered environment with disturbances - Experiment 1: resulting inclination errors of the upper body (IMU-data).

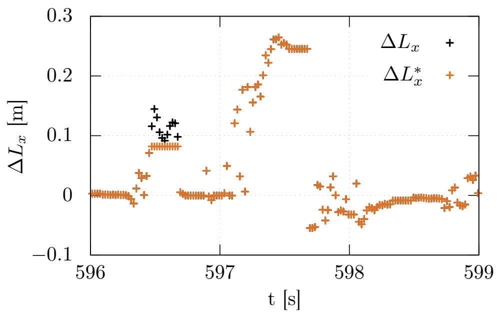


Figure 6.14: Cluttered environment with disturbances - Experiment 1: optimization result of the step length adaptions before (ΔL_x) and after (ΔL_x^*) collision check.

6.4 Stepping Up and Down

Real versatile walking includes not only walking on flat ground but also stepping up and down platforms or stairs. As discussed in Section 5.4 the stepping up and down movement is kinematically demanding. This section further discusses the experiment presented in Subsection 5.4.5. The experimental setup is depicted in Figure 6.18.

Figure 6.19 shows the step sequence executed by the robot. The step sequence and the initial parameters, which are heuristically determined, are the input as $\boldsymbol{p}_{wp}$ to the method presented in Section 5.4. It generates an optimal CoG trajectory. The following *Model-Predicitive Kinematic Planning* optimizes the parameters dz_{Step} and dy describing the swing foot trajectories of the next step. Table 6.1 lists the initial parameters, determined heuristically by the step planner, and the optimized parameter describing the swing foot height. The heuristic provides a good initial guess, which is slightly optimized. For real-time application only a limited number of parameters can be optimized. Thus, the resulting movement may not meet all kinematic constraints including collision avoidance. Therefore, the method, described in Section 5.5, locally modifies the swing foot trajectories. Figure 6.20 shows its influence on the vertical swing foot movement for stepping up and down. Without such a continuous adaption in each control cycle the movement would not be executable.

6.5 Dynamic Environment

So far, only experiments in static environments were presented. The responsiveness of the approach also allows for application in dynamic environments. This section presents an experiment with LOLA moving among static obstacles and a dynamic moving human.

A sequence of snapshots of the experiments is depicted in Figure 6.21. The video of

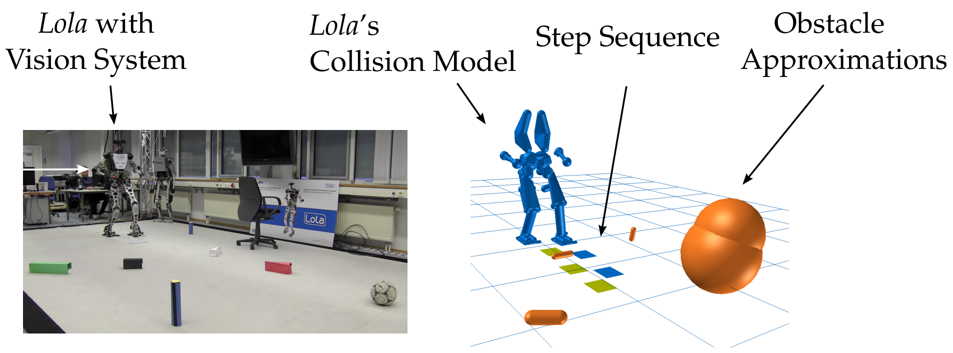


Figure 6.15: Setup for experiments in cluttered environments.

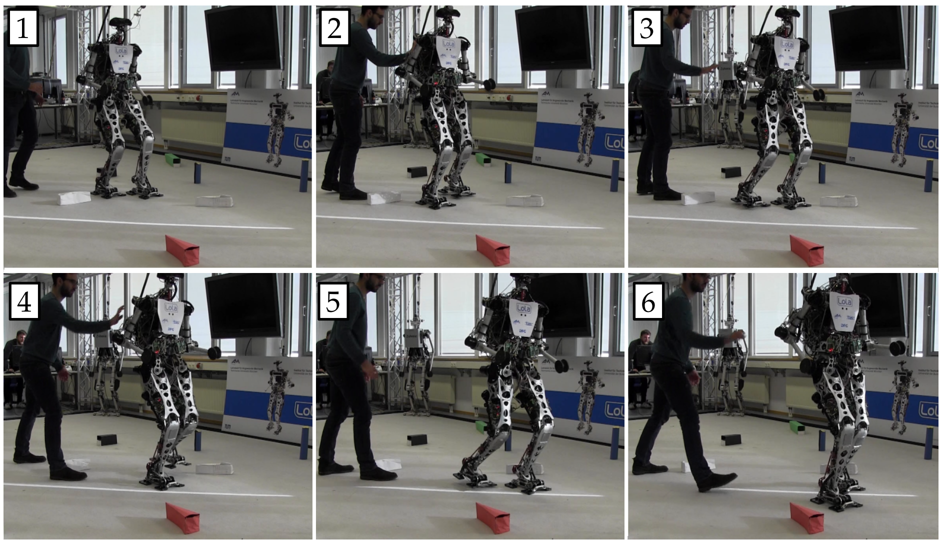


Figure 6.16: Cluttered environment with disturbances - Experiment 2: snapshots at different time instants of Lola walking in cluttered environment while being pushed simultaneously.

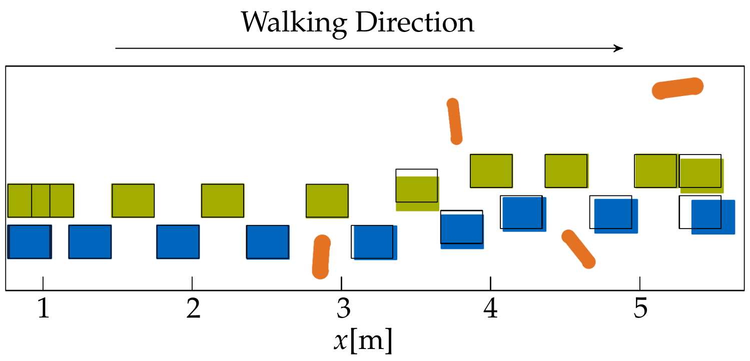


Figure 6.17: Cluttered environment with disturbances - Experiment 2: ideal step sequence is represented as black boxes. Modified and executed steps are represented as filled boxes. Relevant obstacles in orange.

Table 6.1: Optimized parameters for stepping up and down a platform. Only steps up (Step 6 and 7) and down (Step 12 and 13) are listed (see Figure 6.19). dz_{step} denotes the relative height of the swing foot trajectory relative to either the start or the final position of the swing foot depending on the height.

	Initial	Optimized
Step	dz_{step}	dz_{step}
7	0.016m	0.02m
8	0.04m	0.04m
12	0.04m	0.04m
13	0.016m	0.02m

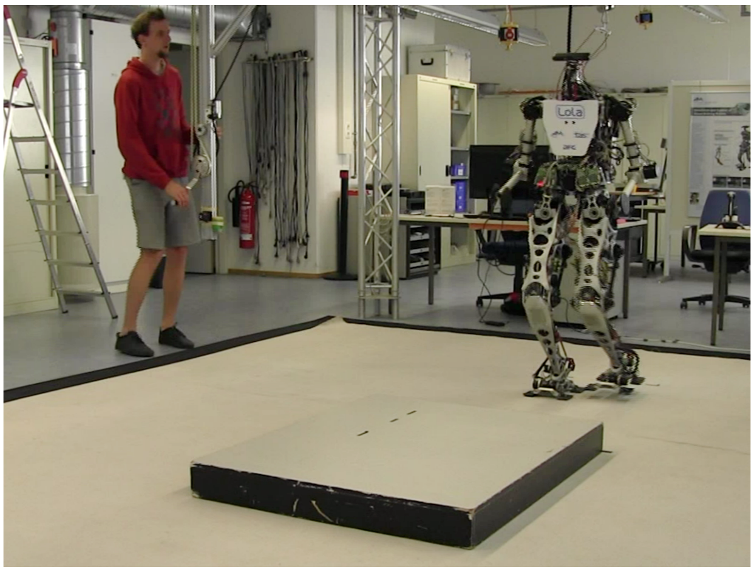


Figure 6.18: Platform test case: snapshots showing Lola in experiment walking ahead to platform. It is the same experiment as discussed in Subsection 5.4.5.

the whole experiment is available online `https://youtu.be/-VvxzFg9ATU`. The experiments validated the real-time capability of the presented methods: the environment in the experiments is completely unknown and dynamic. It is modeled using our on-board vision system.

Figure 6.22 shows the result of the mobile platform planner and the step planner for three different walking steps. Since the field of view is limited, Lola perceives a new part of the environment with each step. Furthermore, Lola has to react to the human dynamically stepping in its way. This is clearly visible in the differences of the environment of snapshots $1-3$ in Figure 6.22. The mobile platform planner adapts to the changing environment and determines executable 2D paths. The step planner uses these 2D paths as a guideline and calculates step sequences. Figure 6.22 emphasizes the real-time character of the step planner. It is aborted after $n_{exp,des}$ steps are calculated or the time limit is reached.

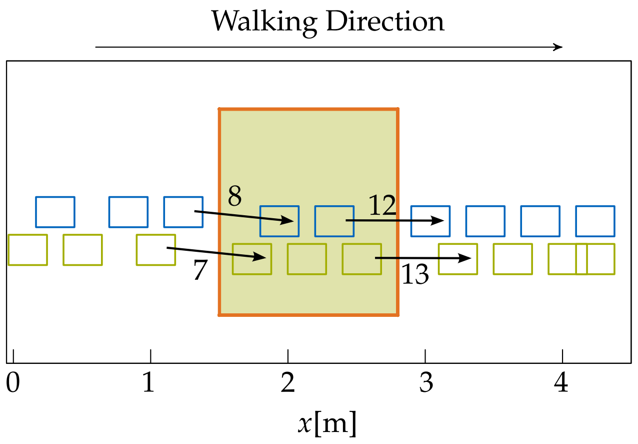
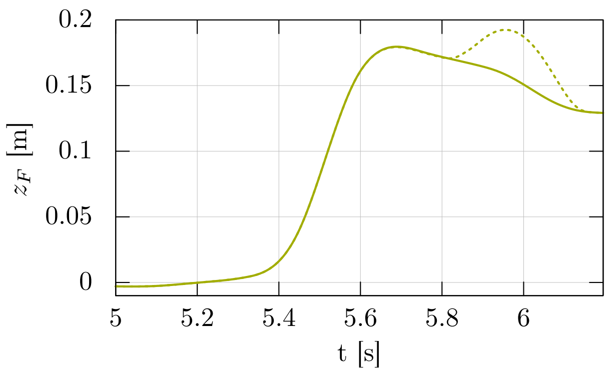
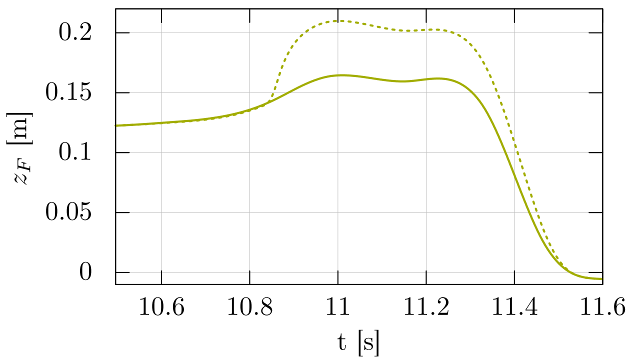


Figure 6.19: Lola stepping up and down a platform - executed step sequence. Experimental setup is depicted in Figure 5.18.



(a) Stepping up.



(b) Stepping down.

Figure 6.20: LOLA stepping up and down a platform - influence of reactive collision avoidance. Vertical movements (denoted by z_F) of the right foot's TCP for step 7 and step 13. Dashed lines show the ideal trajectories, solid lines the trajectories with modifications of the reactive adaptions.

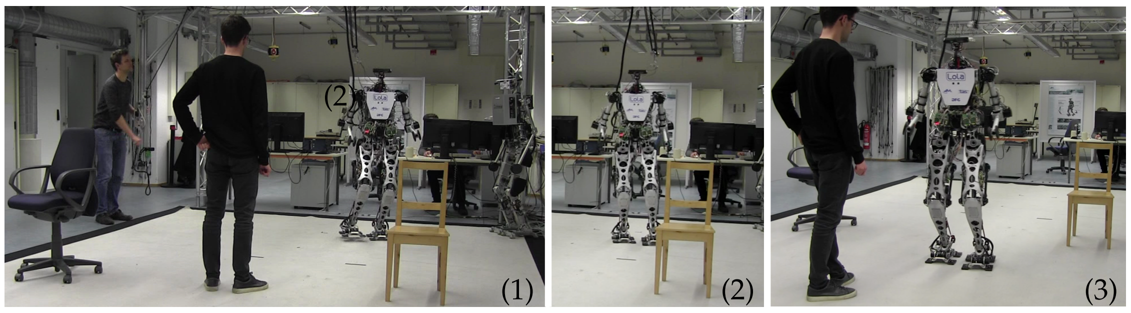


Figure 6.21: Dynamic environment: LOLA autonomously finding its path to a given goal position in dynamic environment with moving human and multiple large obstacles at three different time instances.

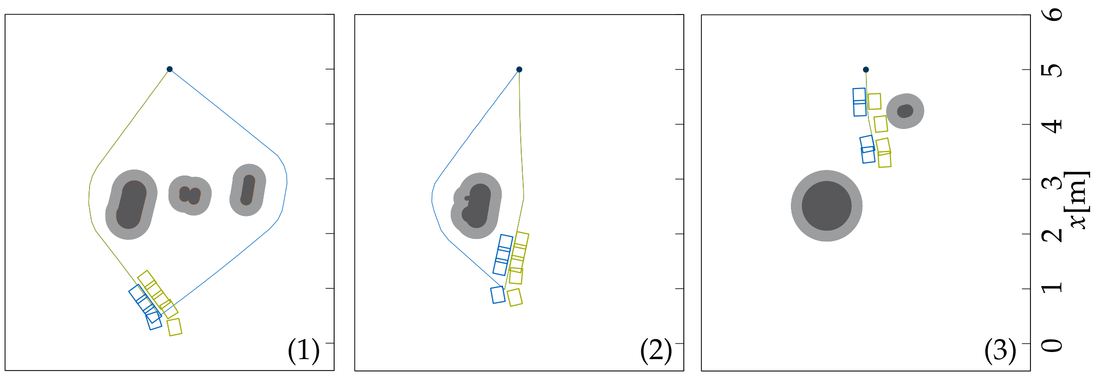


Figure 6.22: Experiment in dynamic environment with moving human and multiple large obstacles: corresponding calculated step sequences and environment approximation at three different instances. Corresponding snapshots are depicted in Figure 6.21.

6.6 Robustness

The robustness of the presented approach allowing for walking in uneven terrain is difficult to quantify. On the one hand, a real quantification of the approach needs an experimental setup which is reproducible. The setup does not only depend on the environment but also on multiple additional factors which change for each experiment and can not be controlled by the author:

- depending on the friction and the velocity of the robot, the robot slides on the ground, which changes its position relatively to the environment;
- the sensors' inaccuracy lead to differences in the perceived environment and the robot's internal state;
- it is difficult to ensure that the robot starts each experiment from the same position in the environment and receives the same user input at the same instances.

These inaccuracies represent small changes in the experimental setups, but they may change the calculated step sequences and therefore the walking situations completely.

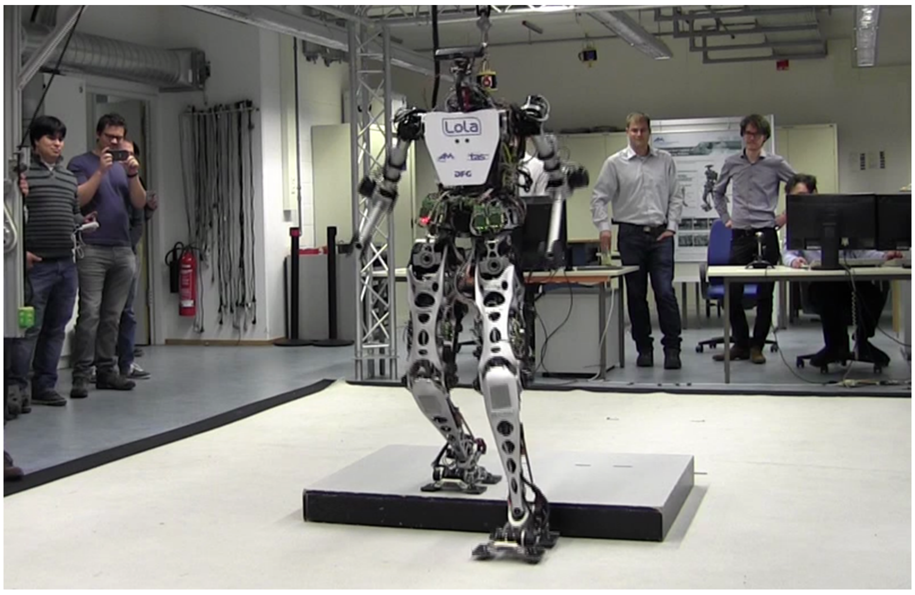


Figure 6.23: Lola stepping up and down a platform at final presentation of DFG project in front of robotic researchers.

On the other hand, a quantification would only consider some specific walking situations. Even in the laboratory, it is possible to set up a large number of different experimental setups which are not all quantifiable. The setups depend on the robot's walking speed, the exact location of the robot in the world and on its previous and future motions. Also the human rarely executes the same steps subsequently and it also may fail in difficult situations.

However, up to now, in the author's opinion the most effictive way to analyze the robustness of the approach is to execute multiple experiments in changing setups. The experiments presented in this chapter are only a small selection of the multiple experiments performed to analyze different aspects of the methods. Furthermore, the methods are presented at several occasions for researchers but also in public presentations. The most relevant one was the final presentation of the DFG-founded project in January 2017 in front of multiple robotic researchers, where Lola performed a selection of the presented experiments in unknown environments live. A picture of LOLA stepping up and down a platform at the presentation are shown in Figure 6.23. A video of the presentation is published at `https://youtu.be/g6UACMHgt20`.

6.7 Chapter Summary

In this chapter, experiments are presented validating the method's applicatbility on real and complex robotic systems. The presented experiments include

- execution of a simple step sequence,
- experiments in cluttered environments with vision system,
- experiments in cluttered environments with external disturbances,
- experiments including steppable platforms and experiments in dynamic environments.

These experiments represent a collection of a wider range of experiments highlighting different aspects of the methods. Special focus is set on the method's real-time capability and the method's integration with the vision system and the methods for disturbance rejection.

Chapter 7

Conclusions

The robots' limited autonomy in unknown and changing scenarios seems to be one of the major bottleneck to employ them in real applications. While redundant hardware design is beneficial in terms of autonomy in cluttered environments, its exploitation in real-time application is one challenge of current robotic research. In the preceding chapters, methods have been presented for motion generation of redundant robots in unknown and dynamic scenarios. This thesis concentrates on humanoids as one prominent representative of redundant mobile robots. Thanks to their kinematic structure, they are especially suited to be employed in diverse and highly complex scenarios. Nevertheless, the methods developed within this thesis are not limited to humanoid robots, but are applied to other robots such as agricultural robots or industrial manipulators working in complex scenarios. The following section gives a brief summary of key ideas and contributions of this thesis. Section 7.2 and Section 7.3 discuss the achievements and point out directions of future research.

7.1 Summary

The work, presented in this thesis, was done as part of the DFG-founded project *Walking in Uneven Terrain* (BU 2736/1-1). The goal of the project was to develop methods which allow to use biped robots outside tightly controlled laboratory conditions and make their usage in real applications realistic. Specifically, the project addresses three shortcomings in current research: (1) on-board environment perception and modeling for fast walking in unknown and dynamic environments. (2) real-time motion generation, which closes the gap between navigation and joint trajectory generation, and (3) robust walking control to stabilize the robot even during large disturbances. A key component of the methods success is their tight integration in one overall framework.

This thesis concentrates on methods for real-time motion generation. The developed overall concept is a hierarchical approach. It is based on three layers, which use models with differing levels of detail to predict and plan future motions. Figure 7.1 shows an overview of the layers. For planning long motion sequences, first a set of way points is calculated. These way points are described by sets of parameters. The robot's motion in between these way points are approximated by computationally simple weighted models and heuristics. That way, long time horizons as well as large areas of the search space can be investigated. In the second layer, the robot's motion is optimized. Different methods are proposed for optimization of task space trajectories as well as optimal redundancy resolution. Since a full kinematic model of the robot is used, the time horizon is reduced in comparison to the sequence planning in order to allow for real-time application. The third layer modifies reactively the robot's motion to take sensor feedback into account. Although, the methods were developed having their application for bipedal walking in

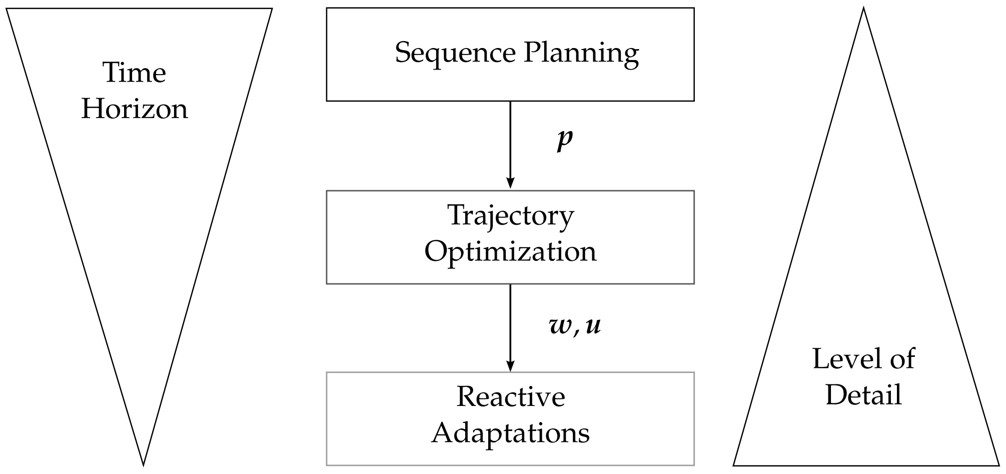


Figure 7.1: Hierarchical approach for motion generation.

mind, an application on a 5-DoF manipulator is shown to emphasize their applicability to a wider range of redundant robots.

This approach for real-time motion generation was applied and extended for bipedal walking as follows. Navigation methods were developed for searching sequences of foothold positions. An A*-search is applied to find the optimal one. Instead of investigating only the footholds an articulated 3D approximation of the lower leg and the foot is considered to find feasible and optimal footstep locations. Additionally, it provides an initial solution for the swing-foot movement. Key for their effectiveness in unknown and dynamic environments is their responsiveness to changes in the environment and the user's input. Methods which integrate a mobile platform planner in the step planner and which continuously optimize the set of discrete step suggestions make fast re-planning even in complex environment possible.

The robot's motion is determined by splines in workspace configured by the parameter sets. The robot's redundancy resolution is dedicated to minimizing potential functions. The robot's full kinematic model is analyzed for one step ahead of the actual executed motion. That way, non-executable motion can be identified before the robot starts executing them. An advanced error handling makes the motion more robust. Methods for the parameter sets which describe the workspace trajectories as well as methods for the redundancy resolution optimize separately and combined the overall motion. At each control cycle step, reactive optimization methods modify the workspace trajectories according to sensor feedback.

The real-time motion generation was verified and validated in multiple simulations and experiments. Selected experiments are presented in this thesis. Furthermore, the methods were integrated in an overall framework for bipedal walking control. They were combined with an on-board environment perception system and stabilization methods. That way, the approach could be validated in complex experiments including unknown and dynamic environments and large disturbances. Autonomous walking in unknown and uneven terrain is only possible when combining the methods. This overall framework was successfully demonstrated at multiple public presentations as, for example, the final presentation of the project *Walking in Uneven Terrain* with invited researchers from all over Germany.

7.2 Discussion

In this thesis, literature of current research was analyzed and discussed. Although, many research groups are focusing on bipedal walking in uneven terrain, very few of them presented experiments with the scope of the experiments of this project. Usually, experiments showing results on stabilization methods, vision systems or motion planning methods are presented separately and for very narrow tasks in predefined environments. The compatibility with one another in changing scenarios and the scalability has yet to be shown.

The methods presented in this thesis were successfully applied and validated on the robot LOLA: They allow for navigation in dynamic environments, stepping up and down platforms, stepping over obstacles and handling external disturbances in the presence of previously unknown obstacles. In contrast to most of current research, they are not limited to predefined scenarios and can be applied in combination.

Like for humans, walking in uneven and unknown terrain is a challenging task for bipedal robots. Even in the labrotary, scenarios may still be too complex to be successfull completed and they may provoke a fail of the system. On the one hand, this is due to the technical limits of the system. LOLA was developed at a university and it is still a prototype. Technical failures are inherent to a prototype. On the other hand, the developed methods, represent approaches how walking in uneven terrain can be achieved. The robustness of the methods have to be improved by further tuning in experiments.

In our simulations and experiments, the cost function design plays a crucial role for the performance of the optimization methods. Although complex motions are feasible when applying the presented methods, the question "What is optimal bipedal locomotion?" can not be answered.

7.3 Directions for Future Work

The direction for future work can be divided in two parts: the first one is related to improvements of the current methods. The second one covers open research questions for which the current approach needs further improvements on the methodological level.

Outlook for Improving Current Implementation

Cost Function Design

In the current implementation of the methods presented in Chapter 5, the influence of the collision avoidance is rather high. Zucker et al. [116] propose an approach of cost weight scheduling to dynamically adapt the cost weighting. The influence of the collision avoidance on the total costs is increased near obstacles. This results in better behavior of the gradient method and could also improve the results in the presented applications. Another possibility would be to introduce a strict task hierarchy as described in Ott et al. [84].

Real-Time Application and Parallelization

The methods presented in this thesis are real-time methods. Based on an initial solution, the methods improve the results. They are implemented such that it is possible to abort the calculations at any given time instant. Thus, all of them could be easily parallelized which could significantly improve their performance. This could be implemented for the *Navigation*-module with multiple solutions of the mobile platform planner's result as suggested in Chapter 4. Parallelization would also be beneficial in the *Model-Predictive Kinematic Planning* as presented in Chapter 5. Instead of starting with one initial solution, the motion generation could be started with different sets of parameter including different foothold positions. Furthermore, the implemented line search algorithms could be accelerated analogously as discussed in Graichen [37].

Dynamic Environments

The presented methods for motion generation are able to deal with dynamic environments due to their re-planning before each executed step. Dynamic moving objects are not explicitly taken into account. First work of integration of dynamic moving objects in the sequence planning explicitly has been done in cooperation with Tobias Kindsmueller [143]. The developed methods does not augment the search space by the dimension of the time but only introduce time-dependence in the object approximations. Thus, the timing of the step sequence is still calculated based on heuristics. In addition to the time-dependence of the object approximations, confidence intervals for their motions are introduced. These intervals represent the error in the estimation of the objects motions. The uncertanties are taken into account in the step planning via additional costs. Till now, this approach has shown good results in simulation but it has to be combined with the vision system as presented in Buttner [137]. Including time as a free variable in the *Navigation*-module would further increase planning time, but the capabilities of the robot would increase.

Robustness

Since the methods developed in this thesis are part of an overall framework for bipedal walking in uneven terrain, the methods' quality has to be always evaluated with respect to their interaction with the whole control of bipedal locomotion. This has to be done in experiments, due to their great variety of requirements. For successfully conducted experiments, robustness of the overall system is crucial. During the project, the robustness of the software framework could be greatly improved by introducing test cases for continous integration. Although, the overall system is evaluated, debugging could be improved by using software testing of each sub module, *unit tests*, of the overall system. Furthermore, defined input and output limits for the sub modules could help evaluating the performance of the overall system.

Artificial Environment

In cooperation with Dipl.-Ing. Daniel Wahrmann, the simulation environment is currently extended by an artificial environment. Instead of testing multi-body simulation and vision system seperately, they have to be evaluated together already in simulation. The objective of the artificial environment is to extend the current multi-body simulation by the possibility to add arbitrary environments and to control the robot via continuous input commands in the simulation, as for example with a Joystick. The vision software module is integrated in the simulation framework to approximate the environment as in the real experiments. That way, the interaction between walking control and vision system can be tested in simulation instead of in time-consuming experiments [148]. Although the combination of artificial environments with the vision system has already been achieved [129], their integration in the simulation framework still has to be finished.

Outlook for Improving Current Methodology

Dynamics

The presented methods for motion generation improve the motion of the robot mainly by using kinematic models. This allows for exploiting the kinematic capacities of the robot. The dynamics of the system are only taken as constraints into account but not as optimization potential. When executing fast motions, the dynamics of the system become more important. Future research should also included the overall multi-body dynamics in the methods for motion generation.

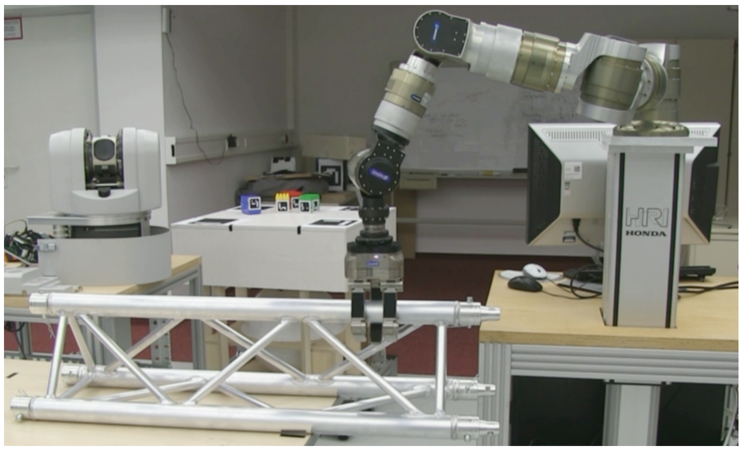


Figure 7.2: Experiments at the Honda Research Institute Europe showing the applicability of the concept in industrial applications [149].

Time Dependency

The methods presented in this thesis always rely on heuristics to determine the time dependency of the motions. Furthermore, the timing is fixed with the calculation of the foothold positions without knowledge about the robot's motion. Nevertheless, the timing of the trajectories has large influence on the motions. Therefore, optimized timing promise large improvement potential, especially when including the dynamics of the robot into the optimization methods, which can still be exploited.

Outside Laboratories and Error-Handling

The current methods have been validated in experiments in the laboratory. Only when evaluating the methods performance outside the laboratory current limitations can be identified. Furthermore, outside laboratories there will always exist scenarios, such as large disturbances, which can not be handled by the robot. Although, the current control architecture includes error handling for system failures, still it has to be extended. For example, falling of the robot has to be taken into account not only by shutting down the control system but by implementing falling strategies.

Human-Robot Interaction

Autonomous robots have to interact with human users. They have to autonomously perform tasks, while respecting the user's input. Even in the experiments in the laboratory, the question arises to which extend the robot has to follow the user's input to allow for an intuitive handling or when it has to search its own path. This question should be a general research question, which is important when trying to bring autonomous robots in real applications.

Supporting with Arms

This thesis focuses on walking of humanoids. But it does not take the arms as additional supporting points into account. The use of arms could largely improve the robot's robustness and versitality in highly complex environments. They enlarge the robot's action radius and allow for using additional areas as, for example walls, to stabilize the robot. The current approach for motion generation can still be applied to this problem, nevertheless the methods have to be extended. In colloboration with Benjamin Kammermeier and Dr.-Ing. Michael Gienger a framework for sequence planning was developed which could also be applied to bipedal walking taking arms into account [142]. First simulations show promising results for a humanoid traversing a bridge while using its arms.

What Is Optimal Bipedal Locomotion?

As stated in the discussion, the methods presented in this thesis allow for complex motions; but they still rely on manual tuning of cost functions. An experienced developer decides by designing functions and choosing weights of the costs what optimal bipedal walking means. When bringing the robot in real environments out of the laboratory, the environments and, therefore also the cost function design, will become more complex. One option would be to use learning methods to choose the parameters of the cost functions. In this context, the proposed *Artificial Environment* could not only be used for evaluation of new methods, but also to produce data sets for the learning methods.

Generalization and Industrial Application

As presented in this thesis, the discussed approach is not limited to bipedal locomotion. In collaboration with Dr.-Ing. Michael Gienger, Benjamin Kammermaier and Michael Meissner the approach was extended to industrial robots. Dr.-Ing. Michael Gienger, Benjamin Kammermaier and Michael Meissner could show simulations and experiments at Honda Research Institute Europe which proves the applicability of the concept [117, 149]. Further work, has to combine the sequence planning with the motion generation. Additionally when executing long motion sequences, small errors accumulate to large deviations to the ideal planned motion. Reactive acting methods, as presented in Chapter 5 could be one way to solve this issue without large computational effort.

Appendix A

Gradients for Optimization

A.1 Gradients for Optimization of Redundancy

Considering the function $\boldsymbol{f} = \boldsymbol{f}(\boldsymbol{q}, \boldsymbol{u}, p) = \dot{\boldsymbol{q}}$ as shown in (2.1) and the Moore-Penrose-Pseudo-Inverse $\boldsymbol{J}^{\#} = \boldsymbol{J}^T(\boldsymbol{J}\boldsymbol{J}^T)^{-1}$, the analytical gradients necessary for redundancy optimization can be formulated as follows:

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{q}} = \frac{\partial \dot{\boldsymbol{q}}}{\partial \boldsymbol{q}} = \frac{\partial \boldsymbol{J}^{\#}}{\partial \boldsymbol{q}} \dot{\boldsymbol{w}} - (\frac{\partial \boldsymbol{J}^{\#}}{\partial \boldsymbol{q}} \boldsymbol{J} - \boldsymbol{J}^{\#} \frac{\partial \boldsymbol{J}}{\partial \boldsymbol{q}}) \boldsymbol{u} \tag{A.1}$$

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}} = \boldsymbol{I} - \boldsymbol{J}^{\#} \boldsymbol{J} \tag{A.2}$$

$$\frac{\partial \boldsymbol{J}^{\#}}{\partial \boldsymbol{q}} = \frac{\partial \boldsymbol{J}^T}{\partial \boldsymbol{q}} \left(\boldsymbol{J}\boldsymbol{J}^T\right)^{-1} + \boldsymbol{J}^T \left[- \left(\boldsymbol{J}\boldsymbol{J}^T\right)^{-1} \left(\frac{\partial \boldsymbol{J}}{\partial \boldsymbol{q}} \boldsymbol{J}^T + \boldsymbol{J} \frac{\partial \boldsymbol{J}^T}{\partial \boldsymbol{q}}\right) \left(\boldsymbol{J}\boldsymbol{J}^T\right)^{-1}\right] \tag{A.3}$$

$$\frac{\partial L_{cmf}}{\partial \boldsymbol{q}} = 2\left(\boldsymbol{q} - \boldsymbol{q}_{cmf}\right) \tag{A.4}$$

$$\frac{\partial L_{vel}}{\partial \boldsymbol{q}} = 2\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{q}}\right)^T \dot{\boldsymbol{q}} \tag{A.5}$$

$$\frac{\partial L_{vel}}{\partial \boldsymbol{u}} = 2\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}\right)^T \dot{\boldsymbol{q}} \tag{A.6}$$

with $\boldsymbol{q}_{cmf}$ a user defined comfort pose. The gradients to the cost functions of collision and joint limit avoidance are detailed in Buschmann et al. [16] and Schuetz et al. [94], respectively.

A.2 Gradients for Parameter Optimization

In the context of real-time application of the parameter optimization described in Section 5.3, the calculation of the gradient $\nabla_{\boldsymbol{p}_{wp}} L_{opt}$ is challenging: for a small search space of $\boldsymbol{p}_{opt}$ it may be advantageous to determine $\nabla_{\boldsymbol{p}_{wp}} L_{opt}$ numerically. The analytical derivation may take more computational time than the additional integrations for the numerical derivation with finite differences. With growing search space the use of an analytical gradient becomes more significant.

Analytical Gradient

The goal function (5.8) consists of a sum of functions which depend on w and q. Therefore, w and q have to be differentiated w.r.t. $\boldsymbol{p}_{opt}$ at each time-step of the numerical integration. Similar to Toussaint et al. [106], differentiation of the discrete form of (5.9)[1]

$$\begin{aligned}\begin{pmatrix} q \\ w \end{pmatrix}_{i+1} &= \\ \begin{pmatrix} q \\ w \end{pmatrix}_{i} &+ \Delta t_i f(q_i, w_i, w_{d_i}(p_{opt}), \dot{w}_{d_i}(p_{opt})), \\ \begin{pmatrix} q \\ w \end{pmatrix}_{0} &= \begin{pmatrix} q_0 \\ w_0 \end{pmatrix}.\end{aligned} \tag{A.7}$$

delivers the recursive formula for the gradients

$$\begin{aligned}\begin{pmatrix} \nabla_{p_{opt}} q \\ \nabla_{p_{opt}} w \end{pmatrix}_{i+1} &= \begin{pmatrix} \nabla_{p_{opt}} q \\ \nabla_{p_{opt}} w \end{pmatrix}_{i} \\ &+ \Delta t_i \nabla_q f_i \nabla_{p_{opt}} q_i \\ &+ \Delta t_i \nabla_w f_i \nabla_{p_{opt}} w_i \\ &+ \Delta t_i \nabla_{w_0} f_i \nabla_{p_{opt}} w_{0_i} \\ &+ \Delta t_i \nabla_{\dot{w}_0} f_i \nabla_{p_{opt}} \dot{w}_{0_i}, \\ \begin{pmatrix} \nabla_{p_{opt}} q \\ \nabla_{p_{opt}} w \end{pmatrix}_{0} &= \begin{pmatrix} 0 \\ 0 \end{pmatrix}.\end{aligned} \tag{A.8}$$

The partial derivatives of f can be calculated analytically. They include derivatives of J_w, $J_w^{\#}$ and ∇H_{RCA}. The foot trajectories and the CoG trajectories are the p_{opt}-dependent components of w_d. The boundary problem of the CoG movement is solved applying the collocation method as described in Buschmann et al. [14]. This generates the CoG horizontal reference trajectories. Since its analytical differentiation is computationally expensive and the calculation time of the horizontal CoG reference trajectories is negligible compared to the integration of the robot's state it is decided to use numerical gradients for the corresponding derivatives. The derivation of the foot trajectories, as the splines are linearly dependent of p_{opt}, is straight forward.[2]

[1]For the sake of simplicity, $v(t_i)$ is replaced by v_i.

[2]Demmeler [138] describes the gradient calculation in more detail.

Appendix B

Inverse Kinematics for Simplified 2D Model of Lola

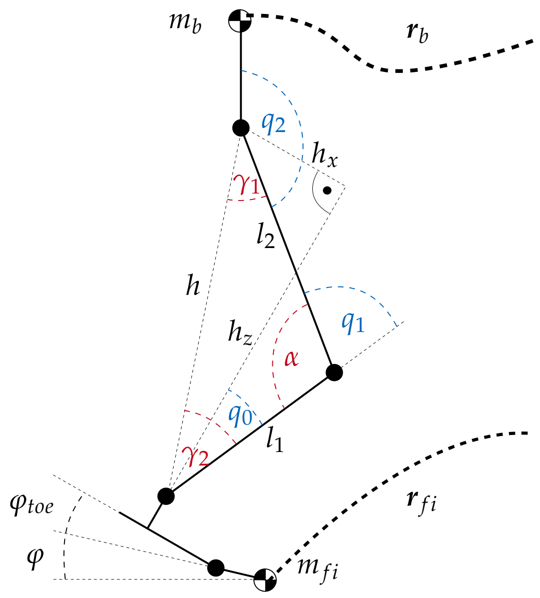


Figure B.1: 2D kinematic model approximating robot's full kinematics. Input values in black, auxiliary joint angles in red, output joint angles in blue.

The relative distances h_x and h_z can be obtained from robot dimensions and the trajectories for feet and body. The law of cosines yields the angles α, γ_1 and γ_2 in the triangle between hip, knee and ankle joint (B.1)-(B.3).

$$\alpha = acos\left(\frac{l_1^2 + l_2^2 - h^2}{2l_1 l_2}\right) \tag{B.1}$$

$$\gamma_1 = acos\left(\frac{h^2 + l_2^2 - l_1^2}{2hl_2}\right) \tag{B.2}$$

$$\gamma_2 = acos\left(\frac{l_1^2 + h^2 - l_2^2}{2l_1 h}\right) \tag{B.3}$$

Together with the sign of h_x, the distance from the origin in the ankle joint to the hip joint projected in x direction of the mid foot, the three leg joint angles can be derived from the angles in (B.1)-(B.3) and some basic trigonometry (B.4).

$$\begin{aligned} h_x > 0 \quad & q_0 = q_{ankle} = -\gamma_2 - acos\left(\frac{h_z}{h}\right) \\ & q_1 = q_{knee} = 180^\circ - \alpha \\ & q_2 = q_{hip} = 180^\circ - \gamma_1 + atan\left(\|\frac{h_x}{h_z}\| \right) \end{aligned} \tag{B.4}$$

References

[1] A. Liegeois. “Automatic Supervisory Control of the Configuration and Behavior of Multibody Mechanisms”. In: *IEEE Transactions on Systems* 12 (1977), pp. 868–871.

[2] Adams, E. *Fundamentals of Game Design*. Ed. by Johnson, K. 3rd ed. 2013, p. 576.

[3] Arbulu, M., Kheddar, A., and Yoshida, E. “An approach of generic solution for humanoid stepping over motion”. In: *IEEE-RAS International Conference on Humanoid Robots*. 2010.

[4] Ayaz, Y., Munawar, K., Bilal Malik, M., Konno, A., and Uchiyama, M. “Human-like approach to footstep planning among obstacles for humanoid robots”. In: *IEEE International Conference on Intelligent Robots and Systems*. 2006.

[5] Bae, H., Lee, I., Jung, T., and Oh, J. H. “Walking-wheeling dual mode strategy for humanoid robot, DRC-HUBO+”. In: *IEEE International Conference on Intelligent Robots and Systems*. 2016.

[6] Banerjee, B. and Chandrasekaran, B. “A Framework of Voronoi Diagram for Planning Multiple Paths in Free Space”. In: *Journal of Experimental & Theoretical Artificial Intelligence* 25.4 (2013), pp. 457–475.

[7] Baudouin, L., Perrin, N., Moulard, T., Lamiraux, F., Stasse, O., and Yoshida, E. “Real-time replanning using 3D environment for humanoid robot”. In: *IEEE-RAS International Conference on Humanoid Robots*. 2011.

[8] Behnisch, M., Haschke, R., Ritter, H., and Gienger, M. “Deformable trees - exploiting local obstacle avoidance”. In: *IEEE-RAS International Conference on Humanoid Robots*. 2011.

[9] Betts, J. T. “Survey of Numerical Methods for Trajectory Optimization”. In: *Journal of Guidance, Control, and Dynamics* 21.2 (1998), pp. 193–207.

[10] Bocek, M. “Conjugate gradient algorithm for optimal control problems with parameters”. In: *Kybernetika* 16.5 (1980), pp. 454–461.

[11] Buschmann, T., Lohmeier, S., Ulbrich, H., and Pfeiffer, F. “Optimization based gait pattern generation for a biped robot”. In: *IEEE-RAS International Conference on Humanoid Robots*. 2005.

[12] Buschmann, T. “Simulation and Control of Biped Walking Robots”. PhD thesis. Technical University of Munich, 2010.

[13] Buschmann, T., Favot, V., Lohmeier, S., Schwienbacher, M., and Ulbrich, H. “Experiments in Fast Biped Walking”. In: *IEEE International Conference on Mechatronics*. 2011.

[14] Buschmann, T., Lohmeier, S., Bachmayer, M., Ulbrich, H., and Pfeiffer, F. “A collocation method for real-time walking pattern generation”. In: *IEEE-RAS International Conference on Humanoid Robots*. 2007.

[15] Buschmann, T., Lohmeier, S., Schwienbacher, M., Favot, V., Ulbrich, H., Hundelshausen, F. von, Rohe, G., and Wuensche, H.-J. "Walking in Unknown Environments - A Step Towards More Autonomy". In: *IEEE-RAS International Conference on Humanoid Robots*. 2010.

[16] Buschmann, T., Lohmeier, S., and Ulbrich, H. "Biped walking control based on hybrid position/force control". In: *IEEE-RSJ International Conference on Intelligent Robots and Systems*. 2009.

[17] Chestnutt, J., Michel, P., Nishiwaki, K., Kuffner, J., and Kagami, S. "An intelligent joystick for biped control". In: *IEEE International Conference on Robotics and Automation*. 2006.

[18] Chestnutt, J. "Navigation Planning for Legged Robots". PhD thesis. Carnegie Mellon University, 2007.

[19] Chestnutt, J., Nishiwaki, K., Kuffner, J., and Kagami, S. "An adaptive action model for legged navigation planning". In: *IEEE-RAS International Conference on Humanoid Robots*. 2007.

[20] Chestnutt, J., Takaoka, Y., Suga, K., Nishiwaki, K., Kuffner, J., and Kagami, S. "Biped Navigation in Rough Environments Using On-board Sensing". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2009.

[21] Chestnutt, J., Takaokaz, Y., Doiz, M., Sugaz, K., and Kagamiy, S. "Safe adjustment regions for legged locomotion paths". In: *IEEE-RAS International Conference on Humanoid Robots*. 2010.

[22] Chevallereau, C. and Aoustin, Y. "Self-Stabilization of 3D Walking via Vertical Oscillations of the Hip". In: *IEEE International Conference on Robotics and Automation*. 2015.

[23] Chitta, S., Sucan, I., and Cousins, S. "MoveIt! [ROS Topics]". In: *IEEE Robotics & Automation Magazine* 19.1 (2012), pp. 18–19.

[24] Choset, H., Lynch, K. M., Hutchinson, S., Kantor, G. A., Burgard, W., Kavraki, L. E., and Thrun, S. *Principles of Robot Motion - Theory, Algorithms, and Implementations*. 2005.

[25] Cupec, R. and Schmidt, G. "An approach to environment modelling for biped walking robots". In: *IEEE-RSJ International Conference on Intelligent Robots and Systems*. 2005.

[26] Deits, R. and Tedrake, R. "Footstep planning on uneven terrain with mixed-integer convex optimization". In: *IEEE-RAS International Conference on Humanoid Robots*. 2014.

[27] Deits, R. and Tedrake, R. "Computing Large Convex Regions of Obstacle-Free Space Through Semidefinite Programming". In: *Algorithmic Foundations of Robotics XI*. Springer International Publishing, 2015, pp. 109–124.

[28] Englsberger, J., Ott, C., Roa, M. A., and Hirzinger, G. "Bipedal Walking Control Based on Capture Point Dynamics". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2011.

[29] Englsberger, J., Werner, A., Ott, C., Henze, B., Roa, M. A., Garofalo, G., Burger, R., Beyer, A., Eiberger, O., Schmid, K., and Albu-Schäffer, A. "Overview of the torque-controlled humanoid robot TORO". In: *IEEE-RAS International Conference on Humanoid Robots*. 2015.

[30] Ersson, T. and Xiaoming Hu. “Path planning and navigation of mobile robots in unknown environments”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2001.

[31] Ewald, A., Mayet, J., Buschmann, T., and Ulbrich, H. “Generating Smooth Trajectories Free from Overshoot for Humanoid Robot Walking Pattern Replanning”. In: *Autonomous Mobile Systems*. Informatik aktuell. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[32] Favot, V. “Hierarchical Joint Control of Humanoid Robots”. PhD thesis. Technical University of Munich, 2016.

[33] Garimort, J., Hornung, A., and Bennewitz, M. “Humanoid navigation with dynamic footstep plans”. In: *IEEE International Conference on Robotics and Automation*. 2011.

[34] Ge, S. and Cui, Y. “Dynamic Motion Planning for Mobile Robots Using Potential Field Methods”. In: *Autonomous Robots* 13.3 (2002), pp. 207–222.

[35] Gienger, M. “Entwurf und Realisierung einer zweibeinigen Laufmaschine”. PhD thesis. TU München, 2004.

[36] Gienger, M., Toussaint, M., Jetchev, N., Bendig, A., and Goerick, C. “Optimization of Fluent Approach and Grasp Motions”. In: *IEEE-RAS International Conference on Humanoid Robots*. 2008.

[37] Graichen, K. *Methoden der Optimierung und Otimalen Steuerung*. Universität Ulm, 2013.

[38] Guan, Y., Yokoi, K., and Tanie, K. “Stepping Over Obstacles with Humanoid Robots”. In: *IEEE Transactions on Robotics* 22.5 (2006), pp. 958–973.

[39] Gutmann, J.-S., Fukuchi, M., and Fujita, M. “3D Perception and Environment Map Generation for Humanoid Robot Navigation”. In: *The International Journal of Robotics Research* 27.10 (2008), pp. 1117–1134.

[40] Hanan Samet. *The Design and Analysis of Spatial Data Structures*. Ed. by Harrison, M. A. Maryland: Addison-Wesley Publishing Company, Inc., 1990.

[41] Herdt, A., Perrin, N., and Wieber, P. B. “LMPC based online generation of more efficient walking motions”. In: *IEEE-RAS International Conference on Humanoid Robots*. 2012.

[42] Hildebrandt, A.-c., Wahrmann, D., Wittmann, R., Rixen, D., and Buschmann, T. “Real-Time Pattern Generation Among Obstacles for Biped Robots”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2015.

[43] Hirai, K., Hirose, M., Haikawa, Y., and Takenaka, T. “The development of Honda humanoid robot”. In: *IEEE International Conference on Robotics and Automation*. 1998.

[44] Hong, Y.-D. and Lee, K.-B. “Stable Walking of Humanoid Robots Using Vertical Center of Mass and Foot Motions by an Evolutionary Optimized Central Pattern Generator”. In: *International Journal of Advanced Robotic Systems* 13.1 (2016), p. 27.

[45] Hornung, A. and Bennewitz, M. “Adaptive Level-of-Detail Planning for Efficient Humanoid Navigation”. In: *IEEE International Conference on Robotics and Automation*. 2012.

[46] Hornung, A., Dornbush, A., Likhachev, M., and Bennewitz, M. “Anytime search-based footstep planning with suboptimality bounds”. In: *IEEE-RAS International Conference on Humanoid Robots*. 2012.

[47] Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C., and Burgard, W. "OctoMap: An efficient probabilistic 3D mapping framework based on octrees". In: *Autonomous Robots* 34.3 (2013), pp. 189–206.

[48] Ioan A. Sucan, Mark Moll, L. E. K. "The Open Motion Planning Library". In: *IEEE Robotics & Automation Magazine* 12.December (2012), pp. 72–82.

[49] J. Nocedal and S. J. Wright. *Numerical Optimization*. 2nd ed. Springer Series in Operations Research and Financial Engineering. New York: Springer New York, 2006.

[50] Kaelbling, L. P. and Lozano-Perez, T. "Hierarchical task and motion planning in the now". In: *IEEE International Conference on Robotics and Automation*. 2011.

[51] Kagami, S., Nishiwaki, K., Kuffner, J., Okada, K., Inaba, M., and Inoue, H. "Vision-based 2.5D terrain modeling for humanoid locomotion". In: *IEEE International Conference on Robotics and Automation*. 2003.

[52] Kajita, S., Kanehiro, F., Kaneko, K., Fujiwara, K., Harada, K., Yokoi, K., and Hirukawa, H. "Biped Walking Pattern Generation by Using Preview Control of Zero-Moment Point". In: *IEEE International Conference on Robotics and Automation*. 2003.

[53] Kaneko, K., Kanehiro, F., Kajita, S., Hirukawa, H., Kawasaki, T., Hirata, M., Akachi, K., and Isozumi, T. "Humanoid robot HRP-2". In: *IEEE International Conference on Robotics and Automation*. 2004.

[54] Karkowski, P. and Bennewitz, M. "Real-Time Footstep Planning Using a Geometric Approach". In: *IEEE International Conference on Robotics and Automation*. 2016.

[55] Kavraki, L. E., Svestka, P., Latombe, J. C., and Overmars, M. H. "Probabilistic roadmaps for path planning in high-dimensional configuration spaces". In: *IEEE Transactions on Robotics and Automation* 12.4 (1996), pp. 566–580.

[56] Kim, J.-O. and Khosla, P. K. "Real-time obstacle avoidance using harmonic potential functions". In: *IEEE Transaction on Robotics and Automation* 8.3 (1992), pp. 338–349.

[57] Klein, C. A. and Huang, C.-H. "Review of pseudoinverse control for use with kinematically redundant manipulators". In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-13.2 (1983), pp. 245–250.

[58] Koch, K. H., Mombaur, K., Stasse, O., Soueres, P., Koch, K. H., Mombaur, K., Stasse, O., and Optimization, P. S. "Optimization based exploitation of the ankle elasticity of HRP-2 for overstepping large obstacles". In: *IEEE-RAS International Conference on Humanoid Robots*. 2014.

[59] Koolen, T., Posa, M., and Tedrake, R. "Balance control using center of mass height variation: limitations imposed by unilateral contact". In: *IEEE-RAS International Conference on Humanoid Robots*. 2016.

[60] Kraft, D. "Algorithm 733; TOMP - Fortran modules for optimal control calculations". In: *ACM Transactions on Mathematical Software* 20.3 (1994), pp. 262–281.

[61] Kuindersma, S., Deits, R., Fallon, M., Valenzuela, A., Dai, H., Permenter, F., Koolen, T., Marion, P., and Tedrake, R. "Optimization-based locomotion planning, estimation, and control design for Atlas". In: *Autonomous Robots* 40.3 (2016), pp. 429–455.

[62] LaValle, S. M. *Rapidly-Exploring Random Trees: A New Tool for Path Planning*. Tech. rep. Computer Science Dept, Iowa State University, 1998.

[63] LaValle, S. M. *Planning Algorithms*. Cambridge: Cambridge University Press, 2006.

[64] Li, Z., Vanderborght, B., Tsagarakis, N. G., and Caldwell, D. G. “Fast bipedal walk using large strides by modulating hip posture and toe-heel motion”. In: *IEEE International Conference on Robotics and Biomimetics*. 2010.

[65] Liu, H., Liu, W., and Latecki, L. J. “Convex shape decomposition”. In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2010).

[66] Lohmeier, S., Loffler, K., Gienger, M., Ulbrich, H., and Pfeiffer, F. “Computer system and control of biped "Johnnie"”. In: *IEEE International Conference on Robotics and Automation*. 2004.

[67] Lohmeier, S. “Design and Realization of a Humanoid Robot for Fast and Autonomous Bipedal Locomotion”. PhD thesis. TU München, 2010.

[68] Maier, D., Lutz, C., and Bennewitz, M. “Integrated perception, mapping, and footstep planning for humanoid navigation among 3D obstacles”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2013.

[69] Maier, D., Stachniss, C., and Bennewitz, M. “Vision-Based Humanoid Navigation Using Self-Supervised Obstacle Detection”. In: *International Journal of Humanoid Robotics* 10.2 (2013).

[70] Mansard, N., Remazeilles, A., and Chaumette, F. “Continuity of Varying-Feature-Set Control Laws”. In: *IEEE Transactions on Automatic Control* 54.11 (2009), pp. 2493–2505.

[71] Mayr, J., Gattringer, H., and Bremer, H. “A Bipedal Walking Pattern Generator that Considers Multi-Body Dynamics by Angular Momentum Estimation”. In: *IEEE-RAS International Conference on Humanoid Robots*. 2012.

[72] Michel, P. and Chestnutt, J. “GPU-accelerated real-time 3D tracking for humanoid locomotion and stair climbing”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2007.

[73] Miura, K., Morisawa, M., Kanehiro, F., Kajita, S., Kaneko, K., and Yokoi, K. “Human-like walking with toe supporting for humanoids”. In: *IEEE International Conference on Intelligent Robots and Systems*. 2011.

[74] Mombaur, K., Kheddar, A., Harada, K., Buschmann, T., and Atkeson, C. “Model-based optimization for robotics”. In: *IEEE Robotics and Automation Magazine* 21.3 (2014), pp. 24–26.

[75] Nakamura, Y. *Advanced Robotics - Redundancy and Optimization*. Ed. by Addison-Wesley. 1991.

[76] Nakamura, Y. and Hanafusa, H. “Optimal Redundancy Control of Robot Manipulators”. In: *The International Journal of Robotics Research* 6.1 (Mar. 1987), pp. 32–42.

[77] Naveau, M., Kudruss, M., Stasse, O., Kirches, C., Mombaur, K., and Souères, P. “A Reactive Walking Pattern Generator Based on Nonlinear Model Predictive Control”. In: *IEEE Robotis and Automation Letters* 2.1 (2017), pp. 10–17.

[78] Nelson, G., Saunders, A., Neville, N., Swilling, B., Bondaryk, J., Billings, D., Lee, C., Playter, R., and Raibert, M. “PETMAN: A Humanoid Robot for Testing Chemical Protective Clothing”. In: *Journal of the Robotics Society of Japan* 30.4 (2012), pp. 372–377.

[79] Nishiwaki, K. “Online design of torso height trajectories for walking patterns that takes future kinematic limits into consideration”. In: *International Conference on Control, Automation and Systems*. 2011.

[80] Nishiwaki, K., Chestnutt, J., and Kagami, S. “Autonomous navigation of a humanoid robot over unknown rough terrain using a laser range sensor”. In: *The International Journal of Robotics Research* 31.11 (2012), pp. 1251–1262.

[81] Nishiwaki, K., Chestnutt, J., and Kagami, S. “Planning and Control of a Humanoid Robot for Navigation on Uneven Multi-scale Terrain”. In: *Springer Tracts in Advanced Robotics*. Vol. 79. Berlin, Heidelberg, 2014, pp. 401–415.

[82] Nishiwaki, K., Kagami, S., Kuffner, J., Inaba, M., and Inoue, H. “Online humanoid walking control system and a moving goal tracking experiment”. In: *IEEE International Conference on Robotics and Automation*. 2003.

[83] Okada, K., Kagami, S., Inaba, M., and Inoue, H. “Plane segment finder: algorithm, implementation and applications”. In: *IEEE International Conference on Robotics and Automation*. 2001.

[84] Ott, C., Dietrich, A., and Albu-Schäffer, A. “Prioritized multi-task compliance control of redundant manipulators”. In: *Automatica* 53 (2015), pp. 416–423.

[85] Pan, J., Chitta, S., and Manocha, D. “FCL: A general purpose library for collision and proximity queries”. In: *IEEE International Conference on Robotics and Automation*. 2012.

[86] Park, C.-S., Ha, T., Kim, J., and Choi, C.-H. “Trajectory generation and control for a biped robot walking upstairs”. In: *International Journal of Control, Automation and Systems* 8.2 (2010), pp. 339–351.

[87] Park, I. W., Kim, J. Y., Lee, J., and Oh, J. H. “Mechanical design of humanoid robot platform KHR-3 (KAIST humanoid robot - 3: HUBO)”. In: *IEEE-RAS International Conference on Humanoid Robots*. 2005.

[88] Perrin, N., Stasse, O., Baudouin, L., Lamiraux, F., and Yoshida, E. “Fast Humanoid Robot Collision-Free Footstep Planning Using Swept Volume Approximations”. In: *IEEE Transactions on Robotics* 28.2 (2012), pp. 427–439.

[89] Quinlan, S. and Khatib, O. “Elastic bands: connecting path planning and control”. In: *IEEE International Conference on Robotics and Automation*. 1993.

[90] Sabe, K., Fukuchi, M., Gutmann, J.-S., Ohashi, T., Kawamoto, K., and Yoshigahara, T. “Obstacle avoidance and path planning for humanoid robots using stereo vision”. In: *IEEE International Conference on Robotics and Automation*. 2004.

[91] Sacerdoti, E. “Planning in a Hierarchy of Abstract Spaces”. In: *Artificial Intelligence* 5.1974 (1974), pp. 115–135.

[92] Sarmiento, A., Murrieta-Cid, R., and Hutchinson, S. “A sample-based convex cover for rapidly finding an object in a 3-D environment”. In: *IEEE International Conference on Robotics and Automation*. 2005.

[93] Schuetz, C. “Trajectory Planning for Redundant Manipulators”. PhD thesis. Technical University of Munich, 2017.

[94] Schuetz, C., Buschmann, T., Baur, J., Pfaff, J., and Ulbrich, H. “Predictive Online Inverse Kinematics for Redundant Manipulators”. In: *IEEE International Conference on Robotics and Automation*. 2014.

[95] Schulman, J., Ho, J., Lee, A., Awwal, I., Bradlow, H., and Abbeel, P. “Finding locally optimal, collision-free trajectories with sequential convex optimization”. In: *Robotics: Science and Systems*. 2013.

[96] Schwienbacher, M. "Vertical Angular Momentum Minimization for Biped Robots with Kinematically Redundant Joints". In: *International Congress of Theoretical and Applied Mechanics* (2012).

[97] Schwienbacher, M. "Efficient Algorithms for Biped Robots". PhD thesis. Technical University of Munich, 2014.

[98] Schwienbacher, M., Buschmann, T., Lohmeier, S., Favot, V., and Ulbrich, H. "Self-Collision Avoidance and Angular Momentum Compensation for a Biped Humanoid Robot". In: *IEEE International Conference on Robotics and Automation*. 2011.

[99] Shafii, N., Lau, N., and Reis, L. P. "Learning a fast walk based on ZMP control and hip height movement". In: *IEEE International Conference on Autonomous Robot Systems and Competitions*. 2014.

[100] Siciliano, B., Sciavicco, L., Villani, L., and Oriolo, G. *Robotics*. Advanced Textbooks in Control and Signal Processing. London: Springer London, 2009.

[101] Stasse, O., Verrelst, B., Vanderborght, B., and Yokoi, K. "Strategies for Humanoid Robots to Dynamically Walk Over Large Obstacles". In: *IEEE Transactions on Robotics* 25.4 (2009), pp. 960–967.

[102] Stumpf, A., Kohlbrecher, S., Conner, D. C., and Stryk, O. V. "Supervised Footstep Planning for Humanoid Robots in Rough Terrain Tasks using a Black Box Walking Controller". In: *IEEE-RAS International Conference on Humanoid Robots*. 2014.

[103] Tajima, R., Honda, D., and Suga, K. "Fast running experiments involving a humanoid robot". In: *IEEE International Conference on Robotics and Automation*. 2009.

[104] Takenaka, T., Matsumoto, T., and Yoshiike, T. "Real time motion generation and control for biped robot - 1st report: Walking gait pattern generation". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2009.

[105] Takenaka, T., Matsumoto, T., and Yoshiike, T. "Real time motion generation and control for biped robot - 3rd report: Dynamics error compensation". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2009.

[106] Toussaint, M., Gienger, M., and Goerick, C. "Optimization of sequential attractor-based movement for compact behaviour generation". In: *IEEE-RAS International Conference on Humanoid Robots*. 2007.

[107] Urata, J., Nakanishi, Y., Okada, K., and Inaba, M. "Design of high torque and high speed leg module for high power humanoid". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2010.

[108] Urata, J., Nishiwaki, K., Nakanishi, Y., Okada, K., Kagami, S., and Inaba, M. "Online Decision of Foot Placement Using Singular LQ Preview Regulation". In: *IEEE-RAS International Conference on Humanoid Robots*. 2011.

[109] Verrelst, B., Stasse, O., Yokoi, K., and Vanderborght, B. "Dynamically Stepping Over Obstacles by the Humanoid Robot HRP-2". In: *IEEE-RAS International Conference on Humanoid Robots*. 2006.

[110] Wahrmann, D. "Environment Modeling for Bipedal Robots (submitted)". PhD thesis. Technical University of Munich, 2017.

[111] Wang, X. W. X., Seet, G., Lau, M., Low, E., and Tan, K. "Exploiting force feedback in pilot training and control of an\nunderwater robotics vehicle: an implementation in LabVIEW". In: *OCEANS 2000 MTS/IEEE Conference and Exhibition.* 2000.

[112] Werner, A., Henze, B., Rodriguez, D. A., Gabaret, J., Porges, O., and Roa, A. “Multi-Contact Planning and Control for a Torque-Controlled Humanoid Robot”. In: *IEEE International Conference on Intelligent Robots and Systems*. 2016.

[113] Whitney, D. “Resolved motion rate control of manipulators and human prostheses”. In: *IEEE Transactions on Man-Machine Systems* 10.2 (1969), pp. 47–53.

[114] Wittmann, R. “Robust Walking Robots in Unknown Environments (submitted)”. PhD thesis. Technical University of Munich, 2017.

[115] Yisheng Guan, Yokoi, K., and Tanie, K. “Feasibility: Can Humanoid Robots Overcome Given Obstacles?” In: *IEEE International Conference on Robotics and Automation*. 2005.

[116] Zucker, M., Ratliff, N., Dragan, a. D., Pivtoraiko, M., Klingensmith, M., Dellin, C. M., Bagnell, J. a., and Srinivasa, S. S. “CHOMP: Covariant Hamiltonian optimization for motion planning”. In: *The International Journal of Robotics Research* 32.9-10 (2013), pp. 1164–1193.

Author's Publications

[117] Gienger, M., Ruiken, D., Bates, T., Regaieg, M., Meißner, M., Kober, J., Seiwald, P., and Hildebrandt, A.-C. "Human-Robot Cooperative Object Manipulation with Contact Changes (submitted)". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2018.

[118] Hildebrandt, A.-C., Demmeler, M., Wittmann, R., Wahrmann, D., Sygulla, F., Rixen, D., and Buschmann, T. "Real-Time Predictive Kinematic Evaluation and Optimization for Biped Robots". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2016.

[119] Hildebrandt, A.-C., Klischat, M., Wahrmann, D., Wittmann, R., Sygulla, F., Seiwald, P., Rixen, D., and Buschmann, T. "Real-Time Path Planning in Unknown Environments for Bipedal Robots". In: *IEEE Robotics and Automation Letters* 2.4 (2017), pp. 1856–1863.

[120] Hildebrandt, A.-C., Ritt, K., Wahrmann, D., Wittmann, R., Sygulla, F., Seiwald, P., Rixen, D., and Buschmann, T. "Torso Height Optimization for Bipedal Locomotion (accepted)". In: *Journal of Advanced Robotic Systems* (2018).

[121] Hildebrandt, A.-C., Schuetz, C., Wahrmann, D., Wittmann, R., and Rixen, D. "A Flexible Robotic Framework for Autonomous Manufacturing Processes: Report from the European Robotics Challenge Stage 1". In: *IEEE International Conference on Autonomous Robot Systems and Competitions*. 2016.

[122] Hildebrandt, A.-C., Schwerd, S., Wittmann, R., Wahrmann, D., Sygulla, F., Seiwald, P., and Rixen, D. "Kinematic Optimization for Bipedal Robots (submitted)". In: *Journal of Autonomous Robots* (2017).

[123] Hildebrandt, A.-C., Wittmann, R., Sygulla, F., Wahrmann, D., Rixen, D., and Buschmann, T. "Versatile and Robust Bipedal Walking in Unknown Environments (submitted)". In: *Journal of Autonomous Robots* (2017).

[124] Hildebrandt, A.-C., Wittmann, R., Wahrmann, D., Ewald, A., and Buschmann, T. "Real-Time 3D Collision Avoidance for Biped Robots". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2014.

[125] Sygulla, F., Ellensohn, F., Hildebrandt, A.-C., Wahrmann, D., and Rixen, D. "A Flexible and Low-Cost Tactile Sensor for Robotic Applications". In: *IEEE International Conference on Advanced Intelligent Mechatronics*. 2017.

[126] Sygulla, F., Wittmann, R., Seiwald, P., Berninger, T., Hildebrandt, A.-C., Wahrmann, D., and Rixen, D. "An EtherCAT-Based Real-Time Control System Architecture for Humanoid Robots". In: *IEEE International Conference on Automation Science and Engineering* (2018).

[127] Wahrmann, D., Hildebrandt, A.-C., Schuetz, C., Wittmann, R., and Rixen, D. "An Autonomous and Flexible Robotic Framework for Logistics Applications". In: *Journal of Intelligent & Robotic Systems* December (2017).

[128] Wahrmann, D., Hildebrandt, A.-C., Wittmann, R., Rixen, D., and Buschmann, T. "Fast Object Approximation for Real-Time 3D Obstacle Avoidance with Biped Robots". In: *IEEE International Conference on Advanced Intelligent Mechatronics*. 2016.

[129] Wahrmann, D., Hildebrandt, A.-C., Wittmann, R., Sygulla, F., Seiwald, P., Rixen, D., and Buschmann, T. "Vision-Based 3D Modeling of Unknown Dynamic Environments for Real-Time Humanoid Navigation (submitted)". In: *International Journal of Humanoid Robotics* (2018).

[130] Wahrmann, D., Knopp, T., Wittmann, R., Hildebrandt, A.-C., Sygulla, F., Seiwald, P., Rixen, D., and Buschmann, T. "Modifying the Estimated Ground Height to Mitigate Error Effects on Bipedal Robot Walking". In: *IEEE International Conference on Advanced Intelligent Mechatronics*. 2017.

[131] Wahrmann, D., Wu, Y., Sygulla, F., Hildebrandt, A.-C., Wittmann, R., Seiwald, P., and Rixen, D. "Time-Variable, Event-Based Walking Control for Biped Robots". In: *International Journal of Advanced Robotic Systems* 15.2 (2018).

[132] Wittmann, R., Hildebrandt, A.-C., Ewald, A., and Buschmann, T. "An Estimation Model for Footstep Modifications of Biped Robots". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2014.

[133] Wittmann, R., Hildebrandt, A.-C., Wahrmann, D., Buschmann, T., and Rixen, D. "State Estimation for Biped Robots Using Multibody Dynamics". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2015.

[134] Wittmann, R., Hildebrandt, A.-C., Wahrmann, D., Rixen, D., and Buschmann, T. "Real-Time Nonlinear Model Predictive Footstep Optimization for Biped Robots". In: *IEEE-RAS International Conference on Humanoid Robots*. 2015.

[135] Wittmann, R., Hildebrandt, A.-C., Wahrmann, D., Sygulla, F., Rixen, D., and Buschmann, T. "Model-Based Predictive Bipedal Walking Stabilization". In: *IEEE-RAS International Conference on Humanoid Robots*. 2016.

Supervised Student Theses

[136] Blume, T. “Development and Kinematic Optimization of a Novel Collision-Free Swing Foot Trajectory for a Humanoid Robot”. Diploma Thesis. Technical University of Munich, 2015.

[137] Buttner, C. “Position and Velocity Estimation of Obstacles for Humanoid Robots”. Interdisciplinary Project. Technische Universität München, 2016.

[138] Demmeler, M. *Gradientenberechnungen für die Optimierung der Fußkurve*. Tech. rep. Munich: Technical University of Munich, 2016.

[139] Dünhuber, S. “Model Predictive Kinematics”. Bachelor Thesis. Technical University of Munich, 2017.

[140] Jeschek, L. “Robustes Gehen unter geometrischen Beschränkungen”. Master Thesis. Technical University of Munich, 2016.

[141] Jonas Wittmann. “Autonome Robotik - Reaktive Pfadplanung in der Landwirtschaftsrobotik”. Semester Thesis. Technical University of Munich, 2016.

[142] Kammermeier, B. “Multi-Pose Planning for Sequential Movements”. Master Thesis. Technical University of Munich, 2017.

[143] Kindsmüller, T. “Humanoide Schrittplanung in dynamischen Umgebungen”. Semester Thesis. Technical University of Munich, 2016.

[144] Kissel, M. “Optimierung von Robotertrajektorien”. Bachelor Thesis. Technical University of Munich, 2015.

[145] Klischat, M. “Autonome Robotik - Flexibles Laufen in unbekanntem Gelände”. Semester Thesis. Technical University of Munich, 2016.

[146] Klotz, J. “Entwicklung eines Schrittsequenzplaners für humanoide Roboter”. Master Thesis. Technical University of Munich, 2016.

[147] Kunze, L. “Adaptive Arbeitsraumdefinition für humanoide Roboter”. Bachelor Thesis. Technical University of Munich, 2017.

[148] Makhani, A. “Simulated point clouds for the evaluation of 3D Computer vision algorithms”. Interdisciplinary Project. Technische Universität München, 2016.

[149] Meissner, M. *Minimum-jerk Movement Optimization for Time dependent Sequential Motions*. Tech. rep. Munich/Offenbach: Honda Research Institute Europe, 2017.

[150] Sattler, M. “Schrittplanung für Zweibeiner”. Bachelor Thesis. Technical University of Munich, 2014.

[151] Scheuermann, T. “Trajektorienoptimierung für den humanoiden Roboter Lola”. Semester Thesis. Technical University of Munich, 2016.

[152] Simon Schwerd. “Modellprädiktive Kinematikplanung bei humanoiden Robotern”. Master Thesis. Technical University of Munich, 2016.

[153] Smith, T. and Hildebrandt, A.-C. *Dokumentation Minimalmodell - Nullraumschalten*. Tech. rep. Munich: Technical University of Munich, 2015.

[154] Wiedmeyer, W. “Entwicklung eines Schrittsequenzplaners für humanoide Roboter”. Semester Thesis. Technical University of Munich, 2015.