AUTONOMOUS ROBOT WALKING IN UNKNOWN SCENARIOS

Perception, Modeling and Robustness in Dynamic Environments

Daniel Wahrmann Lockhart

Fakultät für Maschinenwesen
Lehrstuhl für Angewandte Mechanik

Autonomous Robot Walking in Unknown Scenarios

Perception, Modeling and Robustness in Dynamic Environments

Daniel Wahrmann Lockhart

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. Prof. h.c. Dr. Dr. h.c. Ulrich Walter

Prüfer der Dissertation:

1. Prof. dr. ir. Daniel J. Rixen

2. Prof. Dr. Maren Bennewitz

Die Dissertation wurde am 24. Mai 2018 bei der Technischen Universität München eingereicht und durch die Fakultät für Maschinenwesen am 01. Oktober 2018 angenommen.

Abstract

A fundamental component of a robotic navigation platform is a perception system that can deal with unknown scenarios, providing an environment model on which a planning system can find a feasible path. However, existing perception systems are either too computationally expensive to handle dynamic environments during walking or make strong simplifying assumptions and cannot effectively deal with complex environments. This thesis aims to close the gap between complex perception systems and real-time autonomous navigation.
After an introduction to perception systems and their integration into a humanoid robot, a new 3D modeling strategy for unknown environments is introduced. Based on this environment representation, a new perception system is developed that is capable of processing unknown complex dynamic environments online. It is validated with a standard, inexpensive RGB-D sensor to make it compatible with other robotic systems. With this vision system, the full-size humanoid robot Lola is capable of navigating through previously unknown dynamic scenarios including moving persons, fixed obstacles and platforms. Efficient environment representations and inexpensive sensors, however, come at a cost: high inaccuracies in the provided information which may affect the robot's stability. In order to deal with this issue as well as to increase the robot's robustness against irregular terrains, a new walking controller is presented. The control optimizes the environment model based on the robot's dynamics and deals with unexpected contact situations using direct contact sensing.
Some algorithms presented in this work are additionally validated for their use on robot manipulators. Furthermore, the different tools developed during this project, including an augmented reality system for external visualization devices, are released as open source for the benefit of the community. The contributions of this thesis intend to bring biped robots one step closer to their application in real life scenarios.

Zusammenfassung

In mobilen Robotiksystemen ist die Perzeption unverzichtbar: sie nimmt unbekannten Umgebungen wahr und stellt ein Umgebungsmodel zum Navigieren bereit. Existierende Perzeptionssysteme sind allerdings noch zu rechenintensiv um Navigation in Echtzeit in dynamischen Umgebungen zu erlauben oder nutzen zu viele Vereinfachungen um komplexe Umgebungen akkurat zu modellieren. Ziel dieser Arbeit ist diese Lücke zwischen komplexen Perzeptionsalgorithmen und autonomer Echtzeitnavigation zu schließen.
Nach einer Einführung zu Perzeptionssystemen und ihrer Nutzung in der humanoiden Robotik, wird eine neue 3D Modellierungsstrategie für unbekannte Umgebungen eingeführt. Diese Strategie wurde in einem Perzeptionssystem umgesetzt. Damit ist es in der Lage, unbekannte und komplexe Umgebungen in Echtzeit zu modellieren. Um es kompatibel zu anderen Robotikanwendungen zu machen, nutzt es in der vorgestellten Implementierung einen günstigen Standard RGB-D Sensor. Das vorgestellte System erlaubt es dem humanoiden Roboter Lola sich in unbekannten und dynamischen Umgebungen, welche sich bewegende Menschen, Hindernisse und Plattformen beinhaltet, zu bewegen. Die Messungenauigkeiten der günstigen Sensoren und die effiziente Umgebungsdarstellung beeinflussen allerdings die Stabilität des Roboters. Um diesem Problem zu begegnen und zudem die Robustheit des Roboters in unebenem Gelände zu verbessern, wurde eine neue Laufregelung entwickelt. Die Regelung optimiert die Umgebungsmodellierung basierend auf der Dynamik des Roboters und nutzt Kontaktsensoren um mit nicht erwarteten Bodenkontakten umzugehen.
Einige der vorgestellten Algorithmen wurden zusätzlich im Einsatz mit Manipulatoren validert. Zudem sind alle während des Projekts entwickelten Werkzeuge, inklusive eines Augmented Reallity Systems, als Open Source veröffentlicht. Die Beiträge dieser Arbeit sollen zweibeinige Roboter näher zu ihrer Anwendung in echten Szenarien bringen.

Contents

Abbreviations

3MM	three mass model
AIST	Japan's National Institute of Advanced Industrial Science and Technology
AR	augmented reality
CNRS	French National Center for Scientific Research
CoG	center of gravity
CoP	center of pressure
DARPA	United States Defense Advanced Research Projects Agency
DLR	German Aerospace Center
DoF/s	degree/s of freedom
DRC	DARPA Robotics Challenge
DSP	double support phase
EC	early contact
EM	Expectation Maximization
EoM/s	equation/s of motion
EuRoC	European Robotics Challenges
FoR	frame of reference
FoV	field of view
fps	frames per second
GMM	Gaussian Mixture Model
HRP	Japan's Humanoid Robotics Project
IC	ideal contact
IMU	inertia measurement unit
KAIST	Korea Advanced Institute of Science and Technology
LC	late contact
LEPP	Lola Environment Perception Package
LIPM	linear inverted pendulum model
MLE	maximum likelihood estimate
PC/s	point cloud/s
PCL	Point Cloud Library
RANSAC	Random Sample Consensus
ROS	Robot Operating System
SfR	Structure from Motion
SLAM	Simultaneous Localization and Mapping
SSP	single support phase
SSV/s	swept-sphere-volume/s
TCP	tool center point
ZMP	zero moment point

Chapter 1

Introduction

History and Motivation behind Humanoid Robots

Since early in history, humankind has been fascinated with the idea of creating artificial beings in their own image. Even before technology had reached a stage where autonomous machines became possible, the concept of creating humanoid beings was long present in the literature and folklore. In Greek mythology, the God Hephaestus created artificial helpers to aid him in his workshop; Jewish legends contain several mention of *golems*, anthropomorphic beings created magically out of clay; *Pinocchio* comes to life after being carved by Mr. Geppetto while Victor Frankenstein creates *the Monster* during a scientific experiment.
This fascination with anthropomorphic beings was not restricted to literature. Since early in human history, animated humanoid figures called *automata* were built to entertain people, serve as toys or demonstrations of mechanical principles. Using cleverly build mechanisms, they were able to perform extremely complex but predefined tasks such as playing the flute or writing [10].
Since the industrial revolution, machines started replacing humans in tasks requiring physical labor. However, people were still required for tasks that are not repetitive or require certain intelligence, flexibility or adaptability (and still are). If a machine could be created with the exact shape and capabilities of humans, then it could replace them in all tasks they find either tedious or physically challenging. In his 1920 play "Rossum's Universal Robots", Karel Čapek introduced the term *robot* (meaning "forced laborer") to designate the artificial people created by humans to replace them in production and hard work [22]. Since then, the term has become widely popular, first in modern literature and then in science. Interestingly, the term *robotics* was first used by the science fiction writer Isaac Asimov referring to the study and development of robots and their principles [5]. Even though Asimov's robot stories usually dealt with anthropomorphic machines, the term has since been used to designate all kinds of autonomous machines [167].
If the purpose of a robot is to help or replace humans in tedious or difficult tasks, it is natural to ask if it necessarily must have an anthropomorphic shape. One argument has been mentioned already and refers to its application as a human substitute. As explained by Asimov in one of his early stories:

> "[The robot] is designed like a man so that it can use all the tools and machines that have, after all, been designed to be used by a man." [6]

However, that argument alone is not entirely convincing, especially when considering that the price of robots may be high enough to justify the modification of the tools and machines themselves. Without talking about the particular difficulties in controlling these robots, a humanoid configuration may not be the most adequate or efficient depending on the particular task. For example, large industrial manipulators are better capable of executing tasks requiring high forces; wheeled or flying machines are more adequate for transportation or access of

remote locations than anthropomorphic robots. Additionally, humanoid robots have a high number of degrees of freedom (DoFs), requiring numerous actuators and mechanisms: even though the cost of electronic components has been rapidly declining in the last years and will probably continue to do so, that is not the case of mechanical costs [118]. Therefore, more mechanically-simple machines may be more cost-effective for certain applications.
Nevertheless, anthropomorphic robots are particularly well-suited for applications that depend on human-centered scenarios. Adapting Asimov's argument, humanoid robots are developed to operate in environments that were specifically designed for humans or that are co-inhabited by humans. They are the natural candidates for tasks like collaborating with housework or helping elder people, as well as replacing humans in dangerous scenarios. For example in a Nuclear Plant, where corridors and corners are designed for people, an anthropomorphic robot would have more access than e.g. a wheeled robot and would be therefore better suited to help in case of an accident. Additionally, the development of humanoid robots has enormous potential applications in medicine, from the testing of human accessories to the improvement of bio-mechanical prostheses.
This work deals with the mobility and navigation of humanoid robots, an area that is still in the research phase. The complexity of humanoid robot control lies, on the one hand, in the robot's configuration. Due to the limited one-sided contact with the ground at the feet, these robots can exert limited force and torque against it and cannot arbitrarily adopt any pose or configuration as they might fall down. Therefore, a humanoid robot is *underactuated*, meaning it has more DoFs (its position and orientation with respect to the ground in addition to its own articulations) than actuators. As a result, not every conceivable motion is achievable or safe, resulting in a more complex control system. On the other hand, the high number of degrees of freedom makes it computationally challenging to perform real-time motion planning and control: every desired action must be executed by synchronously controlling all the robot's actuators without compromising its stability.
One popular solution to overcome these problems is the use of reduced models to approximate the robot's dynamics, as explained in chapter 2 of this thesis. Humanoid robots can nowadays execute stable motions in real-time and quickly adapt their trajectories to the user's input [14, 85, 108, 133, 177]. Once the robot can be robustly controlled, that still leaves the problem of achieving autonomous navigation. After all, if humanoid robots are to become really useful one day, they have to be able to safely navigate through irregular terrain without having a human indicating where to place each footstep. In order to exploit their kinematic ability and present an advantage over wheeled platforms, they must be able to perform complex motions such as climbing stairs or walking over obstacles. This multiple number of high-dimensional motions make the problem of navigation a more complex one in humanoid robots than in wheeled ones, from the detail of the required environment representation to the planning, execution and stabilization of robot motion.
In order to advance humanoid locomotion capabilities in such scenarios, the project *Versatile and Robust Walking in Uneven Terrain*, financially supported by the German Research Foundation[1], was started at the Chair of Applied Mechanics, Technical University of Munich. With the parallel work of three Ph.D. candidates, a framework for autonomous biped navigation in unknown environments is developed, including capabilities to model the environment and generate safe trajectories in real-time, as well as adapting those trajectories on-the-fly to stabilize the robot against external disturbances.
In this context, the work presented in this thesis deals with the acquisition and processing of environment information, as well as with the analysis and compensation of inaccuracies of the environment data on the robot's stability. The relationship between these parallel works is presented in chapter 2 and the final developed framework is discussed in chapter 7. In

[1] DFG project BU 2736/1-1

the following, a review of related work is given, followed by an overview of the hardware platforms used throughout this project and the main contributions of this thesis.

Related Work

In this section, a review of relevant work on autonomous biped navigation is given, focusing on the navigation in unknown, irregular scenarios. It is divided into the main existing projects and research groups. More subject-specific reviews are given in each corresponding chapter throughout this thesis. The interested reader can find a thorough review of humanoid hardware and their mechanical complexity in Lohmeier [110] while a review of humanoid simulation and control strategies can be found in Buschmann [13]. In direct relationship with this work, Hildebrandt [66] gives a detailed review of navigation and motion planning approaches for humanoid robots while Wittmann [200] presents a thorough review of humanoid stabilization and robust locomotion strategies.
Traditionally, humanoid robots were a niche area of research, with most work being carried on in Japan and without attracting too much attention or funding due to their limited performance. In 2011, however, a massive earthquake and an accompanying tsunami hit the west coast of Japan with disastrous consequences on the Fukushima Daiichi nuclear power plant. It was the second largest nuclear disaster in history, after the Chernobyl accident [129]. Due to safety reasons, the Japanese government and power plant operators refrained from sending in people to contain the damage. The obvious answer was the use of robots: in the span of several missions, robots were sent to assess and limit the environmental damage. Nevertheless, the mobility of existing robotic systems in such an environment was extremely constrained and difficult [129]; their limitations resulted in an unprecedented interest in anthropomorphic robots. Legged robots are theoretically better suited for navigating through cluttered environments and could potentially be used in such hazardous areas. Unfortunately, there was no robot sufficiently advanced to be used in Fukushima at the time (and there still isn't). A few years later, the United States Defense Advanced Research Projects Agency (DARPA) launched the DARPA Robotics Challenge (DRC) [31], in which teams with different robots tried to solve several tasks inspired by those scenarios (see fig. 1.1). Even though the completion of the tasks relied strongly on teleoperation and none of the participating teams performed completely autonomous navigation (as far as the author knows), the DRC helped to evaluate the state of the art in humanoid technology as well as to motivate new research. Still, not all relevant research groups participated in the DRC. In the following, a review of the most relevant laboratories and projects on autonomous biped navigation is presented.

Japan and Korea

As mentioned before, Japan has been a pioneer in humanoid robotic research. The first functional biped robot, *WABOT-1*, was developed at the Waseda University in 1973. Since then, they have demonstrated various advanced mobility and control capabilities and developed multiple legged robots; the latest one is called *WABIAN-2* [4].
The Tokyo University JSK Laboratory has a long tradition of biped locomotion research. Together with Kawada Industries, they developed the *H6* and *H7* humanoid robots, on which some of the earlier work on autonomous biped navigation was done [82]. By creating a height-map of the environment, the *H7* is capable of generating collision-free trajectories and stepping over simple obstacles. However, motions are planned at the beginning of the

Figure 1.1: The DARPA Robotics Challenge tested robots with tasks inspired by the Fukushima disaster. In this picture, the Japanese robot *HRP-2* tries to solve a task consisting on turning a valve. Photograph by the author.

sequence before walking due to the computational limitations.
Better known are the results of the *Humanoid Robotics Project* (HRP), a joint project between Japan's National Institute of Advanced Industrial Science and Technology (AIST), Kawada Industries and several Japanese universities [89]. One of the prototypes developed, the *HRP-2* [90] (see fig. 1.1), has become a standard platform for humanoid research [83]. Much of today's reference literature for autonomous biped navigation has been developed for this robot [125, 134–137], particularly at AIST. In their work, the robot is able to walk over obstacles, platforms and ramps in real-time. The limitations of the system are the assumption of static environments which are scanned each step with a pivoting laser and the simplicity of the tested scenarios, which are simply represented by a height-map and don't require complex or dynamic motions by the robot. Out of this research group, a spin-off called *Shaft* was founded to participate in the DRC, where it won the first round (or "trials") by a clear margin [31] before being bought by Google and starting working in almost total secrecy [3].
At the Sony Corporation, the 0.6 m-tall *QRIO* robot was developed for entertainment purposes. Even though it wasn't particularly successful as a toy, researchers at the company achieved a high level of autonomy, recognizing and avoiding obstacles and climbing stairs in static environments [59, 60].
Perhaps the most impressive work on humanoid robots in Japan has been done by the Honda Motor Corporation since 1986. After ten years of secret research and development, they disclosed their *P2* robot, the world's first self-stabilizing, two-legged humanoid robot to the public [73]. Since then, they have demonstrated amazing locomotion capabilities and inspired other projects such as the HRP. Their latest series of prototypes, called *Asimo*, were developed to operate in actual human living environments [161] and represent the closest achievement towards that goal. As Honda does not usually publish too many details about their research, it is difficult to know how much level of autonomy their robots have. Nevertheless, from public demonstrations and personal conversations one can safely infer their advanced capabilities on object recognition, manipulation, stair climbing and even ball kicking. Some details of their control system were later published as papers as well [177–180]. While *Asimo*'s capabilities in a household are impressive, it cannot adequately walk in unknown, non-structured environments. That is why, after the Fukushima disaster, Honda set up in developing another prototype specifically built for disaster response which they just recently disclosed [207]. Although details on the level of autonomy are scarce, it is capable of amazing mobility feats, such as bipedal and quadrupedal fast walking, stairs, ladder and piping climbing as well as sideways walking while being resistant to dust, water and high temperatures. These capabilities make the robot a promising platform for future research.
In Korea, the most relevant work on humanoid robotics has been done at the Korea Advanced Institute of Science and Technology (KAIST), where a series of humanoid robots called *HUBO* were developed [141]. Built after Honda's *Asimo* robots, these prototypes have demonstrated walking, running and manipulation capabilities. They became notorious by winning the DRC final round [31]. Among their achievements are a dual walking-wheeling locomotion mode [8] and a robust perception processing strategy [166], developed specifically to recognize different features during the challenge.

United States of America

First at the Carnegie Mellon University and later at the Massachusetts Institute of Technology, researchers led by Marc Raibert did pioneering work on legged locomotion [153]. Although the work was done mainly on hoping (instead of walking), their robots showed impressive and very stable locomotion capabilities in various kinds of terrain using hydraulic actuators [74]. In 1992, Raibert used these advancements to found a spin-off called Boston Dynamics

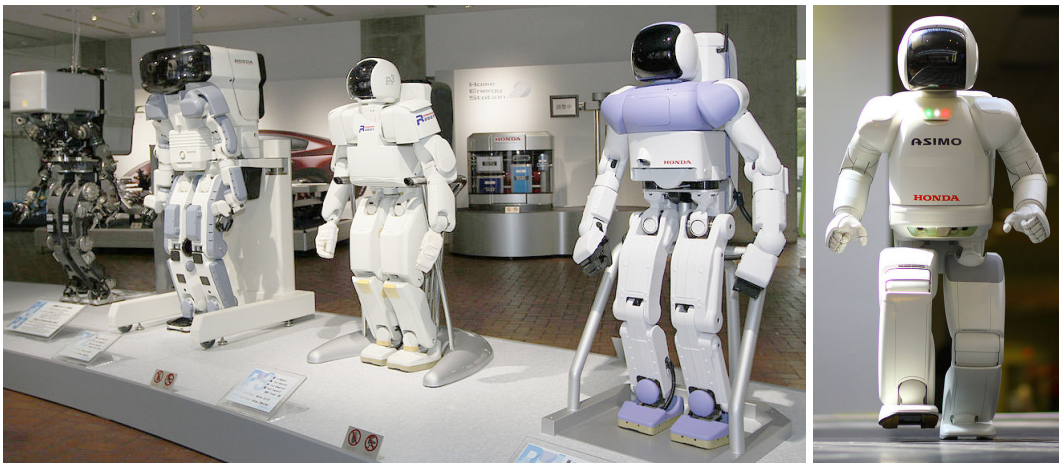


Figure 1.2: Humanoid robots by Honda. From left to right, some of the earlier prototypes (*P1*, *P2*, *P3* and *P4*) [123] and the 1.3 m-tall *Asimo* [186].

[168]. Supported by the DARPA, they have since then continued to disclose videos showing amazing capabilities by their hydraulic legged robots. Perhaps one of their best known creations, the quadruped *Big Dog* is capable of navigating through different kinds of irregular terrain as well as maintaining stability against external disturbances [154]. In 2012 they disclosed their first humanoid robot, *PETMAN*, an hydraulic machine to test chemical protective clothing [131]. In recent years they developed a series of humanoid robots called *Atlas* to serve as the official robotic platform at the DRC for teams that didn't have their own robots (see fig. 1.3). The robot was used at the DRC by teams from the Massachusetts Institute of technology, the Florida Institute of Human and Machine Cognition, the Virginia Institute of Technology together with the German Technische Universität Darmstadt, among many others [31]. There is little doubt that the company has been able to build impressive machines that have little or no rival in terms of mobility and robustness (in partly thanks to the hydraulic actuation, which allows for more powerful movements). Unfortunately, it has always worked under strict secrecy due to potential military applications. A few years ago, it was acquired by Google, which reduced the already limited amount of publications and announcements. Nowadays, it is owned by SoftBank [168], but it is not clear what path the company is going to take in the future. Nevertheless, a little bit can be inferred from the published videos, personal conversations and live demonstrations at the DRC. The robots are mostly teleoperated: the adaption to irregular terrain is a result of their robust control [154] while a human operator gives navigational commands. Although the company has never been specific about the level of autonomy of their robots, displays of their perception and planning system in videos leave little doubt that they make use of techniques which are comparable to the ones presented throughout this work [169].

At the Massachusetts Institute of Technology, there has been interesting work on motion planning and autonomous biped navigation based on optimization strategies. Though they have shown results achieving autonomous and robust locomotion, these techniques have the downside that they require high computational power and are not easily applicable to changing environments [29, 32, 47, 100].

There are many other interesting projects in the USA dealing with humanoid locomotion, especially since the DRC. However, focus lies mostly on control and trajectory generation and not on autonomous navigation. Still, it is worth to mention a few of them, as they follow walking control strategies that differ from the approach used by most bipeds that are based on maintaining the robot's stability by controlling the position of the center of

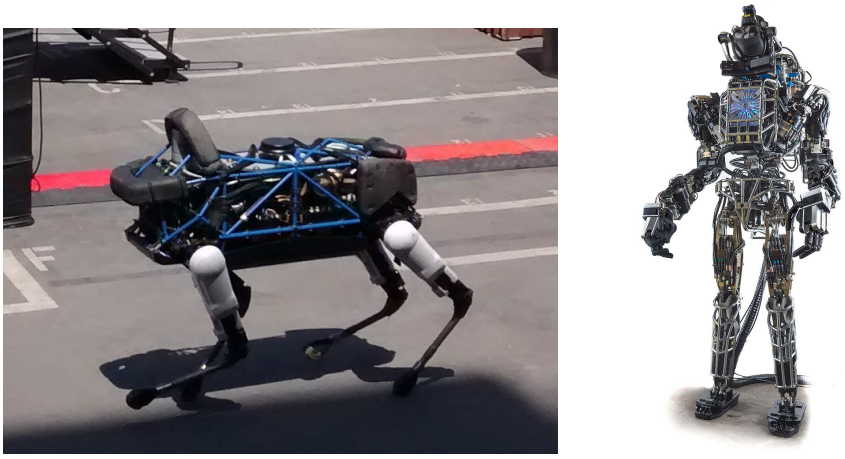


Figure 1.3: Legged robots with hydraulic actuation by Boston Dynamics. On the left, one of their latest quadrupeds, the 0.94 m-tall *Spot* (photograph by the author). On the right, the 1.8 m-tall *Atlas* humanoid robot, built for the DRC.

pressure (see chapter 2). At the Florida Institute of Human and Machine Cognition, they achieve stable walking by calculating "capture points", which indicate the locations where the robot must place its foot to stop its motion [150]. Another interesting approach is called the "hybrid zero dynamics", which obtains a solution for stable walking by including the virtual constraints in an optimization problem that controls the actuated DoFs; the underactuated DoFs are included as inequality constraints in the optimization formulation [195]. These control strategies have been successfully applied to semi-passive robots with compliant joints for energy efficiency [171]. Such a robot, called *ATRIAS*, was recently developed at the Oregon State University. It is able to perform stable walking on two point-feet by controlling the forces between the robot and the ground [78].

Europe

The Chair of Applied Mechanics at the Technical University of Munich has a long tradition of building autonomous robots. In 2002, the humanoid robot *Johnnie*, capable of stable human-like walking and jogging, was presented [108]. In terms of autonomy, it is able to step over obstacles and climb stairs [28]. Although the on-board perception system is capable of localizing those environment features, their geometry was known in advance. In 2010, a follow-up prototype called *Lola* was presented [111]. At the time, it was able to safely navigate changing environments by checking two-dimensional safe paths [16]. It is the main platform used for this project and is referenced thoroughly throughout this thesis.
In recent years, several torque-controlled robot manipulators were developed at the German Aerospace Center (DLR) [99]. In 2013, the biped robot *TORO* was presented based on this technology. Although it is capable of performing remarkable capabilities in terms of balancing thanks to the torque information at the joints, its walking control is still based on a position-based control of the joints and capture points [41]. At the University of Freiburg, interesting work on autonomous navigation was presented for the *Nao* humanoid robot, a research platform originally developed by the French company Aldebaran Robotics for the RoboCup which is 0.58 m tall. They achieved collision-free navigation using texture and color for environment classification [114, 116]. More recently, at the University of Bonn, they presented fast planning capabilities in dynamic environments using a more simple envi-

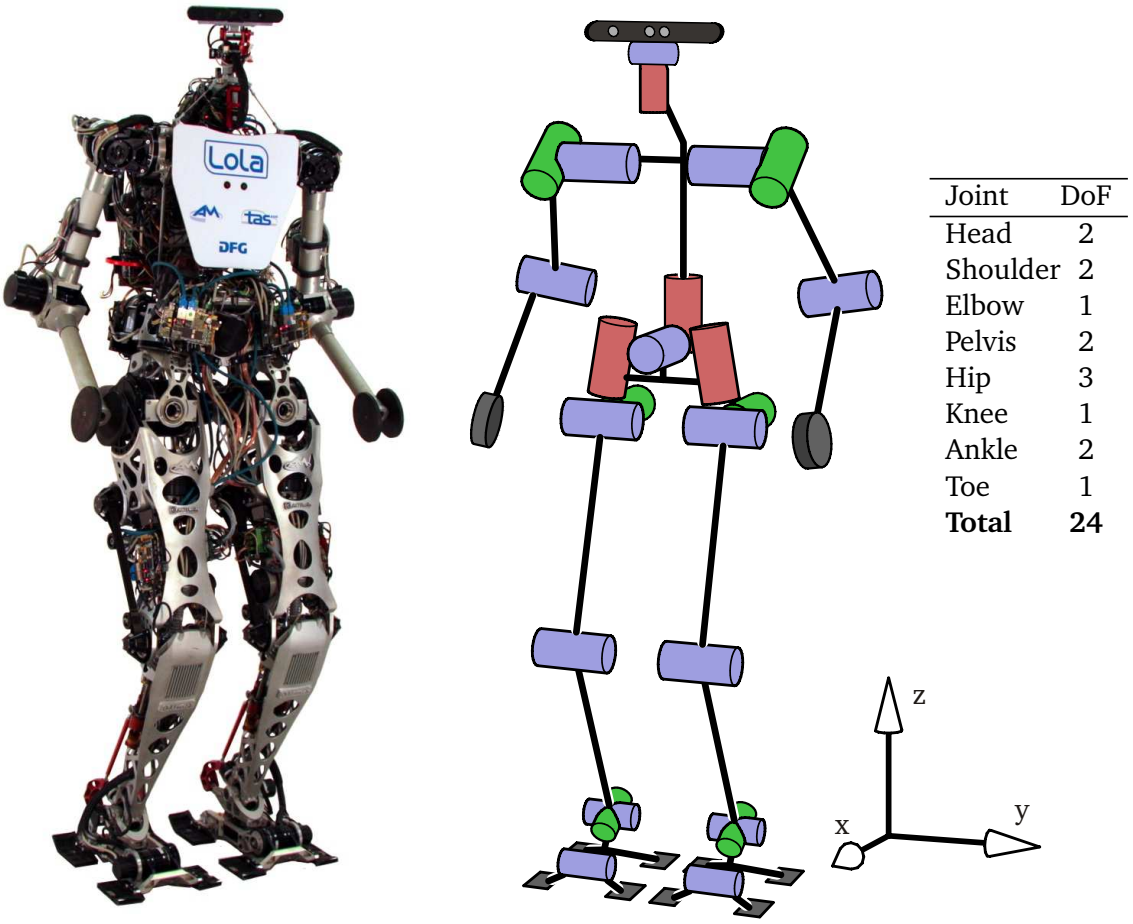
ronment representation model [91, 92]. Other interesting works were developed in France and Italy. At the French National Center for Scientific Research (CNRS), many results have been published based on the *HRP-2* robot and collaboration with Japan's AIST [206]. Their work is mainly focus on trajectory generation, based on a mathematical optimization formulation of the motion problem [39]. Although their system could be integrated with perception information [155], the framework is not yet fast enough to be applicable in dynamic, complex environments. At the Italian Institute of Technology, several robots have been developed including one for participation at the DRC [31, 106].

Autonomous Vehicles

Another related research area worth mentioning in this section is the autonomous vehicles industry, where dynamic scenarios are taken into account for real-time navigation. The main difficulty in assessing the state of the art in this area is that it has mainly been developed by private companies which avoid disclosing too much information in order to protect their know-how. Nevertheless, some information can be inferred from a few available publications [42, 95], magazine or newspaper articles [151] and demonstration videos [182].
The first important characteristic of autonomous driving is that the environment is not entirely unknown. By previously scanning and acquiring a labeled 3D map of streets, traffic signs and landmarks, the car already has detailed information on the environment surrounding it. Using GPS signals and comparing features its map and sensor information, it is able to know its position in the world map, simplifying both the perception and planning strategies [2]. The perception system is constrained to identifying known features of the environment (such as traffic lights wich it already knows where to look for) and recognizing external components to it. These, however, consist mostly of usual elements of the scene, such as other vehicles, bicycles and pedestrians which can be recognized using learning methods [42]. As the problem of vehicle navigation is a two-dimensional one, all these and other objects can be modeled with simple bounding boxes [182] which are usually enough for autonomous driving. On an interesting note, the limitations of such systems become apparent when dealing with strange objects which are not part of their two-dimensional world, such as jumping kangaroos on the road [45]; these and similar cases could be solved with more general environment representations and perception systems such as the one presented in this thesis (chapter 5).
The available information about the environment and its representation together with the small dimensionality of the planning problem for an autonomous car (a two-dimensional robot) make the navigation problem of autonomous vehicles a restricted one compared to the one for high-dimensional, underactuated humanoid robots. Nevertheless, the state of this area of research is arguably more advanced than the one of humanoid robotics and important lessons are to be learned from it, especially when dealing with dynamic environments.

Experimental Platforms

The contributions of this thesis are repeatedly verified in simulations and experiments with the humanid robot *Lola* (fig. 1.4). It was developed at the Chair of Applied Mechanics between 2004 and 2010 based on the lessons learned with *Johnnie*. A detailed description about its mechanical design and realization can be found in Lohmeier [110]. Buschmann [13] developed the main control system for the robot as well as a simulation platform that handles unilateral and compliant contacts with the ground and takes motor dynamics and



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

Figure 1.4: Photo and kinematic structure of the humanoid robot *Lola* with an RGB-D sensor mounted on top.

control loops into consideration.
Lola is an electrically actuated robot with 24 DoFs which weighs approximately 60 kg and is 1.8 m tall. It is capable of stable walking at a speed of up to 0.94 m/s. Each leg has 7 DoFs including an actuated toe. It has no dexterous hands, its arms (3 DoFs each) are mainly used for the compensation of angular momentum and center of gravity dynamics [164]. Besides having position sensors at every joint, *Lola* has a high-precision inertia measurement unit (IMU) at the chest and force/torque sensors at the ankles. As explained in chapter 2, these are used for balancing and tracking of the center of pressure (in other words, these are the sensors that are used to keep the robot from falling down). In fig. 1.4, a photo and the kinematic configuration of the robot can be seen. During the course of this project, it is decided to use a standard RGB-D sensor Asus Xtion PRO LIVE [7] (30 Hz) for 3D sensing (chapter 3) which can be seen mounted on top. The walking controller and the perception system run on two parallel on–board computers with Intel Core i7-4770S@3.1 GHz (4x) processors and 8GB RAM and communicate via Ethernet using UDP and TCP/IP (see chapter 7). An up-to-date description of *Lola*'s hardware and low-level control can be found in Sygulla et al. [176].
The methods discussed throughout this thesis are intended to be as generally applicable as possible, even on other kinds of robots besides humanoids. Therefore, they are repeatedly validated on a robotic manipulator as well. For that purpose, the *CROPS* platform, developed at the Chair of Applied Mechanics, is used (see fig. 1.5). Its development was part of a Euro-

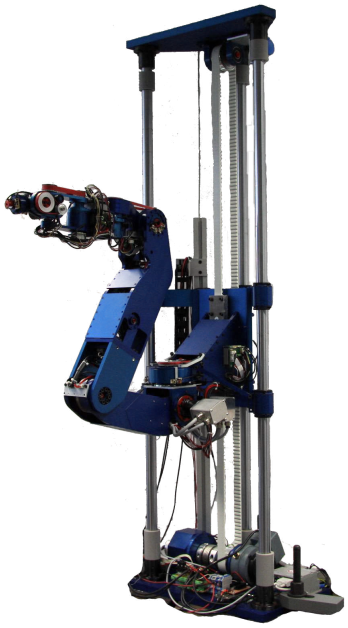


Figure 1.5: The *CROPS* first generation manipulator prototype with 9 DoFs: one translational DoF (vertical) and eight rotational DoFs.

pean project which focused on an autonomous multi-purpose robot system for the selective harvest of sweet pepper and apple as well as the precision spraying on grapes. Details on the project can be found in Bontsema et al. [11] while the manipulator system as used in the experiments is described by Schütz et al. [163] and Ulbrich et al. [184].
The *CROPS* robot has a total of nine DoFs and is mounted on a fixed platform. It is capable of processing and executing motion commands in real-time, as well as generating trajectories between given points in its workspace. In this thesis, it is used to test calibration (chapter 3), environment recognition (chapter 4) and obstacle avoidance (chapter 8) methods.

Objectives and Structure of this Thesis

This thesis aims to close the gap between complex perception systems and real-time autonomous navigation. As mentioned, one of the greatest challenges to develop useful, robust humanoid robots is their navigation capabilities in unknown environments. Most existing solutions assume static scenarios and use simple environment representations which limit their applicability in the real world.
In order to be useful in real scenarios, a framework for autonomous biped navigation based on perception information must satisfy the following requirements:

1. Online processing of perception information during walking to react to dynamic environments.

2. Efficient representation of complex scenarios to enable both real-time planning and elaborated navigation motions.

3. Robust against inevitable uncertainties in environment data.

The problem is approached sequentially. In chapters 2 to 4, the basic concepts and preliminary conclusions about humanoid navigation and perception are presented. An introduction

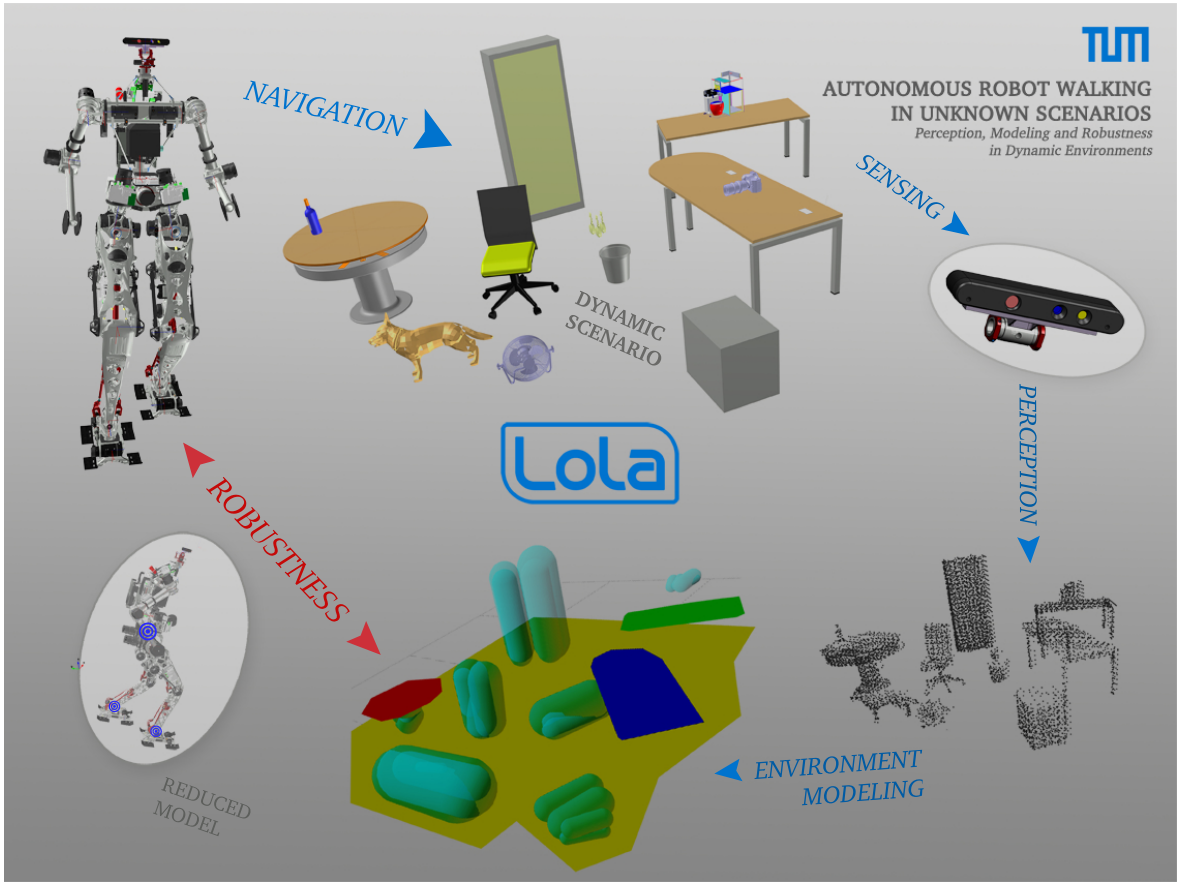


Figure 1.6: Concept of this thesis. A humanoid robot has to navigate dynamic environments using perception information from an adequate sensor; out of that information an abstract model of the environment is created which can be quickly processed by a planning algorithm; a reduced dynamic model of the robot is used to deal with perception inaccuracies.

about the basic principles of humanoid locomotion is given in chapter 2, together with the extended framework developed for this project. Different environment sensing technologies are discussed in chapter 3 together with the considerations for their application to humanoid navigation and new automatic calibration procedures for the integration of sensor data with the robot's planning system. In chapter 4, both existing and new strategies for environment recognition and modeling are presented.
Based on a new 3D environment representation, a perception system is presented in chapter 5 that is capable of processing dynamic environments online. With it, the full-size humanoid robot Lola is capable of navigating through previously unknown dynamic scenarios including moving persons, fixed obstacles and platforms.
Nevertheless, inaccuracies in the perception system and irregular terrain may affect the robot's stability. A new walking controller is presented in chapter 6 that increases the robot's robustness against unexpected contact situations.
A graphical representation of the concept behind this thesis can be seen in fig. 1.6. In it, an example of a complex scenario is shown. In order for a robot to navigate through it (chapter 2), an adequate sensor is required (chapter 3). Perception information must be processed (chapter 4) and an adequate model of the environment is sent to the motion planning (chapter 5). Inaccuracies in perception information are dealt with a new walking controller based on a reduced dynamic model of the robot (chapter 6).
The final framework developed together with other two Ph.D. candidates for this project is

described in chapter 7. In it, an augmented reality system for online visualization of the framework's results is presented. This thesis concludes with the experimental results (chapter 8) and a discussion on accomplishments, limitations and future work (chapter 9).

Chapter 2

Biped Walking

Approach to Biped Walking Control

Humanoid robots are complex machines with high number of degrees of freedom (DoFs), which are not fixed to the ground (see fig. 1.4). As such, the motion and balance of these multi-body systems depend heavily on the dynamics of the different moving parts. These have to be taken into account when designing walking controllers. A first and thorough approach to the problem of motion generation and control would be to write the equations of motion (EoMs) of the complete system and perform an optimization process considering the trajectories of all DoFs. However, there are several problems with this approach. To begin with, it is not clear if a mathematical solution to such a high-dimensional problem even exists. Assuming some kind of solution can be found, the necessary calculations would take too long to be performed in real-time [88]. Additionally (and most importantly), any kind of physical model cannot perfectly reflect real machines in real conditions and some kind of corrective feedback would be needed anyway.
For these reasons, a more pragmatic approach is used when designing both walking robots and controllers: a simplified, approximate physical model is made on which control laws and trajectories can be more easily performed; using a hierarchical structure, a feedback control loop monitors the state of the robot - typically via force/torque sensors on the feet and/or inertial measurement units (IMU) - and applies a corrective signal that enables a stable gait in real environments. In this chapter, an overview of classical robot models and controllers is first given, with special focus on the one used by *Lola*. It is based mainly on the works of Buschmann [13] and Kajita et al. [88], where interested readers may find more details and information. Afterwards, the more encompassing framework for autonomous walking in unknown scenarios, co-developed with A.-C. Hildebrandt and R. Wittmann during this project, is presented. This chapter provides the basic motivation and background for the rest of this thesis and is referred to in subsequent chapters.

Simplified Robot Models

As mentioned before, a reduced model of the robot is needed to deduce basic physical relationships and develop comprehensive control laws. The most simple model that is applicable to a humanoid robot is a two-dimensional, linear inverted pendulum model (LIPM). It has been widely used in literature [33, 61, 84–86, 108, 124, 133, 173, 197] due to its simplicity and the possibility of arriving to closed-form mathematical solutions. It consists of a point mass - on the robot's center of gravity (CoG), where the complete robot's mass is concentrated - balancing with constant height on an extensible leg. In fig. 2.1 the LIPM is depicted on top of a figure of *Lola* in the sagittal plane. From this point on, the analysis of two-dimensional

models is often reduced to the sagittal plane, as it is analogous to the frontal plane. This two-dimensional abstraction results in decoupled equations for the x and y directions[1], simplifying the implementation in the walking controller.
For this and all following models, a few simplifying assumptions are made:

- The robot is standing on one foot only (also called *single support phase* or SSP).
- Only vertical forces between foot and ground are of interest. Horizontal (or friction) forces are assumed to be large enough to prevent slipping.
- The robot is assumed to be in stable contact with the ground (i.e. the robot is not tipping -or about to tip- over and the foot is not leaving -or about to leave- contact with the ground).
- The change rate of angular momentum of the robot with respect to its center of gravity ($\dot{L}^{CoG}$) is assumed to be negligible (the robot keeps standing approximately straight).

The total sum of vertical forces between the ground and the foot can be represented by an equivalent, concentrate force and torque on one single point. Moreover, if that single point is chosen as the foot's center of pressure (CoP), the equivalent torque is zero, reason why it is often referred to as the "Zero Moment Point" (ZMP) [187]. Intuitively, one can guess that the CoP should lie inside the robot's foot during the single support phase. Physically, the position of the CoP represents the imaginary point where the vertical contact forces can be replaced by a single force. If the CoP coincides with the edge of the foot, that means that all contact forces are concentrated on that edge and the robot is balancing itself on that single edge, possibly tipping over. When both feet are on the ground during the double support phase (DSP), the CoP may lie anywere inside the smallest convex polygon containing both feet without the robot tilting over. This polygon is called *Support Polygon* and is generally referred to indicate the smallest convex polygon containing all contact area with the ground - being it one or two feet. For a more thorough discussion on this topic, see Kajita et al. [88].

Linear Inverted Pendulum Model

The LIPM is based on the dynamic relationship between the center of pressure (CoP) and the center of gravity (CoG), where the robot's whole mass is represented by a single point. Both points are joined with an extensible leg (see fig. 2.1). In addition to the assumption that $\dot{L}^{CoG} \simeq 0$, the motion of the CoG is constrained to the horizontal direction (hence the term *linear* and the need for an extensible leg). Therefore, the dynamics of the system can be written as:

$$m\ddot{x}_G z_G - mg\,(x_G - x_P) = 0 \tag{2.1}$$

where m is the mass of the robot, g refers to the gravity acceleration, $(x_G, z_G)^\mathsf{T}$ is the location of the CoG and x_P the location of the CoP. Solving for $\ddot{x}_G$ yields:

$$\ddot{x}_G = \frac{g}{z_G}\,(x_G - x_P) \tag{2.2}$$

which is a direct relationship between the CoG acceleration and the CoP position (z_G is constant). An interesting feature of this relationship is that it sets the basis for a walking controller: *given a trajectory for the CoP, a trajectory for the CoG can be obtained* (eq. (2.2) can be

[1] At some points, the term *horizontal direction* is used as a general term to refer to x or/and y in contrast to the vertical direction z.

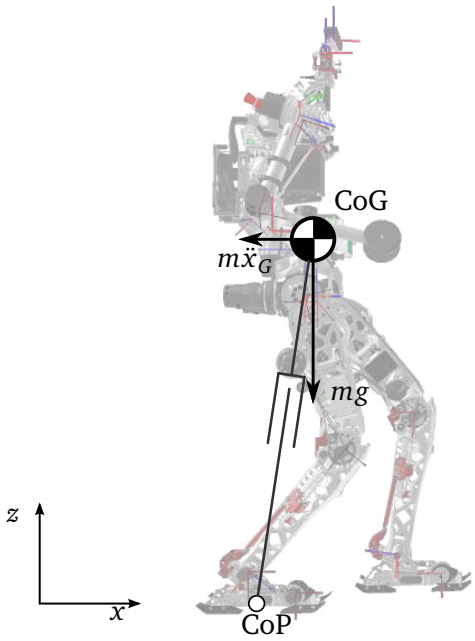


Figure 2.1: Robot and linear inverted pendulum model with one mass m for the upper body at the center of gravity (CoG), with location $(x_G, z_G)^T$. Through an extensible leg it is connected to a support point at the center of pressure (CoP), located at x_p, on which it balances. Sagittal plane (the frontal plane corresponds to $y - z$ in this case).

solved both numerically and analytically [88]). As explained earlier, the CoP should lie inside the support polygon to prevent the robot from tilting over. Therefore, a walking controller can be designed by first planning safe CoP trajectories and then solving the CoG trajectories. Solving then the robot's inverse kinematics yields the full set of trajectories [14, 85, 108, 133, 177]. Feedback control is introduced to compensate for modeling errors and external disturbances [15, 87, 136, 180]. This relationship was first noted by Vukobratovic et al. [187] and resulted in the popularity of the LIPM model, used by most existing walking controllers, also called *ZMP-based Controllers*. These are based on the strategy of keeping the CoP (or ZMP) inside the support polygon at all times[2]. Kajita et al. [88] presents an alternative model, called the "Table-Cart Model", to illustrate the difference between first defining the CoP trajectory and then obtaining the CoG trajectory and otherwise, though both models are described by eq. (2.2).

Three Mass Model

Naturally, the LIPM is an extremely strong simplification of a biped robot, which fails to reflect the dynamics of the extremities. Buschmann et al. [14] propose a three mass model (3MM) instead, with one point mass for the upper body and one point mass for each leg (located at the feet) as can be seen in fig. 2.2. Other authors have proposed ZMP-based controllers using different models as well [88, 177] but the procedure is similar to the one presented here. It is important to note that the right model to use depends strongly on the particular robot. In

[2]Having the CoP inside the support polygon means that the robot is able to exert torque against the ground and is not yet tilting over. Some authors call this a state of "dynamical stability", though it is a slightly misleading term as the torque that the robot can exert on the ground is limited by the polygon's size. The CoP might lie inside the support polygon while the robot's accelerations being such that the robot cannot possibly recover and will become unstable in the future [13].

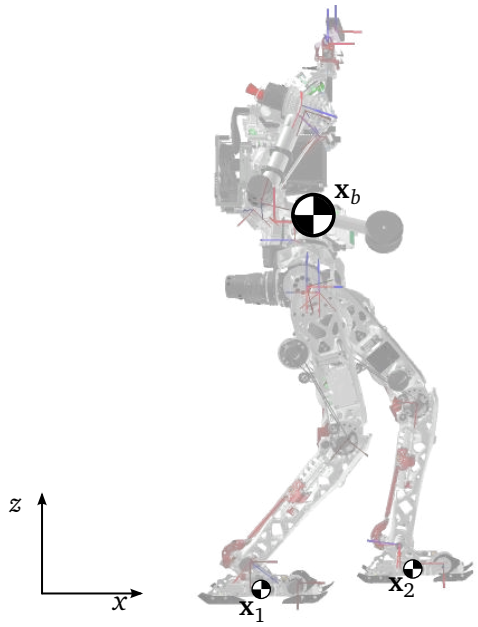


Figure 2.2: Robot and three mass model with one mass m_b for the upper body and two masses $m_{1,2}$ for the feet at the locations $\mathbf{x}_b$ and $\mathbf{x}_{1,2}$, respectively. Sagittal plane (the frontal plane corresponds to $y-z$ in this case).

the case of *Lola*, where 60 % of the robot's mass is located at the legs, the 3MM results in an adequate simplification of the robot's dynamics [13].
Again, it is assumed that $\dot{L}^{CoG} \simeq 0$, but the vertical acceleration of each mass points is not discarded. The dynamics of the system in both sagittal and frontal planes can be represented by the following set of equations (see fig. 2.2):

$$m_b z_b \ddot{x}_b - m_b x_b(\ddot{z}_b + g) = T_y + m_1 x_1(\ddot{z}_1 + g) - m_1 z_1 \ddot{x}_1 + m_2 x_2(\ddot{z}_2 + g) - m_2 z_2 \ddot{x}_2 \tag{2.3}$$

$$m_b z_b \ddot{y}_b - m_b y_b(\ddot{z}_b + g) = -T_x + m_1 y_1(\ddot{z}_1 + g) - m_1 z_1 \ddot{y}_1 + m_2 y_2(\ddot{z}_2 + g) - m_2 z_2 \ddot{y}_2 \tag{2.4}$$

$$m_b \ddot{z}_b - m_b g = F_z - m_1 \ddot{z}_1 - m_2 \ddot{z}_2 \tag{2.5}$$

where:

- m_b, m_1 and m_2 are the condensed masses of the upper body and both legs respectively,
- $\mathbf{x}_b = (x_b, y_b, z_b)^\mathsf{T}$, $\mathbf{x}_1 = (x_1, y_1, z_1)^\mathsf{T}$ and $\mathbf{x}_2 = (x_2, y_2, z_2)^\mathsf{T}$ the positions of the upper body and both feet, respectively and
- T_y, T_x and F_z are the ground moments and forces on the robot.

In a condensed form, eqs. (2.3) to (2.5) can be written as:

$$\mathbf{M}_b \ddot{\mathbf{x}}_b + \mathbf{h} = \mathbf{f} - \mathbf{M}_1 \ddot{\mathbf{x}}_1 - \mathbf{M}_2 \ddot{\mathbf{x}}_2 \tag{2.6}$$

where $\mathbf{f} = (T_y, -T_x, F_z)^\mathsf{T}$, $\mathbf{h}$ contains the gravity terms and $\mathbf{M}_b$, $\mathbf{M}_1$ and $\mathbf{M}_2$ the corresponding mass matrices for the upper body and both feet. In this case, the trajectory of the CoP determines $\mathbf{f}$. Additionally, the trajectories of $\mathbf{x}_1$ and $\mathbf{x}_2$ have to be defined and then eq. (2.6) can be used to obtain $\mathbf{x}_b$ [13].
To summarize, regardless of the model used, solving its reduced dynamic equations allows to obtain a relationship between the trajectories of the elements of the model and the CoP which can be used to build a hierarchical control for real-time motion generation. In the following, such a control for the robot *Lola* is explained in detail.

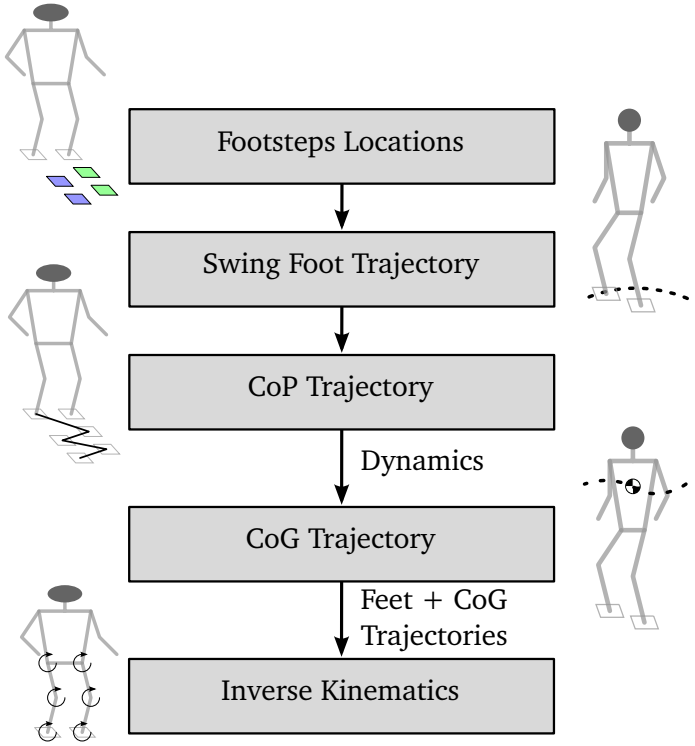


Figure 2.3: Walking Pattern Generation: obtaining feasible trajectories for a humanoid robot in real-time.

Walking Pattern Generation

One of the main components of a walking controller is a module that generates feasible trajectories for the the robot's different joints. The key to enable reactions to the environment or high-level commands is generating these trajectories in *real-time*[3], instead of pre-calculating them using complex optimization methods [39]. This can be achieved by sequentially generating different components' trajectories based on the dynamic relationships of a simple model (section 2.2), following the design objective of keeping the CoP inside the robot's support polygon. Such a process is usually called *Walking Pattern Generation* and runs as following (see fig. 2.3):

1. High-level commands define basic parameters such as desired walking speed and direction.

2. A step *duration* is defined and step lengths are adjusted based on the desired walking speed; with these parameters, future footstep positions are obtained.

3. The swing foot follows a predefined, parameterized trajectory; its parameters are adjusted based on the step duration, height and length.

4. The CoP trajectory is constrained to remain inside the resulting support polygon: confined to the stance foot during the SSP and shifting between both feet during the DSP.

5. The CoG trajectory can be quickly obtained from the model dynamics (eq. (2.6)) and the feet and CoP trajectories.

[3] In this case, the [real-time] requirement refers to the ability to generate new trajectories during each walking step, in order to adapt the robot's motion to a changing environment.

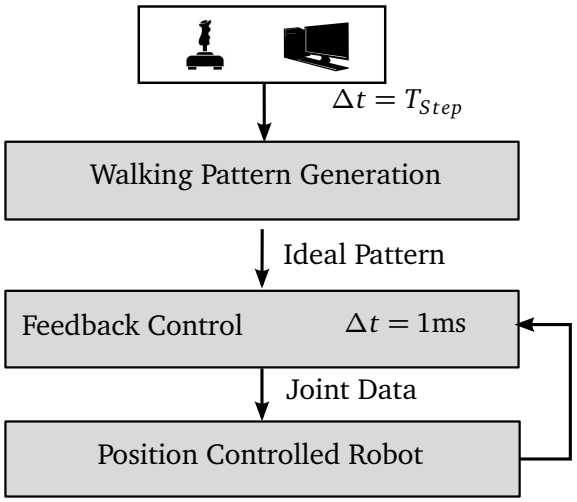


Figure 2.4: *Lola*'s real-time basic walking control system.

6. The joint trajectories can be generated using inverse kinematics.

In the case of the *Lola* robot, the following predefined trajectories are used as a first approximation for generating the final motion (for more details, see [13]).

- *Swing foot*: is parameterized by piecewise quintic polynomials between footstep positions, where it reaches zero velocity and acceleration.
- *CoP*: shifts linearly between both feet during the DSP and along one foot during the SSP.
- *CoG*: is generated from eq. (2.6). A solution to the initial value problem is numerically unstable and could diverge. Thus, periodicity conditions are imposed on position and velocity in order to achieve a periodic gait after the third and fourth steps. The resulting boundary value problem is solved by spline collocation.

Naturally, these ideal trajectories cannot be perfectly executed in a real robot with model and system errors as well as external disturbances. Therefore, a feedback module has to be included in a hierarchical control to stabilize the robot during walking.

Hierarchical Control

The robot's hierarchical control starts with a state machine in charge of defining the walking state, which defines the dynamics of the system. The different walking states depend first on the intended action (e.g. start/stop walking) and the planned contact state (e.g. single/double support) and are synchronized with predefined timings. These determine the phase of the walking controller and thus the control strategy according to the assumed contact state. As explained before, the *Walking Pattern Generation* module generates feasible, ideal trajectories based on high-level parameters or commands from the user. These, however, cannot be perfectly executed due to errors and disturbances and the robot has to be stabilized using sensor feedback as shown in fig. 2.4.
The robot's walking is affected by system and modeling errors as well as by external disturbances, all of which result in a deviation from the ideally planned trajectories. Within the

reduced model (section 2.2), these multiple error sources result in errors in the upper body (or CoG) height and orientation (robot's inclination with respect to the ground) [13]. The *Feedback Control* module[4] is in charge of reducing those errors which, by the basic laws of mechanics, can only be compensated by exerting forces and torques against the environment (the ground in this case).
The *Feedback Control* is based on information from the force/torque sensors located on the feet and the inertia measurement unit (IMU) in the upper body (see section 1.3). Using a PD control law the height and orientation of the upper body can be stabilized via the forces and torques between the feet and the ground: the load is distributed between both feet depending on which foot is on the ground; during the DSP, this *Load Distribution* shifts linearly between both feet. Additionally, the feedback control module keeps track of the CoP and makes sure it stays inside the support polygon to prevent the robot from tilting over. A hybrid position/force control law [15] is implemented to track the resulting forces and torques of the CoP trajectory (component $\mathbf{f}$ of eq. (2.6)).
The total effect of the feedback control module is modifying the reference trajectories in task space[5]. Afterwards, joint trajectories can be computed using inverse kinematics. Due to the redundancy of the robot configuration, these are solved using a resolved motion rate control as presented by Whitney [196]. Joint trajectories (position and velocity) are finally sent to the distributed drivers that control the individual robot joints.

Walking in Unknown Environments

Even though the framework presented above can achieve stable bipedal walking in laboratory conditions [17], there are certain limitations which prevent its application to real world scenarios:

(a) An ideal environment is assumed.

(b) A collision-free, horizontal ground is assumed when defining footstep positions.

(c) Timings are fixed to the predefined step duration, which makes the robot extremely sensible to irregular terrain.

(d) The presented scheme depends on a stabilizing feedback control for error or disturbance compensation which is inherently limited, as the maximum torque that can be exerted against the ground is determined by the foot's size.

The objective of the present project is to deal with these issues and provide the robot with the capability to autonomously walk in unknown environments. For this purpose, an extended framework for biped walking is depicted in fig. 2.5 that introduces three main extra modules, of which the first one is the subject of this thesis:

- **Perception System**. In order to navigate in complex scenarios, the robot has to obtain information about them (item (a)). However, this is not a simple task if the environment is completely unknown and non-static. Moreover, meaningful information has to

[4] In *Lola*'s case, the *Feedback Control* module is implemented before the inverse kinematics computation.
[5] In robotics, *task space* refers to the mathematical description (or parameters) of the geometrical coordinates where the robot acts (usually the cartesian or "world" coordinate system), in contrast with the *joint/configuration space* which refers to the mathematical description (or parameters) which define a robot's geometrical state (usually its joint angles).

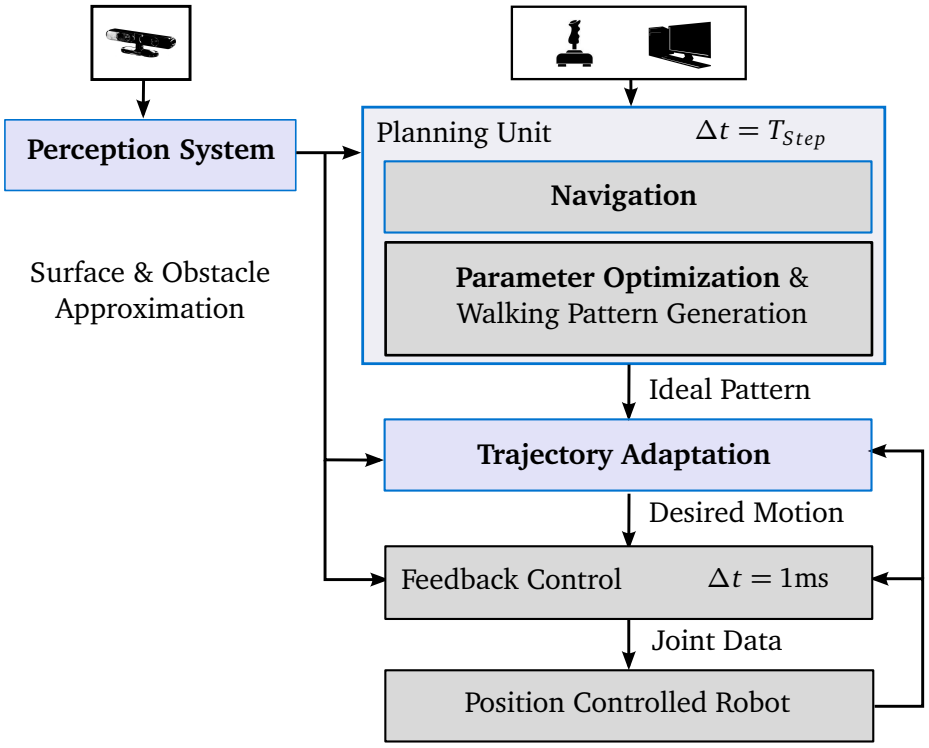


Figure 2.5: *Lola*'s real-time extended walking control system.

be obtained quickly in order to react and adapt the robot's walking in real-time. This is the main subject of this thesis: the acquisition of perception information by a robot is discussed in chapter 3 while the different strategies to process this information are discussed in chapter 4 and the perception system developed for this work is presented in chapter 5. Additionally, a modified walking controller is introduced in chapter 6 to overcome perception limitations and terrain irregularities (item (c)).

- **Planning Unit** Instead of a "blind" walking pattern generation module (item (b)), a planning unit is introduced to adapt the robot's motion to the environment model resulting from the perception system. A *Navigation* module takes the walking parameters and collisions with the environment into account to generate safe footstep locations for the next several steps. These are optimized in the *Parameter Optimization* module by considering the robot's kinematics. This work is developed in a parallel thesis by Hildebrandt [66].

- **Trajectory Adaptation** During walking, wrong assumptions about the environment or large external disturbances may destabilize the robot beyond the limits of the feedback control (item (d)). Once the robot starts tilting over, its actuation capabilities are severly reduced and a quick modification of the swing foot trajectory becomes necessary to avoid completely falling over. This work is developed in a parallel thesis by Wittmann [200].

In chapter 7 a summary of the final control framework is given, followed by experimental results of the robot *Lola* in unknown scenarios in chapter 8.

Chapter 3

Environment Sensing

Background

Animals are able to navigate specific environments by first obtaining information about them. In the case of humans, this is done via different senses, mainly sight, touch and hearing. Out of these, the sense of sight gives us some of the most relevant information and allows us to quickly adapt to different kinds of scenarios by providing a geometrical model of the world ahead, even regions of it we are not in contact with. The sense of touch, meanwhile, gives us more precise physical information (other than shape) of these regions and allows us to adapt our stride to different kinds of terrain, even when visual information is not sufficiently precise or incomplete. The sense of hearing allows us to identify some dynamic components (or vibrations) of the environment and to maintain vertical stability [19].
Just as humans with their sense of hearing and touch, robots such as *Lola* use an IMU and force/torque sensors in the feet to maintain vertical stability while walking (see chapter 2). Later in this thesis (see chapter 6) it is shown how more detailed touch information can be used to improve the robot's robustness against perception errors while walking over irregular terrain. In this chapter, it is shown how to obtain perception information in the first place. Christensen et al. [25] include the perception process as part of the more general process of *exteroception*, which refers to the "sensing and estimation to recover the state of the external world". Throughout this thesis, the term "perception" is used to indicate the acquisition of geometric information about the environment. Humans and many other animals have the sense of sight but other animals have developed different senses for this purpose, such as the animal echolocation (based on emitting and hearing the echo of ultrasounds, also called bio sonar) typically present in bats[1] or the magnetoreception used by many birds for orientation and navigation [93]. In the case of robots, different technologies have been used throughout history (some of them, but not all, inspired by nature) depending on the application requirements, available computational power and technical capabilities [167].
This chapter deals with perception technologies for robotic navigation. In order to understand the idea and motivation of the environment recognition and modeling strategies discussed in chapters 4 and 5 it is useful to understand the kind of information that is available to the robot. The first objective of this chapter is to give an overview and assessment of the main existing perception sensors available. In order to do that, the difference between the different types of perception information is explained. The conclusions reached in this chapter set the path to the development of a perception system (chapter 5) that is as general as possible in order to be applicable to other systems and scenarios. The second objective of this chapter is to discuss the integration of these sensors into a robotic system and present new calibration strategies for imprecise sensors in humanoid robots (as well as manipulators). Even though these calibration procedures are fundamental for a correct performance of a robotic systems, they are difficult to find in the existing literature. The topics discussed

[1] Interestingly, echolocation capabilities can sometimes also be found on humans [183]

Figure 3.1: Unidirectional sensors are usually applied for proximity detection. Among the most popular types are ultrasonic (left) or infrared (right) [170].

throughout this chapter form the basis to adapt the framework presented in this thesis to other robotic systems.

Perception Sensors

Due to the vast amount of existing sensors and information available on them, some system of classification becomes useful when analyzing a robotics application. One popular criterion is the one presented by Christensen et al. [25], who classifies robotic sensors according to sensing objective and method. For example, in the case of GPS, the sensing objective is exteroception and the method is "active". A further classification is made based on typical application (such as haptic sensors or speed/motion sensors). Among those used for perception information, only sensors that do not require contact (such as tactile or haptic sensors) are considered here. The reason behind is that the planning of the robot's future motion depends on the environment information being available in advance. Contact sensors are used in a further stage, as shown in chapter 6. Beacon based sensors are also not considered as they are usually used for the inverse problem: finding the location of the sensor with respect to an external frame of reference which is known, instead of finding information about the environment surrounding the sensor [25]. As the focus of this work lies on the application of perception sensors (and not the principles behind them), these are classified according to their output for the purposes of this thesis:

(a) unidirectional sensors (e.g. ultrasonic),

(b) 2D sensors (e.g. monocular cameras) or

(c) 3D sensors (e.g. lidars).

In fig. 3.1, two examples of unidirectional sensors can be found. Naturally, the use of mechanical devices can turn any directional sensors into full-range 3D sensors (as will be explained later). Sonar sensors, for example, have been used by some authors to obtain environment maps [122] in the past. Nevertheless, even though the use of advanced signal processing algorithms has increased the resolution and applicability of these sensors, their low accuracy and focusing capabilities compared to other technologies still prevent their application as reliable range sensors [96]. Even though these and other kinds of unidirectional sensors can be used for simple obstacle detection [25], advanced navigation algorithms depend on more complex environment representations [49]. Therefore, they are discarded for the present application.
In the case of direct 2D computer vision, a similar problem is encountered. Pure monocular vision is typically used for abstract recognition or scene understanding [162]. For 3D navigation applications, cameras are usually found in pairs (also called "stereo cameras") with a

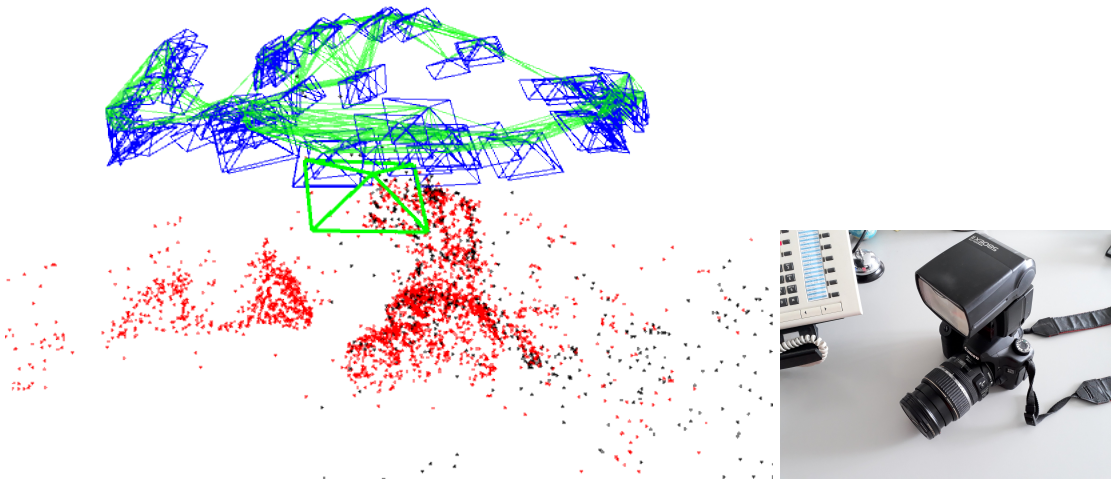


Figure 3.2: Monocular SLAM. Using single RGB images from different viewpoints a 3D point cloud can be obtained and the camera itself can be tracked.

fixed distance between them so that the stereo problem can be solved and thus 3D information can be obtained [49]. However, assuming that the stereo matching problem is solved (there are many proven algorithms to do it, see Lazaros et al. [104]), these cameras can be considered as 3D sensors due to the information they provide.
Recently, some authors have shown autonomous navigation using only a monocular sensor [157]. However, this is achieved by creating a 3D map out of different monocular images. In other words, by solving the stereo problem *a posteriori* out of different, non simultaneous perspectives from a single camera (see fig. 3.2). This field of research is often called *Structure from Motion* (SfR) or *Monocular Simultaneous Localization and Mapping* (Monocular SLAM) and has been actively developed in recent years [40, 127, 132]. While achieving impressive results, even in real-time, these methods have several drawbacks. First and most importantly, as there are no references, these methods cannot correctly estimate scale [181]. Secondly, as they depend on easily recognizable, invariant features in the scene, they can loose tracking from time to time [127]. For the same reason, they are not robust to dynamic environments where those same features may move independently from the camera. These drawbacks can be compensated using external references, such as IMU data or machine learning [181], but these techniques are relatively new and still in the experimental phase.
In order to be able to work in complex environments and ensure a safe interaction with them, robotic navigation systems use as much information about their surroundings as they can possibly get. That is why almost all perception systems are based on some kind of 3D sensor (also called “range sensors”), sometimes combined with other sensors as well [49]. 3D sensors are used to create an accurate representation of the environment, which can be later used for processing (see chapter 4). In order to ensure maximum compatibility with other robotic systems, this work is based on standard 3D information and not especially restricted to a particular kind of sensor. In the following, the most popular types of 3D sensors are mentioned, together with their most relevant characteristics.

- **Stereo Cameras**. These sensors consist of two monocular cameras at a known distance and angle from each other. Similarly to the human vision sense, distances can be calculated by matching image features between both cameras and using the known transformation between them as reference to solve a triangulation problem (see fig. 3.3). While they have been the subject of intensive research, stereo-solving algorithms can be con-

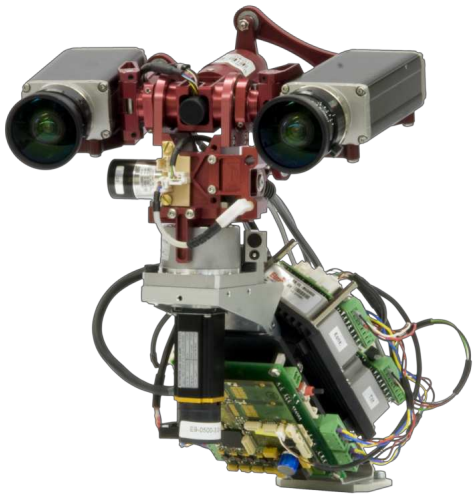


Figure 3.3: Stereo Cameras on the *Lola* robot. Previously, *Lola* was equipped with a pair of monocular cameras for autonomous navigation [16].

sidered state-of-the-art nowadays [104]. These sensors have the advantage that the 3D information is inherently matched against the RGB image, allowing for interesting computer vision algorithms. Additionally, they can work outdoors (provided lighting conditions are favorable) and can have high precision over a range depending on the distance between cameras (error increases for points too far away due to triangulation). Even though higher precision comes with the cost of more requirements on computer power, there already are cameras which are able to provide near real-time information by solving the stereo algorithm directly on the computer's GPU. The ZED camera [209] is a perfect example of this, providing up to 60 Hz update rates using standard hardware.

One important disadvantage is that images have to be rich in features in order to be able to match points between both images and texture-less areas will not be recognized correctly. This can sometimes be solved using *active stereovision*. The Intel RealSense [79] is a popular stereo camera that actively projects patterned light to the scene to improve performance on texture-less surfaces but computing requirements are high. Other interesting works consist of modifications to improve the field of view (FoV) such as the one by Jamaluddin et al. [81], who present a combination of a stereo camera with additional fish-eye cameras for this purpose. These are areas of ongoing research. It is worth mentioning that these are the main kind of *passive* sensors, while almost all other 3D sensors are based on actively projecting the measurement source into the environment (explained in the following).

- **Time-of-Flight Sensors**. The name of these sensors is self-explaining. By projecting a beam into the environment and measuring the time it takes for the reflection to reach the sensor the distance to the reflection point can be measured (assuming a known and constant beam velocity, naturally). The easiest way to do this is using ultrasound due to its relatively low velocity. However, as explained before, its resolution is not good enough for reliable map generation [96]. However, light can be used as well. The oldest and most known of these sensors in popular culture is the RADAR, which has wide applications in the aerospace and military sectors but poor spatial resolution and high cost has prevented its use in robotic applications [25].
  Sensors that use a laser source are called LIDARs or LADARs [49]. Among its multiple

Figure 3.4: Autonomous cars depend on LIDAR sensors for perception. Here such a sensor can be seen on top of an autonomous vehicle prototype by Google's spin-off Waymo [194]. Adapted from Grendelkhan [56].

advantages, they provide high robustness (they work outdoors under varying conditions), high resolution, low level of noise and greater field of view - usually, by means of a rotating mirror [49]. These advantages have made these sensors the most popular sensors for high-end robotic applications such as autonomous navigation [77, 151] or biped locomotion [47]. Some authors have even proposed object recognition applications in larger areas such as airports [126]. However, their high price still prevent them from being used more often. Due to the industry's effort to make autonomous cars affordable (see fig. 3.4), an objective which is mainly hindered due to high sensor costs, there are now several companies that promise to provide low-cost LIDAR sensors in the near future [1]. This promises to have an enormous impact on robotic applications.

- **Modulation Range Sensors**. Instead of timing, these sensors measure a shift in phase between the projected and reflected light. This makes them less complicated and cheaper to produce which makes them more popular among the community [27]. They generally do not perform very well outdoors and have generally less precision and robustness than time-of-flight sensors [49]. However, in order to overcome these shortcomings, some authors have proposed a combination between modulation sensors and 2D sensors using low-cost components [52].

- **Triangulation Range Sensors**. Triangulation refers to the process by which depth information of an observed point in space can be inferred if such point is observed from two different viewpoints. As explained before, this is a basic step of stereo cameras, where the most difficulty consists of matching features between both images. The term *triangulation sensors* is applied to those sensors which manage to overcome the matching step so that only the triangulation step has to be performed. Typically this is done by projecting a laser beam of a particular frequency on the scene, which can be easily identified by a corresponding sensor. Thus, by knowing exactly in which direction the beam was projected and unequivocally recognizing it on a displaced sensor depth can be obtained via triangulation [49].
  If, instead of projecting a single point on the scene, several are projected simultaneously using a known pattern which can also be easily identified, 3D information can be obtained with higher update rates. Sensors using this technology are often called

Figure 3.5: Structured Light Sensors: Microsoft's original Kinect [120].

Structured Light Sensors[2]. As their frequencies usually lie in the infrared range these sensors do not perform well outdoors (or with reflective surfaces). Additionally, some regions can be worse recognized than others, introducing noise (these sensors usually present higher levels of noise than other kinds, see Khoshelham et al. [95]). Nevertheless, despite presenting high levels of noise, they can be relatively robust (they don't depend on textured surfaces) for a smaller price than other sensors. A few years ago, Microsoft commercialized such a sensor, called *Kinect*, as an accessory to their Xbox console to recognize gestures (see fig. 3.5). However, its ability to provide ready 3D information for a very low price (thanks to mass production) and adequate support made it a perfect tool for robotic research applications [102], completely transforming the field [210]. These applications include not only tracking of people and gestures [112], but robotic navigation as well [80, 95]. Other companies quickly provided compatible sensors (using the same technology developed by the company PrimeSense, see Zhang [210]) with wider support for researchers and different operating systems.

Sensor Requirements for Autonomous Navigation

When choosing any kind of sensor, its requirements are determined by its application in the overall system. The objective of a perception sensor for autonomous robotics is to provide the most complete, robust and accurate information about the environment surrounding it. As explained before, of all perception sensors available, 3D sensors are the ones which provide most information about the environment for the purposes of autonomous navigation. However, choosing the right 3D sensor for a robotic application is not an easy task due to the many technologies available. Moreover, these kinds of sensors are being continuously improved [1], which makes the choice even more difficult. For this reason, it is important to remain flexible: the complete system should be adaptable to a possible sensor update in the future.
In the field of 3D computer vision, the availability of small and inexpensive 3D sensors [210] motivated new research and developments in 3D data processing (see for example Rusu et al. [160]). The most general representation of 3D information is a 3D point cloud (PC), which consists of a set of 3D coordinates, corresponding to the discrete measurements of the environments (analogous to pixels of an RGB frame). If color information is available as well, as is the case of stereo cameras, it can be either passed along as a separate RGB frame

[2]Sometimes these sensors are referred to as "2.5D sensors", as they provide a depth map measured from the sensor's image plane. However, any kind of localized sensor is affected by occlusion and limited by the sensor's position. Weather the information is obtained as a depth matrix with a fixed FoV or by a revolving laser unit, it can still be expressed as a point cloud. Therefore, the term "3D sensor" is loosely used to refer to any sensor capable of providing information in the form of a 3D point cloud throughout this work, including structured light sensors.

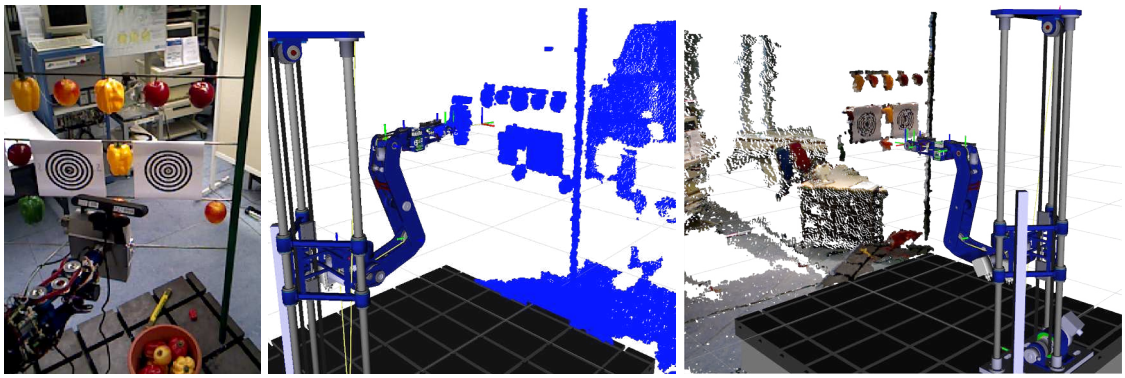


Figure 3.6: Different levels of information. Left: an RGB image from a *CROPS* scene using a 2D camera. Middle: a 3D point cloud obtained by a structured light sensor against the robot's simulation. Right: the same point cloud with color information. Adapted from Grundner [57].

or included in the point cloud by associating an RGB value with each coordinate, resulting in a "colored" PC (see fig. 3.6). In this context, the open-source *Point Cloud Library* (PCL) [144] has been developed. It provides tools to efficiently handle and process 3D point clouds (PCs) (for further information about its basic principles and algorithms, see Rusu [158]).
In order to obtain flexibility and adaptability to different sensors and systems, it was decided to develop a vision system based on direct 3D point cloud processing for this work (see chapter 5). That way, the sensor becomes interchangeable and the vision system can be applied to other robots as well. Moreover, it was decided to rely only on 3D information (and not on RGB), to make this flexibility as broad as possible (most 3D sensor types don't provide color information). To summarize, the perception sensor for this application just has to provide 3D information (in the form of a PC) and not have any special characteristics that would make it irreplaceable.
However, there are still some considerations which must be taken for choosing a sensor that directly depend on the considered application of autonomous robotic navigation in unknown environment:

- **Framerate**. For real-time applications, it is important to consider the frequency of information acquisition. It is usually measured in Hz or frames per second (fps). From experiments performed, it has been found that a 10 Hz update rate (or higher) is enough to reliably consider different scenarios while taking the robot's motion and reaction times into account. However, higher update rates allow for more accurate environment modeling, especially in dynamic scenarios (see chapter 5). Additionally, some sensors such as pivoting laser units require the robot to stand still while scanning and become less reliable in dynamic environments. For this reason, they are discarded for the present application.

- **Field of View** (FoV). When navigating through unknown terrain, the robot should be able to gather as much information about its surroundings as possible. However, the section of the environment right in front of the robot tends to be the most relevant. In this work, the field of view was chosen as a function of the tested scenarios and available sensors. The developed algorithms can be applied regardlessly of how much of the surroundings are detected, though naturally the degree of robustness and ability to navigate unknown environments depend on the information available.

- **Range**. A sensor has to reliably cover the area surrounding the robot to provide complete information to the motion planner. In full-size humanoid applications the first few

centimeters in front of the sensor are usually not considered (if something is detected in this range it is already too late to react to it) while information further than 5 m away become less relevant when considering dynamic scenarios and real-time capabilities.

- **Accuracy**. As explained in chapter 6, errors of more than a few cm in perception information can have disastrous consequences for a humanoid robot. According to our experiments, sensors with a better accuracy than 5 cm are desirable.
- **Compatibility**. As explained before, the objective of this project is to provide a system as open and flexible as possible. That means that sensor information should be easy to access and compatible with standard tools (such as PCL) to make them interchangeable.

There are numerous sensors that comply with the mentioned characteristics. It was found out, however, that the most critical aspect was the sensor's accuracy. Detecting a platform in the wrong location can potentially destabilize the robot (see chapter 6). However, if an extremely accurate sensor is chosen, there is a risk of developing an application that is strongly dependent on high sensing accuracy and thus incompatible with many other systems. Moreover, it could limit the application of the robot to a restricted kind of scenarios: certain kinds of terrain, such as grass or other non-rigid floors are inherently difficult to model correctly, regardless of the sensor used.
Therefore, it was decided to use a standard structured light (or RGB-D) sensor for this work, in order to ensure compatibility with other systems and scenarios (see fig. 1.4). Authors usually name two main disadvantages of these kind of sensors, which are their low accuracy and poor performance under sunlight. However, choosing such a sensor serves to ensure compatibility with other systems:

- As *Lola* is developed as a prototype to be used inside the laboratory, the sunlight incompatibility is not an issue at this stage of research.
- The sensor's low accuracy serves as a performance test. A system that performs correctly with this sensor is clearly compatible with more accurate sensors.
- Additionally, the low accuracy serves as a simulation of real world scenarios. The robustness of the robot against irregular terrain can be tested by performing laboratory experiments over regular surfaces with inaccurate perception information (see chapter 6).

To summarize, the objective is to develop a framework as general as possible that can be applied to other systems and scenarios. If it performs correctly with an inaccurate sensor, it will probably perform correctly with other sensors or scenarios as well.

Calibration

Regardless of the sensor chosen, a calibration process always has to be performed in order to make the sensor information compatible with the planning system. A 3D sensor provides spatial information with respect to a local coordinate system, which almost always differs from the one used by the robot's motion planner. Most often sensors don't even provide a physical indication of their reference coordinate system. Ideally, all sensory information should be joined together into one coordinate system to make it compatible with the rest of the robot's modules. If many different sensors are available, this process is known as "sensor fusion"[37]. Before continuing, it is important to make a distinction between two main kinds of calibration:

(a) **Intrinsic Calibration**. Due to errors during fabrication, differences in temperature or other conditions, each particular sensor may provide slightly different information with varying degrees of accuracy. If it is compared to a reference, more accurate sensor, a calibration function may obtained to correct data before its processing.

(b) **Extrinsic Calibration**. After the intrinsic calibration is performed, the sensor itself must be calibrated against the robot in order to correctly identify its coordinate system. If a physical procedure is not supplied by the sensor's provider, the extrinsic calibration usually consists of identifying some reference feature (that is already calibrated against or even part of the robot itself) from the sensor information and then matching both references.

Therefore, the intrinsic calibration is the process performed on each particular sensor to correct fabrication errors, while the extrinsic calibration has to be performed when mounting the sensor in each particular robot to make its data compatible with the robot's control. As it is strongly dependent on each particular kind of sensor, the intrinsic calibration is only briefly mentioned here for the chosen sensor. On the other hand, the extrinsic calibration depends mostly on the particular robot and different systems require different procedures. In this section, a standard extrinsic calibration process is mentioned for the *CROPS* robot while a new one is introduced for the *Lola* robot.

Intrinsic Calibration

In the previous section, it is explained how the choice of an RGB-D sensor was done after considering that a noisy sensor would ensure the system's robustness against other sensors and scenarios. However, RGB-D sensors are not intended as measurement devices and their errors are higher than the ones admissible by *Lola*'s control system (see chapter 6). Nevertheless, a simple intrinsic calibration procedure helps greatly reduce them. It is based on the works by Jalobeanu et al. [80] and Khoshelham et al. [95], where the interested reader may find additional information. It is important to know that the greater source of error in these systems comes from the miss-alignment of the structured light projector and sensor, which results in a triangulation error. In order to solve this, Jalobeanu et al. [80] mounted the sensor's internal components into a fixed metal case. However, for many applications, a simple correcting function is sufficient [95].
A simplified characterization of the sensor was performed against a laser range sensor[3], measuring perpendicularly and averaging along a plane in the central half FoV of the camera. Several sensors were tested, each of them presenting a similar response. The measurement error can be, for present purposes, roughly approximated using a linear regression line [95], which significantly improves the sensor's precision (see fig. 3.7).

Extrinsic Calibration on Manipulators

If a sensor is mounted on a robot in such a way that part of the robot itself lies inside the sensor's FoV, the extrinsic calibration can be performed straightforward with the help of markers and fixed references to the robot. The interested reader may find a thorough analysis in the work by Wang [193]. Here it is shortly illustrated taking the *CROPS* robot as an example (the methods presented are the result of a collaboration with Sahand Yousefpour [208]). In the current setting, an RGB-D sensor is mounted on top of the robot's base (see fig. 3.8). This

[3] Bosch PLR 50 Digital Laser Measure, Typical accuracy = 2.0mm.

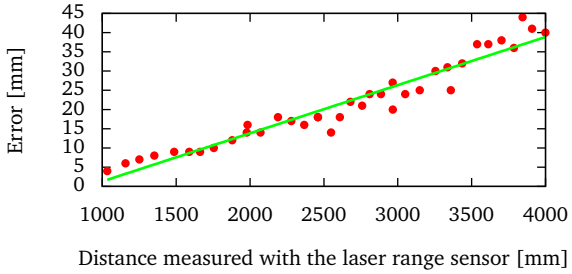


Figure 3.7: Characterization of the Xtion Kamera. Error of the measurements (red) together with the linear regression approximation (green)

stationary position ensures maximum possible field of view towards the workspace environment, where fake lab fruits are hanged for the robot to pick them up.
The objective of the extrinsic calibration is to find the precise location and orientation of the sensor with respect to the robot's *world frame*. In order to do this, an initial "guess" is provided based on the position where the camera is mounted. Then, if a closed chain of known transformations that include the sensor itself can be obtained, the error of that closed chain can be interpreted as the transformation between the initial guess and the actual sensor pose. The whole idea consists of using a known reference which can be identified by the sensor and precisely located from the robot. In this case, the simplest reference consist of a marker - which can be easily identified by the sensor using standard tools such as the ArUco library [53] - attached to the manipulator, as can be seen in fig. 3.8.
This way, the following closed chain of transformations is obtained (ref. fig. 3.8):

$$ {}^{M}_{tcp}T \; {}^{tcp}_{W}T = {}^{M}_{C}T \; {}^{C}_{\hat{C}}T \; {}^{\hat{C}}_{W}T \tag{3.1} $$

where ${}^{B}_{A}T$ denotes the transformation from frame A to frame B, following the convention of Craig [26]. ${}^{C}_{\hat{C}}T$ is the transformation between the sensor's initial guess and it's actual pose, and is the unknown in eq. (3.1). ${}^{tcp}_{W}T$ contains the transformation between the robot's world coordinate system and the end-effector pose (measured by the robot) during the marker detection step. ${}^{M}_{tcp}T$ is a fixed transformation between the end-effector and the marker board. ${}^{\hat{C}}_{W}T$ is the guessed pose of the sensor in the robot's world coordinate system and ${}^{M}_{C}T$ is the pose of the marker board measured by the sensor.
If, instead of obtaining one closed chain, the process is repeated for a number of different observations, the real sensor position can be more accurately obtained via least squares. The manipulator is instructed to move to different positions, making the system observe the marker. At each position, an average value of marker poses are computed to eliminate any possible local error. Then, a set of equations (3.1) for the different detected values of ${}^{M}_{C}T$ is obtained. For the sake of implementation, the transformation attributes are decoupled into rotation and translation. The rotational component of eq. (3.1) reads as:

$$ {}^{M}_{tcp}R \; {}^{tcp}_{W}R = {}^{M}_{C}R \; {}^{C}_{\hat{C}}R \; {}^{\hat{C}}_{W}R \tag{3.2} $$

where ${}^{C}_{\hat{C}}R$ is unknown; eq. (3.2) can be rewritten as:

$$ J = \left({}^{M}_{tcp}R \cdot {}^{tcp}_{W}R\right)^{-1} \cdot {}^{M}_{C}R \cdot {}^{C}_{\hat{C}}R \cdot {}^{\hat{C}}_{W}R \tag{3.3} $$

where J is a 3x3 matrix which, in an ideal case, would be equal to the identity matrix. However, during multiple observations, system errors result in different solutions to eq. (3.2).

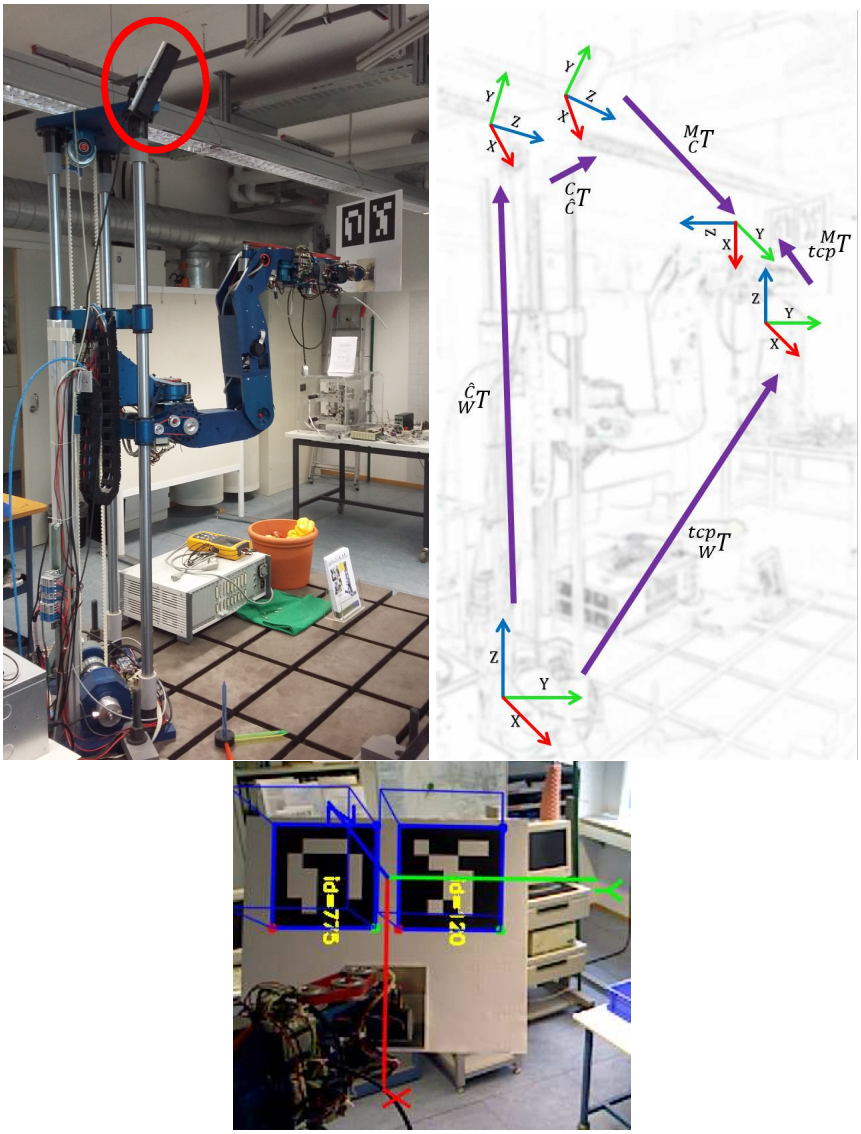


Figure 3.8: Extrinsic calibration using markers. Top: robot with sensor (indicated in red) and marker and the corresponding kinematic chain. Bottom: recognition of the marker pose using standard tools. Adapted from Yousefpour [208].

The least squares method finds an optimum value for ${}^{C}_{\hat{C}}R$ that "minimizes" the different values J when the sum, S, of squared residuals:

$$S = \sum_{i=1}^{n} {}_{R}r_i{}^2 \tag{3.4}$$

is minimum, where n is the number of observations. To build the rotational residuals ${}_{R}r_i$, the Euler angles are extracted from each J_i matrix:

$${}_{R}V_i = \begin{bmatrix} \alpha_i & \beta_i & \gamma_i \end{bmatrix}^T \tag{3.5}$$

where α, β and γ denote the rotation around x, y and z axes respectively and the rotational residuals become:

$${}_{R}r_i = \|{}_{R}V_i\| \tag{3.6}$$

Thus, the value of ${}^{C}_{\hat{C}}R$ that minimizes the residual angles can be found.
The translational component of ${}^{C}_{\hat{C}}T$ is computed analogously. Similar to eq. (3.3), rearranging eq. (3.1) gives:

$$K = \left({}^{M}_{tcp}T \cdot {}^{tcp}_{W}T\right)^{-1} \cdot {}^{M}_{C}T \cdot {}^{C}_{\hat{C}}T \cdot {}^{\hat{C}}_{W}T \tag{3.7}$$

where K is a 4x4 matrix. The first three components of the last column of this matrix (containing the translation) are extracted for each equation:

$${}_{T}V_i = \begin{bmatrix} x_i & y_i & z_i \end{bmatrix}^T \tag{3.8}$$

and the translational residuals are computed as:

$${}_{T}r_i = \|{}_{T}V_i\| \tag{3.9}$$

Applying least squares yields the optimal transformation ${}^{C}_{\hat{C}}T$ and the sensor is now calibrated against the robot, resulting in the coherent point clouds shown in fig. 3.6.

Extrinsic Calibration on Humanoids

The extrinsic calibration of a sensor on a humanoid robot follows the same idea as the one on a manipulator, but with a fundamental difference: there is no section of the robot inside the camera's FoV and the range that is of interest is the area away from the robot (calibrating using the robot's hand would not yield the desired results due to the varying performance of the sensor over distance). Additionally, the robot is not fixed to the environment, which complicates the task of finding an adequate reference. Unfortunately, even though the calibration of a sensor of a humanoid robot is a topic of interest for the community due to its unique characteristics, authors systematically omit this subject in publications. Moreover, most other works depend on off-board systems. In this work, a simple but effective calibration routine and tool for on-board sensing are proposed which may be useful to future researchers. The idea consists on using the environment itself as reference.
Rewriting eq. (3.1) for this case leads to:

$${}^{E}_{W}T = {}^{E}_{C}T \; {}^{C}_{\hat{C}}T \; {}^{\hat{C}}_{W}T \tag{3.10}$$

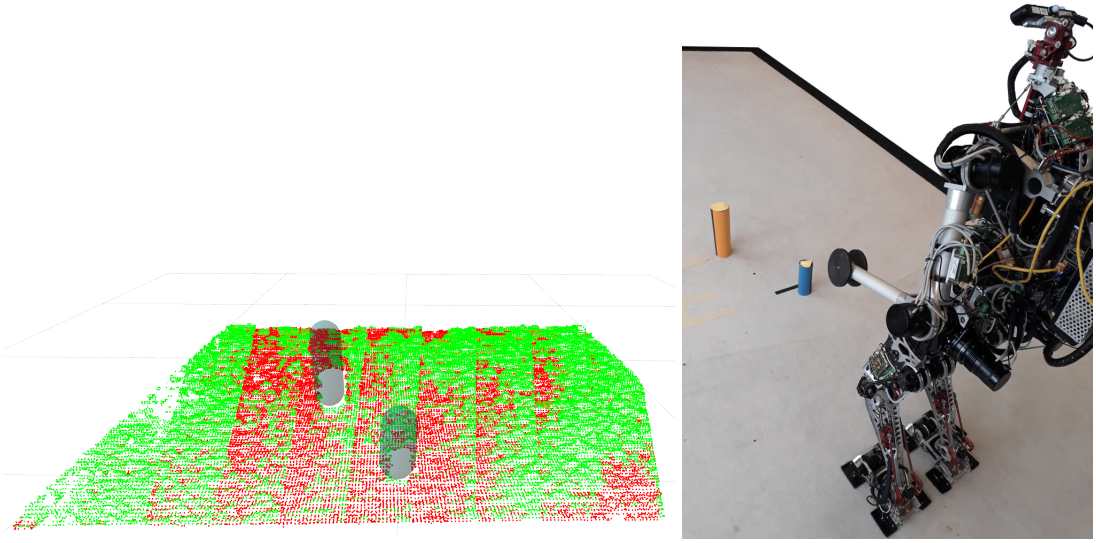


Figure 3.9: Extrinsic calibration using the environment as reference. Right: the robot is placed on a flat ground with a couple of simple objects in front. Left: both objects and the ground can be identified and located using a simple tool (green and red represent points above and below the ideal ground, respectively).

where E refers to the environment reference. Similarly to eq. (3.2), the rotational components of eq. (3.10) can be written as:

$$ {}^{E}_{W}R = {}^{E}_{C}R \; {}^{C}_{\hat{C}}R \; {}^{\hat{C}}_{W}R \tag{3.11} $$

If the robot is placed on a flat ground and the stabilization control is active (see chapter 2), the feet - as well as the robot's local coordinate system - can be assumed to be parallel to the ground (or as parallel to the ground as they can be). If the ground itself is E, ${}^{E}_{W}R$ becomes the identity matrix. The orientation of the ground as seen from the camera ${}^{E}_{C}R$ can be obtained by fitting a plane to the detected point cloud. However, the detected point cloud shows only an homogeneous section of the ground. Applying a vertical rotation or an horizontal translation to the camera would result in an identical point cloud, assuming the ground is perfectly flat. Thus, the orientation around the vertical axis is thereby not determined. Similarly, the translation along the vertical axis can be obtained from the plane identification but not the other components of the translation.
In order to find these unknowns, a few simple objects that can be easily identified (chapter 5) are precisely placed in front of the robot at known distances from the robot's feet (see fig. 3.9). This way, the identification of these objects together with the ground can be combined to obtain ${}^{E}_{C}T$.
Based on the author's experience, RGB-D sensors exhibit variable behavior with time and temperature, reaching a stationary state after around one minute. Additionally, high levels of noise may compromise the whole calibration process. It becomes relevant, therefore, to visualize the calibration results online. Using a simple tool which is published open source (see chapter 5), live results from the calibration chain can be projected to a viewer (see fig. 3.9) which indicates both positon and orientation of the ground and obstacles (these are detected using the algorithms presented in chapter 5). By manually changing the rotational and translational components of ${}^{C}_{\hat{C}}T$ the extrinsic calibration can be easily completed without the risk of reaching local minima with an unsupervised least squares method. The viewer calculates and displays the mean vertical value of all detected ground points and their variance against the mean horizontal plane, which can be used as indicators for the manual calibration (see algorithm 1). Nevertheless, this process can naturally be automated if needed (for example,

when using more accurate and reliable sensors). This tool can be additionally used to perform the intrinsic calibration mentioned earlier, by detecting the wall instead of the ground; in this case, the variance is used for aligning the 3D sensor and the mean is compared against an accurate unidirectional sensor (such as a laser range sensor).

Algorithm 1 Extrinsic Calibration on a Humanoid Robot

1: Start the calibration tool to display the *mean* value (along the vertical axis) and the *variance* value (against the mean horizontal plane) of the ground plane as well as the locations of detected obstacles
2: **repeat**
3: Modify the rotational components (Euler Angles) of ${}^{C}_{\hat{C}}T$
4: **until** the *variance* reaches a minimum
5: **repeat**
6: Modify translational components of ${}^{C}_{\hat{C}}T$ (and the rotation around the vertical axis)
7: **until** the *mean* is closest to 0 and the object's locations are correct

Chapter 4

Environment Representation

Interpretation of the Environment: Recognition vs. Modeling

In the previous chapter, different perception technologies were discussed and the integration of 3D sensors into a robotic system was presented. These sensors provide information in the form of a 3D point cloud which, by itself, is useless for robotic navigation. In order to be able to navigate along the sensed environment, this information must be *interpreted* in a meaningful way. This process is often called 3D image processing. In the particular case of autonomous navigation, there are two main strategies to do this.

The first one is called *Environment Recognition* and consists of finding previously known components (such as objects) in an image with the aim of executing predefined actions. This has been the topic of extensive research, both in 2D and 3D image processing [36]. It has the advantage of reducing the number of unknowns in complex scenarios: if the main components of such a scenario are known, predefined actions can be taken reducing both planning time and failure risk. This strategy is widely used for robot grasping and manipulation as shown below, but is very limited for robot navigation as it cannot easily deal with new, or previously unknown, scenarios.

On the contrary, the second strategy, *Environment Modeling*, specifically deals with these scenarios. Its main idea consists of generating an abstract, geometrical model of the environment which can later be handled by a corresponding planning system. For example, in the case of a small wheeled robot navigating on a flat surface the environment could be represented by a 2D map, which would suffice for that particular problem. In general, the adequate modeling strategy differs greatly depending on the application: an autonomous rover navigating on the open surface of mars will require different level of detail and world representation than an autonomous quad-copter flying inside a factory.

In recent times, however, new developments in machine learning methods as well as available computational power are increasing the applicability of environment recognition methods: by using large amounts of data, more and more components of the environment can be recognized, even if they don't exactly match the previous ones [162]. Using an everyday example, Google Photos can presently filter pictures containing elements such as dogs, sunsets, or a beach scenery with impressive accuracy [109]. It is possible that, in the future, systems based on sufficiently large amounts of data are able to correctly interpret all components of the scene in which a specific robot may find itself.

Nevertheless, these environment recognition methods cannot guarantee a 100% effectiveness and there will always be cases where unknown scenarios will have to be dealt with. Moreover, some applications such as collision avoidance can easily and effectively be solved by an environment modeling strategy and there is no need to rely on these more complicated and less reliable methods. The author considers that ideal perception systems for advanced robots should include a combination of both strategies: sections of the environment that cannot be correctly (or confidently) recognized shall be approximated using a modeling strategy

to ensure correct behavior. An example of such a combined environment recognition and modeling system is presented in this chapter, where a robot performs manipulation tasks on known objects while avoiding unknown obstacles.
The rest of this thesis deals mainly with environment modeling strategies. The objective is to provide a system capable of dealing with completely unknown environments for robot navigation which could also be combined with environment recognition algorithms in the future. After the aforementioned example, existing environment modeling strategies along with their limitations are discussed. Finally, a new environment modeling strategy with key advantages for humanoid navigation is presented. Some of the results presented in this chapter have been previously published in more compact form in international journals and conferences [69, 189].

Example of a General Perception System: the European Robotics Challenges

In order to illustrate the perception system concept, an example is given in the context of an industrial application (a more complete overview on environment recognition algorithms can be found in Daniilidis et al. [30]). In 2014, the EU-funded project European Robotics Challenges (*EuRoC*)[1] [44] was started. In three concurrent tracks, competitor teams from European research institutes and universities are developing integrated robot systems for autonomous manufacturing and inspection applications in close cooperation with technology suppliers, system integrators and end-users. The team *AM-Robotics*, composed of four PhD candidates of the Chair of Applied Mechanics (including the author) and nine students, obtained first place (among 39 international contestants) during the first phase of the *EuRoC*'s Challenge 2. This track consists of the development of autonomous robot systems for use in logistics and as robotic co-workers. The challenge platform is a KUKA omniRob (consisting of a mobile platform with a KUKA iiwa manipulator and several sensors on top, see fig. 4.1). For the first phase of the challenge, an intelligent and flexible framework for autonomous pick-and-place tasks in previously unknown scenarios was developed using the Robot Operating System (ROS)[152]. The vision module, for which the author was responsible, combines algorithms for environment recognition and modeling and is presented in this section.

Framework and Software Architecture

The classical application of robots in industry involves repetitive processes. A manipulator must follow pre-defined motions for specific tasks in completely known and enclosed environments. New robotic systems with enhanced sensing capabilities allow for more complex applications, where the robot shares a dynamic workspace with humans. In order to exploit this potential, the robot has to adapt to changing conditions and tasks. The use of additional sensor feedback offers many options for these applications, including detection of various objects and obstacles, safe navigation in human workspaces and precise positioning with known uncertainties. Manipulator platforms are able to combine advanced sensing and handling capabilities in an autonomous, mobile unit that seems ideal for these kinds of applications.
Several of these platforms with integrated sensors and control systems for autonomous tasks have been presented for academic and research use (e.g. *PR2* by *Willow Garage* [198], *youbot* by *KUKA* [101]). In robotics research, algorithms and software frameworks have been developed over many years aimed at achieving high levels of autonomy. However, autonomous

[1] European Robotics Challenges, 2014–2017, Grant Agreement No.608849.

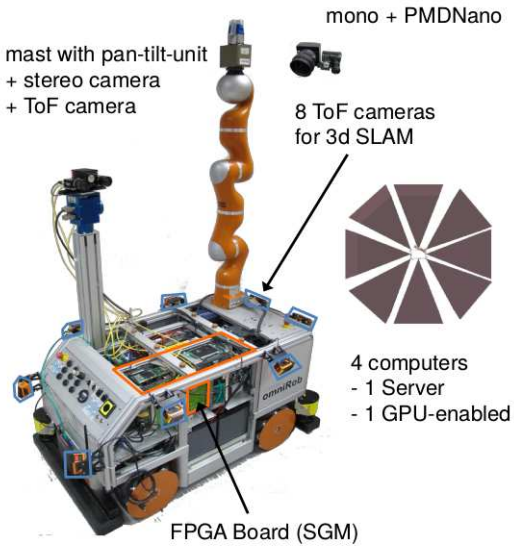


Figure 4.1: Challenge platform for Challenge 2 of the *EuRoC* [44].

mobile platforms for manipulation tasks are not common in industrial production since they have not yet shown the necessary level of robustness [69]. At the time of the challenge, to the best of the author's knowledge, no appropriate framework for the tasks of the *EuRoC* were available. For that reason, the team *AM-Robotics* developed an in-house approach which was released as open source[2], in the hope that it helps not only other *EuRoC* teams but other researchers in the field as well.

Stage 1[3] of the *EuRoC* took place between July and November of 2014 and consisted of different pick-and-place tasks in a simulated environment (see fig. 4.2). The simulated robotic system (running on Gazebo [54]) consists of a KUKA iiwa 7 DoF manipulator with gripper mounted on a 2 DoF mobile basis, emulating the omniRob platform. Two simulated RGB-D sensors with artificial noise record the scene, one is mounted on the gripper and the other one on a fixed stand with pan-tilt-unit. Both the pan-tilt-unit and the manipulator's DoF can be position-controlled. The tasks to be solved consist of different pick-and-place scenarios, which are previously unknown.

During each task, the simulator provides a list of objects to be found. These are defined by their respective geometry and color, selected from a predefined set. The objective is to find and place those objects according to a predefined sequence in different kinds of pick-and-place scenarios that include obstacles, puzzle solving and moving objects.

The framework consists of independent modules, each of which is responsible for a certain subtask of the overall pick-and-place process. These are: state machine, state observer, vision system, grasping and motion planning. An overview of the overall system architecture is shown in fig. 4.3.

The central communication and coordination, including the overall task sequence and the interaction between the modules, is performed by the state machine. The sequence to solve a basic task is depicted in algorithm 2. To maintain consistent coordinate systems between the main modules, an additional supporting node is introduced. It uses the ROS tf-package and continuously publishes all relevant coordinate transformations. The main modules use it to express location data in the "world" coordinate system. The vision module, for example, transforms the data from each sensor into world coordinates.

[2]Available under https://github.com/AppliedMechanics/EuRoC.

[3]The challenge is divided into 3 stages: Stage 1, simulation contest; Stage 2, realistic labs; Stage 3, field tests.

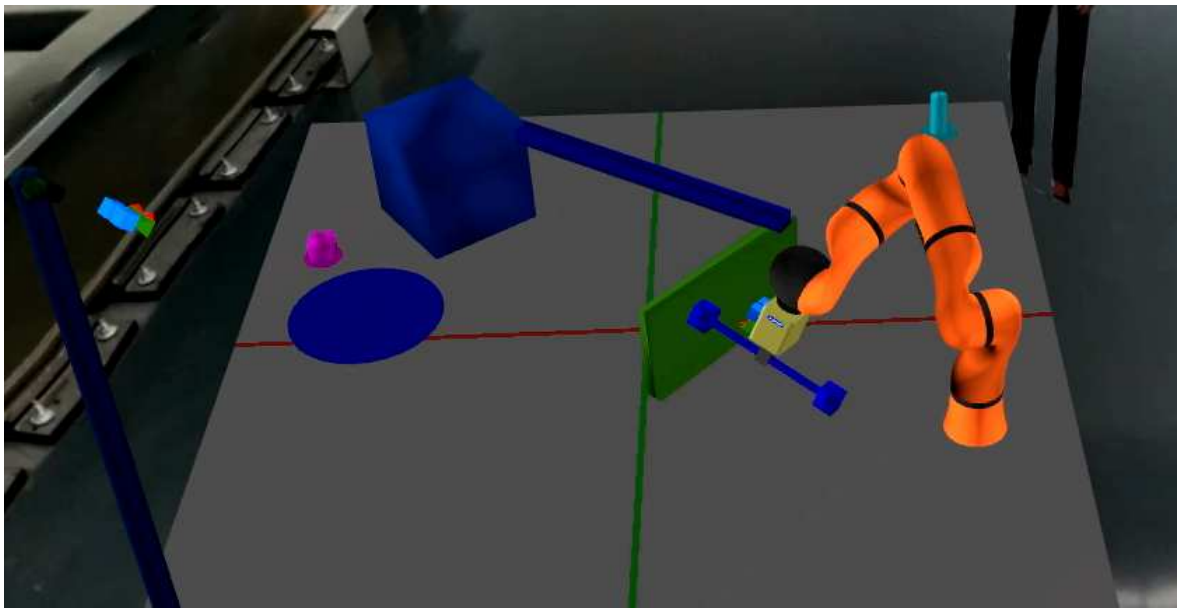


Figure 4.2: Simulation environment of course 2, stage 1. The manipulator (7 DoF) has a movable base and performs several pick-and-place tasks while avoiding obstacles. All environment data is based on RGB-D sensors mounted on a stationary pan-tilt unit and on the manipulator's end-effector.

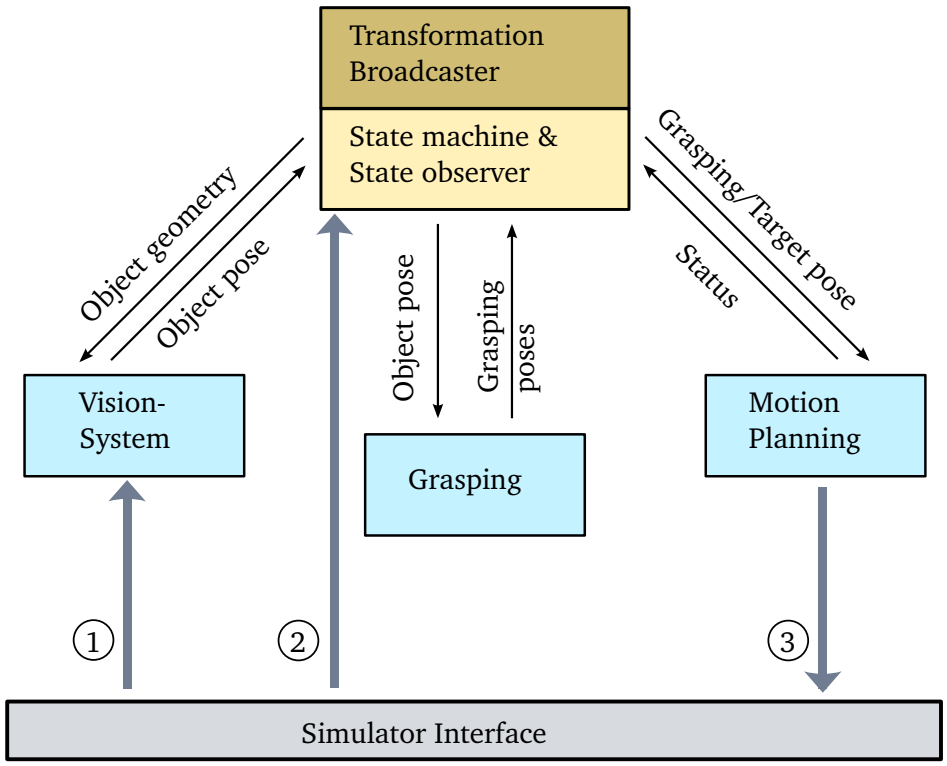


Figure 4.3: Control system (challenger application) overview including the main modules and communication data.

Algorithm 2 Pick & Place control sequence

```
1: Request task & start simulation
2: Get task-description
3: Scan environment
4: Set n = 1
5: repeat
6:     Locate Object n
7:     Calculate possible grasping poses
8:     Move to object
9:     Grasp object
10:    Move to target zone
11:    Place object n
12:    if Object in zone then
13:        Set object finished
14:        n → n+1
15: until All objects finished
16: save log & stop simulator
```

Vision Module

The perception information is processed and handled by the vision module. Its purpose is to find the location of the sought-after objects and model surrounding obstacles, performing both environment recognition and modeling functions. The vision module serves as a typical example of a combined environment recognition and modeling system as mentioned at the beginning of this chapter. In this case, it obtains an RGB image and a 2.5D depth stream from each of the simulated RGB-D sensors. This data contains varying levels of Gaussian noise depending on the task (refer to fig. 4.5). Pose and geometric properties of the camera are available so that depth and RGB information can be correlated. This correlation becomes relevant for filtering out and determining the pose of the objects - which differ in shape and color. Unknown sections of the environment are considered as obstacles and modeled as a 3D map for safe navigation.
An overview of conventional and modern approaches for 3D object recognition can be found in Wöhler [204]. As explained, most 2D (and some 3D) pose estimation algorithms rely on previous feature learning from a training set of images of the object. These were not considered optimal for these tasks because object characteristics (shape, color) are only specified once the task has started. Instead, geometric methods (also called 3D descriptor methods or template methods, see Hinterstoisser et al. [72]) are more appropriate in this case. The objects consist of combinations of regular surfaces for which normal-based descriptors are particularly robust. The presented application is based on PCL, which is integrated into ROS. The *Fast Point Feature Histogram* as described by Rusu et al. [159] is used for pose estimation as it allows taking noise and incomplete data into account. The general strategy for image processing is shown in fig. 4.4 and comprises the following steps:

a) *Scan Environment*. As the environment is not dynamic, multiple images are taken from different viewpoints (using both the scene and the tool center point (TCP) camera). By transforming the 3D PCs into world coordinates, they are combined and known parts of the environment (e.g. table, robot and pan-tilt camera unit) are filtered out. In the challenge tasks, objects can have only one out of six different colors. Therefore, for later object recognition, each PC is separated into six colored sub-PCs via HSV-filtering. The same procedure could be applied for a more general case by storing a colored PC (where each point is assigned an RGB value) and performing the color-filtering afterwards. An example of this procedure is shown in figs. 4.5 to 4.7.

b) *Environment Recognition*. For object pose estimation, an ideal PC is created based on

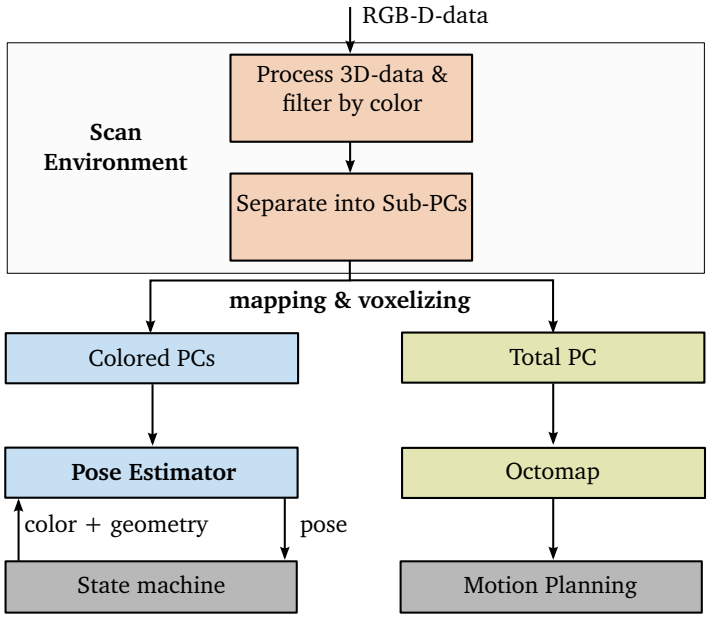


Figure 4.4: Image processing strategy and integration in the overall framework. Bottom left: environment recognition. Bottom right: environment modeling.

its geometric description (with a point density similar to the voxelized PCs). The corresponding colored PC is clustered into separate sub-PCs. These are classified based on their bounding box size to filter out obstacles or other objects. Finally, using the *Fast Point Feature Histogram* both PCs are compared iteratively in order to find the correct transformation (see fig. 4.7).

c) *Environment Modeling*. The PCs are created, combined and voxelized (both for computational efficiency and to obtain a homogeneous distribution of points, necessary for the pose estimation algorithm). The total (not color-filtered) PC is modeled using a compact 3D discretization called *Octomaps* [76] and sent to the planning module for collision checking (see fig. 4.6).

Results for Autonomous Pick and Place Tasks

In this section an exemplary result for a pick-and-place task is presented. A video showing the different simulation tasks can be found online at https://youtu.be/OTWEZd6BMk8. In these tasks, the robot must locate objects, pick them and place them in the corresponding target zones while avoiding obstacles and complying with its dynamic and kinematic limits. A plan view of the setup is shown in fig. 4.8 and algorithm 2 describes the overall strategy. First, the environment is scanned (since some parts of it are not visible for the scene camera, the environment has to be scanned successively using the views of the TCP camera as well). This way, a fuller, more complete PC is created by fusing together PCs from different perspectives. Using the PC-data from the vision module the *Octomap*-based environment model is generated for collision checking.
Subsequently, objects are sequentially located. For a located object the grasping module calculates feasible grasping poses and the pick-and-place procedure is performed as follows (see fig. 4.9): the robot moves to a predefined *transport*-configuration and approaches the located object (1). Once the distance between the robot and the object is less than a predefined threshold the grasping sequence is performed (2). After picking up the object, the

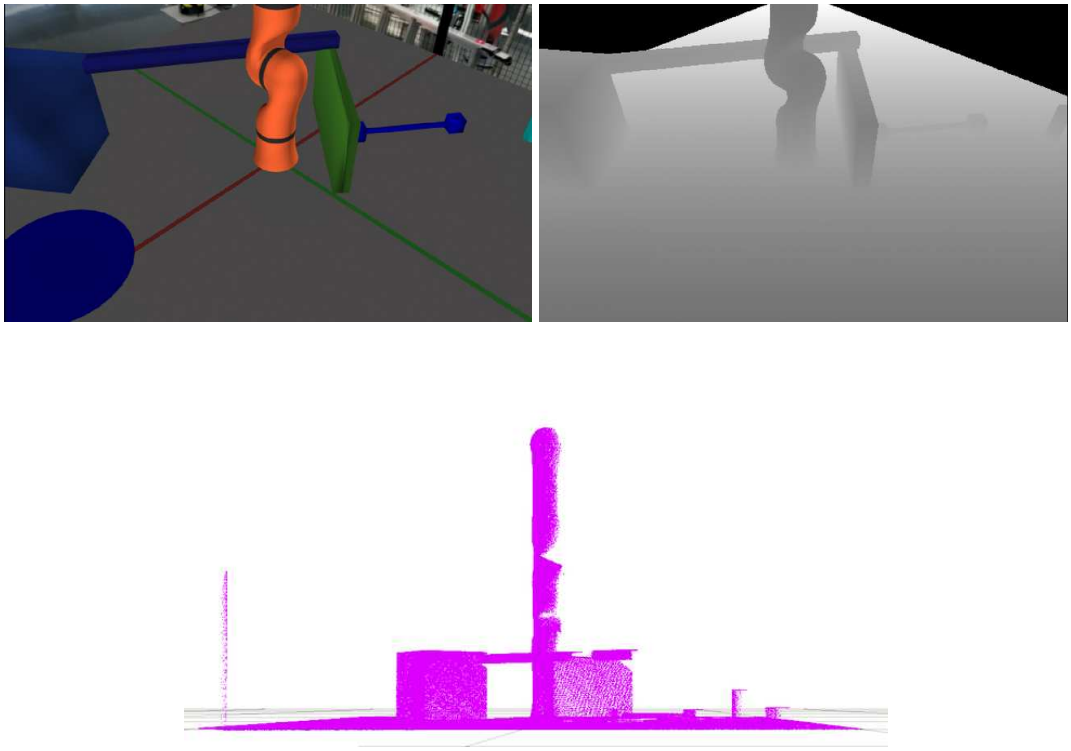


Figure 4.5: RGB image and 2.5D depth stream (top) sent by the simulated sensor, and the resulting 3D PC (bottom).

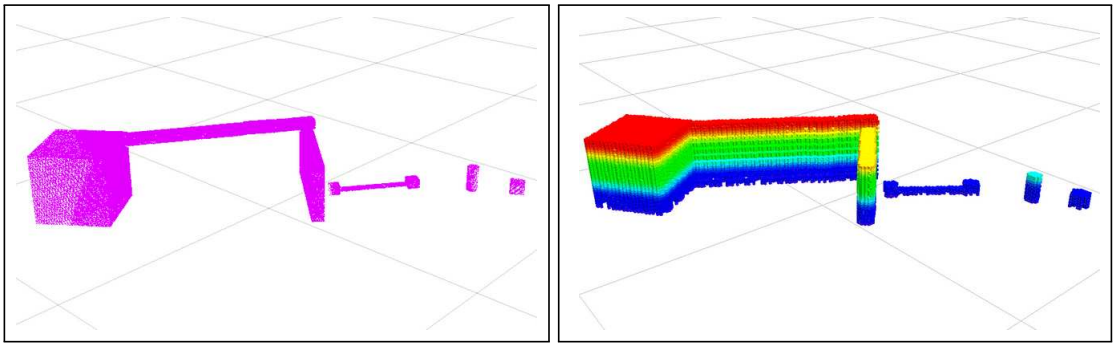


Figure 4.6: Filtered total PC (left) and *Octomap*-based environment model sent to the motion planning module for collision avoidance (right).

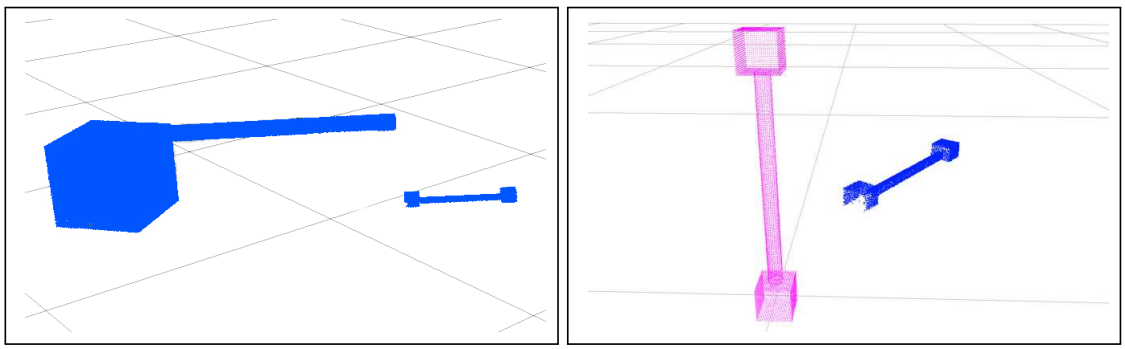


Figure 4.7: Environment Recognition. Color “blue” PC (left) and the feature comparison (right) of the ideal PC (purple) and the clustered color PC (blue) for pose estimation.

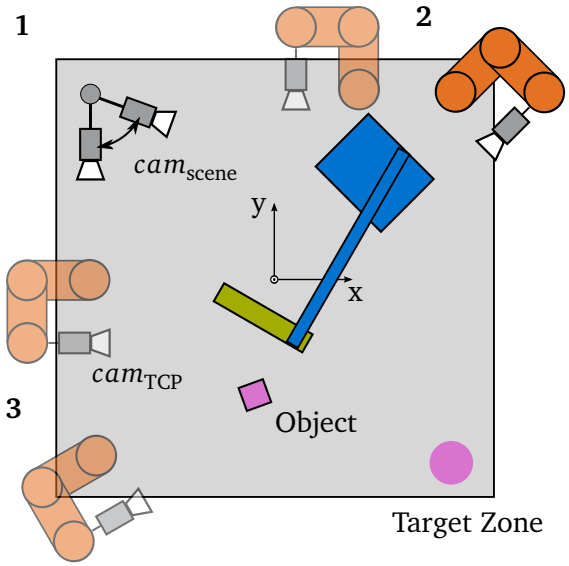


Figure 4.8: Plan view of the scene *Pick-and-place in obstructed environments*. The object (magenta) has to be located and placed in the target zone while avoiding obstacles (blue, green). Four different poses of the TCP camera for exploring the environment are shown.

robot moves to the corresponding target zone switching back to the *transport*-configuration (3) and performs a placing procedure similar to the grasping sequence (4).
It is important to note how both the environment recognition and modeling strategies come into play here. The object pose estimation algorithm allows to generate a grasping pose which becomes the objective of motion planning module. In order to reach this objective, it generates a feasible, collision-free motion based on the environment model.

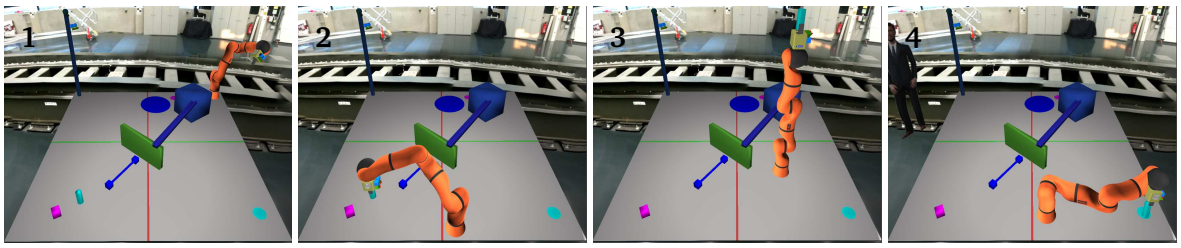


Figure 4.9: *Pick-and-place* task in obstructed environment: 1) *Scan environment*. 2) *Grasp object* (cyan cylinder) 3) *Move to target zone* in 2 DoF *transport*-configuration. 4) *Place object* into target zone.

Application and Perspectives

When applying this strategy to real, non-simulated, scenarios, more factors have to be taken into account (e.g. lighting conditions for color filtering). Nevertheless, this approach can be applied to the *CROPS* robot for the autonomous harvesting of fruits. Such real-world applications involve further challenges for an autonomous robot as it must work outdoors, under changing conditions and in obstructed environments. Nevertheless, the vision module presented above has already been tested on artificial sweet peppers in the laboratory. Only some of the parameters used were modified. Using the calibrated sensor mounted on the top of the manipulator (section 3.4) and an ideal sweet-pepper model as reference, the robot is able to find and pick the fruit autonomously (fig. 4.10).

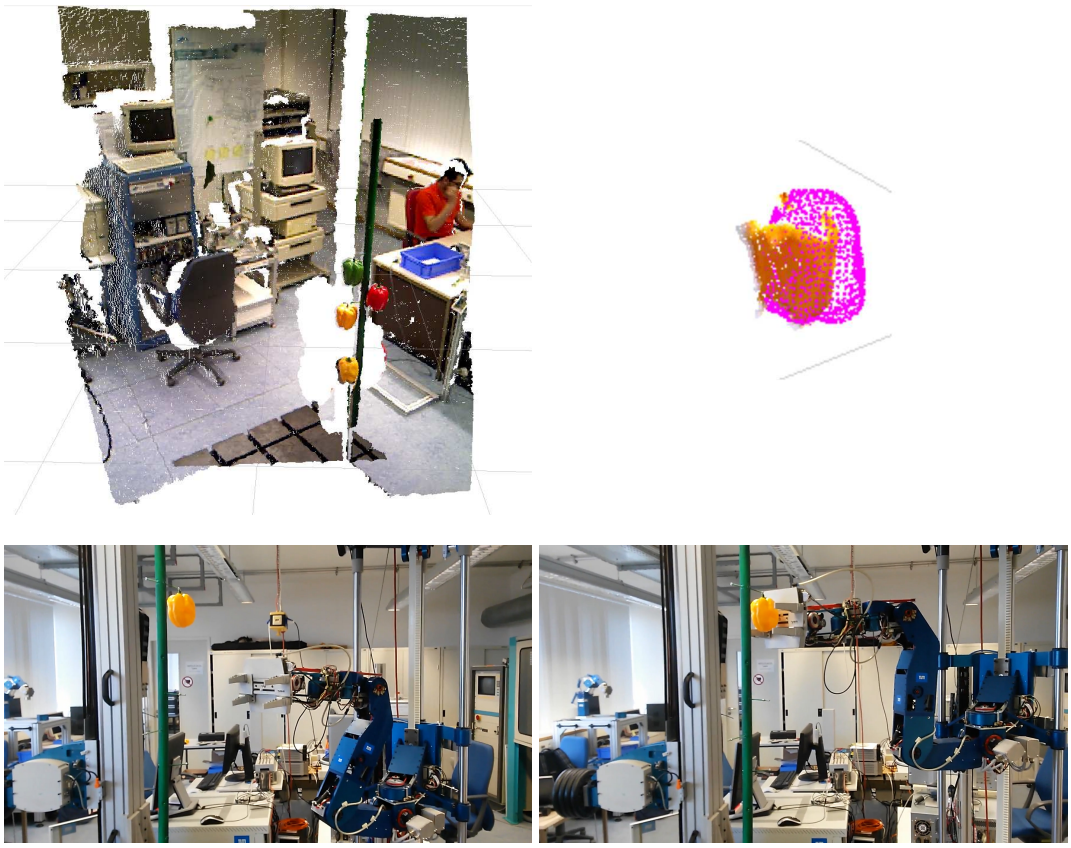


Figure 4.10: Testing the vision system in a real environment. Top left: an external RGB-D sensor mounted on top of the manipulator captures the scene. Top right: the filtered point cloud is compared against an ideal model of a sweet pepper. Bottom: the object's pose is sent to the robot which picks it up.

The framework presented in this section serves as an example of a combined vision system for environment recognition and modeling. As was shown, such a system can be successfully applied to pick-and-place tasks. However, it is not exempt from certain limitations which prevents its use in other kinds of applications. The main ones concern timing and its applicability to dynamic scenarios:

(a) *Data acquisition*. The Scan Environment routine assumes a static scene and can therefore be used for sensor fusion during post-processing. In real dynamic scenarios, data-acquisition and sensor fusion must be performed in real time [126], limiting the availability of additional viewpoints and the possibility of filtering out noise. Thus, 3D data becomes incomplete and less reliable.

(b) *Processing time*. The results of the presented framework show an underlying problem of the selected strategies, which is runtime. Both the vision and motion planning modules require several seconds to process the *EuRoC*'s simple scenes using modern computers. In the case of the vision module, the use of geometrical descriptors requires more computational power than machine learning strategies but, as explained before, these are not easily applicable in the considered scenarios. Furthermore, the presented strategy relies on simple color filtering based on previously known information. If color information was not available, both runtime and robustness would be badly affected. Even without considering the vision system's performance, the motion planning module would still be a problem: during experiments, it took sometimes up to one minute to find a collision-free path in complex scenarios[4]. The reason for this is that 3D map representations such as the one used are not exactly efficient for collision avoidance as the number of distances that have to be calculated becomes extremely large. This makes them unsuitable for dynamic scenarios.

Due to these reasons, the methods presented above cannot be presently applied to systems which either contain a large number of DoFs or which require low processing times due to either their own dynamics or changing environments. Humanoid robots in particularly fall into all of these cases. Therefore, previous frameworks for autonomous walking have always included strategies for environment modeling with a wide range of complexity. In the following, these are discussed and an original environment representation is presented afterwards.

Unknown Environments

A general concept for a perception application for autonomous robots can be seen in fig. 4.11. As explained before, object recognition techniques, which have lately become more robust with the use of deep learning methods [94], enable all kinds of applications such as manipulation and interaction. Still, there will always be sections of the environment (or even complete scenarios) that will not be recognized by the most comprehensive database; these can be modeled in parallel using basic geometries, which the robot's planning can handle. Furthermore, for obstacle avoidance applications a rough approximation of an object is not only sufficient but desirable for computational reasons.
The interaction between environment modeling and motion planning is discussed in this section. These modules represent two of the main components of an autonomous navigation system. In order for a robot to navigate in a real environment, the perception module has

[4]For the final framework, many motions were reduced to the DoFs at the base using a rigid manipulator to reduce processing time as can be seen in the videos

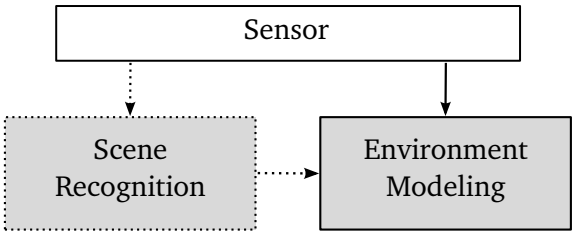


Figure 4.11: Concept for a Perception System for Autonomous Navigation: sections of the environment which are not recognized by learning methods (*Scene Recognition*) are modeled by an approximation strategy (*Environment Modeling*) with a set of pre-defined geometries.

to acquire an abstract representation (or model) of it, which is then used by the planning module to find a feasible path while avoiding collisions [62]. The modeling strategy restricts the kind of environments the robot can navigate through and determines the capabilities of the motion planner.
In what is usually considered to be the first example of autonomous robotic navigation, 2D maps were used to represent the environment and the A* algorithm was presented to navigate an unknown scenario avoiding static walls [62]. Later works included height information in a 2D grid to generate "height-maps" and navigate in uncluttered, relatively horizontal terrain [122]. Using this approach, wheeled robots have been able to achieve impressive feats [63]. As explained in chapter 1, legged robots are better suited than wheeled robots for navigating through cluttered environments. However, providing legged robots with autonomous navigation capabilities is a more challenging task. The complexity of this problem lies, on the one hand, in the particular dynamics of biped robots. Besides being naturally underactuated, their high number of degrees of freedom makes it computationally challenging to perform real-time motion planning and control. In order to solve this, reduced models are used for approximating the robot's dynamics (chapter 2).
On the other hand, a detailed model of the environment (such as the *Octomap* representation used in section 4.2) is not suited for real-time navigation due to its computational costs, both on the computer vision and the motion planning side. Therefore, real-time humanoid navigation[5] has only been achieved using simplified environment models on relatively uncomplicated, static scenarios (see fig. 4.12). In the first application of height-maps to biped navigation, static scenes could be traversed in real-time due to its simplified representation [82]. An extension to this representation, consisting on the segmentation and classification of these "2.5D" maps, is a very popular approach in humanoid research as it permits a real-time solution of the motion planning problem [24, 59, 60, 68, 91, 137]. However, it still limits the complexity of applicable scenarios.
In summary, existing environment modeling strategies are either limited in their representation of complex scenarios (2D, 2.5D) or require too much processing time to be implemented in real-time, dynamic applications (see table 4.1). This work tries to bridge this gap by introducing a full 3D environment representation which can represent complex, dynamic scenarios and can be processed in real-time by the motion planner. In the following, the basic concept of this representation is presented. The next chapter is dedicated to explain how such a representation can be obtained online from 3D point clouds when facing unknown dynamic environments.

[5]Compared to a robotic manipulator in a limited workspace, the problem of biped navigation in large environments is more challenging in terms of processing time constraints. A biped navigation system is considered to run in real-time if trajectories and motions can be adapted or recomputed in the span of one walking step.

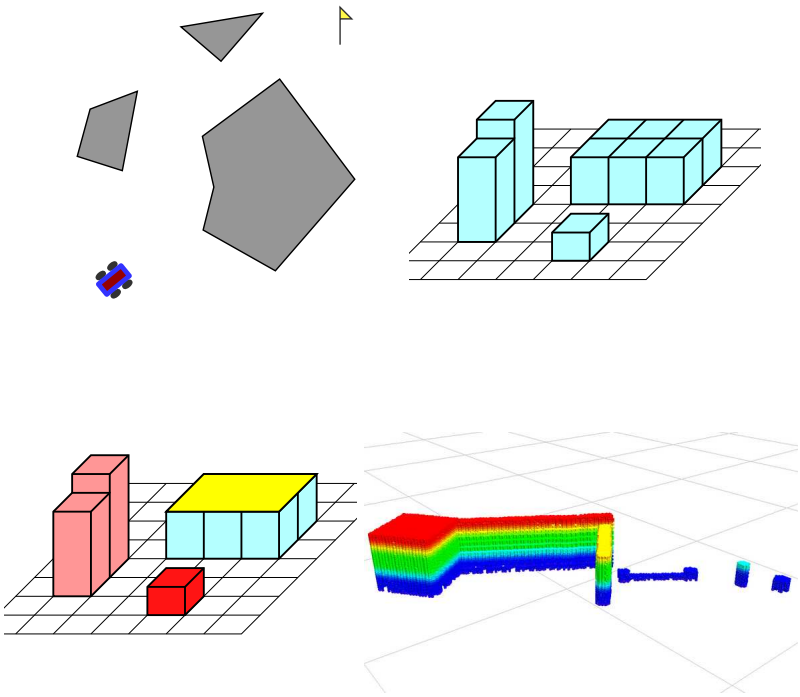


Figure 4.12: Different Environment Modeling Strategies. A simple 2D model can be useful for e.g. wheeled robots (top left). If height information is included, a discrete 2.5D map can be obtained (top right). By segmenting such a 2.5D map, surfaces or obstacles in simple environments can be recognized (bottom left). A dense 3D voxelization offers detailed information at the cost of increased runtime (bottom right).

Table 4.1: Environment Modeling Strategies

Type	Computational Cost	Dynamic Environments	Example
2D	very low	yes	Hart et al. [62]
2.5D	very low	yes	Kagami et al. [82]
Segmented 2.5D	low	possible	Nishiwaki et al. [137]
3D	high	no	Hornung et al. [76]

3D Environment Modeling

The following approach for environment modeling is based on using simple geometries that are particularly well suited for collision calculation and motion planning. This new environment modeling strategy is particularly well suited for humanoid robots. When performing autonomous biped navigation, the surrounding environment can be classified in two: sections over which the robot can walk and obstacles which have to be avoided. Walkable areas of the environment have to be represented in a way that is compatible with a fast walking pattern generation strategy (chapter 2), while obstacles are to be represented with shapes that allow for quick collision checking.
An example of such geometries are the swept-sphere-volumes (SSVs), defined as the expansion of basic geometric shapes (e.g. point, line or triangle) in all possible directions with a fixed offset or radius. They consist of rounded shapes (e.g. spheres, capsules or rounded triangles) that have been repeatedly used for efficient collision checking by different authors [65, 68, 165]. The advantage of these shapes becomes evident when calculating the distance between two of them:

$$dist\left(SSV\left(a,r_a\right),SSV\left(b,r_b\right)\right) = dist\left(a,b\right) - r_a - r_b \tag{4.1}$$

where $SSV\left(n,r_n\right)$ is the SSV expansion of shape n with radius r_n. For example, the distance between a sphere and a capsule is equal to the distance between a point and a segment minus both radii (see fig. 4.13). This representation of objects allows for fast planning and obstacle avoidance in cluttered environments. Collisions -both internal and external- are checked in 3D in real-time [67]. Due to their efficiency for distance calculation with respect to other shapes, point- and line-SSVs (spheres and capsules) are used to represent obstacles.

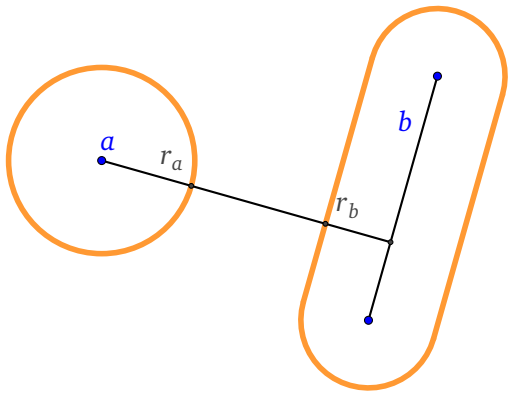


Figure 4.13: SSV approximation for fast collision checking. The distance between two SSVs out of the geometric shapes a and b is calculated as the distance between a and b minus the corresponding radii, r_a and r_b. In this example, a and b are a point and a segment, respectively.

Walkable areas are meanwhile represented with convex polygons, which facilitate the task of finding safe footstep positions (chapter 7). However, it is not safe for the robot to step on the border of the surface and it is therefore represented as a set of line-SSVs. The complete environment representation can be seen in fig. 4.14, where scenarios are represented as a list of polygons and SSVs that represent safe and non-safe regions for the robot. Additionally, the robot itself is represented by SSVs in order to easily calculate 3D collisions against itself and the environment. As shown in different examples throughout this thesis, complex scenarios can be efficiently represented by this full 3D modeling strategy and, more importantly, processed by the motion planning module in real-time.
This representation is not only useful for humanoids but can be applied to other kinds of robots as well. In the following section an example application of this modeling strategy to

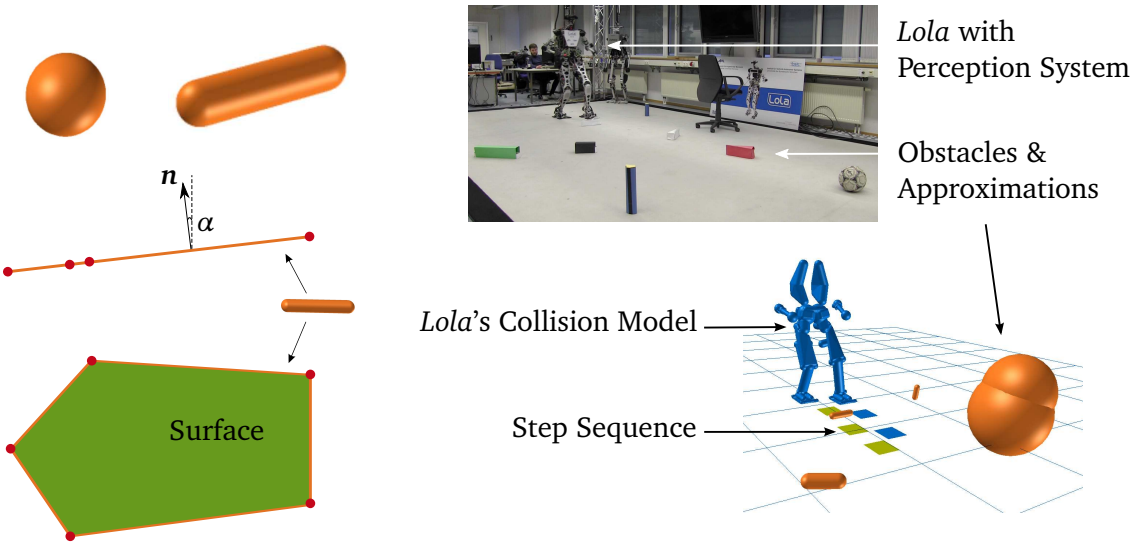


Figure 4.14: Environment modeling. Left: obstacles are represented as point- and line-SSVs; surfaces as polygons with line-SSVs at the edges. Right: environment representation enables real-time 3D planning and collision avoidance.

obstacle avoidance with manipulators is given. The following chapter deals with the applicability to real humanoid robots, explaining how such an environment representation can be computed online, during walking, without previous information about it and depending only on on-board sensing.

Obstacle Modeling for Robotic Manipulators

The most convenient application of the presented object modeling strategy is 3D obstacle avoidance, which is a general problem for any kind of robot. Besides the robot *Lola*, this concept can be potentially applied to other kinds of robots such as robotic manipulators. It has already been successfully tested on the *CROPS* manipulator [199] for evaluating different strategies for collision avoidance. In fig. 4.15, it can be seen how the SSV approximation can be used to perform obstacle avoidance[6]. A video of the experiments can be found in https://youtu.be/uDiXij2O5Y4.

[6]In this example, the perception system is not present.

Figure 4.15: Obstacle Avoidance with robot manipulators. Left: both the robot and the obstacles can be modeled using SSVs. Right: the robot is controlled via tele-operation but adapts its path to avoid a vertical object.

Chapter 5

Perception System

Related Work: Robot Perception

In previous chapters, it has been discussed how robots obtain raw data from the environment (chapter 3) and how to extract meaningful information from it (chapter 4). As explained before, for this work it was decided to develop a perception system that is compatible with open source libraries and a broad range of 3D sensors in order to make it adaptable to other robots and systems (chapter 3). The requirements of this perception system are to model completely unknown, dynamic scenarios online (while the robot is moving) and based only on on-board processing. This chapter describes the perception system itself, which generates a representation of the environment out of 3D point clouds using basic geometries (chapter 4). It is the result of collaboration with Fabian Bräu, Marko Lalić, Irem Uygur, Sahand Yousefpour, Christian Buttner, Dominik Gutermuth, Gregor Schwarz, Adnan Makhani, Tamas Bates and Stefan Floeren [12, 20, 50, 58, 103, 117, 185]. Some of the results presented in this chapter have been previously published or submitted for publication in more compact form in international journals and conferences [188, 192].
Existing perception systems for biped navigation are considerably limited in their application to unknown environments. In 2003, 2.5D maps were used to represent a sufficiently structured terrain [82]. The authors could generate collision-free trajectories for their *H7* robot. Other early approaches include a 2D classification of the environment [28] in which the biped robot *Johnnie* could navigate over obstacles and surfaces of which the geometry was previously known. In order to deal with more complex scenarios, a 3D occupancy grid was added to the 2.5D map in order to recognize obstacles [59, 60], but relied on textured surfaces to climb stairs with the *QRIO* robot. In one of the few works dealing with dynamic environments [23], the *Asimo* robot managed to safely navigate between 2D moving obstacles; these were previously known and moved only with constant speed in the lateral direction. An impressive degree of autonomy was showed by the *HRP-2* robot [24, 137]. Out of a structured environment, it could extract planes and label other regions as obstacles for walking over them and onto platforms. It relied on a pivoting laser scanner for which static scenarios were assumed. In 2010, a 2D-based occupancy approach was used for quickly generating collision-free trajectories in non-static environments [16], but it didn't consider complex obstacle avoidance motions such as stepping over. A different representation of the environment using *octrees* was presented later [114, 116]. The authors achieved collision-free navigation with the *Nao* robot, though they used texture and color for classification. Complex motions such as walking over obstacles were achieved when using height-maps [115]. Using this representation, they recently presented impressive results in dynamic environments [92]. Motivated by the DRC, some authors [47, 172] presented autonomous navigation results of the Atlas robot. However, they do not consider dynamic environments. In one of these works, a height-map of relatively simple scenarios was generated while the robot was standing [172] and simple collision checking was done via *Octomaps*. In another one [47], accurate and dense 3D maps

of the environment were used to extract walkable surfaces (but not obstacles) from static terrains. The robot was able to plan future footsteps during walking.
The works mentioned above present several limitations. In most cases, the scenarios considered are relatively simple, such that they can be represented using height-maps. Additionally, collision checking is based only on footstep locations and heuristics. Most importantly, either the perception system, motion planner or both take too long to be applied to unknown dynamic environments. The system presented in this thesis overcomes all of these issues with an efficient environment representation based on direct point cloud processing. On the perception side, it is fast enough to be generated online while the robot is moving. On the planning side, it enables reactive footstep planning and real-time 3D collision avoidance. Compared to all previous works, it presents a greater degree of applicability, as it can simultaneously process unknown walkable surfaces and obstacles of different sizes, shapes, static or dynamic. No other work, as far as the author knows, is able to handle unknown dynamic obstacles. The perception system is designed in a modular fashion so that it can be integrated as a mix of vision processing and visualization libraries easily into other systems. These libraries are available open source in the group's repository[1] and include tools for local visualization[2] and mixed reality with either an external camera or Microsoft's HoloLens (see chapter 7). More information about the visualization library can be found in appendix A.

Framework and Software Architecture

In fig. 5.1, the structure of the perception system for environment modeling can be seen. It can process 3D point clouds directly and provides the motion planning system with a set of walkable surfaces and obstacles. In order to allow for a fast update rate, a parallel structure allows to speed up the whole system and to make use of multi-threading computation: each process runs with its own cycle time and can access the latest scene information regardless of the other parallel processes. The first step to achieve this kind of architecture is defining classification criteria. In this work, the environment is classified according to whether it is walkable or not. For other robotic systems, different criteria could be applied, making use of the same approximation algorithms.
The main idea behind this system is to have one process for classifying points in the point cloud (either as surfaces or obstacles) and other parallel processes to obtain the geometric models for each subset of points. As shown in fig. 5.1, the classification process (*Plane Segmentation*) finds points in the scene (planes) which can potentially be walked over by the robot. Using the latest plane coefficients obtained, both the *Surface* and *Obstacle Modeling* filter out the new incoming point cloud (the *Surface Modeling* keeps points belonging to the planes and the *Obstacle Modeling* discards them) for generating the corresponding basic geometries. Note that, even though the coefficients are taken from previous frames (less than 50 ms old, see chapter 8), they should still be valid in the considered application scenarios. Fast moving surfaces, such as escalators, are therefore treated as obstacles because their points do not correspond with earlier plane parameters (which is acceptable, as they cannot be handled by the existing motion planner anyway). In fig. 5.2, the approximation of an example scene using polygons and SSVs can be seen.
Both approximation processes follow a similar strategy that consists of the following steps:

- *clustering*, where points are grouped into separate "clusters",
- *tracking*, where each surface or obstacle is tracked and filtered across frames and

[1] https://github.com/am-lola/lepp3
[2] https://github.com/am-lola/ARVisualizer

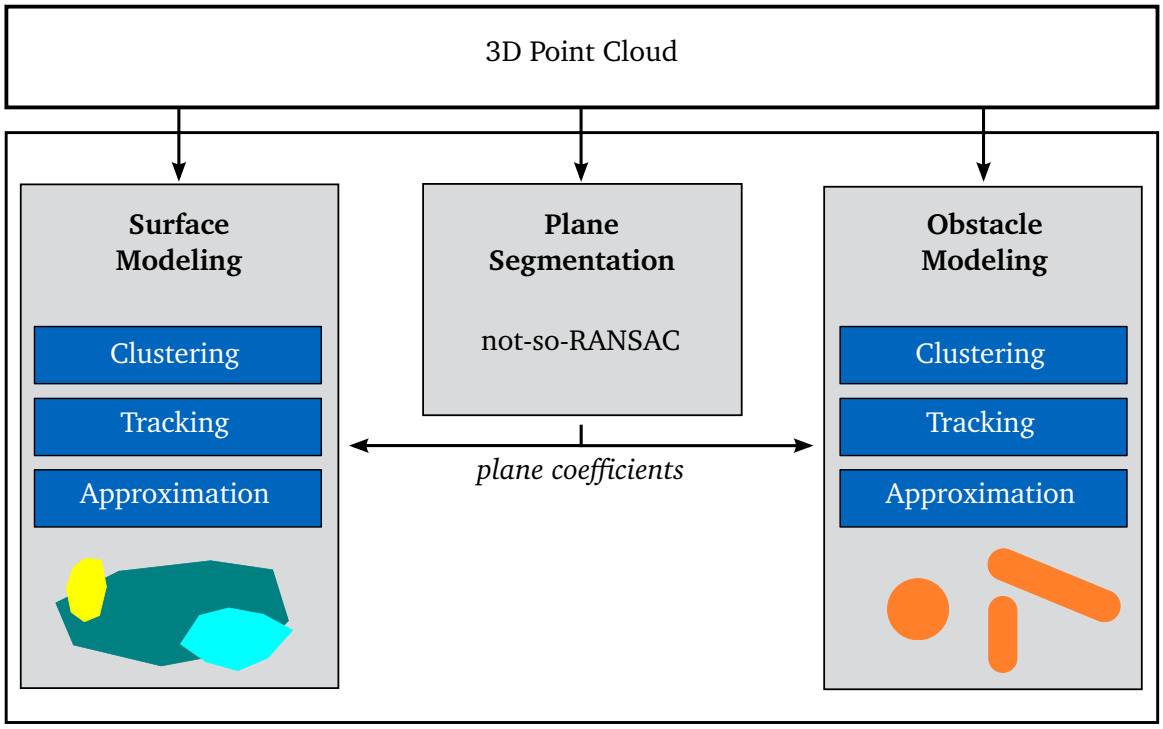


Figure 5.1: Environment Modeling Structure. The *Plane Segmentation* process classifies the incoming point cloud into walkable surfaces (for the *Surface Modeling*) and collision objects (for the *Obstacle Modeling*).

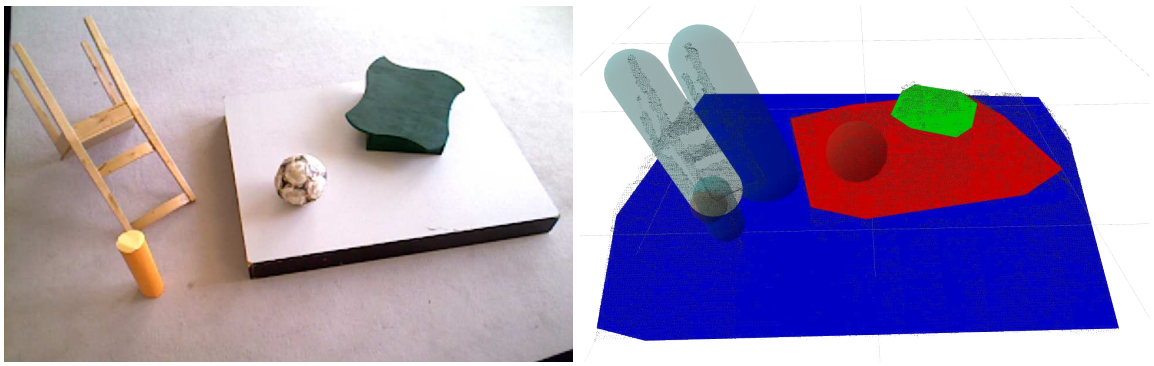


Figure 5.2: Result of the vision system. Left: RGB image of an example scene (for reference). Right: surfaces (including the ground) are modeled with polygons and obstacles with SSVs.

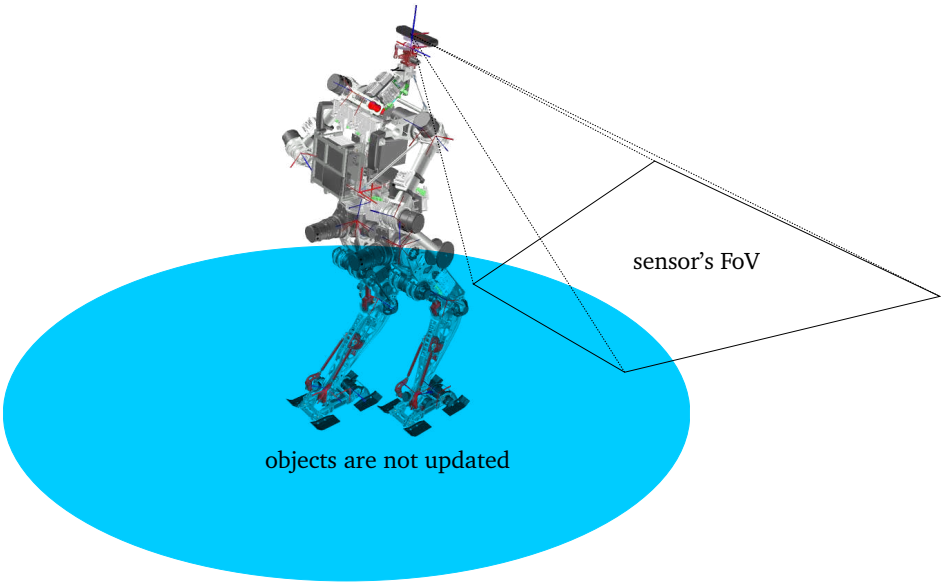


Figure 5.3: The sensor's field of view is limited. Objects that enter the skyblue region stop being updated and are not removed. Objects outside both the FoV and skyblue region are removed.

- *approximation*, where each cluster of points is assigned one or more approximating geometries.

In the *tracking* step the data is additionally checked for consistency. High levels of sensor noise (chapter 3) can cause the false detection of non-existing objects. Therefore, surfaces or obstacles have to appear in several consecutive frames before they are considered *real* and can be sent to the planner. Moreover, as the sensor's FoV does not include the section of the ground nearest the robot, objects stop being updated once they get close to the robot and before they leave the FoV (see fig. 5.3).
The walking control system continuously updates the kinematics transformation from the camera to a reference *world coordinate system* (W), fixed to the ground at the starting position of the robot. By transforming the incoming point cloud (PC) from the camera coordinate system into W, objects can be successfully tracked despite the robot's motion. It is worth noting that this is not intended as a SLAM strategy, as a map of the environment is not kept or updated; instead, each incoming PC is analyzed separately. W will be affected by inaccuracies in the robot's motion (e.g. sliding on the ground) and will change slowly with time, but that is not relevant for this application as only on-board sensing is considered and measurements of the environment are continuously updated and correct with respect to the robot. Therefore, odometry errors are always kept low enough for safe navigation. Additionally, a simple voxelization of the PC into a 3D grid with 1cm size is applied in order to have a homogeneous PC in the following steps regardless of the sensor used. Thus, the dependence of the processing algorithms on the type of sensor is reduced and parameters don't have to be re-adjusted. In the following, the different processes in fig. 5.1 are explained in detail. As stated before, a real-time application is the main motivation behind the proposed system. Throughout its implementation, strategies were chosen by prioritizing robustness and runtime over optimality in order to achieve safe, if not optimal, navigation. The following algorithms satisfy these criteria and are the result of extensive testing and iterations, where both new and state-of-the-art algorithms are combined to make this system as efficient and robust as possible.

Plane Segmentation

The segmentation submodule classifies points according to whether they belong to a walking surface or an obstacle. Several standard algorithms were evaluated against a set of complicated, dense scenes to assess their worst-case performance. Algorithms tested can be split in three main categories:

a) *Depth-map-based*. These algorithms are based on the direct processing of the depth-map of structured light sensors. Besides being incompatible with other kinds of sensors such as LIDARs, they often fail to segment plane regions from objects that lie on top of them, such as ramps or prisms.

b) *Normal-based*. These algorithms first calculate a map of vectors normal to the point cloud's points (and their corresponding neighbor points). Regions are then separated based on the normal gradients. Apart from their sensibility to noise, the main problem with these algorithms lie in the long processing time needed to calculating the normal maps.

c) *Random Sample Consensus* (RANSAC). This algorithm was introduced by Fischler et al. [48] and is based on the random fitting of sensor data to a model. Due to their direct evaluation approach (as explained below) it can be very efficient to fit ideal models such as planes.

Other novel approaches to fast plane detection were discarded for their dependency of particular sensing technologies [146], RGB data [64] or height-maps [119]. The chosen segmentation strategy is based on RANSAC. Besides the characteristics mentioned above, it proved to be faster, more robust and transferable throughout different frames (key for this parallel approach).
The PCL's *Sample Consensus Segmentation* [145] provides a RANSAC implementation which randomly selects 3 points of a point cloud to get a plane's coefficients $((a,b,c,d) \mid ax+by+cz+d=0)$ and tests them against the remaining points. If enough points belong to the plane (within a certain threshold), these are removed and the process starts over. As the algorithm finds different planes' sections separately, these are joined and clustered in the end to obtain complete surfaces. The system selects only surfaces that are sufficiently large to fit the robot's foot and which are nearly horizontal (up to 20° slope in the present configuration) and, thus, walkable. Additionally, the following two modifications are introduced:

1. Not-so-RANSAC. In order to speed up the process, old plane parameters are tested against the incoming point cloud before starting RANSAC, so that previously existing surfaces can be quickly identified. This can speed up the segmentation routine up to seven times.

2. Classification. The standard segmentation process is not robust against intersecting surfaces with similar inclinations (such as the grond and a ramp); when joining sections of planes together, both surfaces may be joined into one single plane (see fig. 5.4). Therefore, before this step, planes' sections are classified according to their inclination so that all different plane parameters are correctly identified and adjacent surfaces separately approximated. As shown in the next section, this simple modification also helps speed up the surface modeling routine.

The final list of plane coefficients $\{a_i, b_i, c_i, d_i\}, 1 \leq i \leq n_{\text{planes}}$ is sent to both *Surface* and *Obstacle Approximation* processes which are explained in the following.

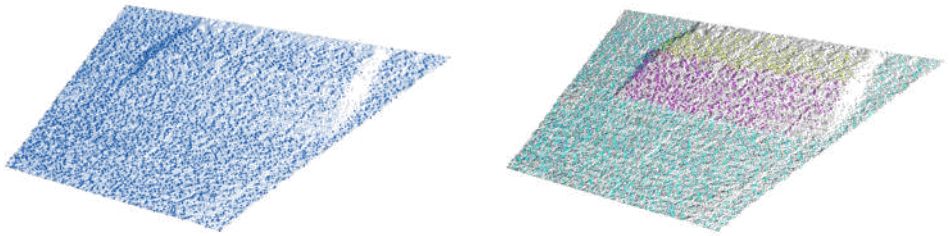


Figure 5.4: RANSAC Classification. Left: the standard RANSAC implementation includes a clustering step that may merge planes with different inclination together (such as the floor and the ramp). Right: preliminary surfaces are classified according to their inclination before clustering so they can be separately approximated.

Surface Modeling

Clustering

After filtering surface-points with the plane coefficients, these must be clustered into separate surface objects. Even points with the same plane coefficients might belong to separate surfaces (e.g. two separated platforms with equal height). Standard clustering algorithms (e.g. PCL's *Euclidean Clustering* [145]) are computationally expensive, partly because they are implemented for 3D point clouds (see fig. 5.5). However, thanks to the modified RANSAC implementation, the clustering step is reduced to a 2D problem: as planes are already classified according to their inclination and position, only points belonging to the same plane have to be clustered further. Thus, points can be projected into their corresponding plane and clustered using local 2D coordinates.

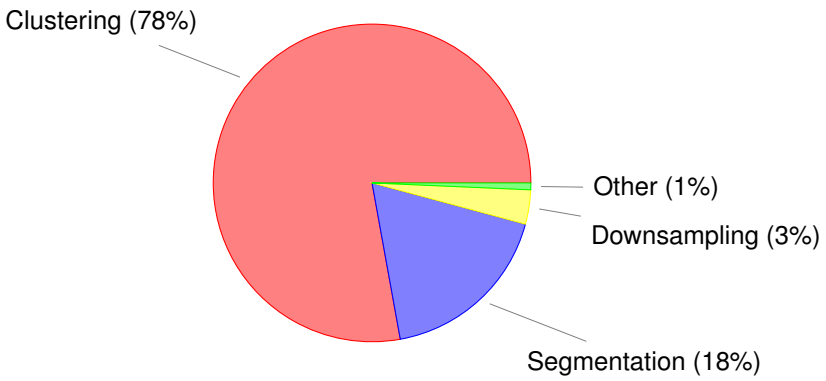


Figure 5.5: Profiled performance of a sequential Surface Modeling implementation using the standard PCL's *Euclidean Clustering* algorithm for surface clustering. Adapted from Gutermuth et al. [58].

For this application, clustering criteria depend on the robot's feet. A cluster can be defined such that the distance between neighboring points is considerably smaller than the foot size. For clustering a given plane, a simple grid discretization is used: all points are grouped according to a grid with a relatively small unit length (5 cm in the present implementation) and define connectivity based on neighboring occupied cells (note that the grid is only used for clustering while the original points are passed along to further stages).
This implementation was tested against different clustering algorithms using varied scenes with multiple separated platforms of equal height. Algorithms tested include: PCL's 3D Euclidean Clustering [145], *OpenCV*'s 2D Euclidean Clustering [139] and a local implementation of *DBSCAN* [43]. The difference in clustering results is always less than 1% of the

number of points while the difference in runtime is significant: the grid strategy (runtime of less than 2 ms in the most complicated scenarios) performs more than 50 times faster than every other algorithm (see appendix B for more details).

Approximation

As explained before, surfaces are approximated by simple polygons. At present, concave or incomplete surfaces are not considered due to the way footstep locations are optimized for collision avoidance (chapter 7). The approximation starts by projecting the cluster point clouds to the corresponding ideal plane and applying the popular *QuickHull* algorithm [21, 38] (chosen for its runtime and easy parallelization). It iteratively expands a polygon until it contains all points, as shown in fig. 5.6.

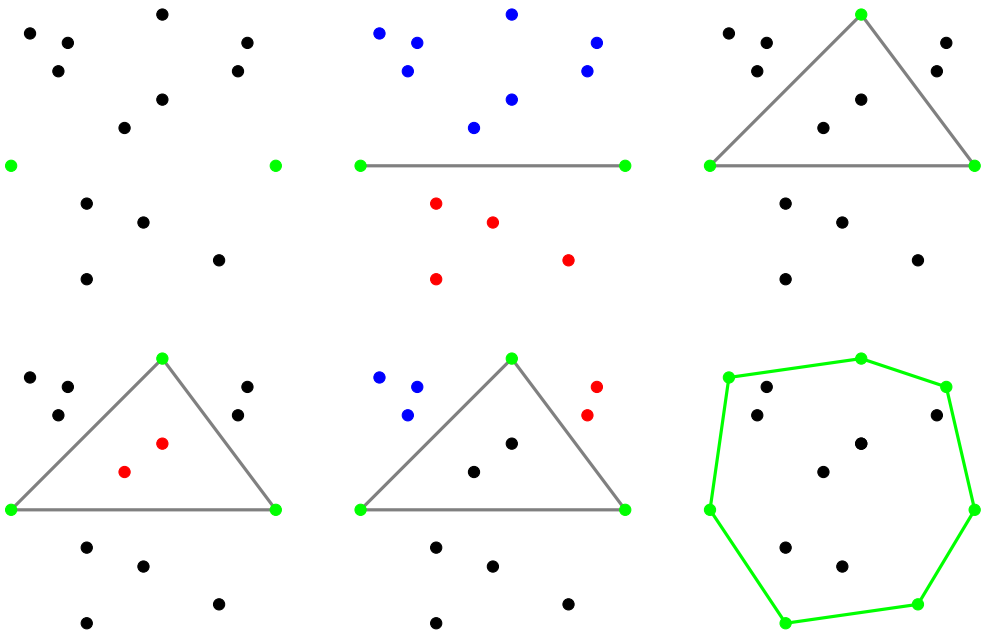


Figure 5.6: A simple example to illustrate the QuickHull algorithm (from left to right and top to bottom). First, the two extremal points of one dimension are selected; the line connecting these two points splits the set into two halves (red and blue), continuing with each half separately. Then, for each half, the point with the maximal distance to the line is found and added to the hull; all points within the triangle (red) are removed from the set. The remaining points are again split into two separate groups (red and blue) and the process continues until there are no points left. Adapted from Gutermuth et al. [58].

The result is a group of polygons with varying number of vertices. In order to facilitate the integration with the motion planner these are reduced to a maximum number of vertices n_{vertices} ($n_{\text{vertices}} = 8$ in the experiments) by iteratively removing those vertices which subtract the smallest area from the polygon (see algorithm 3). A graphic example of the algorithm is shown in fig. 5.7.

Algorithm 3 Reduction of convex polygons

1: Polygon P with adjacent vertices $P[i]$
2: **repeat**
3: **for all** modified vertices j **do**
4: Update area $\triangle P[j]$ of the triangle $\langle P[j-1], P[j], P[j+1] \rangle$
5: Remove vertex corresponding to $\triangle P[m] = \min(\triangle P[i])$
6: Update P
7: **until** Polygon contains desired number of vertices

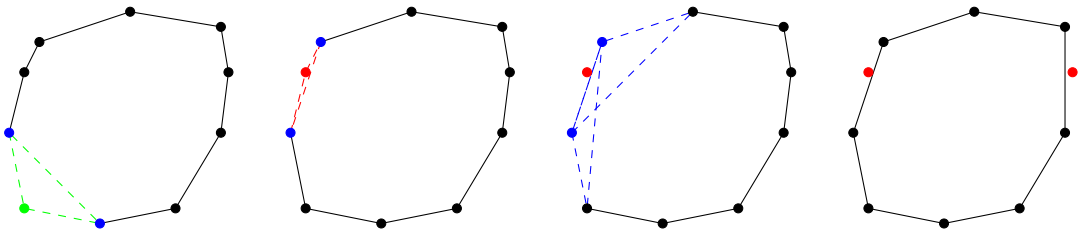


Figure 5.7: Reduction of convex polygons (from left to right). First, areas are computed for all triangles made of three consecutive vertices (example). Then, the vertex corresponding to the triangle with the smallest area is removed. For the next iteration, only two new triangle areas have to be computed. Finally, the algorithm finishes when the desired number of vertices is reached. Adapted from Gutermuth et al. [58].

Tracking

Due to sensor's noise and a limited FoV, the approximation process results in different polygon approximations from frame to frame. Surfaces are matched between frames by comparing plane coefficients and positions. If a corresponding previous polygon is found, it is updated (at the *approximation* step) by averaging both polygons. For this purpose, a geometric interpretation of the classic low-pass filter, or *geometric low-pass filter* is introduced. The main problem when averaging two polygons is that there is no clear correspondence between both sets of parameters (vertices). Methods which simplify filtering using the polygon's center or area often result in inaccurate approximations of the surface's perimeter, which is extremely relevant to this application[3]. Instead of matching vertices from both polygons with one another, each polygon vertex is matched with its closest point (or projection) in the other polygon. Then, the positions of each vertex and its corresponding projection are averaged with a factor α. This can be interpreted as a low-pass filter (with inverted factors for both polygons for consistency) as seen in fig. 5.8. This results in a polygon with double as many points that can be reduced using algorithm 3 (see algorithm 4).

Algorithm 4 Geometric low-pass filter

1: Polygons A, B, with $n_{\text{vertices}_{A,B}}$ vertices $A[i], B[j]$
2: **for all** i, j **do**
3: Compute projection of $A[i]$ in B, $A[i]_B$
4: Compute projection of $B[j]$ in A, $B[j]_A$
5: Compute pre-filtered polygon $C =$
$\{\alpha A[i] + (1-\alpha)A[i]_B\} \cup \{(1-\alpha)B[j] + \alpha B[j]_A\} \; \forall i, j$
with $0 \leq \alpha \leq 1$
6: Perform algorithm 3 on C

[3] As shown in the next section, using an object's center for tracking is instead applicable to obstacles, where the object's margins can be represented with less accuracy.

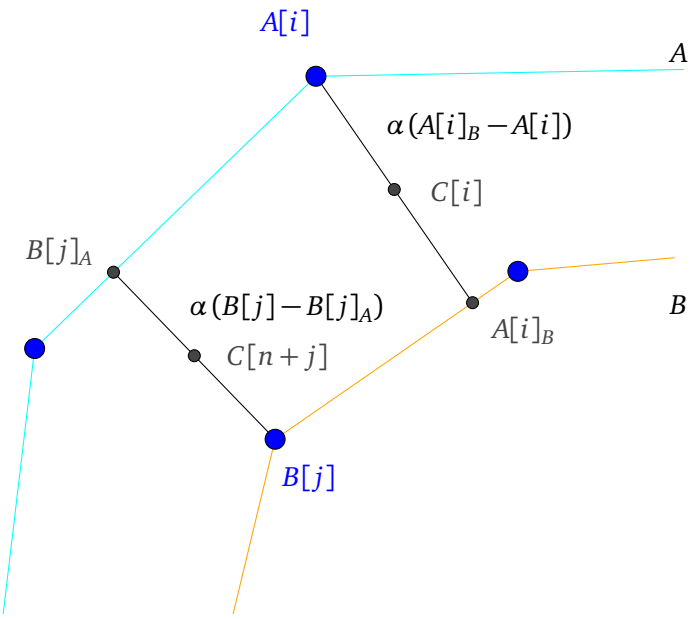


Figure 5.8: "Geometric low-pass filter" for filtering successive polygon approximations of surfaces (see algorithm 4). Note that the value of α determines the weight between both polygons A (cyan) and B (orange). By shifting it closer to the old or new polygons, the algorithm becomes either more or less damped, respectively ($\alpha = 0.5$ in the experiments).

Obstacle Modeling

Clustering

When processing 3D point clouds, there are several standard clustering strategies available, such as PCL's *Euclidean Clustering* [145]. One of their main limitations is long runtime (see fig. 5.5), which usually depends on the complexity of the scene (effectively restricting the amount and velocity of objects). In figs. 5.9 and 5.10, the performance of the obstacle modeling system using an euclidean clustering implementation is shown for two separate experiments. The robot walks among different scenarios with varying complexity. As can be seen, the total cycle time is strongly correlated with the number of objects in the scene.
In order to overcome this problem, clustering and tracking methods are presented which are both more robust and dynamic, handling extremely complex scenarios faster than the sensor's frame rate (30 Hz). They are adapted from probabilistic theory and machine learning methods to dynamic scenarios.
In machine learning theory, data clustering is a fundamental problem of unsupervised learning. The objective is to determine correlation relationships between variables out of training data sets, without any additional information. A common approach [128] is the *Gaussian Mixture Model* (GMM), which consists of a set of probabilistic Gaussian distributions (or Gaussians) with the form:

$$p(x_i|\theta) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x_i|\mu_k, \Sigma_k) \tag{5.1}$$

where K is the number of Gaussians and $\mathcal{N}(x_i|\mu_k, \Sigma_k)$ is the normal distribution of Gaussian k with mean μ_k, covariance Σ_k and mixing weight π_k. These weights satisfy $0 \leq \pi_k \leq 1$ and $\sum_{k=1}^{K} \pi_k = 1$, to ensure a correct probability distribution $p(x_i|\theta)$ of the n points x_i (in this case, x_i has three coordinates) with the list of parameters θ, which consist of $\{\pi_k, \mu_k, \Sigma_k\}$ $\forall k$

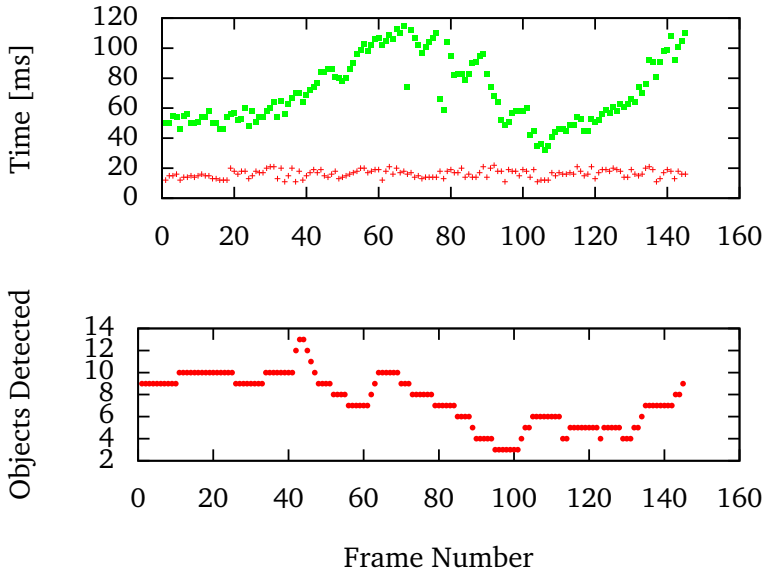


Figure 5.9: First experiment. Top: calculation time of the PC transformation and filtering step (red) and total time performance of the obstacle modeling (green) for each frame. Bottom: number of objects recognized in each frame.

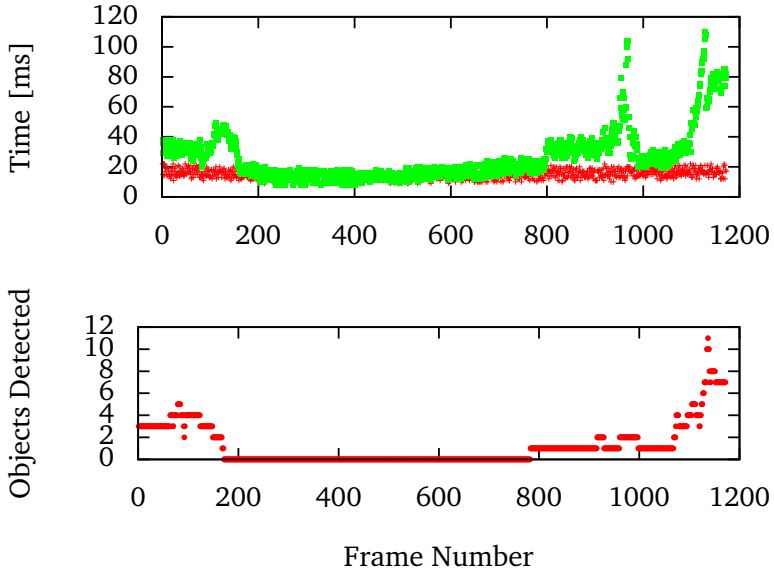


Figure 5.10: Second experiment. Top: calculation time of the PC transformation and filtering step (red) and total time performance of the obstacle modeling (green) for each frame. Bottom: number of objects recognized in each frame.

in this case [128].
These models are potentially well suited for the present application as they don't require many assumptions on the data and their probabilistic nature make them robust against sensor noise. Additionally, the implicit principal axis decomposition analysis can be directly used for the SSV approximation, as explained below. However, classical implementations are based on static data sets and require an iteration procedure which converges to a local maximum likelihood estimate (MLE). Even though other authors have recently shown applications to object tracking [98], these are still too complex for real-time applications (other works achieve faster times but depend on the color image as well as the point cloud [105, 112, 142]). For the obstacle *clustering* and *tracking* application, the following adaptation of the *Expectation Maximization* (EM) algorithm is proposed which, combined with a Kalman Filter, can be successfully applied for online tracking of unknown, dynamic objects.
In order to improve runtime, only one EM-iteration is performed for each new point cloud. They classically consist of:

- Expectation (E) step: compute an auxiliary *responsibility* r_{ik} for each point x_i and Gaussian k.

$$r_{ik} = \frac{\pi_k \mathcal{N}(x_i|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_i|\mu_j, \Sigma_j)} \tag{5.2}$$

- Maximization (M) step: for each Gaussian k, compute new estimates for the weights π_k^{new} and parameters $\{\mu_k^{\text{new}}, \Sigma_k^{\text{new}}\}$.

$$\pi_k^{\text{new}} = \frac{1}{n}\sum_{i=1}^{n} r_{ik} = \frac{R_k}{n} \tag{5.3}$$

$$\mu_k^{\text{new}} = \frac{1}{R_k}\sum_{i=1}^{n} r_{ik} x_i \tag{5.4}$$

$$\Sigma_k^{\text{new}} = \left(\frac{1}{R_k}\sum_{i=1}^{n} r_{ik} x_i \,(x_i)^\mathsf{T}\right) - \mu_k^{\text{new}} \left(\mu_k^{\text{new}}\right)^\mathsf{T} \tag{5.5}$$

Even though an MLE is not guaranteed for dynamic data, experiments in real environments show that it typically converges within a few frames and effectively tracks existing objects afterwards. Nevertheless, eqs. (5.4) and (5.5) do not take into account dynamic scenarios (or changing data). In order to deal with these scenarios, several modifications are introduced to the EM algorithm that take objects' dynamics directly into account and are explained in the following.

Tracking

It is interesting to note that the use of GMMs makes tracking much easier. Compared to the euclidean segmentation [188], where new objects have to be matched against old ones, the iterative nature the EM algorithm means that cluster identities and parameters are kept from frame to frame. However, data is supposed to be static, meaning that new values of μ_k^{new} do not take into account the object's velocities and are always "lagging behind" the motions of the objects. For this reason, a Kalman filter is applied to the values of μ_k^{new} which results in a more dynamic update procedure and better tracking[4].

[4]Even though the obstacles might be correctly approximated after some frames without the Kalman filter if the update rate is fast enough, it is still useful as a velocity estimator.

A standard version of the *random accelerations model* is taken for the Kalman filter [128], where both position and velocity are subject to Gaussian noise and only the position is observed. The state vector $\mathbf{x}_{k_t}$ for each Gaussian k at time t is defined as:

$$\mathbf{x}_{k_t}^{\mathsf{T}} = (x_{k_t}, y_{k_t}, z_{k_t}, \dot{x}_{k_t}, \dot{y}_{k_t}, \dot{z}_{k_t}) \tag{5.6}$$

where $(x_{k_t}, y_{k_t}, z_{k_t})^{\mathsf{T}}$ is the value of μ_k at time t. The corresponding transition matrix $\mathbf{F}_{k_t}$ for a linear motion is:

$$\mathbf{F}_{k_t} = \begin{pmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{5.7}$$

where Δt is the sampling time (or time between frames). The observation matrix $\mathbf{H}_k$ is constant:

$$\mathbf{H}_k = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \tag{5.8}$$

The system is described by the state transition:

$$\mathbf{x}_{k_t} = \mathbf{F}_{k_t} \mathbf{x}_{k_{t-1}} + \mathbf{w}_{k_t} \tag{5.9}$$

where $\mathbf{w}_{k_t} \sim \mathcal{N}(0, \mathbf{Q}_{k_t})$ is the system noise, assumed as a normal distribution with covariance $\mathbf{Q}_{k_t}$. Every time t, an observation of the true state is performed. In this case, the observation corresponds to μ_k^{new}. In the Kalman model, it can be expressed as:

$$\mu_{k_t}^{\text{new}} = \mathbf{H}_k \mathbf{x}_{k_t} + \mathbf{v}_{k_t} \tag{5.10}$$

where the observation noise $\mathbf{v}_{k_t}$ is also assumed to be a normal distribution with covariance $\mathbf{R}_{k_t}$: $\mathbf{v}_{k_t} \sim \mathcal{N}(0, \mathbf{R}_{k_t})$. $\mathbf{Q}_{k_t}$ is defined as $\text{diag}\{\sigma_{pos}, \sigma_{pos}, \sigma_{pos}, \sigma_{vel}, \sigma_{vel}, \sigma_{vel}\}$ and $\mathbf{R}_{k_t}$ as $\sigma_{obs} I$. The noise parameters σ_{pos}, σ_{vel} and σ_{obs} can be set by the user and are based on sensor noise and odometry errors.
Each new value of μ_k^{new} is introduced in the Kalman *update* step to obtain the estimated value:

$$\hat{\mu}_k^{\text{new}} = \mathbf{H}_k \mathbf{x}_k \tag{5.11}$$

Therefore, eq. (5.11) is used to replace eq. (5.4). This combination of the EM and Kalman filter algorithms results in the effective tracking of multiple dynamic elements[51], as can be seen in fig. 5.11. Furthermore, the estimate of the velocity can be passed along to the robot in order to directly consider objects' motions in the motion planning module.
For what concerns Σ, the estimation in eq. (5.5) depends strongly on the present data and thus can vary from frame to frame. In the present application, it means that separate dynamic objects are quickly joined once they are close, preventing correct tracking and velocity estimation. Therefore, it is replaced with a *maximum a posteriori* estimation [128], which can be interpreted as a low-pass filter that keeps separate obstacles separated (see fig. 5.12):

$$\hat{\Sigma}_k^{\text{new}} = \lambda \Sigma_k + (1 - \lambda) \Sigma_k^{\text{new}} \text{ with } 0 \leq \lambda \leq 1 \tag{5.12}$$

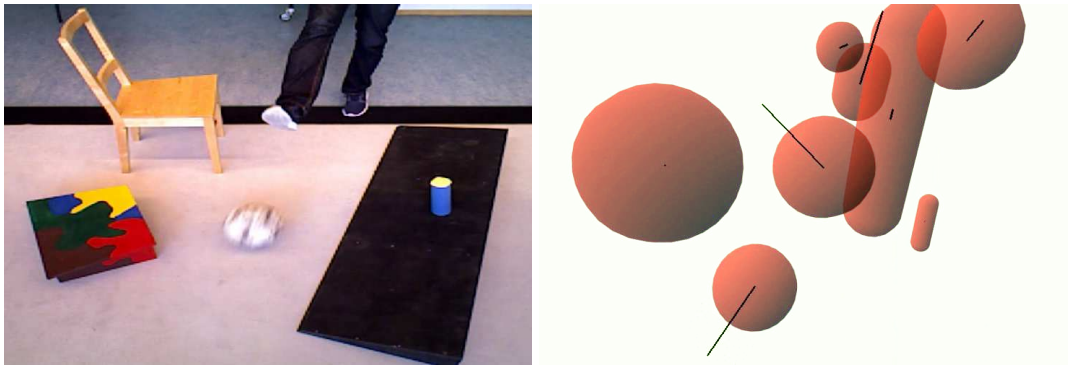


Figure 5.11: GMM clustering with Kalman filter for obstacle tracking. Left: in the reference RGB figure it can be seen how a person suddenly kicks a ball. Right: several SSVs can be tracked simultaneously (surfaces are discarded for this example); velocity vectors are shown as black segments.

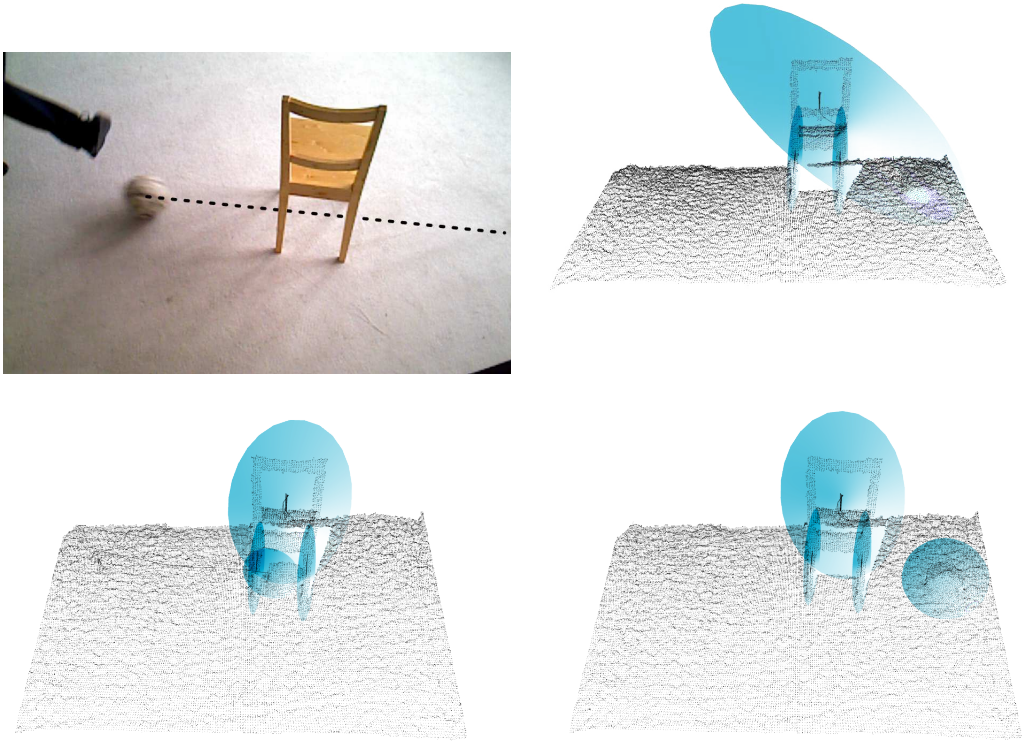


Figure 5.12: Keeping separate obstacles separated. In this example, a ball rolls under a chair (top left). The standard application of the EM algorithm doesn't take into account dynamic scenes and obstacles are joined together (top right, $\lambda = 0$). A low-pass filter on Σ (see eq. (5.12)) keeps a better track of the ball (bottom, $\lambda = 0.95$). Only the Gaussians are shown here for simplicity.

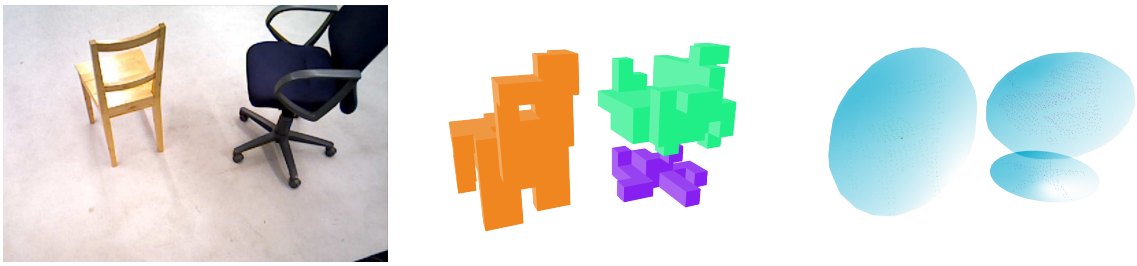


Figure 5.13: Determination of the correct number of Gaussians. From left to right: two chairs, voxel grid for quick clustering and point cloud with approximating Gaussians (the ground is hidden).

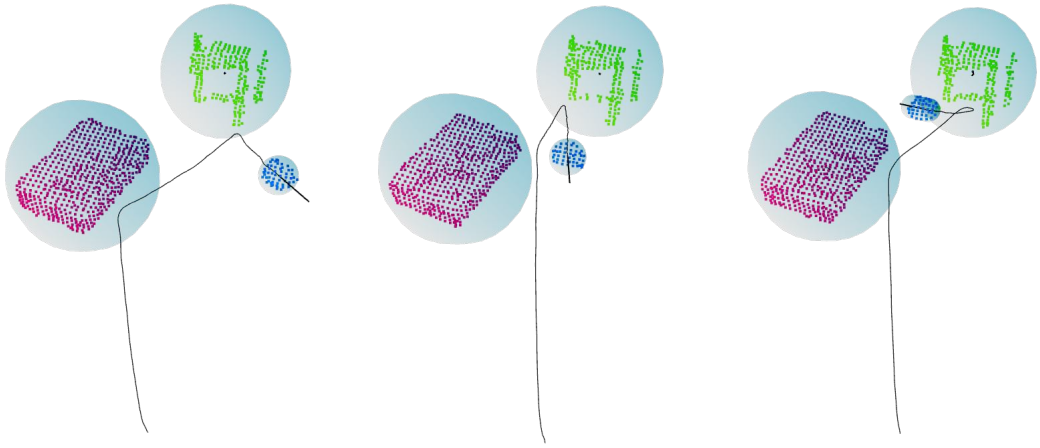


Figure 5.14: Filtered trajectories and estimated velocities of a ball bouncing off two obstacles. The ball is correctly tracked in three separate experiments.

An intrinsic problem of such a probabilistic method consists of determining the correct number of Gaussians (or obstacles) in the scene, especially when handling dynamic scenarios. In order to solve it, the points are grouped according to a simple voxel grid with a coarse resolution (see fig. 5.13) such as the one used for surface clustering (section 5.4) but in 3D (note again that the 3D grid is only used for clustering while the original points are passed along to further stages). Again a quick clustering of the scene is obtained, consisting of several "qclusters". These are used to split Gaussians that contain a considerable amount of points in more than one qcluster (see middle image in fig. 5.13). Additionally, new obstacles are created when a qcluster is found on which most points don't belong to previous Gaussians. Obstacles are removed when they present low weight values π_k, as they are not needed to represent existing points. Using these criteria to add, split and remove Gaussians from the scene, the distribution is initialized with one Gaussian for the entire point cloud and iterated further. Even in complex scenarios, initialization time is fast enough for the present application (note that each obstacle can be approximated with more than one SSV later). In fig. 5.14 the final result of the tracking algorithm can be seen: a ball is consistently tracked when bouncing against two other static objects.

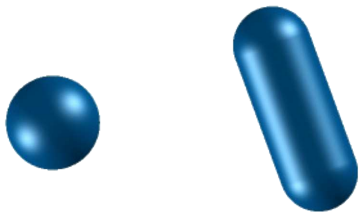


Figure 5.15: Geometries used by the vision system for the approximation of objects: point-SSVs (left) and line-SSVs (right)

Approximation

The SSV approximation is based on the maximum, middle and minimum principal moments of inertia $I_{\max}$, I_{mid} and $I_{\min}$ of every obstacle point cloud (sometimes called "principal axis decomposition"). The quotients $\xi_1 = \frac{I_{\min}}{I_{\max}}$ and $\xi_2 = \frac{I_{\text{mid}}}{I_{\max}}$ are geometric invariants, as they don't depend on the scale of the point cloud. Other authors used similar invariants to fit PCs using prisms or superellipsoids [34, 35]. In this case, the environment approximation uses only point- and line-SSVs (see fig. 5.15), which satisfy:

- Ideal point-SSVs have $\xi_1 = \xi_2 = 1$
- Ideal line-SSVs have $\xi_1 < \xi_2 = 1$

These criteria could be directly used to approximate objects. However, as detected point clouds are incomplete, they have to be adapted to experimental values. The geometric invariants ξ_1 and ξ_2 were analized for a series of objects resembling point- and line-SSVs. They differ from the ideal values due to the point clouds being hollow, incomplete and noisy. Based on results of these PCs, the following characteristics were obtained:

- Real spheres satisfy $(\xi_1 > 0.1) \cap (\xi_2 > 0.8)$
- Real capsules satisfy $(\xi_2 < 0.25)$

In this way, point clouds (PCs) that satisfy either condition are directly approximated with a corresponding sphere or capsule. When neither criterion is satisfied, a more detailed approximation could be achieved by iteratively splitting the point cloud (again) and assigning more than one SSV object to every obstacle (e.g. the chair approximation in fig. 5.2). However, splitting is not always helpful or necessary. To prevent the scene approximation from reaching unnecessary complexity levels, PCs are split until any of the following conditions apply:

1. It "resembles" a perfect sphere or capsule (using the above mentioned criteria).
2. The distance to the robot is bigger than a certain threshold.
3. Its volume is smaller than a certain threshold.
4. It is the result of a certain number of "splitting steps".

These conditions have the purpose of using more detailed representations of obstacles only when they are needed. They keep the necessary number of SSVs to represent the environment at a minimum, thus reducing the computational cost of the step-planner and collision avoidance modules. This hierarchical strategy only improves the approximation of objects that are relevant to the robot's navigation. As mentioned before, objects that resemble ideal

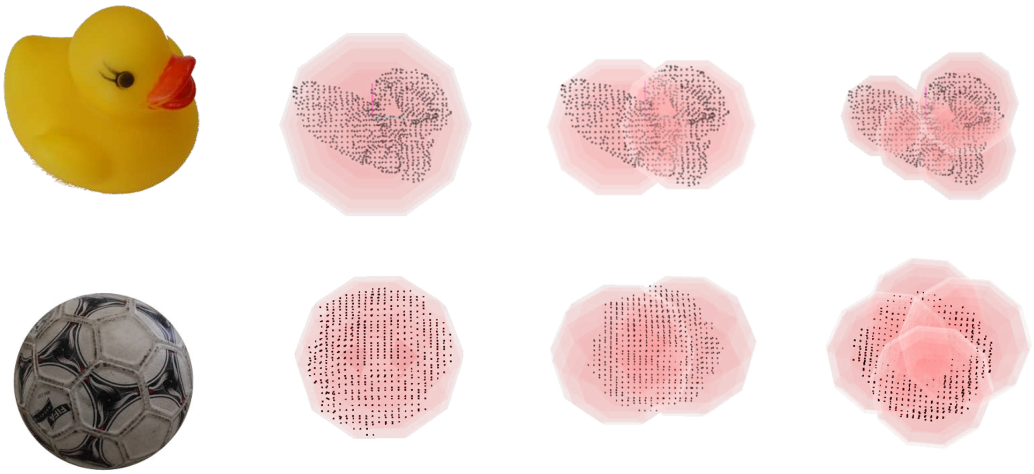


Figure 5.16: Approximation of point clouds. From left to right: a picture of the object, the point cloud (black) and SSV approximation (pink) without splitting, after 1 splitting step, after 2 splitting steps. In the second example, it can be seen that the splitting operation can result in worse approximations for objects that already resemble ideal geometries. Adapted from Lalić [103].

geometries don't require a more detailed approximation. In fact, in these cases the splitting operation can be counterproductive, as shown in fig. 5.16.
The other conditions are implemented to prevent the algorithm from splitting objects that are either too far away to have an influence in the planning process or too small compared to the robot to require a detailed approximation. Additionally, the number of splitting steps is limited to prevent excessive runtime. A sphere is chosen as the default approximation geometry for a non-ideal PC due to its reduced distance calculation times. In the following, a strategy for splitting point clouds is introduced.
Each incoming point cloud PC_i is split using a plane ψ that satisfies:

$$\overline{p}^{(PC_i)} \in \psi \tag{5.13}$$

where $\overline{p}^{(PC_i)}$ is the PC's centroid (or center of mass). Thus, each of the resulting sub-PCs PC_{i_1} and PC_{i_2} will have approximately the same number of points. The next question is to decide which plane to choose (in other words, to define the orientation of ψ). In order to obtain an efficient representation of the environment, ψ should be chosen in a way that results in sub-PCs with more resemblance to perfect spheres or capsules, so that they don't have to be split again. This is achieved by analyzing the following example of a full symmetric PC (see fig. 5.17):
Let PC_s be a PC plane-symmetric with respect to three orthogonal planes. The coordinate system $S(x,y,z)$ is defined coincident with the eigenvectors of $I^{(PC_s)}$ (principal axes of inertia) and its origin in $\overline{p}^{(PC_s)}$. If ψ is chosen as the $x-y$ plane, PC_s is split into two *identical* sub-PCs PC_{s_1} and PC_{s_2} (any points $\in \psi$ are not taken into account). PC_{s_1} and PC_{s_2} are symmetrical with respect to the planes $x-z$ and $y-z$, so their principal axes of inertia are parallel to the axes of $S(x,y,z)$. Applying Steiner's Parallel Axis Theorem,

$$I_{ij}^{(PC_s)} = 2\left[I_{ij}^{\left(PC_{s_1}\right)} + m^{\left(PC_{s_1}\right)}\left(\sum_k a_k^2\delta_{ij} - a_i a_j\right)\right] \tag{5.14}$$

where

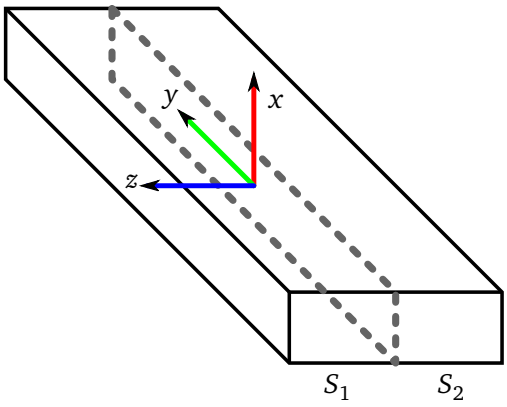


Figure 5.17: Splitting point clouds. In order to illustrate the strategy, a symmetric point cloud is used as an example (here a symmetric prism is shown for clarity). In this case, the splitting plane corresponds to I_{mid} which results in objects more similar to ideal line-SSVs.

- $I_{ij}^{(PC_x)}$ is the point cloud's PC_x inertia tensor
- $i, j, k \in \{1, 2, 3\}$
- $m^{\left(PC_{s_1}\right)}$ is the number of points (mass) of PC_{s_1}
- δ_{ij} is the Kronecker-delta
- The vector $a = (a_1, a_2, a_3)^\intercal = \overline{p}^{(PC_s)} - \overline{p}^{\left(PC_{s_1}\right)}$

Because of the choice of ψ, $a_1 = a_2 = 0$.

$$\therefore I_{ij}^{\left(PC_{s_1}\right)} = \begin{cases} \frac{1}{2}\left(I_{ij}^{(PC_s)} - m^{(PC_s)} a_z^2\right) & : i = j = 1, 2 \\ \frac{1}{2}\left(I_{ij}^{(PC_s)}\right) & : i = j = 3 \\ 0 & : i \neq j \end{cases} \tag{5.15}$$

It is important to observe the relationships obtained in eq. (5.15): as the term $\left[m^{(PC_s)} a_z^2\right]$ is always positive, the values of I_{xx} and I_{yy} in the new sub-PCs are proportionally smaller than the value of I_{zz} with respect to the corresponding values in the initial PC. Supposing that PC_s doesn't resemble a point- or line-SSV and the axes corresponding to I_{max}, I_{mid}, I_{min} are the same for PC_{s_1} as for PC_s, the relationships shown in Table 5.1 are obtained. Out of

Table 5.1: Choosing the splitting plane

z corresponds to I_{min}	z corresponds to I_{mid}
$\xi_1^{\left(PC_{s_1}\right)} > \xi_1^{(PC_s)}$	$\xi_1^{\left(PC_{s_1}\right)} < \xi_1^{(PC_s)}$
$\xi_2^{\left(PC_{s_1}\right)} < \xi_2^{(PC_s)}$	$\xi_2^{\left(PC_{s_1}\right)} > \xi_2^{(PC_s)}$

these relationships it can be concluded that, in this example, splitting a PC through a plane perpendicular to the axis corresponding to I_{min} or I_{mid} results in sub-PCs having a closer resemblance to ideal point-SSVs or line-SSVs respectively. That means, applying these splitting strategies to the hollow PCs of symmetric objects would favor reaching the criteria for real sphere or capsule approximation. Although PCs rarely correspond to symmetrical objects,

experiments with all kinds of objects and scenarios showed a similar trend. Regardless of the chosen strategy, a safe approximation is assured in the next step by the chosen fitting values. The fitting values for each incoming point cloud PC_i are also based on its inertial parameters. The potential collision region is covered by including all obstacle's points in the SSV. This is especially relevant as it guarantees a safe approximation, regardless of the identification and splitting parameters. The fitting values (or SSV approximation parameters) are then heuristically determined in the following way (see fig. 5.18):

1. Fitting a point-SSV with center o and radius r:

$$o = \overline{p}^{(PC_i)} \tag{5.16}$$

$$r = max\left(\left|p - \overline{p}^{(PC_i)}\right|\right) \tag{5.17}$$

2. Fitting a line-SSV with centers o_1 and o_2 and radius r:

$$o_1 = \overline{p}^{(PC_i)} + \kappa_1 \mathbf{c} \tag{5.18}$$

$$o_2 = \overline{p}^{(PC_i)} + \kappa_2 \mathbf{c} \tag{5.19}$$

$$r = max\left(\left|\left(p - \overline{p}^{(PC_i)}\right) \times \mathbf{c}\right|\right) \tag{5.20}$$

where

- $\mathbf{c}$ is the directional vector of the principal axis of inertia corresponding to $I_{\min}$
- $\kappa_1 = max\left(\left(p - \overline{p}^{(PC_i)}\right)\mathbf{c}\right)$
- $\kappa_2 = min\left(\left(p - \overline{p}^{(PC_i)}\right)\mathbf{c}\right)$
- max and min functions are evaluated $\forall$ point $p \in PC_i$

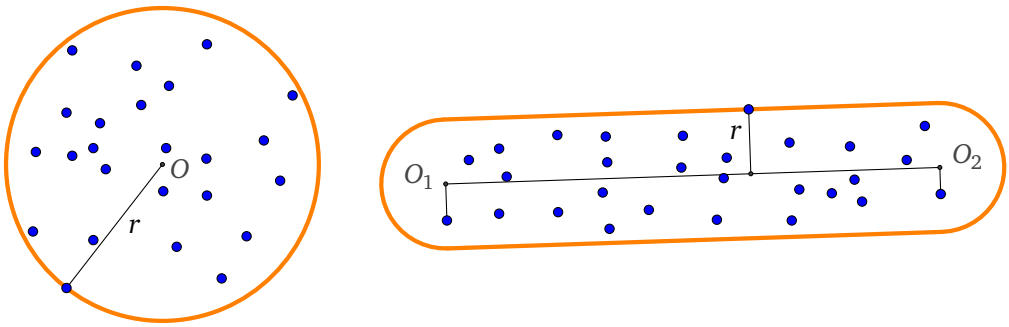


Figure 5.18: SSV approximation of point clouds. After clustering, identification and splitting, point-SSVs (left) and line-SSVs (right) are fitted using the centroid O and projections O_1 and O_2 to the $I_{\min}$-corresponding axis, respectively. The radius r corresponds to the farthest-away-point in each case. On the right, the axis of inertia and projections on it are represented with black lines.

Evaluation

Before implementing such a system on a real robot, it is useful to evaluate its performance in simulation. By creating artificial scenarios, perception and planning algorithms can be tested

and adjusted to obtain the desired results. This strategy has become especially important in the last few years thanks to the development of autonomous vehicles [113]. In order to evaluate the effectiveness of the perception algorithms, several environments are simulated using synthetic point clouds.

Figure 5.19: Generation of synthetic point clouds. Using a large database of predefined objects and a random list of positions and transformations, infinite scenarios can be simulated. The result is filtered using the camera's position and FOV and then sent to the perception system.

The process is illustrated in fig. 5.19. First, a dataset of 3D files is created, including one for a large floor area from which point clouds are generated and transformed into point cloud files. By scaling, transforming and combining these objects, increasingly complex scenes can automatically and randomly created. Moreover, moving objects can be simulated by applying frame-varying transformations and generating a stream of point clouds. Additionally, in order to better recreate the real scenario, the robot's point of view and its resulting occlusion effect are also taken into account by *frustum culling* and *ray casting* [145]. They consist of filters that simulate the existence of a camera. By defining a camera position, orientation and parameters, points are first reduced to the ones existing inside the camera's field of view and range (frustum culling); then, they are iteratively checked for occlusion against the camera position and removed (ray casting). The result can be seen in fig. 5.20.
With these synthetic scenarios, the approximation of obstacles, surfaces and tracking of dynamic objects are evaluated. By randomly varying scale, transformation and combinations of platforms and objects with different shapes, parameters are adjusted and the results of both the *Surface* and *Obstacle Approximation* can be compared against scaled ideal values of the original object dataset. Around 100 different scenarios were tested.
For the polygon evaluation, inclined platforms were used with different sizes and the following shapes: circles, rectangles with normal/round corners, ellipses, regular and irregular convex polygons and polygons with some rounded corners. The error in inclination is negligible in all cases. While the error in area lies in the range of 0-4 % for polygons, the error in area can be up to 10 % in the case of rounded shapes: this is mainly caused by the limited number of vertices used in the polygon approximation. However, the approximated area is always smaller than the original area and the remaining points are approximated with SSVs so the result is always safe for navigation.
In the case of the SSV evaluation, prisms, cylinders, platforms and combinations of more than one shape were used. The volume of the approximating SSVs varies between 100-300 % of the original shapes, with the best results corresponding to rounded shapes and higher number of splitting steps. As expected, the volume of the approximation is consistently higher than the original shapes (due to the conservative fitting strategy).
Using a stream of point clouds, the obstacle tracking is first evaluated (see fig. 5.21). The obstacle tracking algorithm, running at 30 Hz, is capable of tracking objects moving with constant speeds up to 3 m/s (at higher speeds, the displacement between frames is too large to be correctly matched). Around 200 simulations of randomly-sampled velocities between 0.01 m/s and 3 m/s were performed. Convergence of the estimated velocities to a value with less than 3 % error takes between 30-60 frames for the fastest moving objects[5].
The evaluation of synthetic point clouds is a valuable tool for development. Additionally, it helps to validate the capacities of the developed system. It is capable of correctly approximating a large variety of dynamic scenarios. Errors may become significant due to the sim-

[5]The initialization time of the algorithm required when starting the robot is not taken into account.

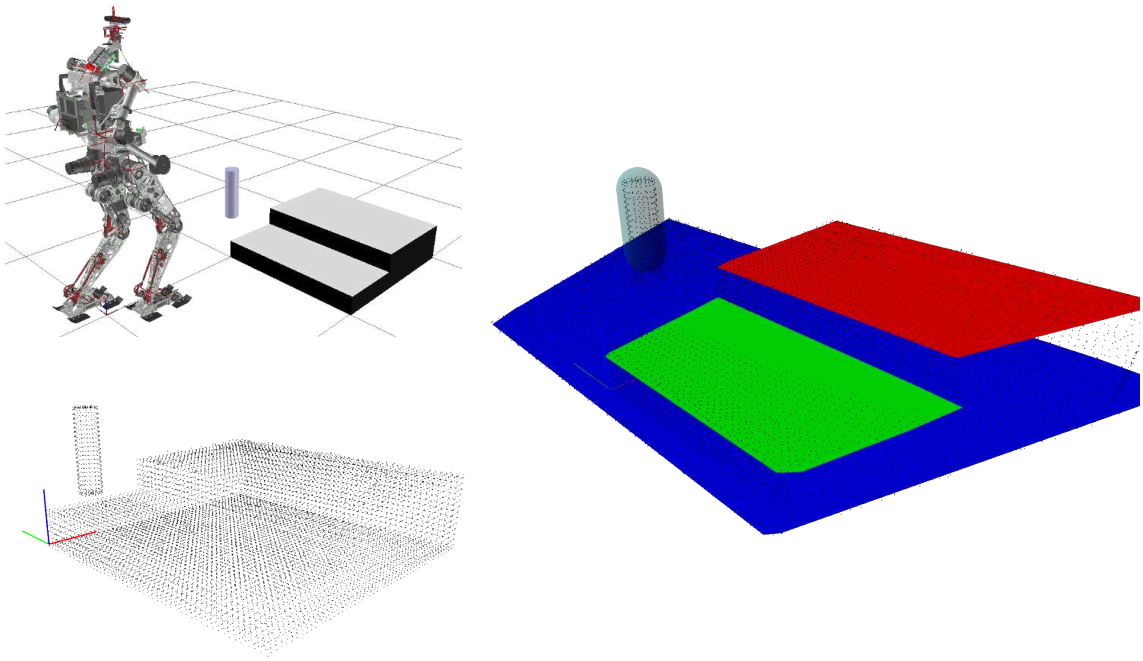


Figure 5.20: Evaluation of the algorithm via synthetic point clouds: 3D objects (top left) are transformed into point clouds (bottom left) which are combined with the floor, filtered via frustum culling and ray casting, and approximated by the vision system (right).

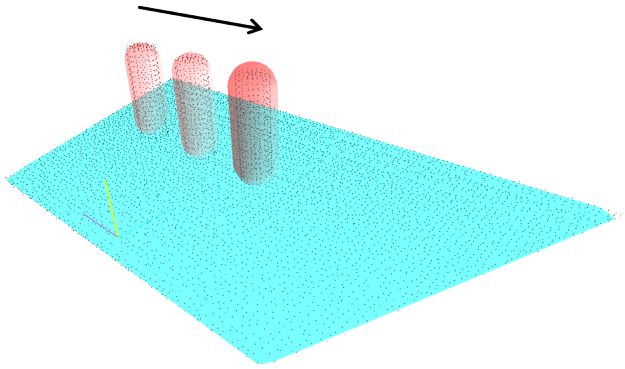


Figure 5.21: Obstacle tracking performance. An object moves with constant speed and is approximated with an SSV. Objects moving with constant speeds up to 3 m/s can be successfully tracked.

plifications necessary for fast processing, but they are always conservative and safe for robot navigation. The performance of the perception system is further validated with experiments on *Lola* as shown in chapter 8.

Chapter 6

Robustness against Perception Errors

Effect of Perception Errors

As with any sensing process, a robotic perception system contains different kinds of error sources which affect its accuracy. These are mainly the underlying error in the sensing technology itself (see section 3.2), in the calibration process (see section 3.4) and the processing algorithms (see chapter 5). Some errors are inherent to the robotic system (for example, occlusion errors cannot be avoided when relying only on on-board sensing). Others are variable: sensor errors may depend on lighting conditions or surface reflectivity; the accuracy of an approximation strategy may depend on the particular scene being approximated; the filtering algorithms are affected by the scene's and the robot's dynamics (for example, the odometry information used for calibration may become less accurate when the robot is moving and the filtering of a particular object depends on the number of frames it can be perceived).
Naturally, by limiting the environment complexity or the robot's behavior some of these errors sources could be reduced. For example, a scenario could contain only components of certain material and regular shapes; the robot could move slowly to reduce odometry errors and obtain a more accurate representation through filtering and further processing. Moreover, additional or better sensors could be used to reduce occlusion or measurements errors. However, even though perception errors could be reduced, they cannot be eliminated. There will always be situations where the robot may deal with inaccurate perception information or irregular environment.
Instead of trying to improve the sensor's precision, another possible strategy to deal with perception errors consists of increasing the robustness of the walking controller against them. One of the main advantages of this approach is that it simultaneously improves the robustness against irregular terrain: a robot walking over structured scenarios with large perception errors can be compared to a robot with a more accurate perception system over more irregular terrain[1]. The perception sensor used in this work is considerably noisy (see section 3.2); additionally, as explained in chapter 5, perception strategies were chosen for runtime and performance, not accuracy (in fig. 6.1 the error in the approximation of a platform can be clearly seen). Therefore, in order to achieve fast walking over irregular terrain, the walking controller has to be robust against these errors. This chapter deals with this topic: by considering possible errors explicitly in the planning module this robustness can be greatly improved. It is the result of collaboration with Tilman Knopp and Yizhe Wu [97, 205]. Some of the results presented in this chapter have been previously published in more compact form in international journals and conferences [190, 191].
As explained before, the environment is classified in two: obstacles which the robot has to avoid and surfaces over which it can walk. Dealing with the obstacle errors is straightforward:

[1] In this case, "irregular terrain" is used to refer to scenarios where the location of hard contact with the ground is unclear (such as cobbles) and not as an indication of the terrain properties.

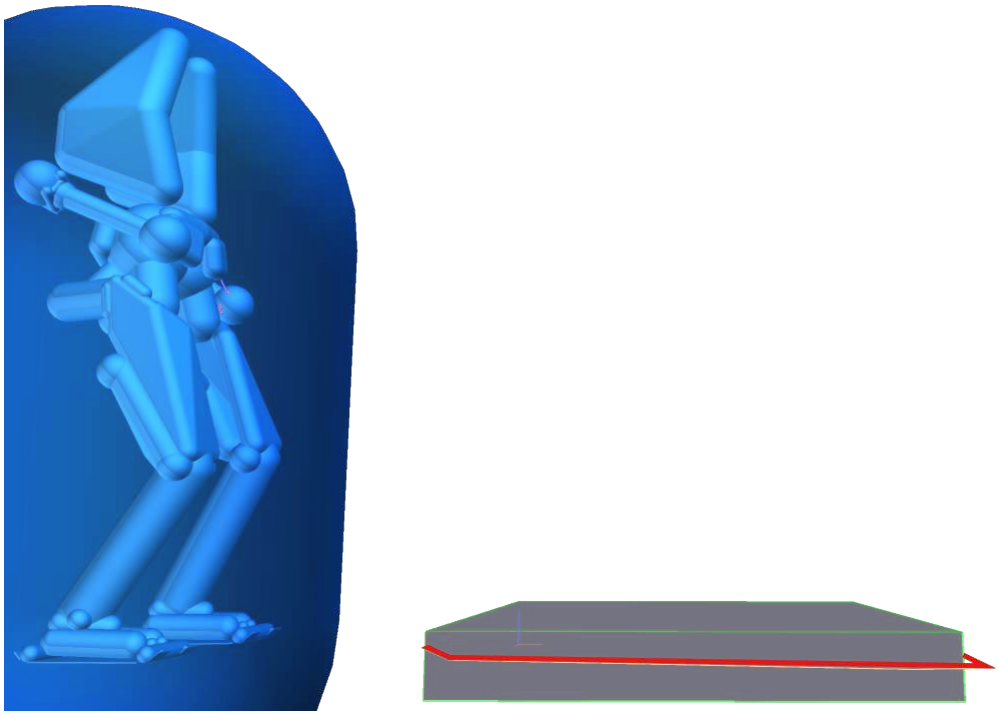


Figure 6.1: Errors in the vision system. The surface modeling of a 12.5 cm high platform during an experiment is shown against an ideal model of the platform in the robot's planning system.

a safety factor can be included to maintain a more conservative distance against the obstacles. However, dealing with errors in surface approximations is considerably more complex (see fig. 6.1), as these are the objects which the robot is in contact with and on which its stability depends. In this chapter, the effect of surface approximation errors is analyzed in detail and a new walking controller is presented which takes them directly into consideration. The modifications to the walking controller are performed at two different stages. First, before the planning takes place, the landing location is optimized to reduce the destabilizing effect of sensor errors depending on the robot's dynamics. This optimization is complemented at a later stage with a more robust control: the planning module is modified to include time-variable phases that directly deal with unexpected contact situations. The methods presented here can be implemented on any robot following a ZMP-based control, regardless of the perception sensor used.

Related Work: Robust Walking

Classic biped walking controllers assume a perfectly flat, rigid surface on which the robot walks (see chapter 2). When using a vision system to detect those surfaces, most authors assume perception errors to be small enough to be compensated by control [24, 28, 47, 59, 60, 65, 68, 119, 137, 140, 172, 188]. These perception systems employ different kinds of on-board or external sensors that are subject to errors, regardless of the environment representation used afterwards. In the case of Kagami et al. [82], the 2.5D map for their H7 robot had an accuracy of around 1.5cm. This error didn't have an influence, as experiments were performed over flat ground with considerably larger obstacles. In Gutmann et al. [59] and Gutmann et al. [60], the segmented 2.5D map used to recognize surfaces above the ground presented an error below 1.5cm. Chestnutt et al. [24], Michel et al. [119], and Nishiwaki

et al. [137] used a similar representation that allowed their full-sized biped robot to walk over complex scenarios; reported perception errors lay around 1cm. In the vision systems for the Atlas robot by Stumpf et al. [172] and Fallon et al. [47] the ground detection error was less than 3cm. In all these works, experiments were performed on static environments with relatively flat walking surfaces and at considerably slow walking speeds. Perception errors in these almost quasi-static conditions could be compensated by the robot's feedback stabilization system.
When walking over more cluttered terrain and at higher speeds, biped robots may experience larger perception errors with a more pronounced effect on their stability [18]. In order to cope with such *Early Contact (EC)* or *Late Contact (LC)* events, many authors exploit additional sensor information (typically, contact sensors in the robot's feet) and phase-switching mechanisms [18, 125, 147, 149, 171]. Additional robustness may be achieved with state estimation algorithms and real-time footstep modifications [201–203].
In summary, works available in the literature always (as far as the author knows) compensate perception errors with robust control. As mentioned in chapter 2, this strategy has its advantages and is therefore applied in this work (section 6.4). However, it is possible to mitigate the effect of these errors before reaching to that point. Specifically, by modifying the detected ground height before it is sent to the motion planner. The premise is simple: "is it possible to find an optimal value for the assumed ground height in terms of stability?". This question can become even more relevant when walking over irregular surfaces (e.g. gravel, grass), where a perception system could provide a variable accuracy factor (e.g. the standard deviation) for each detected surface. In order to answer that question, the term "stability" has to be accurately defined and quantified. In the following, a simplified model of the robot *Lola* is used to analyze how *EC* and *LC* events affect the robot's walking and deal with this issue.

Ground Estimation Modification

Problem Statement

The following analysis is not based on specific sensor data. The results presented rely only on the robot's dynamics and the consequences of *EC* and *LC* events. Therefore, it may be applicable to many different systems where the robustness may be influenced by this open-loop control strategy. It is worth mentioning, however, that it is inspired by a scenario consisting of a biped robot and a perception system for ground (or surface) detection.
It is assumed that the error distribution p of the ground estimation is known beforehand. For this work, only the error in the vertical (z) direction is considered. In order to obtain computable results (section 6.3.5), it is assumed that the error is constrained between finite limits (i.e. errors outside those bonds are neglected). In general:

$$1 = \int_{-\infty}^{+\infty} p(z)\,dz \simeq \int_{z_{\min}}^{z_{\max}} p(z)\,dz \tag{6.1}$$

with the limits $z_{\min} < z_{\max}$.
By considering the system's expected ground height $z_{\exp} \in [z_{\min}, z_{\max}]$ and the ground's real height $z_{\text{real}} \in [z_{\min}, z_{\max}]$, the *EC*, *LC* and *Ideal Contact (IC)* events can be defined:

$$\begin{cases} EC & : z_{\exp} < z_{\text{real}} \\ LC & : z_{\exp} > z_{\text{real}} \\ IC & : z_{\exp} = z_{\text{real}} \end{cases} \tag{6.2}$$

In usual cases, z_{exp} is defined as the sensor's output or expected value $z_{\text{exp,sensor}} = E(Z) = \int_{-\infty}^{+\infty} zp(z)\,dz$. The objective is to find an optimal value $z_{\text{exp,opt}}$ such that the negative effect of all possible *EC* and *LC* events on the robot's *stability* is minimized. In order to be applicable, the solution has to be computable online while the robot is walking.
To reduce computational costs and simplify the analysis, it is restricted to the following case:

- The robot's control uses a fixed step duration T_{step}.
- Fixed phase durations and no footstep modification mechanisms are assumed since they are supposed to work in a later stage.
- The ground is horizontal and $z_{\text{exp,sensor}} = 0$.
- Only the dynamics in the sagittal plane are considered.

As explained throughout this section, these hypothesis enable a fast calculation of an adequate solution which can be applied to more general scenarios with small modifications.
Classical concepts of *stability* are difficult to apply in non-linear, non-smooth systems such as biped robots. Although some authors use the concept of *basin of attraction* to define regions where a periodical gate might be stable [148], there is no consensus on a practical definition of stability applied to humanoid robots and it is unclear how it could be quantified [13]. In this section, different variables related to the robot's stability are introduced and their potential applicability as quantifiable *stability indicators* to the present problem is analyzed. Throughout this work, a *stability indicator* is defined as a function of the robot's state that is convex with respect to $\left(z_{\text{real}} - z_{\text{exp}}\right)$ and presents a global minimum at $\left(z_{\text{real}} - z_{\text{exp}}\right) = 0$.

Model and Simulation

The reaction of a robot to unexpected ground heights could be evaluated in experiments, but other model errors and disturbances make it difficult to isolate a single factor. Repeatability can only be achieved in simulation. Therefore, *Lola*'s full multi-body simulation [13], which handles unilateral and compliant contacts and takes motor dynamics and control loops into consideration, is used as reference.
However, its high computational cost makes this simulation impractical for generating large amounts of data or implementing it in a real-time scenario (ref. section 6.3.5). Therefore, a simplified three-mass model (chapter 2) is used as reference. For this analysis an extension of the model presented in Wittmann et al. [201] which can handle different ground heights is used. In the following, the model's main properties are explained.
The robot is modeled with one point mass for each leg m_f at the feet and an upper body with mass m_b (near the center of mass) and inertia Θ_{zz} (ref. fig. 6.2).
The upper body and feet are assumed to follow the ideal trajectories $\mathbf{x}_b$ and $\mathbf{x}_{1,2}$ in the robot's Frame of Reference (FoR). These are obtained from a reference run of the multi-body simulation (the ideal trajectories are generated by the simulated robot control, see chapter 2). The underactuated state is simulated via two passive DoFs between the FoR and the ground $(x_{\text{I}} - z_{\text{I}})$: a vertical displacement z_{FoR} and an inclination on the sagittal plane φ_{FoR}.
Contact interactions between feet and ground are modeled as point vertical forces, with a unilateral, linear spring (k_c) and damper (d_c) model. The center of pressure is fixed to the contact point. The force control and upper body stabilization is taken into account with an additional stabilization variable T_{stab}. It follows a PD-control (control gains K_p and K_d) based on the one implemented in *Lola*'s control:

$$T_{\text{stab}} = sat(-K_p \varphi_{\text{FoR}} - K_d \dot{\varphi}_{\text{FoR}}) \tag{6.3}$$

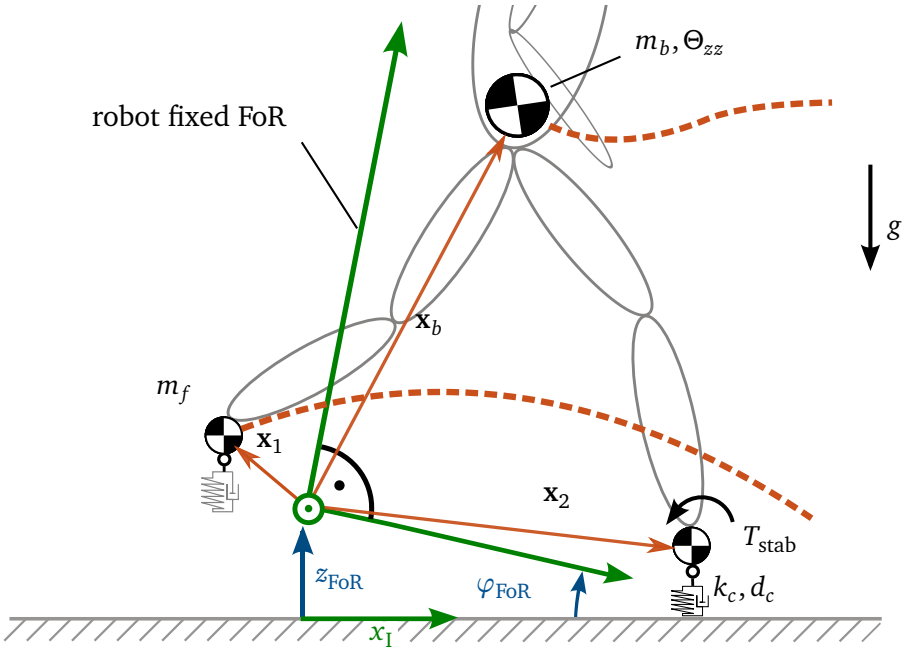


Figure 6.2: Three-mass model in the sagittal plane with two unactuated DoFs and unilateral compliant contacts.

and is saturated at $\pm\frac{\text{sole length}}{2}\left(m_b + 2m_f\right)g$. The equations of motion of the system can then be written as:

$$\mathbf{M}(\mathbf{q}, t)\ddot{\mathbf{q}} + \mathbf{h}(\mathbf{q}, \dot{\mathbf{q}}, t) = \boldsymbol{\lambda}(\mathbf{q}, \dot{\mathbf{q}}, t) \tag{6.4}$$

where $\mathbf{M}$ is the mass matrix, $\mathbf{h}$ contains the Coriolis, centrifugal and gravitational forces and $\boldsymbol{\lambda}$ the contact forces. The state vector $\mathbf{q} = [z_{FoR}, \varphi_{FoR}]^T$ consists of the system's unactuated degrees of freedom. The trajectories of the three masses $\mathbf{x}_b$ and $\mathbf{x}_{1,2}$ and their derivatives are expressed in terms of the inertial coordinate system ICS:

$${}^{ICS}\mathbf{x} = {}^{ICS}_{FoR}\mathbf{T}(\mathbf{q})\,{}^{FoR}\mathbf{x} \tag{6.5}$$

where ${}^{ICS}_{FoR}\mathbf{T}$ is the transformation between both coordinate systems. It consists of a rotation around φ_{FoR} and a translation along z_{FoR}.
Out of (6.3) and (6.4) the dynamics can be expressed as a first order differential equation system:

$$\dot{\hat{\mathbf{x}}} = \mathbf{f}(\hat{\mathbf{x}}, t) \tag{6.6}$$

where $\hat{\mathbf{x}} = [z_{FoR}, \varphi_{FoR}, \dot{z}_{FoR}, \dot{\varphi}_{FoR}]^\mathsf{T}$. An empirical initial value for $\hat{\mathbf{x}}$ is chosen so that a stable gait is fast achieved. Every integration step consists of:

1. calculating the point mass locations,
2. solving the contact forces and the stabilization torque T_{stab} and
3. integrating (6.6) with an explicit Euler-integrator.

Stability Indicators

Using the model from section 6.3.2, *EC*, *IC* and *LC* events are simulated by changing the ground height after achieving periodic walking. In figs. 6.3 and 6.4, a simulation example

of an *EC* and an *LC* event, respectively, is shown. A different ground is defined for each foot to avoid horizontal impacts and evaluate only changes in the z direction. The forces on each foot and the inclination of the sagittal plane are shown, as well as the position of the robot (depicted with three segments: one for the upper body and two between $\mathbf{x}_b$ and $\mathbf{x}_{1,2}$) at three time instances a, b and c for reference.

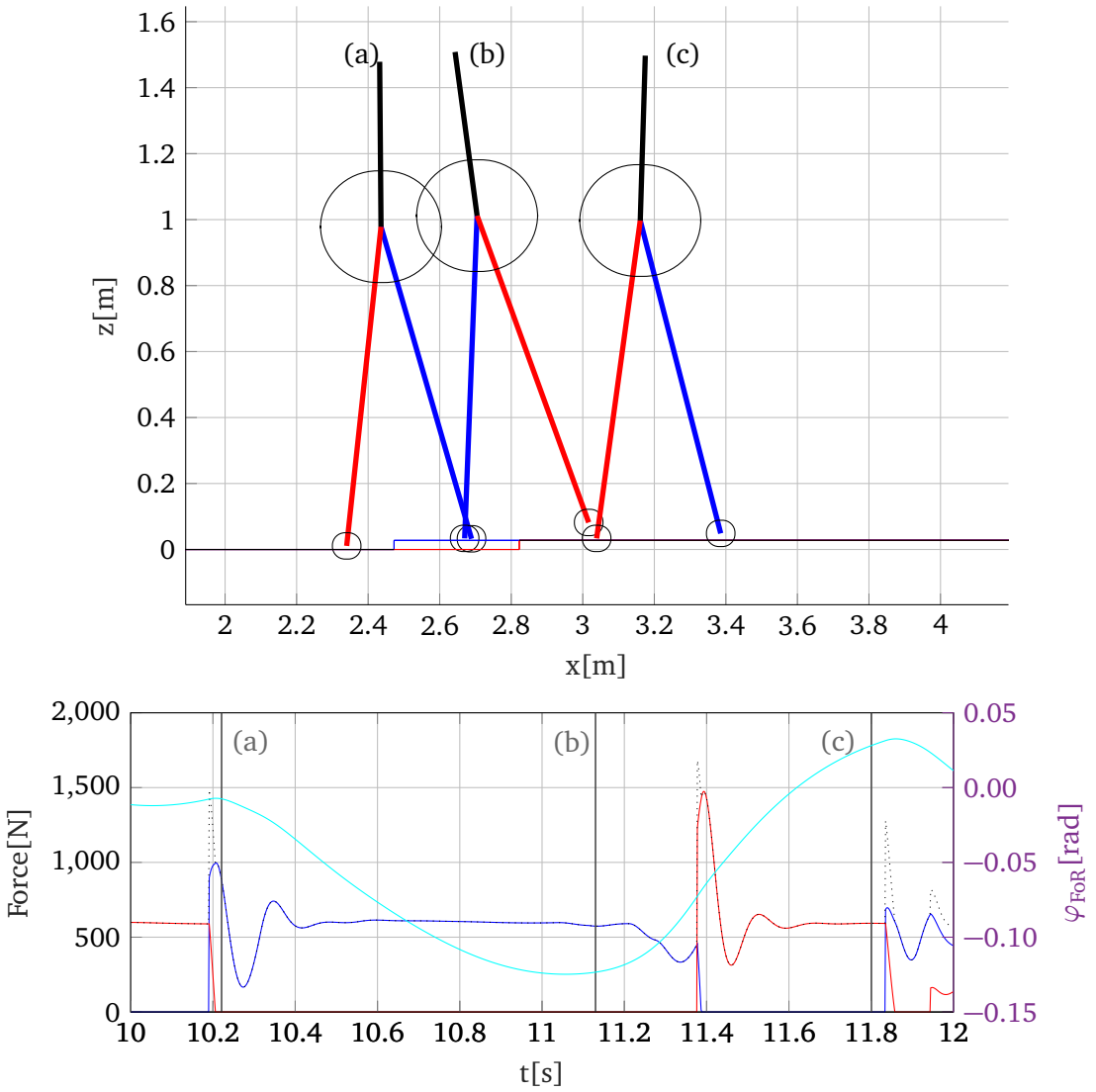


Figure 6.3: A simulation of the three-mass model. Top: the robot position at 3 instances (depicted with three segments and one circle for each point mass). Bottom: the progress of the contact forces for each foot (blue and red), total force (dashed) and the inclination of the sagittal plane (cyan) can be seen for an *EC* event.

These simulations are repeated for different ground heights (positive for *EC* and negative for *LC* events). Thus, the value of different indicators can be evaluated against the *IC* case. Additionally, this is performed for different walking parameters. As an example, fig. 6.5 shows the plot of the angular momentum with respect to the origin for one set of walking parameters and z_{real} varying between -0.05 m and 0.03 m. As the foot's trajectory has a maximum height h of 0.03 m, higher values of z_{real} are not relevant in this case (using standard walking parameters). Besides, the simulations result in the robot loosing balance before reaching those limit values. In the following, some of the stability indicators considered are discussed. Due to the high number of parameters, it is not possible to include all simulation results. Instead, a small selection based on the parameters provided in table 6.1 is be presented. The sagittal

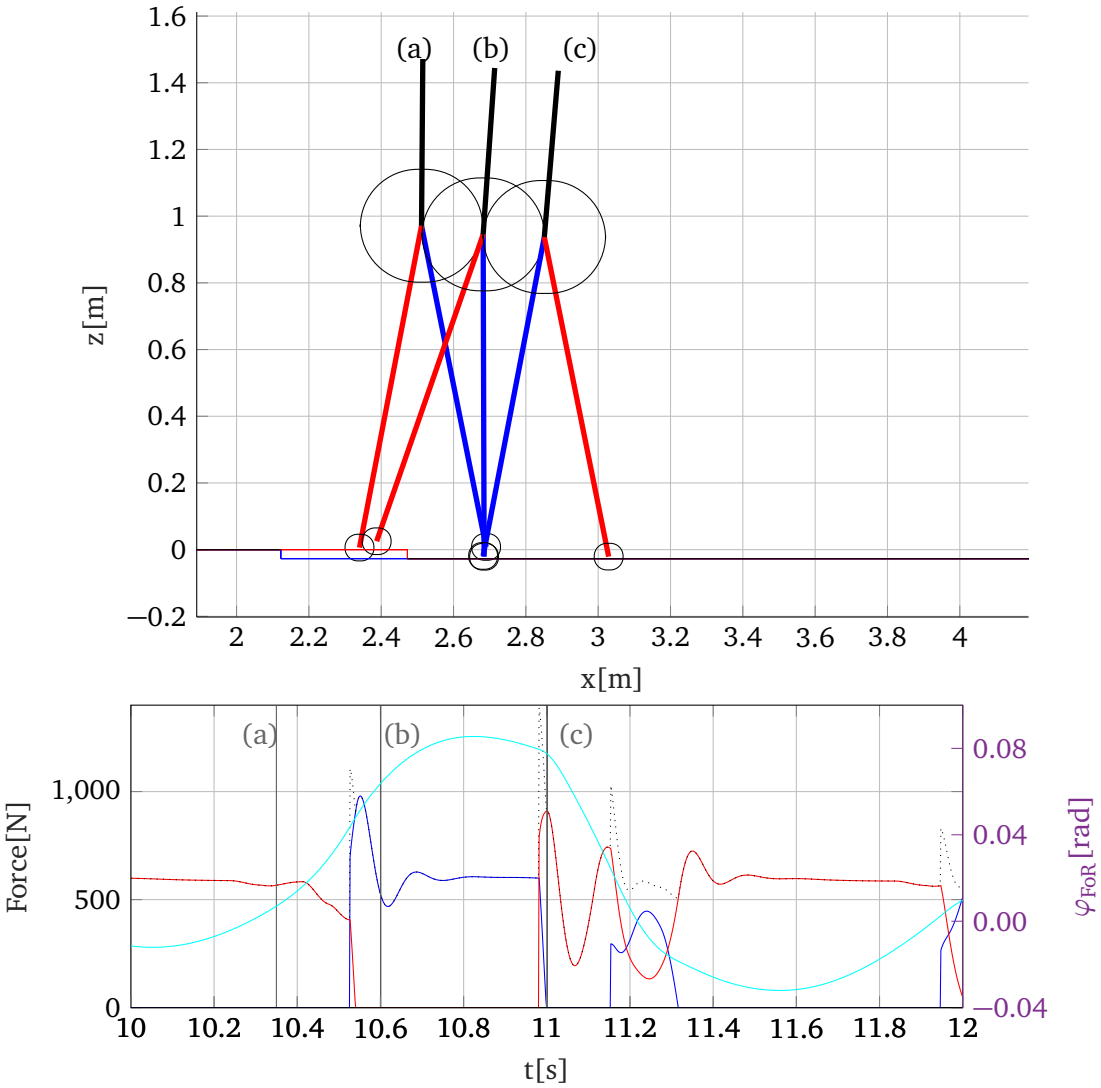


Figure 6.4: A simulation of the three-mass model. Top: the robot position at 3 instances (depicted with three segments and one circle for each point mass. Bottom: the progress of the contact forces for each foot (blue and red), total force (dashed) and the inclination of the sagittal plane (cyan) can be seen for an *LC* event.

Table 6.1: Simulation Parameters

Walking step duration T	0.8s
h	0.03m
Time discretization	0.0015s
Saturation of T_{stab}	±80Nm
Walking step length s	0.25m 0.3m 0.35m 0.4m 0.45m
z_{real}	between -0.05m and 0.03m
Simulation duration	4.8s

inclination of the FoR φ_{FoR} (upper body inclination) is a potential indicator: a high absolute value means that the robot is falling. Nevertheless, during normal walking, it oscillates between -0,012 rad and +0,011 rad. Its periodic motion is not easy to compare or quantify, so the minimum and maximum values φ_{min}, φ_{max} along one run are considered. In the case of an *EC*, the robot tilts backwards and φ_{min} decreases while φ_{max} stays relatively constant. On the other hand, the robot tilts forward in the case of an *LC* and φ_{max} increases while φ_{min} stays relatively constant (this can be observed in fig. 6.9 in the next sub-section). This behavior is strongly dependent on z_{real} but not on the walking parameters, except for large values of $|z_{\text{real}}|$. An indicator considering both effects (e.g. $max\,(|\varphi_{\text{FoR}}|)$) can be applied to the present problem.
Note that the robot's stability is not only influenced by the first contact with an unexpected floor height: if no more changes on the floor height are assumed, the step after an *EC* event consists of an *LC*, and vice-versa; this is due to the change in the robot's state, especially how φ_{FoR} is affected by *EC* and *LC* events (see figs. 6.3 and 6.4). Thus, the values of φ_{FoR} are observed at the first and second contact with the ground after the change in z_{real}, φ_{con_1} and φ_{con_2}. In the performed simulations, φ_{con_1} shows quasi-linear behaviors for small values of z_{real}. More interesting is the behavior of φ_{con_2} which is almost 0 for a range of low values of $|z_{\text{real}}|$ and shows a sudden increase of absolute value outside of that region. This can be explained by the effect of T_{stab}, which is able to counteract small variations of z_{real}. Note that φ_{con_2} indicates the state of the robot at the contact with the ground; at this point, the sudden change in the acting force (through the new contact situation) might hinder the stabilization controller. An indicator such as $|\varphi_{\text{con}_2}|$ is therefore meaningful for this problem.
The angular momentum L^0 with respect to the inertial coordinate system ($x_{\text{I}}-z_{\text{FoR}}$ in fig. 6.2) at its origin[2] also indicates future states of instability. In fig. 6.5, the value of L^0 for one set of walking parameters and different values of z_{real} is shown.

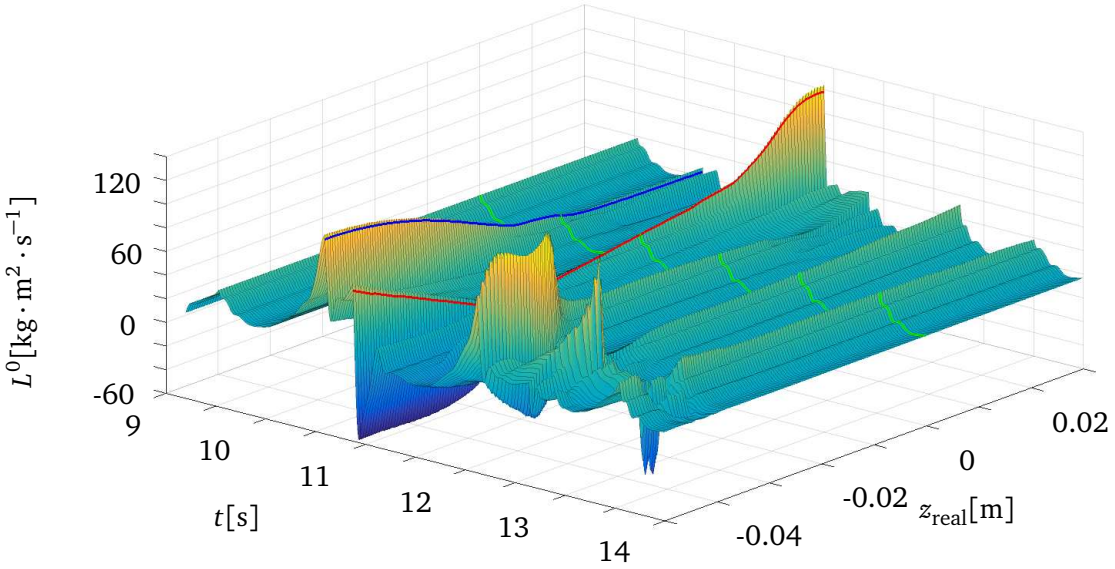


Figure 6.5: Angular momentum with respect to the inertial coordinate system for a walking step length of $s = 0.35m$. The *IC* simulation is depicted in green while the values at the time of the first and second contacts with the modified ground are highlighted in blue and red, respectively.

[2]Choosing the coordinate system's origin as reference point might result in fluctuations of L^0 according to whether the robot is closer or further away from it. However, results show this effect to be negligible.

As with φ_{FoR}, the maximum and minimum values of L^0, L^0_{min} and L^0_{max} are plotted. These are shown for different walking parameters in fig. 6.6, which can be interpreted as the limits of the projection of fig. 6.5 on the $z_{\text{real}} - L^0$ plane.

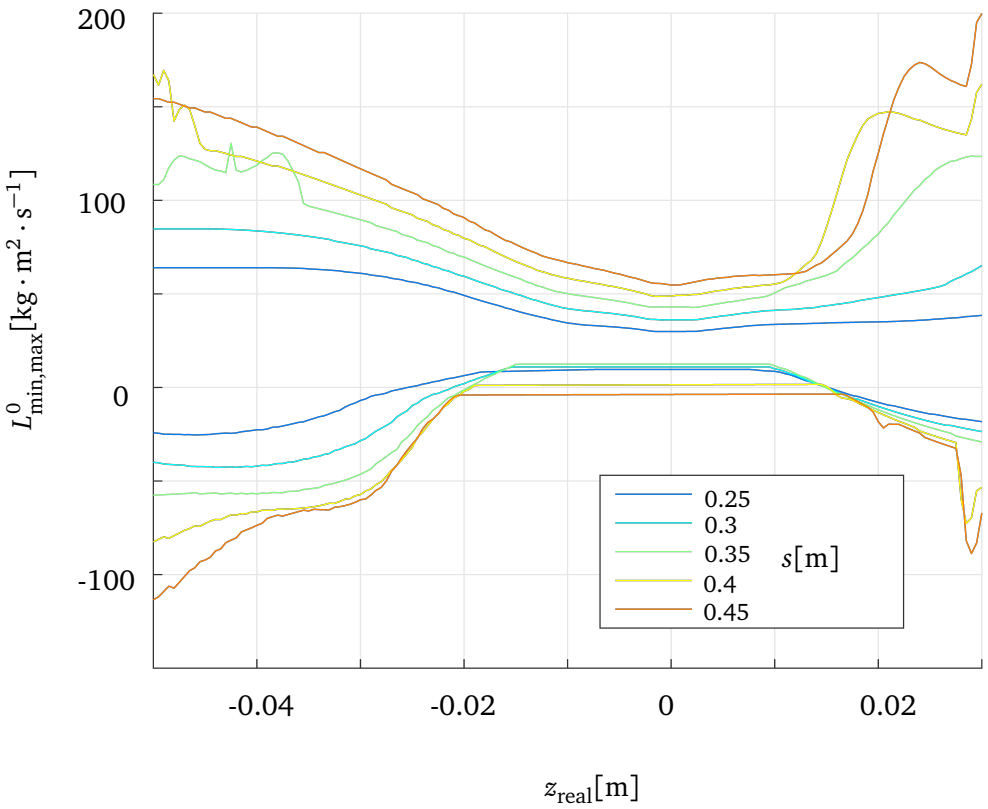


Figure 6.6: Maximum and minimum values of L^0 along z_{real} for different simulations.

Even though no discernible trend can be found on L^0_{max}, L^0_{min} shows a clear constant value for a range around $z_{\text{real}} = 0$ and an increasing deviation outside of it. This behavior allows for a convex indicator in the form of e.g. $\left(L^0_{\text{min}}(0) - L^0_{\text{min}}(z_{\text{real}})\right)$. Another indicator is obtained when considering the timespan t_{sat} in which T_{stab} is saturated, as any values above zero indicate a limitation of the stability control and thus a potential state of instability (see fig. 6.7). Other indicators were tested but do not satisfy the problem's hypotheses. In contrast to L^0, the angular momentum with respect to the FoR L^{FoR} (equivalent to the angular momentum with respect to the CoG) has been proposed as a stability indicator in Goswami et al. [55], but was already dismissed by Buschmann [13] for the robot *Lola*. The performed simulations showed no clear trend (no convexity) as well. Other indicators analyzed were the distance of the *CoP* to the foot's edge, the value of T_{stab} and its minima and maxima. However, none of them show a near convex/concave behavior with respect to $\left(z_{\text{real}} - z_{\text{exp}}\right)$ and are not applicable to the present problem.

Discussion and Model Validation

The results presented here are strongly dependent on *Lola*'s particular walking controller and model parameters. Nevertheless, the presented strategy and indicators can be easily applied to other robots. Of all variables analyzed, several fulfill the considered premise (see section 6.3.1) and can be applied to this problem (namely, $\varphi_{\text{max,min}}$, φ_{con_2}, L^0_{min}, t_{sat}). The

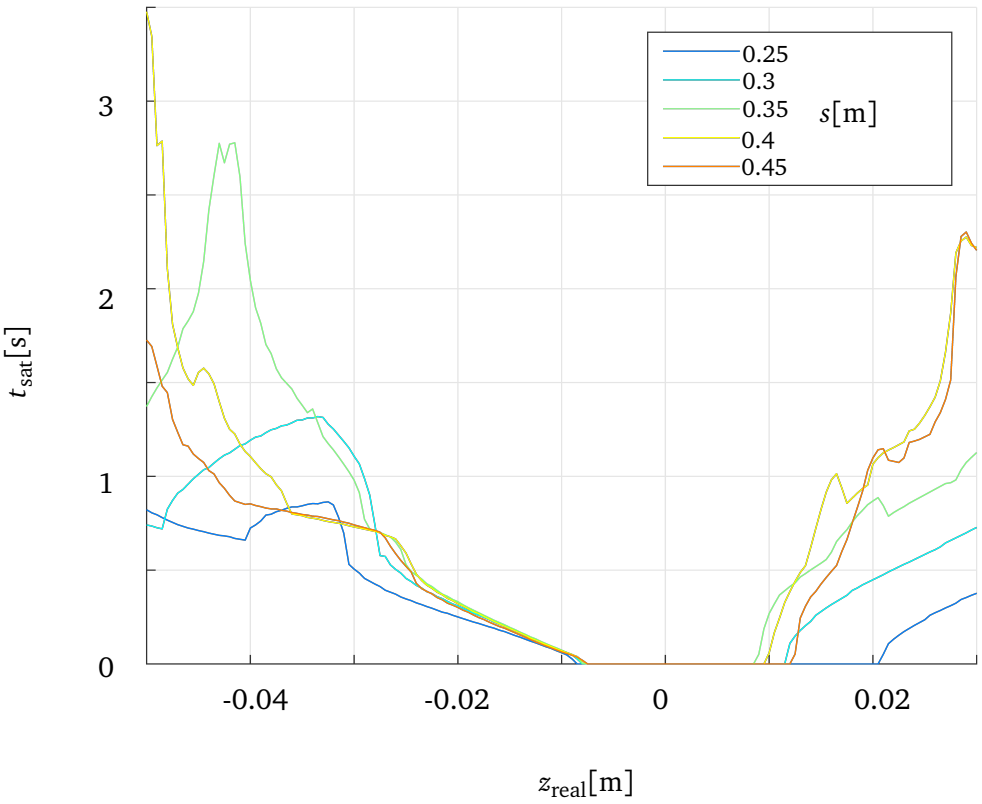


Figure 6.7: Timespan in which T_{stab} is saturated along z_{real} for different simulations.

indicators can be seen in fig. 6.8 for one set of walking parameters. They are all related to stability in the sense that they can indicate states in which the disturbance is too large to be compensated by the stabilization torque. There are a priori no criteria by which some may be better than others for this problem. Those that are less dependent on the walking parameters could potentially simplify the implementation (see section 6.3.5).
As shown in previous work, the dynamics of a robot such as *Lola* can be well described by the presented three-mass model; it allows one to reliably predict future states of instability and reactions to disturbances [201–203]. In order to confirm the validity of the results, full, multi-body simulations were performed for different ground heights. In fig. 6.9, it can be seen that they show a similar trend and are qualitatively equivalent, therefore it does not affect the validity of the following solution.

Application in Walking Controller

As explained in section 6.3.1, the objective of the proposed strategy to mitigate the effect of perception errors is to find the optimal expected ground height $z_{exp,opt}$ according to a stability indicator. Naturally, a weighted combination of several of the presented indicators can be used as an indicator.
In order to obtain the optimal value, let $z_{max} - z_{min} = l$. For the solution, any indicator K defined in $[z_{exp} - l, z_{exp} + l]$ can be used such that a smaller value of K can be associated with a more stable walk (in other words, K satisfies the definition for stability indicator presented in section 6.3.1).
Considering any possible value of z_{exp}, $\tilde{z} = z_{real} - z_{exp}$ is defined. Thus, the indicator value

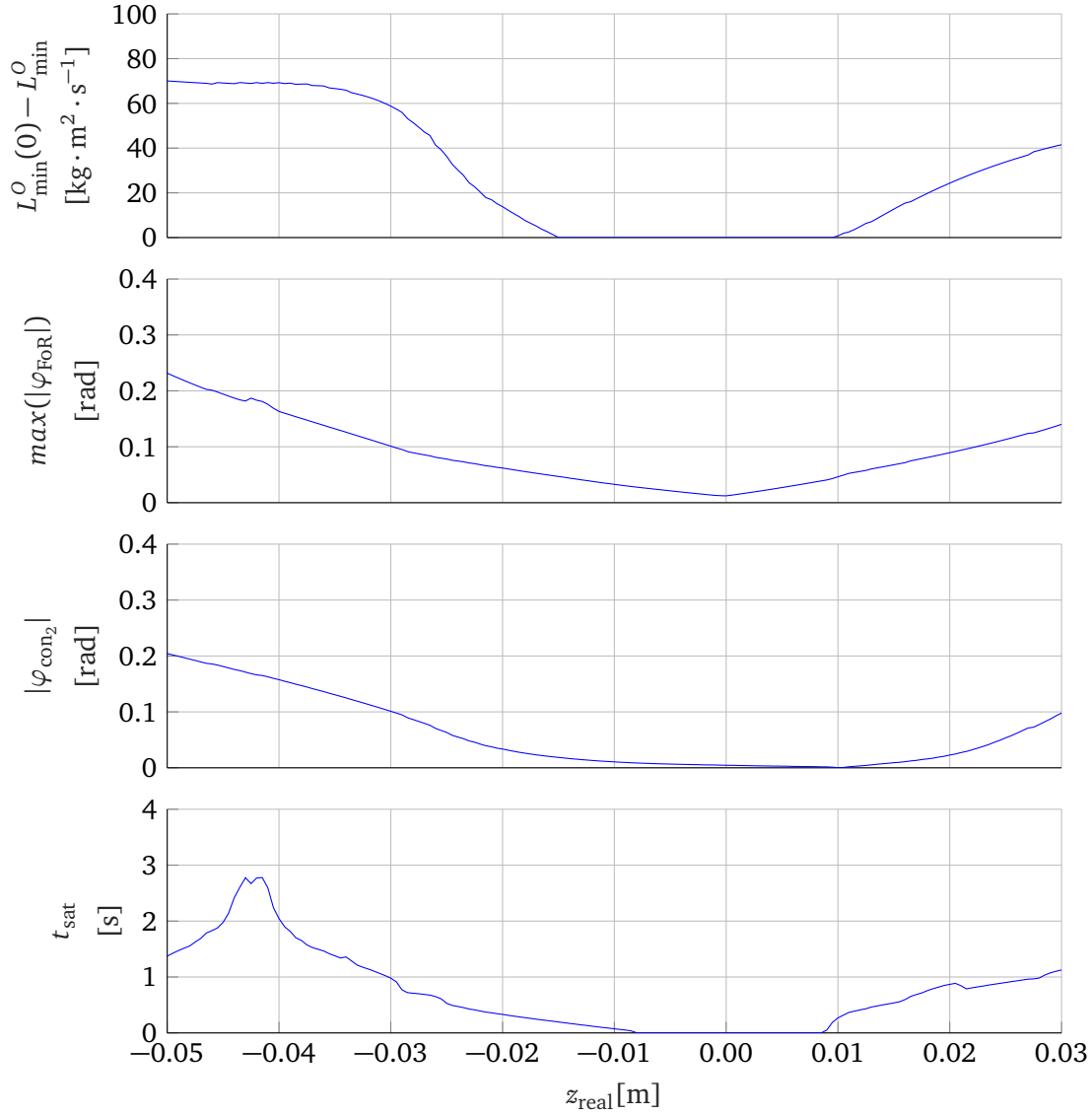


Figure 6.8: A comparison of selected stability indicators for $s = 0.35m$. Any combination of these can be used as indicator K.

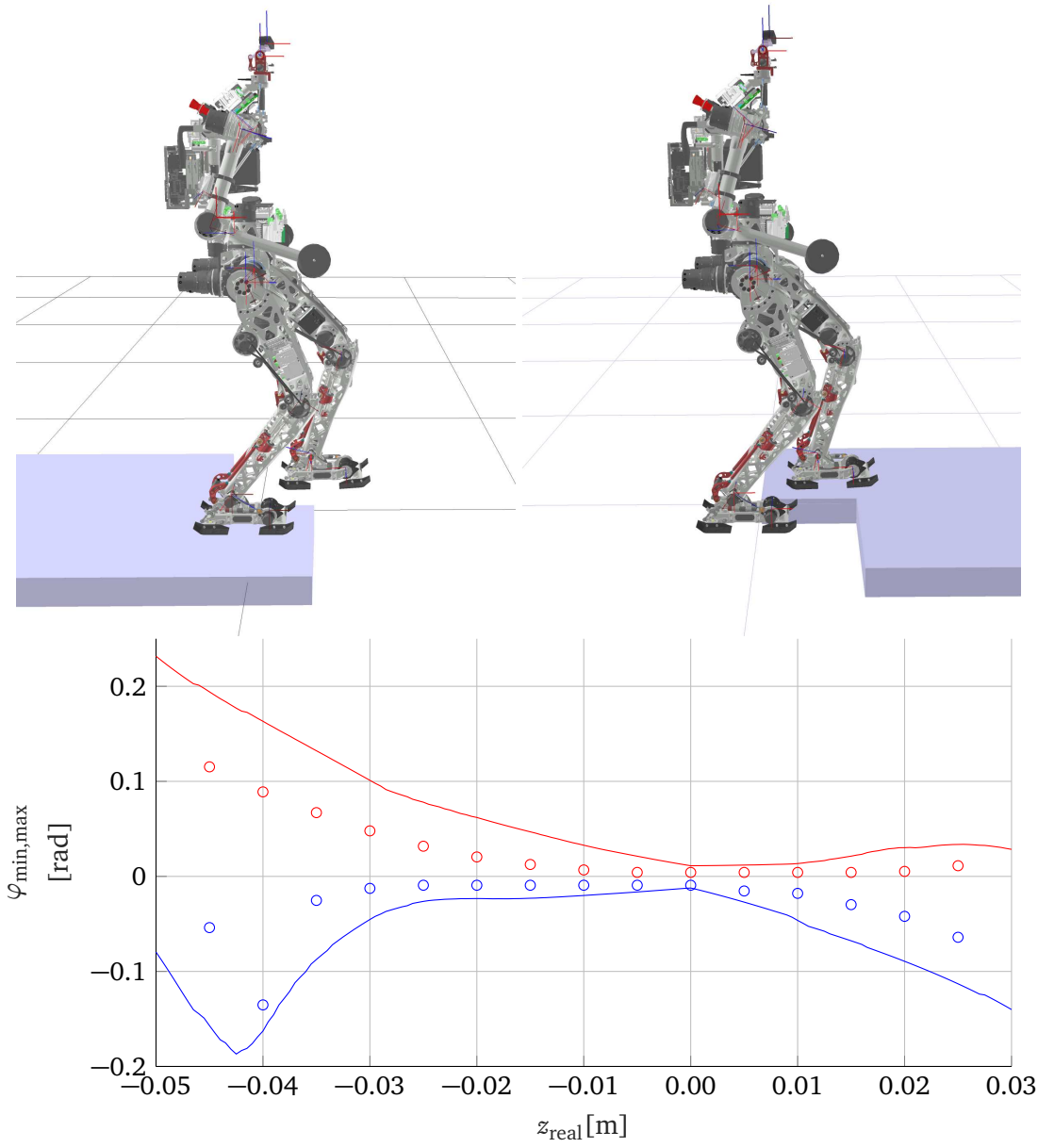


Figure 6.9: The model can be validated with the full multi-body simulation (top). In the bottom, results for $\varphi_{\max}$ (red) and $\varphi_{\max}$ (blue) with $s = 0.35m$ are shown. Lines: simple model. Dots: full multi-body simulation.

can be calculated as:

$$K_{z_{\text{exp}}} = K\left(z_{\text{real}} - z_{\text{exp}}\right) = K\left(\tilde{z}\right) \tag{6.7}$$

Thus, simulations for $z_{\text{exp}} = 0$ can be applied to other values of z_{exp}. The optimal expected value is defined as the one for which the accumulated effect of all possible *EC*, *LC* and *IC* events is minimal:

$$z_{\text{exp,opt}} = \arg\min_{\hat{z}\in[z_{\text{min}},z_{\text{max}}]} \int_{z_{\text{min}}}^{z_{\text{max}}} p\left(z\right) K\left(z - \hat{z}\right) dz \tag{6.8}$$

At this point it is interesting to assume that p is symmetric with respect to $z_{\text{exp,sensor}}$. A symmetric indicator K results in $z_{\text{exp,opt}} = z_{\text{exp,sensor}}$. In that case, the influence of *EC* and *LC* events would be equivalent. It is interesting to observe that solving $z_{\text{exp,opt}}$ for several simulation scenarios with the indicators presented before results in $z_{\text{exp,opt}} > z_{\text{exp,sensor}}$. This result, which is consequent with the asymmetry of fig. 6.8, indicates that EC events have a greater effect on stability than LC events, and z_{exp} should be overestimated. The precise value eq. (6.8) depends both on the sensor's error and robot's dynamics and cannot be computed beforehand.
Thus, the worst *EC* and *LC* events for z_{exp} can be calculated:

$$\begin{aligned} K_{EC,max} &= K\left(z_{\text{max}} - z_{\text{exp}}\right) \\ K_{LC,max} &= K\left(z_{\text{exp}} - z_{\text{min}}\right) \end{aligned} \tag{6.9}$$

The presented solution is very general so that it can be applied regardless of the model used for analysis. Nevertheless, it is computationally expensive. A thorough implementation would involve the following process:

(a) perform *EC*, *LC* and *IC* event simulations for a full robot model in the range $\tilde{z} \in [-l, l]$, and obtain the values for $K\left(\tilde{z}\right)$

(b) with the result from (a), calculate the integral term in (6.8) for the range $z_{\text{exp}} \in [z_{\text{min}}, z_{\text{max}}]$ and find the minimum

that would have to be performed every step, as (a) depends on the sensor's input and walking parameters. Out of both operations, (b) is computationally inexpensive while (a) presents the most difficulty.
In order to implement (a) in the real-time walking controller, an initial solution could be to precompute it offline. Out of a database with detailed simulation results, the values of $K\left(\tilde{z}\right)$ could be interpolated during walking. This method has several drawbacks. One of them is that it makes a change in the stability indicator difficult. Besides, the amount of variables involved would result in a large amount of data (out of simulations it is observed that results vary according to the robot's velocity, step length, swing foot and CoG trajectories). On the other hand, if the discretization is coarse, then the result's precision would decrease. Furthermore, it is not clear if an interpolation could work in this high-dimensional problem.
The author proposes using a reduced robot model (see section 6.3.2) to perform a large number of simulations online and obtain values for $K\left(\tilde{z}\right)$. As shown in previous work [201], simulations of such a reduced model are considerably efficient and can be implemented into the real-time control. Besides, as explained before, such a reduced model can still reliably predict the robot's dynamics. Considering that a small modification of the ground height does not necessarily imply a re-planning of the footstep positions, these simulations may be performed throughout one walking step, which is enough for obtaining acceptable results.

Phase-Switching Strategies and Time-Variable Control

In the previous section it was explained how the landing point for the swing foot can be modified in order to reduce the effect of surface estimation errors, without modifying the walking controller itself. However, this modification may not be sufficient to maintain the robot's stability over uncertain terrain if the errors are large enough. Robots such as *Lola*, which use a ZMP-based control are particularly sensible to unexpected contact situations. As explained in chapter 2, the sequential trajectory planning performed by the *Walking Pattern Generation* module is based on synchronous trajectories of different components with fixed timings. The time duration of each step is predefined and executed completely, regardless of when the actual contact with the ground occurs. Thus the planned and real walking phases can never be perfectly synchronized. Typically, control and modeling errors as well as small perception errors in laboratory conditions can be compensated using feedback control [24, 47, 59, 137]. Nevertheless, perception errors become larger when walking faster or over other kinds of non-rigid, non-flat terrain, such as grass or stones (as explained before). In order to overcome real-world scenarios, some authors presented methods for quick trajectory regeneration [133, 177] that are based on balance compensation without changing the step duration. Nishiwaki et al. [136] proposed different strategies for online modification of ZMP trajectories that allowed their HRP-2 robot to walk over carpet tiles. Other authors treat these errors as disturbances to be compensated afterwards by modifying future footstep locations [201–203].

In contrast, in this section a flexible walking control that is intrinsically more robust against irregular ground is proposed. Instead of fixing the phase durations beforehand, the step duration is variable and depends on sensed contact with the ground, thus making sure that the planned and walking phases are always synchronized. Previously, Buschmann et al. [18] proposed a strategy to adapt the walking phase of *Lola* to a detected early contact. It was able to walk over unexpected obstacles on the ground. Here it is combined with a time variable phase that directly deals with late contacts with the ground from the motion planning stage (in contrast to using feedback control). The resulting control adapts the walking phase and motion to a direct ground contact detection, improving the robustness of *Lola*'s walking over irregular terrain.

Other authors have also proposed phase switching mechanisms for walking control. A review of the biologically inspired *central pattern generators* is given in Buschmann et al. [18]. These have not (as far as the author knows) yet been successfully applied to full-sized humanoid robots. Among others, Pratt et al. [149] and Sreenath et al. [171] propose an interesting control strategy that directly depends on the position of the upper body instead of time and demonstrate their application in planar robots. Morisawa et al. [125] also use contact with the ground as a phase switch mechanism, but they expect it inside a fixed time window. More recently, Hubicki et al. [78] presented a robot capable of walking over irregular terrain. It shows impressive results but uses a completely different strategy for walking control based only on controling the contact forces of the legs against the ground. Instead, the method presented here can be easily implemented in other ZMP-based systems. It allows making the controller more robust against irregular terrain by eliminating the fixed timing restriction of phase switching, using direct ground contact detection instead.

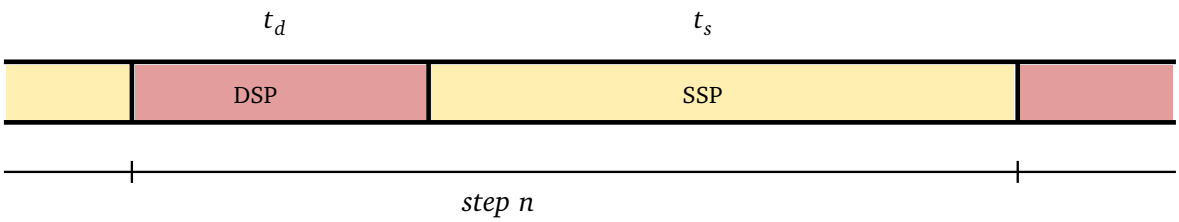


Figure 6.10: Main walking phases during ideal walking. During one step, the robot transitions between a double support (DSP, duration t_d) and a single support (SSP, duration t_s) contact state. Step duration is as planned.

Walking Control Concept

As explained in chapter 2, a state machine is in charge of defining the walking state, which defines the dynamics of the system. The different walking states depend first on the intended action (e.g. start/stop walking) and the planned contact state (e.g. single/double support) and are synchronized with pre-defined timings. These determine the phase of the walking controller and thus the control strategy according to the assumed contact state. For the purpose of this chapter, focus is put on the two main phases of periodical walking: single (SSP) and double support (DSP), see fig. 6.10.
The *Walking Pattern Generation* module of *Lola*'s control system generates the following trajectories sequentially: *Swing foot*, *CoP*, *CoG* and *Load Distribution* (see sections 2.3 and 2.4). In *Lola*'s original system [13], as well as in most humanoid robot controllers, the timings for these phase transitions are fixed (see fig. 6.10) independently of the exact time the foot touches the ground. In this work two event-based transition phases that react to the sensed ground contact are added, adapting the *Walking Pattern Generation* and making the step duration effectively time-variable. Changing the step duration results in a displaced CoG: if the step duration is shortened or extended, the CoG will be behind or ahead of its planned position with respect to the feet, respectively. In order to compensate for this effect, the next step is adapted accordingly, as explained in the following:
Early Contact Response. Previously, Buschmann et al. [18] introduced an additional *impact* phase that is activated when a contact with the ground is detected during the SSP. As seen in fig. 6.11, the state machine switches to the impact phase through this *early contact* (EC) event, thus shortening the step's duration:

$$t_s^* = t_s - t_i \tag{6.10}$$

t_s and t_s^* are the planned and modified duration of the SSP respectively and t_i the time during which the impact phase is active. After the EC event, the swing foot is immediately stopped, using the "stop trajectory" presented in Ewald et al. [46]. During the impact phase, the Load Distribution re-computed for the longer double support state (consisting of the impact phase and the DSP) and the CoP and CoG trajectories are unmodified. For more details, see Buschmann et al. [18].

Late Contact Response. For this work the *impact* phase is complemented with a new *glide* phase that specifically targets a *late contact* (LC) scenario. This phase deals with an unpredicted, extended single support state of the robot until a contact with the ground is detected, thus extending the step's duration as seen in fig. 6.12. As explained before, the following DSP is shortened to prevent a displaced CoG motion:

$$t_d^* = \max\left(t_d - t_g, t_{min}\right) \tag{6.11}$$

where the duration of the glide phase t_g is the time from the end of the SSP until the contact is detected, t_d and t_d^* are the planned and modified duration of the following DSP, respectively,

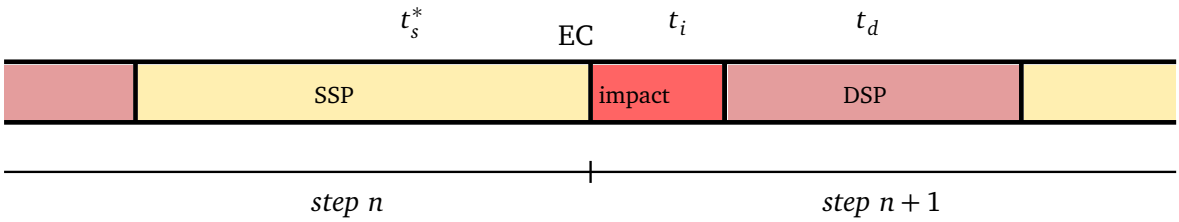


Figure 6.11: Early contact response. The step is interrupted and the state machine switches to the impact phase (where the robot is in a double support state) for the rest of the planned step duration.

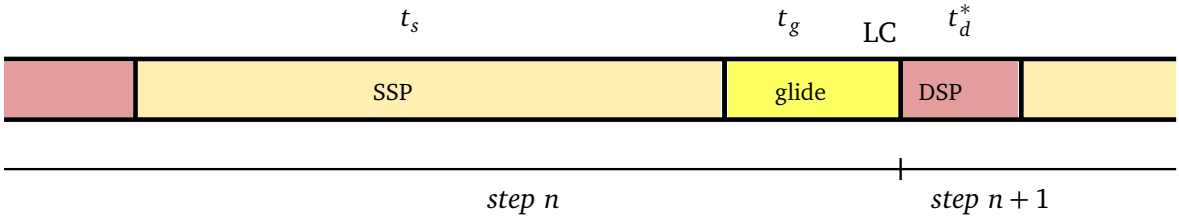


Figure 6.12: Late contact response. The step is extended by switching to a glide phase, until contact with the ground is detected and the DSP can start.

and t_{min} is a predefined minimum possible duration of the DSP (in this case one control cycle, or 1 ms). Note that if $\left(t_d - t_g\right)$ is smaller than t_{min}, the next step is effectively displaced forward in time and not only shortened by t_g (see fig. 6.12). The *Walking Pattern Generation* process during the glide phase is explained in detail in the next section.

Time-Variable Phase for Late Contact

In an LC scenario, the predefined phase-switch timing results in the planned CoP making the transition between both feet before the swing foot is in contact with the ground. The robot can tilt over when the CoP reaches the limit of the stance foot (when tilting over, the robot is not able to exert momentum directly to the ground and has a very limited ability to stabilize itself). Additionally, the inclined state of the robot may cause an initial contact with the toe, causing further destabilization [13]. The main strategy in this scenario becomes keeping the robot from tilting over and achieving a more stable contact situation between the swing foot and the ground. This translates into keeping the CoP inside the stance foot and maintaining the upper body in a vertical orientation, thus obtaining less strained walking. In the following, the behavior of the time-variable glide phase throughout all stages of the *Walking Pattern Generation* is explained. These stages are the CoP, swing foot and CoG trajectory generation as well as force control and a necessary footstep re-planning. For the sake of simplicity, the analysis is limited to the sagittal plane (fig. 6.13), as it is equivalent for the frontal plane.

CoP trajectory generation. Similar to the normal DSP, during the glide phase the CoP is shifted until the boundary (with a safety margin) of the stance foot (in the forward or x direction):

$$x_P(t) = \min\left(h_g t + c_g, x_B\right) \tag{6.12}$$

for $t \in glide$, where x_P is the location of the CoP (the y component stays constant). h_g and c_g are chosen to satisfy continuity in position and velocity with the trajectory of x_P during the SSP and x_B is defined as a safety margin from the boundary. Instead of moving linearly between both feet, the CoP is stopped until contact with the ground is detected, in order to

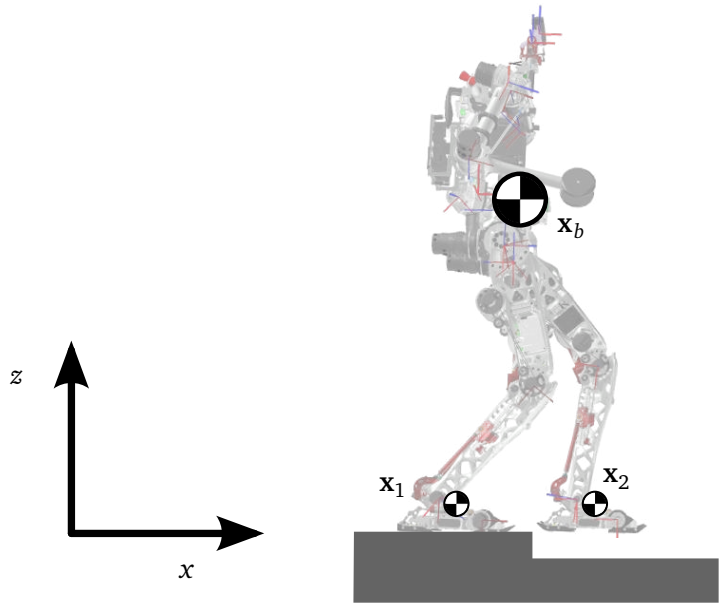


Figure 6.13: Robot and three mass model at the end of the SSP in an LC scenario. Sagittal plane.

keep the foot against the ground and maintain the ability to exert torque against the ground. When the DSP finally starts (after contact is detected), the CoP motion is adapted to the new phase duration and continued:

$$x_P(t) = h_d t + c_d \tag{6.13}$$

for $t \in DSP$. h_g and c_g are chosen such that the location of the swing foot is reached in t_d^*. The y component is analogous.

Swing foot vertical trajectory and load distribution. By the end of the SSP, the ground is expected and the swing foot's vertical motion is stopped (see chapter 2). As there is no information on where the ground may be, planning a vertical motion to reach the ground becomes difficult. However, if nothing else is done, the foot stays in the air and does not reach the ground. While no motion is planned directly, the foot is affected by the modifications imposed by the force control (which can react to a contact with the ground more quickly). The load distribution is then shifted between both feet, just as it would during the DSP (see chapter 2).

CoG trajectory generation. By the end of the SSP, the CoG is located ahead of the stance foot. In order to keep the CoP from leaving the support polygon, the CoG position is integrated further in eq. (2.6). The inputs of the equation ($\mathbf{x}_1$ and $\mathbf{x}_2$) correspond to the initially planned trajectories of the CoP (eqs. (6.12) and (6.13)) and swing foot (constant). The final trajectories for both the CoP and CoG are obtained from the approximate solution of eq. (2.6) by spline collocation, as explained in Buschmann et al. [14]. This results in an accelerated CoG trajectory in the horizontal direction (intuitively, it can be interpreted as the necessary moment to keep the CoP away from the boundary), increasing the CoG position and velocity along the forward direction while the stance foot stays fixed to the ground. If the swing foot's position is not modified, this increased CoG position and velocity can quickly result in the robot falling forward on the next step. In order to compensate for this displacement, a horizontal trajectory is introduced for the swing foot.

Swing foot horizontal trajectory. At this point it is important to recall section 2.2 and note that a swing foot horizontal motion can easily be taken into account by the robot's EoM while generating the CoG trajectory. In this case, the linear inverted pendulum model (LIPM) is considered (see section 2.2). Kajita et al. [84] define the pendulum's orbital energy as the

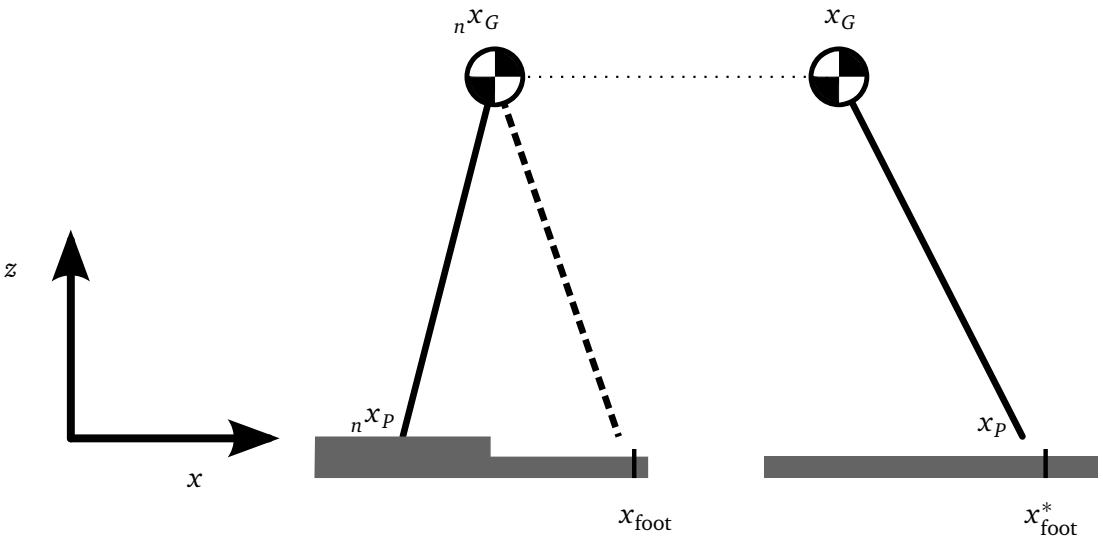


Figure 6.14: The swing foot horizontal trajectory is generated by analyzing the orbital energy of the linear inverted pendulum model. Left: reference energy at the end of the SSP. Right: an accelerated CoG requires a displaced pivot point to slow it down.

sum of the kinetic energy and an imaginary potential energy around the pivot point x_P:

$$E = \frac{1}{2}\left(\dot{x}_G^2 - \frac{g}{z_G}(x_G - x_P)^2\right) \tag{6.14}$$

where g refers to the gravity acceleration, $(x_G, z_G)^\mathsf{T}$ is the location of the CoG and x_P the location of the CoP. To compensate for the CoG velocity, a foot displacement is calculated, such that the pendulum's orbital energy is maintained (using the one at the end of the SSP as reference):

$$\dot{x}_G^2 - \frac{g}{z_G}(x_P - x_G)^2 = {}_n\dot{x}_G{}^2 - \frac{g}{z_G}\left({}_nx_G - {}_nx_P\right)^2 \tag{6.15}$$

where ${}_nx$ refers to the value of x_G at the end of the SSP of step n. The right side of eq. (6.15) is stored at the beginning of the glide phase and x_P is calculated every cycle for the actual values of x_G and $\dot{x}_G$ (see section 2.2). Note that for the new energy value, the pendulum with the swing foot is considered instead of the stance foot as shown in fig. 6.14. After obtaining x_P from eq. (6.15), the new modified swing foot position is computed:

$$x^*_{\text{foot}} = x_{\text{foot}} + \left(x_G - {}_nx_G\right) + k\left((x_P - x_G) - \left({}_nx_G - {}_nx_P\right)\right) \tag{6.16}$$

where x_{foot} and x^*_{foot} are the initial and modified planned footstep location respectively and the second term accounts for the CoG displacement (see fig. 6.14). The third "energy term" consists of the variation of the CoG-CoP distance with respect to the one at the end of the SSP, in order to obtain a continuous trajectory. A heuristic factor k is introduced to compensate for model errors, such as the unaccounted-for swing-foot motion ($k = 0.75$ in this implementation). Taking the time derivative of eq. (6.16) and eq. (6.15), the horizontal velocity becomes:

$$\dot{x}^*_{\text{foot}} = \dot{x}_G + \frac{kz}{g}\frac{\dot{x}_G\ddot{x}_G}{x_P - x_G} \tag{6.17}$$

and both position and velocity can be sent to the *Feedback Control* module.

Footstep re-planning Even though the swing foot horizontal motion helps to reduce the

effect of an accelerated CoG, its high speed can still have a destabilizing effect on further steps. If the CoG velocity increases beyond a certain margin, the next footstep location is also modified in order to slow the robot down to the planned walking speed. In the present implementation, the footstep re-planning is activated when $t_g > 0.2s$.
Again the orbital energy criterion eq. (6.15) is used and the energy at the end of the SSP is compared for both steps:

$$ {}_{n+1}\dot{x}_G{}^2 - \frac{g}{z_G}\left({}_{n+1}x_P - {}_{n+1}x_G\right)^2 = {}_{n}\dot{x}_G{}^2 - \frac{g}{z_G}\left({}_{n}x_G - {}_{n}x_P\right)^2 \tag{6.18} $$

where the planned values of ${}_{n+1}x_G$ and ${}_{n+1}\dot{x}_G$ are taken. Again eq. (6.18) can be solved for ${}_{n+1}x_P$ and the new footstep location is obtained with eq. (6.16).

Simulation Results

In order to obtain comparable results, the presented control strategy is validated with the multi-body simulation. A video of the simulations can be found at https://youtu.be/FPpyDLKVlCY.
The first considered scenario is an unexpected LC. The robot starts walking on a platform which abruptly ends after a few steps. For different heights, the following simulations are considered:

1. the normal control system [13],
2. only the impact phase active [18] and
3. both impact and glide phases active (this work).

For all the tested scenarios, the impact phase shows a consistent improvement with respect to the normal control (as was already shown in Buschmann et al. [18]). The reason behind this is that due to the LC the robot tilts, resulting in an EC for the next step where the impact phase becomes relevant. Adding the glide phase leads to better results than both previous cases, as the effect of the initial LC is reduced before the next step is reached.
In fig. 6.15, a screen shot of the normal and "impact+glide" cases for the same platform height is shown. It can be observed how the glide phase shifts the swing foot away from the planned position and the stance foot stays in full contact with the ground instead of tilting over the toe. In terms of robustness, the robot can also consistently overcome a higher change in the platform height without tilting over. In order to test this, simulations were performed for each control mode by sequentially increasing the platform height by 0.1 cm (starting at ground level) until the robot falls down or a joint limit is reached. The maximum height difference which it can safely overcome can be seen in table 6.2.

Table 6.2: Robustness

Control Strategy	Maximum Platform Height
normal	3.4cm
impact	7.7cm
impact+glide	8.7cm

From the stability indicators considered previously, the upper body orientation is taken to compare the different control schemes. In fig. 6.16 the upper body orientation is plotted for

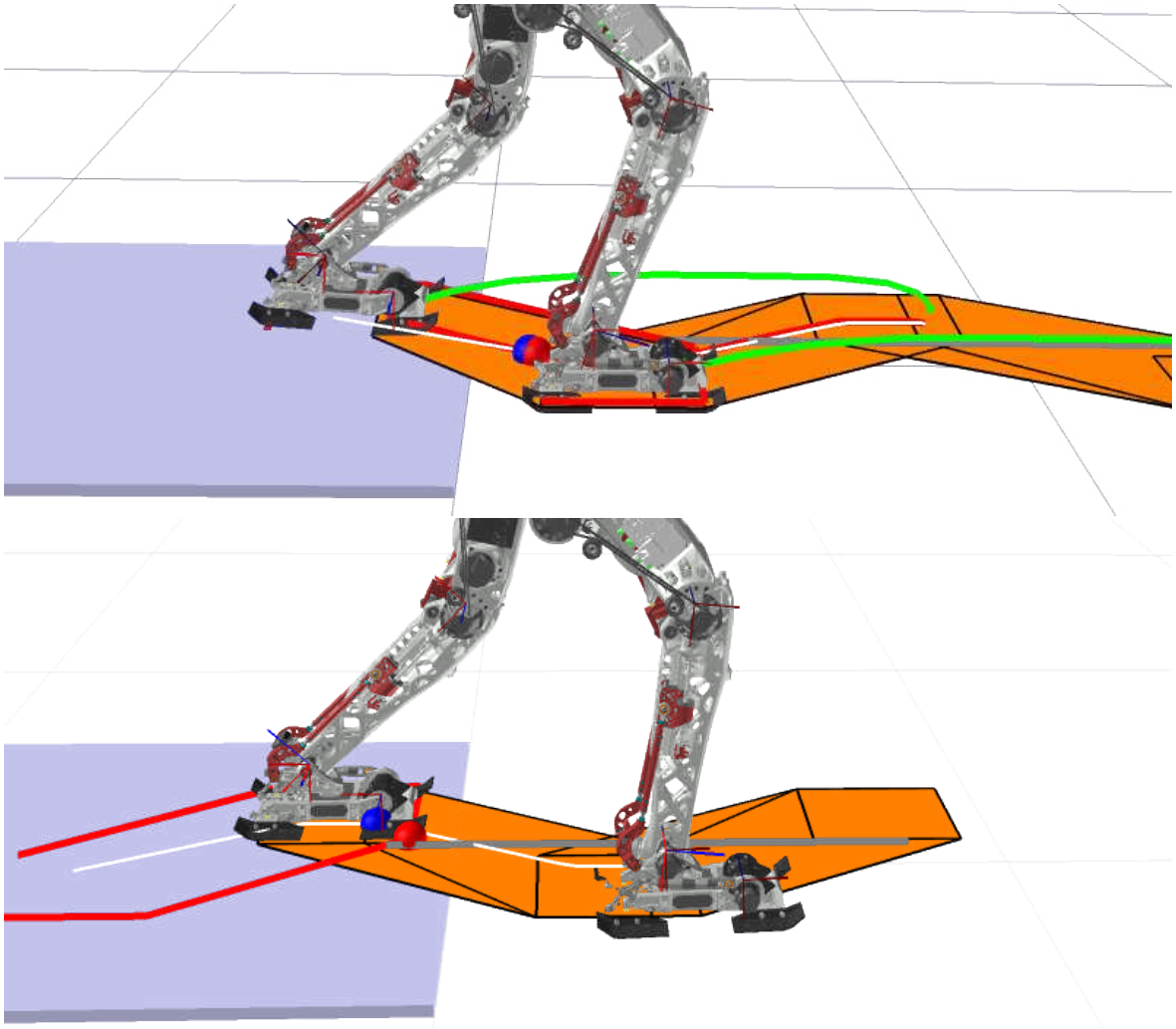


Figure 6.15: Simulation of an LC scenario (3 cm height in this case), before the swing foot touches the ground. The planned footstep positions, along with the support polygons (orange), are projected on the expected ground. Top: normal control. The robot starts tilting over the toe and the support polygon is reduced, while the CoP (red) shifts forward. Bottom: impact and glide phases active. The glide phase keeps the stance foot against the ground, the CoP (red) inside the actual support polygon and accelerates the swing foot and CoG away from their originally planned trajectories.

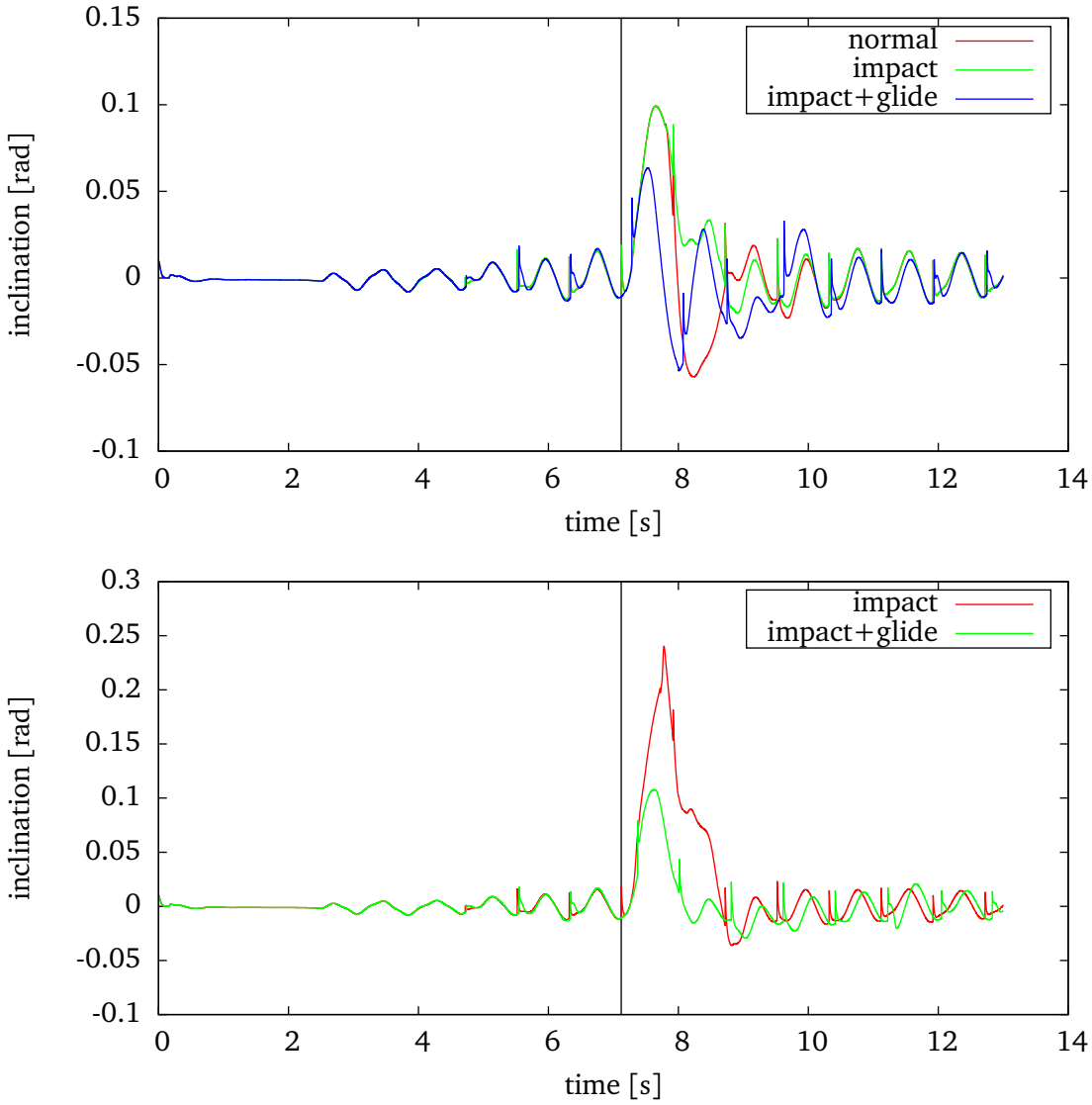


Figure 6.16: Upper body inclination (around the y axis) for a 3 cm (top) and a 7 cm (bottom) LC scenario with different control strategies. The vertical line indicates the end of the SSP.

two exemplary height values to compare the different cases. The normal control results in the robot tilting over for all height values over 3.4 cm (see table 6.2). Therefore, its result is only plotted for the smaller height value, where the normal control and impact phase show consistent behavior during the LC up until the next contact with the ground ($t \simeq 8$s). Afterwards, the impact phase prevents the robot from tilting backwards and maintains a smaller value for the upper body orientation henceforward. In contrast to both previous cases, the glide phase already maintains a low value of the upper body orientation during the LC, preventing high values which could lead to the robot tilting over (this effect can be better appreciated in the bottom plot). The glide phase results in a less pronounced EC by the next step and a more stable walk than both previous cases.

As a final scenario an unexpected obstacle after the platform is considered, to also analyze how the glide phase performs in an EC situation. A screen shot of the experiment is shown in fig. 6.17 and the resulting upper body orientation can be seen in fig. 6.18. Similarly to fig. 6.16, the introduction of the glide phase maintains the upper body orientation during the LC. This time it becomes even more relevant as it allows the robot to return to a stable walk

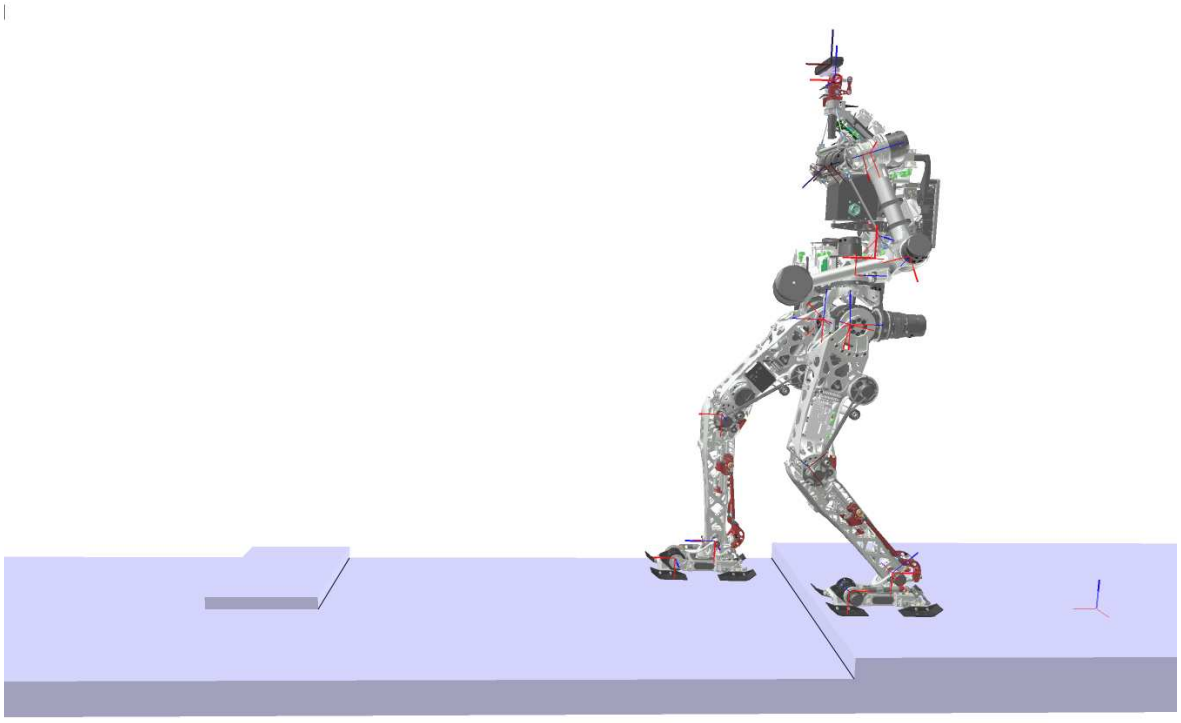


Figure 6.17: Simulation of complex scenario. A few steps after a 6 cm late contact there is an unexpected 4 cm obstacle (the obstacle location is adapted so that it is directly met by the swing foot in both control cases).

more quickly, thus being able to overcome the latter obstacle. Without the glide phase, the robot does not recover in time and tilts over after encountering the obstacle.

Discussion

In this chapter, several modifications to ZMP-based walking controllers were presented to make biped robots more robust against irregular terrain in general, and perception errors in particular. First, several stability indicators were proposed and analyzed using a reduced robot model. Using this same model and the presented indicators, an algorithm to calculate the optimal value for the assumed ground height in the presence of bounded uncertainty was suggested. This can potentially improve the robustness of the robot against perception errors, without making any changes to the walking control itself. An accuracy factor describing irregular surfaces (e.g. gravel, grass) can also be taken into account. According to simulation results, an overestimation of the ground height leads to a more stable walk in most cases.
Additionally, a walking controller for ZMP-based systems that is inherently robust against unexpected irregular terrain was presented. It complements previous work [18] with an extra time-variable phase that specifically deals with late contact scenarios. The result is an event-based walking controller with a variable step duration that depends directly on the detected ground contact. It improves the robustness and stability of biped robots against irregular terrain and perception errors.
There are, however, a few limitations in the presented strategies that leave the field open to further work and discussion.
For the evaluation of the stability indicators as well as the solution to an optimal value for the surface height value a simplified three-mass model was used. This model was only con-

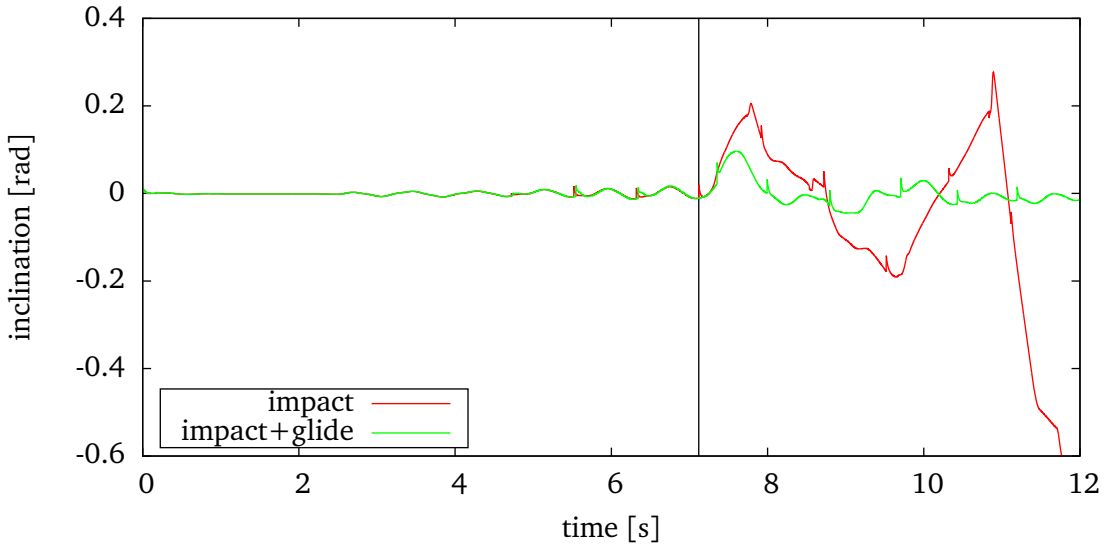


Figure 6.18: Upper body inclination (around the y axis) with different control strategies for a combined scenario: a 6 cm LC followed by a 4 cm EC. The vertical line indicates the end of the SSP before the LC. Without the glide phase, the disturbance becomes so large that the robot tilts over after the EC.

sidered in the sagittal plane. From the simulations performed it is considered that this choice offers at first more interesting results than the frontal plane and a valid starting point for analysis. Nevertheless, by considering the frontal plane as well, the robot's dynamics could be represented more accurately. A simple way of doing this would be to compute the chosen indicator using a combination of the values for both planes, though it would result in twice as much processing time. Another point to consider is the simplicity of the model itself. For the considered application and available hardware, it represents a good compromise between an accurate simulation and low processing time. However, this could change depending on the robotic system and available processing power. Besides the three point masses, the feet point contacts also represent a strong simplification. More complex feet models (including e.g. more contact points or horizontal friction) would result in more accurate results and additional considerations (such as the effect of initial toe- or heel-contact). Moreover, the present analysis was performed on a simulated horizontal, planar ground with a sudden increase in height. By extending the simulation to include more complex scenarios, more detailed results could be obtained.

More importantly, there is considerable potential applications of the real-time analysis of stability indicators. This analysis was motivated by the errors in the results of the vision system and is only used for the optimization of those results in this work (as well as to validate control strategies). However, these tools can also be applied to directly modify the walking parameters in real-time. For example, by simulating the behavior of the robot at different speeds, its velocity could be adapted to the present case. Initial results show that the robot shows a more stable behavior against vision errors at lower speeds, so that its velocity could be reduced when walking over areas with less trustworthy information (still, these results vary when the robot is walking up or down stairs so no general conclusion can be obtained yet). Other walking parameters such as step length, CoP or swing foot trajectory could be similarly modified. For example, by modifying the CoP trajectory to stay closer to the foot's center during the SSP, the robot would be less prone to falling down when walking over irregular terrain (however, the modified walking pattern would have to be analyzed as well as it might have a negative effect).

Regarding the walking controller, future work includes the testing on the actual robot in more

complicated scenarios. For some particular cases the swing foot trajectory may be counterproductive (in the case of unexpected narrow holes, for example, the swing foot would laterally collide with the side of the hole). One possible solution would be the specific monitoring of horizontal contact (using the force/torque sensors) to better adapt the footstep trajectory [174]. Additionally, a more reactive and comprehensive force control strategy could yield enormous benefits [175]. Another important point to consider is that these and further modifications to walking controllers result in increasingly more complicated systems. An interesting research topic would be the development of parameterized walking controllers that are intrinsically time-variable and can be more simply implemented.

Chapter 7

Robust and Flexible Walking

Overview

As explained in chapter 2, the contributions of this thesis were developed in parallel to other Ph.D. candidates with the common objective of achieving flexible and robust walking over unknown terrain. With this in mind, an extended framework for biped control (fig. 2.5) is proposed with special modules for perception, time-varying walking control, intelligent navigation and stabilizing trajectory adaptation. The first two modules correspond to the main contributions of this thesis and are presented in chapter 5 and chapter 6, respectively. This chapter presents the final developed framework in detail, focusing on the interaction between its different components. The last two modules are briefly described; more details can be found in the corresponding theses [66, 200]. The author's own contributions in this chapter comprise the final configuration of the robotic system (explained in the following), part of the communication interfaces between the separate modules and, more importantly, two different solutions for augmented reality systems that are presented at the end of this chapter.
In order to better understand the relationship between the different modules of the extended framework (fig. 2.5), the final configuration of the robotic system is shown in fig. 7.1. The walking control system runs on a real-time QNX computer and the perception system (chapter 5) runs on a separate linux computer (see section 1.3). Both computers are mounted on the robot. The Lola Environment Perception Package (LEPP) sends the environment approximation results to the control computer (Control) while receiving odometry information from it (State Server). Additionally, two other computers are included in the network for augmented reality. They both receive the environment approximation results from LEPP, the actual and planned footstep positions from Control and the odometry information from State Server.

Extended Hierarchical Control

The final walking control system is shown in fig. 7.2; the perception (chapter 5) and time-varying control (chapter 6) modules are left out for clarity. It follows a hierarchical approach and is divided into a *global control* and a *local control* module[1]. Both receive the environment approximation of the perception system (chapter 5) to plan and execute collision-free trajectories. The global control module calculates an *ideal walking pattern* over a time horizon of multiple robot steps. It gets desired step parameters, such as step length, desired goal positions or a desired velocity vector as an input from the user (chapter 2). Thanks to a cycle

[1] In this case, *global control* refers to the long-term planning of future footstep positions and global trajectories while *local control* refers to the short-term trajectory adaptation and control.

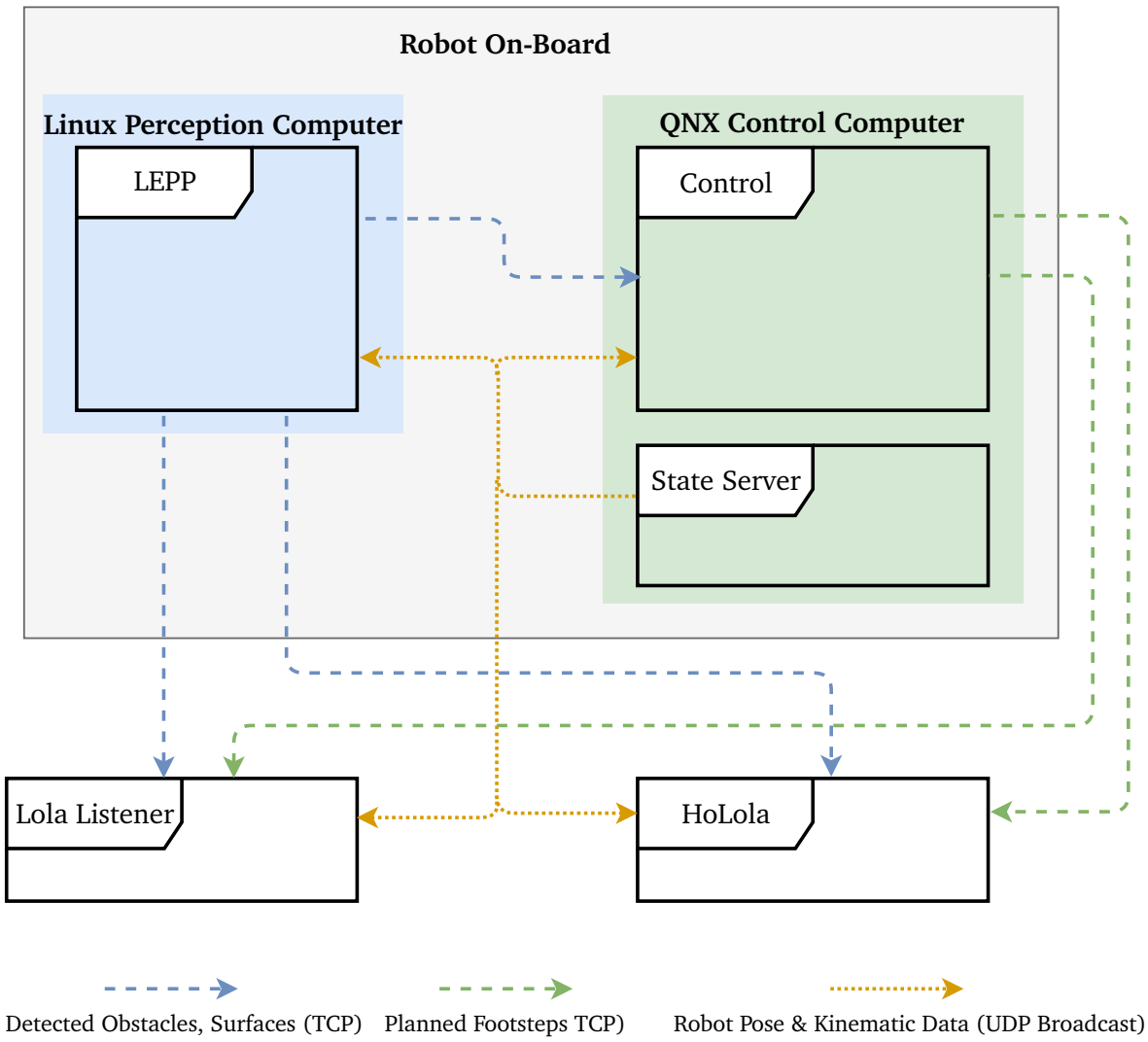


Figure 7.1: Final setup for autonomous walking. The walking controller and perception system run on two parallel computers on-board. Additionally, two external systems for augmented reality receive information from them online.

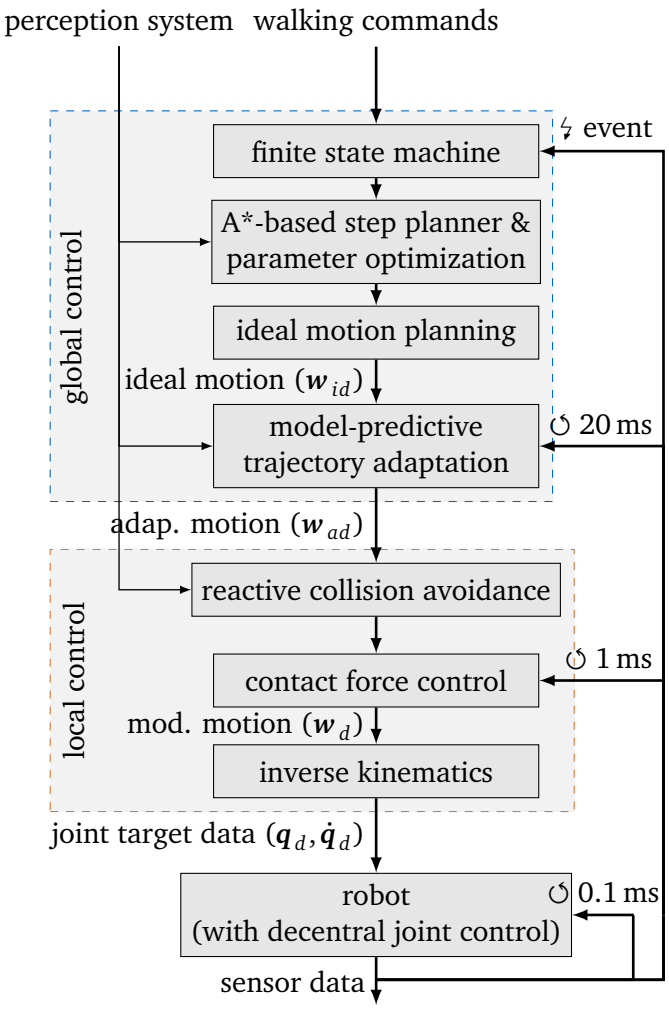


Figure 7.2: *Lola*'s real-time walking control system.

time of one walking step (T_{step}), it is very responsive to changes in the environment or in user input. The trajectory adaptation submodule monitors the state of the robot and modifies footstep positions in real time to compensate for model inaccuracies or external disturbances and stabilize the robot. These modified trajectories are the input to the *local control* submodule. It is called in a cycle time of 1 ms and adapts the ideal planed trajectories locally according to sensor feedback and taking the full approximation of the robot and the environment for collision avoidance into account. Each of these processes is explained in the following.

Motion Planning

The sub-system for navigation and motion planning was developed by Arne-Christoph Hildebrandt. Here, this module is shortly explained, focusing on its integration with the main framework. For more details, see Hildebrandt [66].
Based on the environment representation and the user's input, the *step planner and parameter optimization* submodule calculates a sequence of parameter sets describing the walking pattern based on an A^* search implementation [68]. Using the output of the A^* search as an initial solution to the motion planning problem, it optimizes the parameter set and calculates kinematically feasible, collision-free and dynamically executable trajectories taking the envi-

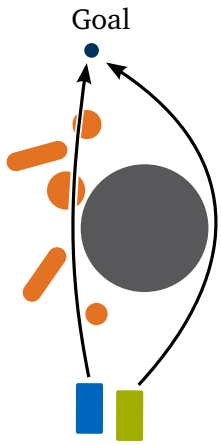


Figure 7.3: A 2D map for goal-oriented navigation. Before planning a discrete set of footsteps, the global walking direction is determined by avoiding large obstacles (gray). A set of continues paths is evaluated against the rest of the environment (consisting of surfaces and smaller obstacles, in orange) and one is chosen to guide the A* search.

ronment representation and the full robot approximation into account [65].
The objective of the navigation module follows ideas from other authors [16, 137]. Its purpose is not to find long distance paths but to give the user a 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 (see fig. 5.3). According to the application, the user has the possibility to guide the robot with a joystick, give it desired walking parameters or set intermediate goal positions which the robot should reach. The robot then executes these high-level commands while taking care of navigating a safe and optimal path. In the following, the different steps in this process are explained.

2D path and A* search

Usual search algorithms for discrete planning require a great amount of computational power and can only be used to plan a few footsteps in advance [68]. However, it can be useful to consider the complete trajectory towards a predefined goal to take further areas of the environment into account. In order to achieve this objective in real-time, a reduced 2D map of the environment containing only large obstacles is created (see fig. 7.3). In this case, large obstacles are defined as those too high for the robot to step over. A set of continuous splines avoiding large obstacles are graded according to their length and the presence of smaller obstacles or surfaces which may hinder the robot's navigation (grading is performed taking the complete environment representation into account), and the easiest (or shortest) path is chosen for global navigation [70].
Based on the walking direction and a discrete set of footsteps an implicit A*-search is applied to solve the planning problem and to find a sequence of n_{Steps} steps. The robot's state s, representing the nodes of the used A*-search, is described by the current stance foot $stance = (left, right)$, the global position $\mathbf{r}_f = (x_f, y_f, z_f)$ and orientation $\theta_f = (\theta_{x_f}, \theta_{y_f}, \theta_{z_f})$ of the stance foot. It follows $s = (\mathbf{r}_f, \theta_f, stance)$.
Since the possible footstep locations are symmetric for the left and the right stance foot, one action model is defined as $a = (\Delta x_f, \Delta y_f, \Delta\theta_{z_f}, h_{obst}, h_{Step}, c_a)$. The variables Δx_f, Δy_f and $\Delta\theta_{z_f}$ represent the possible displacements and rotations relative to the current stance foot.

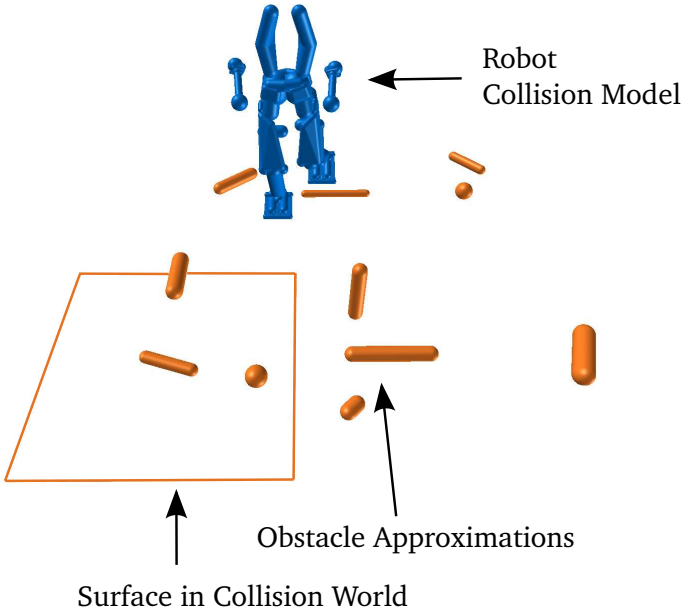


Figure 7.4: Collision model of the step planner for *Lola* stepping over an obstacle with movable leg segments. Both the robot and obstacles are modeled with SSVs. Surfaces are represented as polygons with edges modeled as SSVs to avoid stepping on them using the collision avoidance framework.

Δz_f, $\Delta\theta_{x_f}$ and $\Delta\theta_{y_f}$ are not taken into account as part of the action model since they are directly determined by the current position (x_f, y_f) of the robot in the represented world. The action model is augmented by h_{obst}, which takes into account obstacles the robot has to step over to reach the next s and by h_{step}, which denotes the height change the action can make. The cost for each action is denoted by c_a. Further details are explained in Hildebrandt et al. [68].

Collision Checking

The main difference to other A*-search based footstep planners for biped locomotion [75, 137] is the environment representation (chapter 4) and its consistent consideration in all modules of trajectory generation – from footstep planning to reactive collision avoidance [67]. Instead of a grid-based environment representation which is able to check collisions in a binary way for a 2.5D map, the environment representation presented in this thesis allows to check for collisions in full 3D. In the footstep planner, collisions are checked not by a planar rectangle but by a 3D model (based on SSVs) of the lower leg including the foot and a leg segment approximation. This gives a better representation of the full robot movement, especially for large strides, and allows for reduced safety margins.

3D Walking

In addition to the viability of a state s and the corresponding action model a, the step planner has to evaluate the 6D pose of the foothold. As introduced above, the rotation $\theta_{y_f}, \theta_{z_f}$ and the height z_f are a direct function of the environment, x_f, y_f and θ_{z_f}. It is classified into regions the robot has to avoid (obstacles) and areas the robot can step on (surfaces), as can be seen in fig. 7.4. As explained in section 5.4, surfaces are represented by convex hulls (polygons) and a normal to the surface (see fig. 4.14). In order to prevent the robot from stepping

onto the edges of the surfaces, these are modeled as obstacles using line-SSVs. Thus, the complete foot is in contact with the surface, which helps to maintain the robot's stability and prevents additional modifications to the walking controller. This representation has several advantages:

1. Based on the current x_f and y_f value of s, the step planner is able to determine the whole 6D pose of the foot just by checking in which polygon the current s is lying, which is computationally efficient.

2. Surfaces are completely defined by the corner points and the normal of the surface. This is an extremely dense, memory-efficient representation that simplifies communication between planning modules and the perception system. In the current implementation, a maximum of eight corner points are used. Depending on the desired level of detail it can be easily extended to a higher (or lower) number of corner points.

3. Additionally, surfaces are included consistently using SSV elements in the collision avoidance framework. That way, the motion generation modules are able to generate collision-free whole body motion [65, 67].

Trajectory Adaptation

The sub-system for stabilizing trajectory adaptation was developed by Robert Wittmann. Here, this module is shortly explained, focusing on its integration with the main framework. For more details, see Wittmann [200].
To enable the biped to react to unknown disturbances, a *model-predictive trajectory adaptation* is introduced (fig. 7.2). It adapts the ideal planned motion $\boldsymbol{w}_{id}(t)$ based on current sensor data. This is done with a reduced robot model which allows trajectory modifications to be optimized in real-time for a certain time horizon. This way, the robot dynamics, desired motion and current state are taken into account. The leg trajectories are modified by changing final values for Δx_f and Δy_f as well as the final horizontal orientation [201]. The horizontal CoM trajectories can furthermore be adapted [203].
The main goal of this module is to stabilize the robot's absolute inclinations with regard to the ground. For stiff position-controlled robots, they are considered to be the main DoFs that deviate from the ideal values during disturbances and an indicator of stability (chapter 6).
An overview of the trajectory adaptation strategy is presented in fig. 7.5. In order to enable fast reaction times, the simplified model presented in section 6.3 is used to predict the future state of the robot and enable real-time trajectory modifications. Similarly as before, the sagittal and frontal directions are assumed to be decoupled and both planes are analyzed separately.
First, the robot's absolute inclinations are estimated by filtering the IMU's measurement data with a state observer [202]. It is based on an extended Kalman filter with compensation for model errors and external disturbances.
The footstep displacements are calculated using an optimization formulation. Using the robot's estimated state, the robot model is integrated with a certain time horizon into the future. The optimization problem is formulated to minimize the error of the upper body inclination (defined as the difference between the original planned value and the result of the model integration). It is solved using a direct shooting method [201].

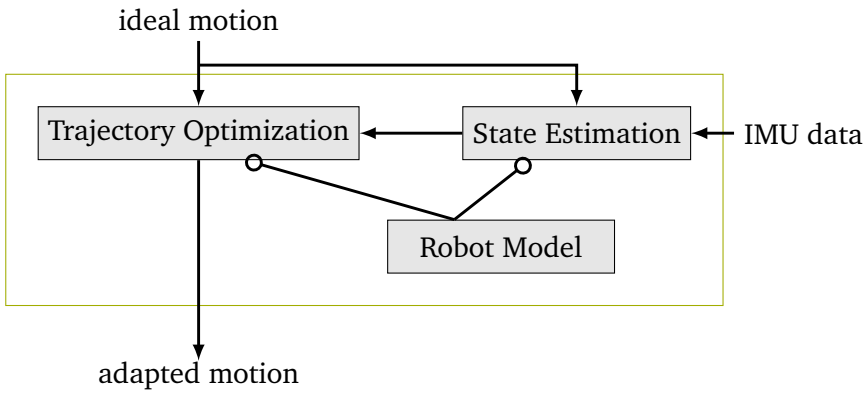


Figure 7.5: Overview of the trajectory adaptation strategy.

Simultaneous Obstacle Avoidance and Robot Stabilization

The modules presented above are both directly related to footstep placement. While the *motion planning* module defines safe footstep positions based on obstacle avoidance and environment navigation, the *trajectory adaptation* module modifies these in real-time to stabilize the robot in the presence of external disturbances. However, it is not trivial to activate both modules simultaneously, as the modifications planned by the *trajectory adaptation* module could result in unintended collisions with the environment.
To solve this problem, a strategy was developed to include the environment's collision model in the *trajectory adaptation* module (while a brief description is given here, more details can be found in Hildebrandt et al. [71]). In order to be able to do this, a criterion on the robot's priorities has to be defined. Namely, does it modify its trajectories to stabilize itself regardless of any potential collision with the environment, does it try to avoid collisions at all cost or something in between? The first case is most simple to implement although potentially dangerous. For this work, it is defined that the robot has to avoid collisions at all costs, as any other case may result of a small modification of this one.
To perform collision avoidance inside the *trajectory adaptation* module it is important to note that collision checking needs to be performed at a considerably faster rate than before (it runs with a cycle time of 20 ms). For this reason, the environment representation is further simplified. In fig. 7.6 the process is depicted. Obstacles are projected onto the ground and approximated with convex polygons. Furthermore, the robot's foot and the area of the ground that is reachable by it (determined by its kinematic capabilities) are also approximated by a convex polygon. The remaining safe area is checked for safe footstep positions using the foot's polygon. The result of this process are further convex regions which are safe for the robot to step into. These are then included in the optimization problem as linear inequalities. In order to enhance the robot's stability in spite of this limitation, further walking steps are also taken into account.

Augmented Reality

When developing an autonomous navigation system, as well as when performing teleoperation with a robot, it is useful to visualize the results of the different framework modules. During development, it helps finding errors and optimizing the system. During task execution, a clear understanding of the robot's intentions is of tremendous value to the user and facilitates human-robot interaction.

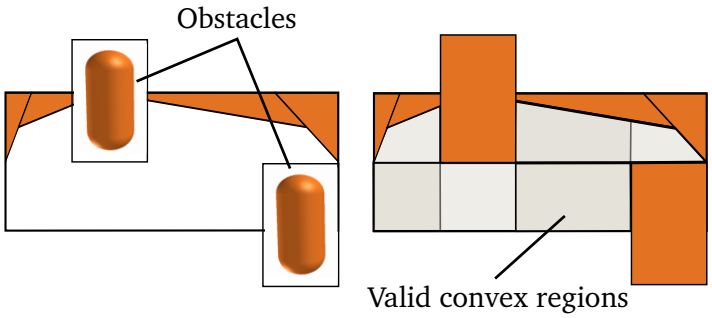


Figure 7.6: Collision avoidance during footstep modification. Obstacles (orange) are represented as convex polygons in 2D. The kinematically reachable area (white) is represented by a convex polygon as well. The safe area is divided into possible regions for footstep placement (gray).

In order to help visualize both the results of the perception system as well as its influence on the motion planner, an augmented reality (AR) system is developed. It projects collision geometries, surface approximations and step positions online into the scene. This is done either via an external RGB video feed from the scene[2] or using Microsoft's HoloLens[3]; both tools are available open source in the repository for the benefit of the community.
The objective of these systems is to provide immediate feedback about the perception system's accuracy and the quality of the motion planner's output in a context which can be immediately understoood at a glance. In the following, both implementations are described. They are the result of collaboration with Tamas Bates [9].

External RGB-D Camera

The results of the first system (called "Lola Listener" in fig. 7.1) are shown in fig. 7.7. In order to render the data correctly and obtain a good registration between the virtual objects and the RGB video feed, two things need to be taken care of. First, the camera's intrinsic parameters have to be measured and used to modify the projection. Virtual objects are rendered to reproduce (as much as possible) the physical camera's characteristics (such as FoV or lens distortion) with a virtual camera. This ensures that if a virtual object and a physical object lie at the same location relative to the camera, they should cover the same pixels in the RGB image. Second, in order to place virtual objects correctly, the transformation between the camera 6D pose and the robot's coordinate system has to be found.
While the first point can be solved using standard camera calibration routines, there is no simple, adequate solution available for the second one. Markers (see chapter 3) are typically used for such applications, but the precision of marker detection algorithms that are based only on RGB data is too low to obtain a correctly rendered image. To solve this, a sensor with correlated RGB and 3D data is used[4]. The transformation between the camera and robot coordinate systems is obtained in two steps. First, the camera's RGB feed is used to locate a marker on the robot (whose transformation to the robot's origin is known from odometry data), using the ArUco library [53]. It can provide the full 6D pose of a detected marker, but the accuracy is too low for this application. Therefore, by estimating the marker's plane using its corresponding 3D information the pose is improved. However, it is still not accurate enough to obtain a precise transformation.
Because the camera in the lab is typically fixed in place, it is possible to further exploit the

[2] https://github.com/am-lola/LolAR
[3] https://github.com/am-lola/HoLola
[4] The system is tested using a standard RGB-D sensor such as the one used for perception.

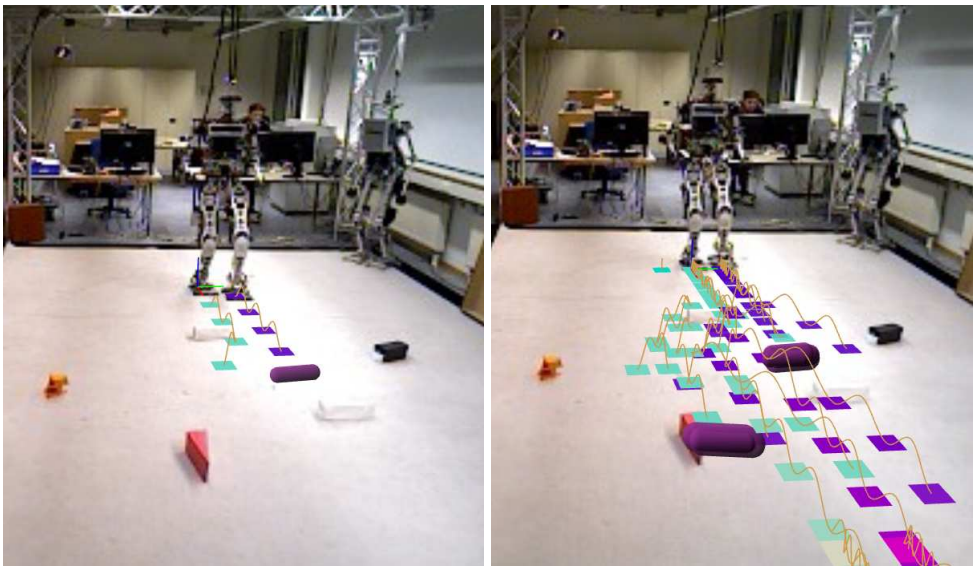


Figure 7.7: Augmented reality system with a structured light sensor. In this picture, calibration is performed manually. Left: the planned steps and results of the perception system at one point are projected into the RGB feed. Right: the accumulated results of the perception system and the step planner during the complete run against the initial RGB frame.

3D information to improve the estimation of the camera-robot transformation further. The robot's origin is initialized on top of the ground, so an estimate of the ground's location and orientation relative to the camera also helps improving the transformation between the camera and the robot coordinate system. To summarize, the transformation between the robot and the camera can be expressed as:

$$ {}^{R}_{C}T = {}^{R}_{C}T\left({}^{R}\alpha_C, {}^{R}\beta_C, {}^{R}\gamma_C, {}^{R}x_C, {}^{R}y_C, {}^{R}z_C\right) \tag{7.1} $$

where ${}^{R}_{C}T$ denotes the transformation from the robot coordinate system to the camera (following the convention of Craig [26]). It can be expressed as a set of six independent variables: $({}^{R}\alpha_C, {}^{R}\beta_C, {}^{R}\gamma_C)$, which correspond to the rotation around the robot coordinate system's x, y and z axes respectively and $({}^{R}x_C, {}^{R}y_C, {}^{R}z_C)$, the translation vector along those same axes. Out of these variables, ${}^{R}\alpha_C$, ${}^{R}\beta_C$ and ${}^{R}z_C$ are obtained from the localization of the ground while ${}^{R}\gamma_C$, ${}^{R}x_C$ and ${}^{R}y_C$ are obtained from the improved marker estimation[5]. Having obtained these parameters and the camera intrinsic parameters, the information can be easily displayed using the developed visualization software (appendix A).

Augmented Reality Glasses

Even though an augmented reality system on an external screen can be useful, an immersive system using AR glasses is considerably more helpful according to the author's experience. Therefore, a less budget-sensitive[6] augmented reality system (called "HoLola" in fig. 7.1) is developed for the Microsoft HoloLens [121]. The visual feed through the HoloLens can be seen in fig. 7.8. A short video is available online at https://youtu.be/EeDR1UNDpIY. The HoloLens is a self-contained head-mounted computer with an array of on-board sensors designed specifically for 6-DoF head-tracking and AR rendering in arbitrary environments. This

[5]The robot coordinate system lies on the ground with a vertical z axis (see fig. 1.4).
[6]Compared to the external RGB-D camera.

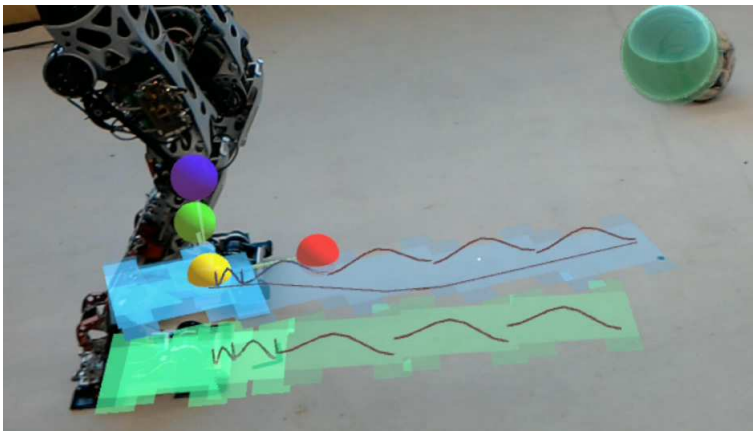


Figure 7.8: Augmented reality system with the HoloLens. Both the results of the perception system and the motion planning are projected into the environment (the ground plane is left out for clarity). The executed footsteps are depicted in blue and green and the planned ones in the same, but semi-transparent colors. The calibration is performed by placing a virtual coordinate system (depicted with a yellow sphere at the origin and red, green and blue spheres corresponding to the x, y and z axes, respectively) coincident with the one of the robot at the foot. The small error in the obstacle approximation is due to self-occlusion (chapter 5).

allows users to not just visualize data from a fixed point of view, but to actually walk around and examine it up close in detail, which makes inspecting things like footstep path trajectories much more practical. The HoloLens application is built in two layers: a simple visualization layer which draws/removes data as the robot generates new object approximations and footstep plans, and a low-level networking layer for communication with the robot's computers (section 1.3). Because the HoloLens localizes itself very accurately, rather than trying to identify the device in the robot's coordinate system a common reference is used. The users can place a coordinate frame anywhere in the HoloLens' map of the world and, by aligning this with the robot coordinate system, all data from the robot can easily be rendered. The location selected by users in this way is anchored to the features that the HoloLens uses for tracking and persists between executions of the application, so it only has to be adjusted when the location of the robot coordinate system's origin changes. It can also be updated on the fly at any time. The HoloLens also provides an RGB feed from a front-facing camera on the device which could be used to locate a marker on the robot, as with the external camera application.

Chapter 8

Autonomous Walking Results

Experimental Validation

Using the complete framework developed throughout this project (see chapter 7), the robot *Lola* was able to navigate different kinds of unknown scenarios. As mentioned before, the objective of this work is to provide humanoid robots with the ability to autonomously navigate over irregular terrain in order for them to be applied in real-world environments where wheeled robots have inherent limitations. In order to validate the proposed methods, several experiments where performed with the robot *Lola* in the laboratory, including static and dynamic obstacles of different sizes, platforms, persons and disturbances. Many of the results presented in this chapter have been published in conferences and journals, and videos of them are available on the Chair's YouTube Channel[1]. Moreover, demonstrations of these results were presented in front of live audiences [2].
At the time of writing, the results presented here represent the fastest demonstration of autonomous humanoid walking in unknown dynamic environments (and the only published work dealing with unknown dynamic scenarios). Thanks to its combination of fast environment modeling and real-time planning, *Lola* is able to walk among static and dynamic obstacles as well as other surfaces without stopping at an average speed of 0.4 m/s. This is faster than other full-size robots [24, 47, 130, 137] and considerably faster than the demonstrations at the DRC[31]. The performance results indicate that higher speeds are achievable with this same system though it would require additional tuning efforts. In this chapter, a selection of experiments is discussed and the performance of the perception system presented in chapter 5 is analyzed.

Walking in Unknown Scenarios

The framework presented in chapter 7 was tested repeatedly by making the robot *Lola* autonomously navigate in different scenarios. It is important to note that, throughout all experiments performed, the robot walks continuously without stopping and both the perception system and the motion planning module react to previously unknown scenarios during walking. In order to highlight the features presented throughout in this work, a selection of them are presented in this chapter. A video including most of them is available at https://youtu.be/VceqNJucPiw (other videos are mentioned where pertinent). In the following, the experiments are classified according to the kind of scenario that the robot must navigate.

[1] All videos available at https://www.youtube.com/appliedmechanicstum

[2] A video showing the demonstration of these results in front of live audiences is available at https://youtu.be/g6UACMHgt20

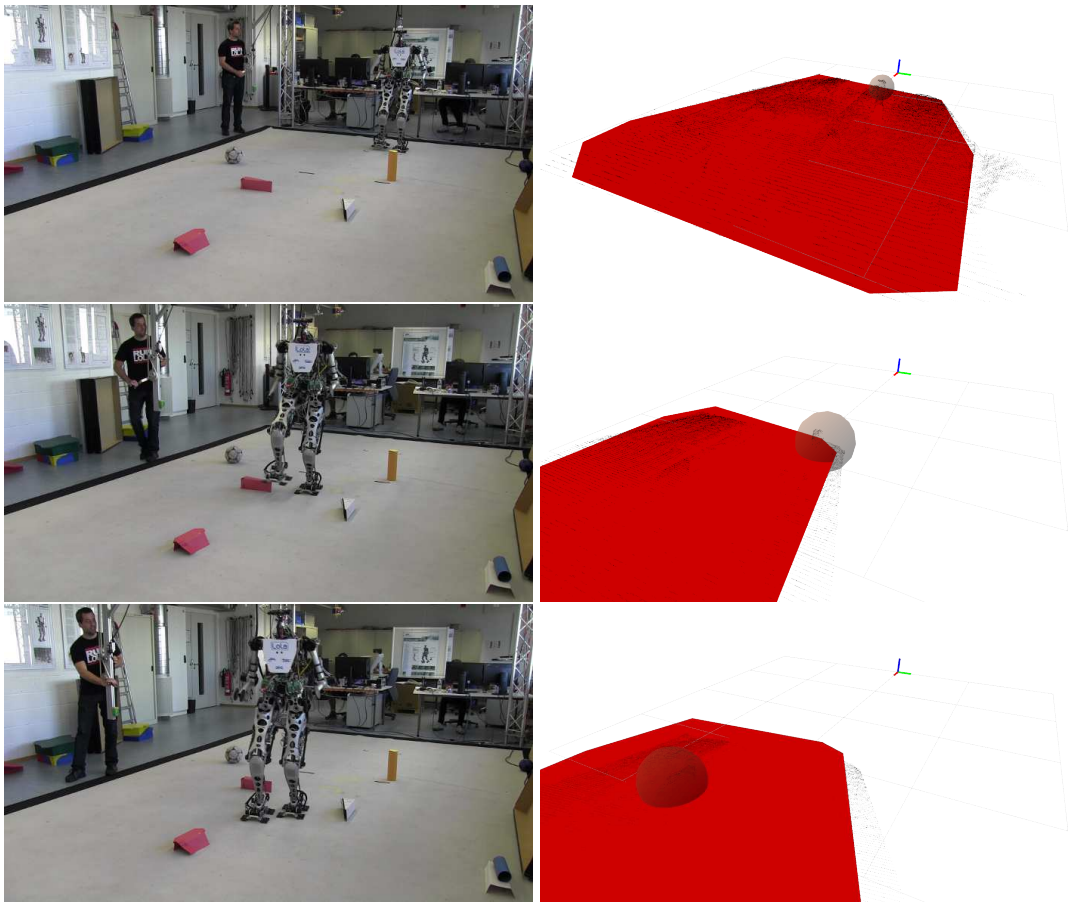


Figure 8.1: Experiment with unexpected obstacles. Left: the robot tries to find a way through the room obstructed by obstacles on the ground. Right: the results of the perception system. The ground is approximated with a polygon (red) while obstacles are approximated with SSVs (cyan); the point cloud is cropped to the walking area.

Walking amongst Obstacles

The first group of experiments validate the ability of the robot to avoid unexpected obstacles, using motions such as stepping over or sideways. A sequence of the video and the results of the perception system for one experiment are depicted in fig. 8.1. The robot has to find a way through the room which is blocked with relatively small, previously unknown obstacles in the ground. It receives a signal to walk forward. As can be seen, the robot deviates its path to find the easiest way to advance through the room, stepping over obstacles when necessary. In fig. 8.2, the real robot is compared to the world representation during another experiment where the robot steps over an obstacle (a video is available at https://youtu.be/6PLN6B4vSHM).

Walking over a Platform

These experiments test the reaction of the framework to uneven terrain, such as platforms or stairs. They validate both the ability of the perception system to detect and model surfaces accurately and fast enough during walking and the flexibility of the motion planner to adapt the walking sequence in real-time. A sequence of the video and the results of the perception system are depicted in fig. 8.3. A platform (12 cm high) together with a few obstacles (to

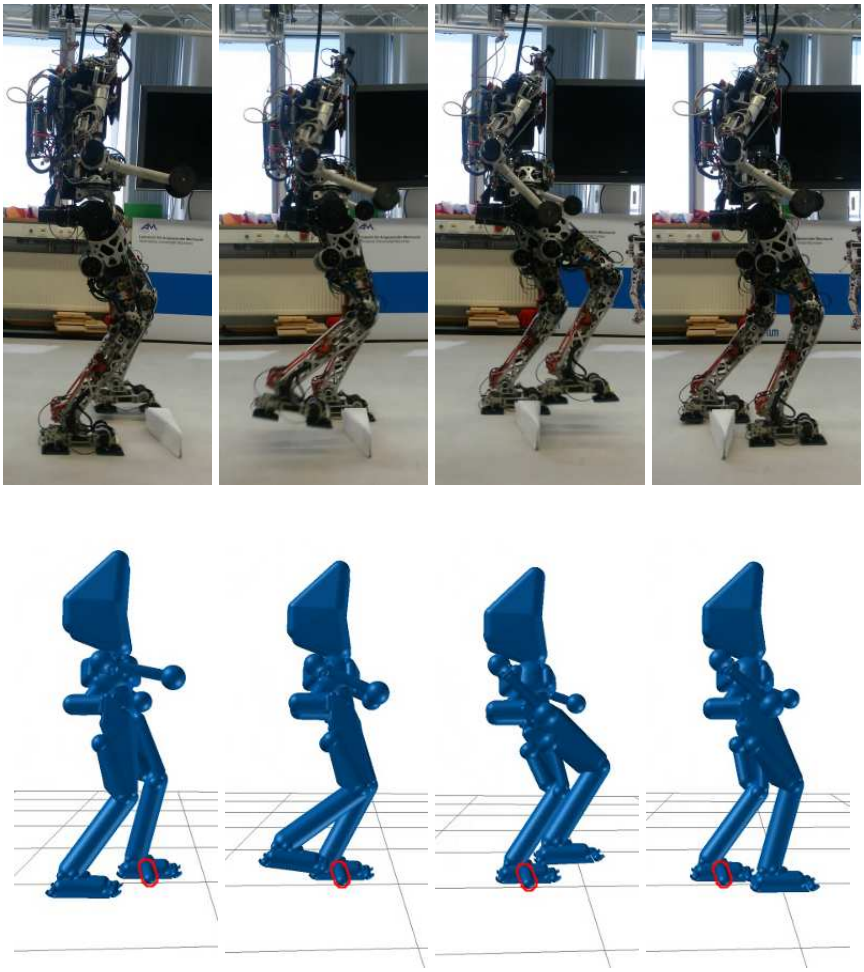


Figure 8.2: Side view of obstacles experiment. The robot is shown stepping over an obstacle (top) and the collision model of the robot (blue) and the obstacle (blue with red border) is shown for reference.

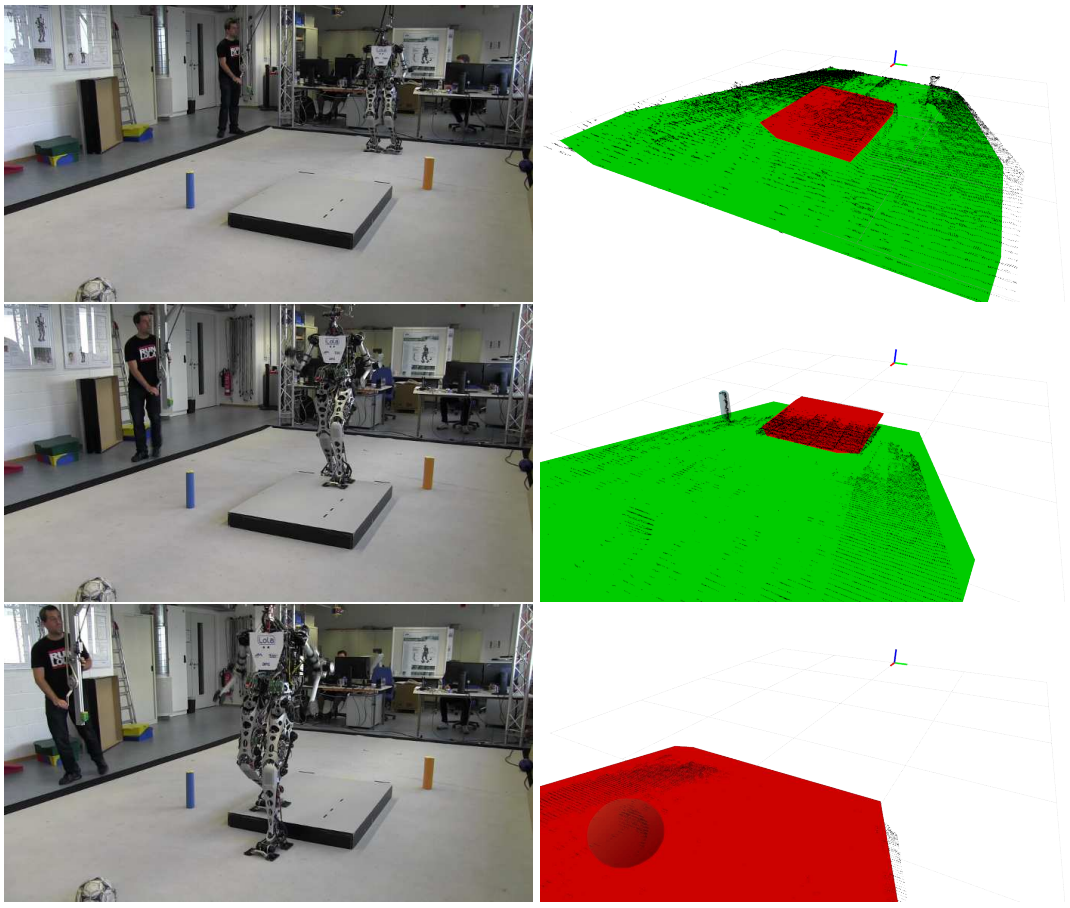


Figure 8.3: Experiment with unexpected platform. Left: the robot walks up and down an unexpected platform which stands in the way to the end of the room. Right: the results of the perception system. Surfaces are approximated with polygons (red and green) while obstacles are approximated with SSVs (cyan); the point cloud is cropped to the walking area.

block alternative paths around the platform) are placed on the ground, obstructing the way through the room. Finding no other alternative, the robot walks up and down the platform on its way forward without stopping (see fig. 8.4).

Highly Dynamic Scenarios with Large Obstacles and Humans

These experiments test one of the most important contributions of this work: the modeling and reaction to unknown dynamic scenarios. They validate both the ability of the perception system to track large dynamic objects during walking and the fast reaction times from the motion planner [70]. Furthermore, the FOV of the camera is manually reduced to provoke more sudden reaction and shorter planning times. A sequence of the video and the results of the perception system are depicted in fig. 8.5. The robot gets a goal position in an initially empty area. When the robot starts moving, a person walks in, blocking its path. When the robot turns to avoid the person, the person blocks its path again with a chair. After avoiding the chair and the person, the robot's path is blocked yet again so that it is prevented from reaching the goal position. The robot's reaction (which depends on the sensor's limited FoV) and sudden change of direction can be appreciated in fig. 8.5, emphasizing the real-time capabilities of this framework.

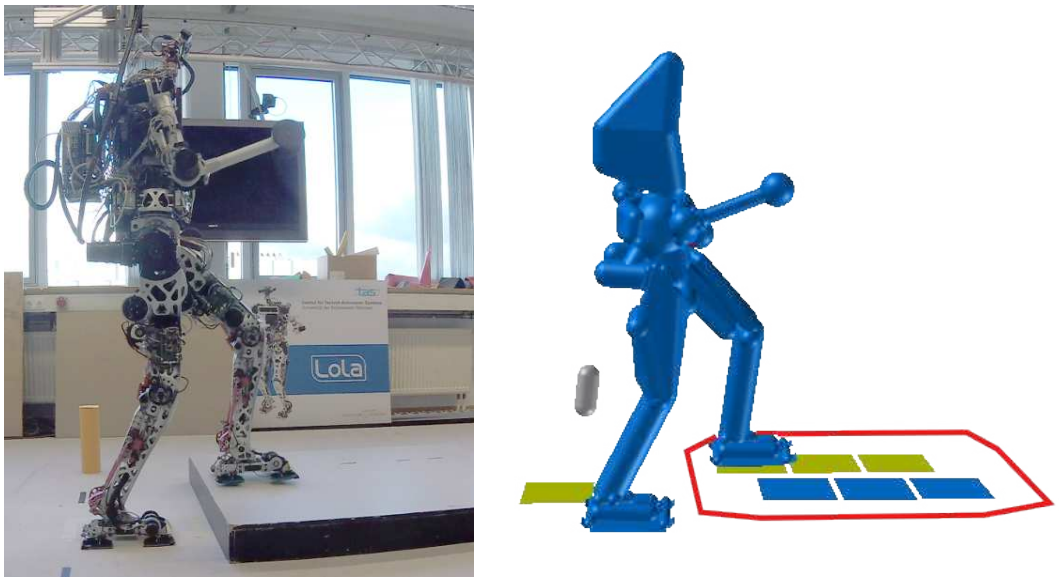


Figure 8.4: Collision world. Left: the robot walks up an unexpected platform which stands in the way to the end of the room. Right: representation of the 3D collision world including the robot and external obstacles modeled as SSVs, the platform modeled as a polygon and the planned footstep locations.

Static Obstacles and External Disturbances

These experiments put the stability of the robot to test. Besides the perception system and the motion planning, the stabilizing trajectory adaptation is active. A sequence of the experiment is shown in fig. 8.7. The robot is commanded to walk forward in a room with scattered obstacles on the ground. The difference with previous experiments is that the robot is pushed several times during walking. While the robot is avoiding obstacles detected by the perception system, it must adapt its swing foot motion in real-time to compensate for external disturbances (see chapter 7). The overall ideal step sequence and modified footstep positions are shown in fig. 8.6. A video of this experiment is available under https://youtu.be/RjqAh3Blxng. As can be seen, the system is capable of simultaneously detecting and avoiding obstacles while compensating external disturbances.

Perception System Performance

Here the performance of the perception system during experiments is analyzed. In order to obtain a greater amount of data and validate the capabilities of the system in more complex environments, a series of dynamic scenarios with people, objects and platforms are recorded in front of the robot's camera. These include dynamic scenes with several objects and platforms of different shapes and sizes (e.g. figs. 5.2 and 5.11 to 5.14 and transitions between them). The duration of the different processes can be seen in table 8.1. In the *Surface Modeling* process, higher runtimes correspond to frames that consist mostly of walkable planes, as the different algorithms have to iterate through a high number of points. In the case of the *Obstacle Modeling* process, higher runtimes are the result of sudden changes of the scene where the algorithm has to re-converge; they improve considerably after a few frames. The final list of obstacles and surfaces are sent to the motion planning module with a set frequency of around 5 Hz. The future eight walking footsteps are re-planned every walking step

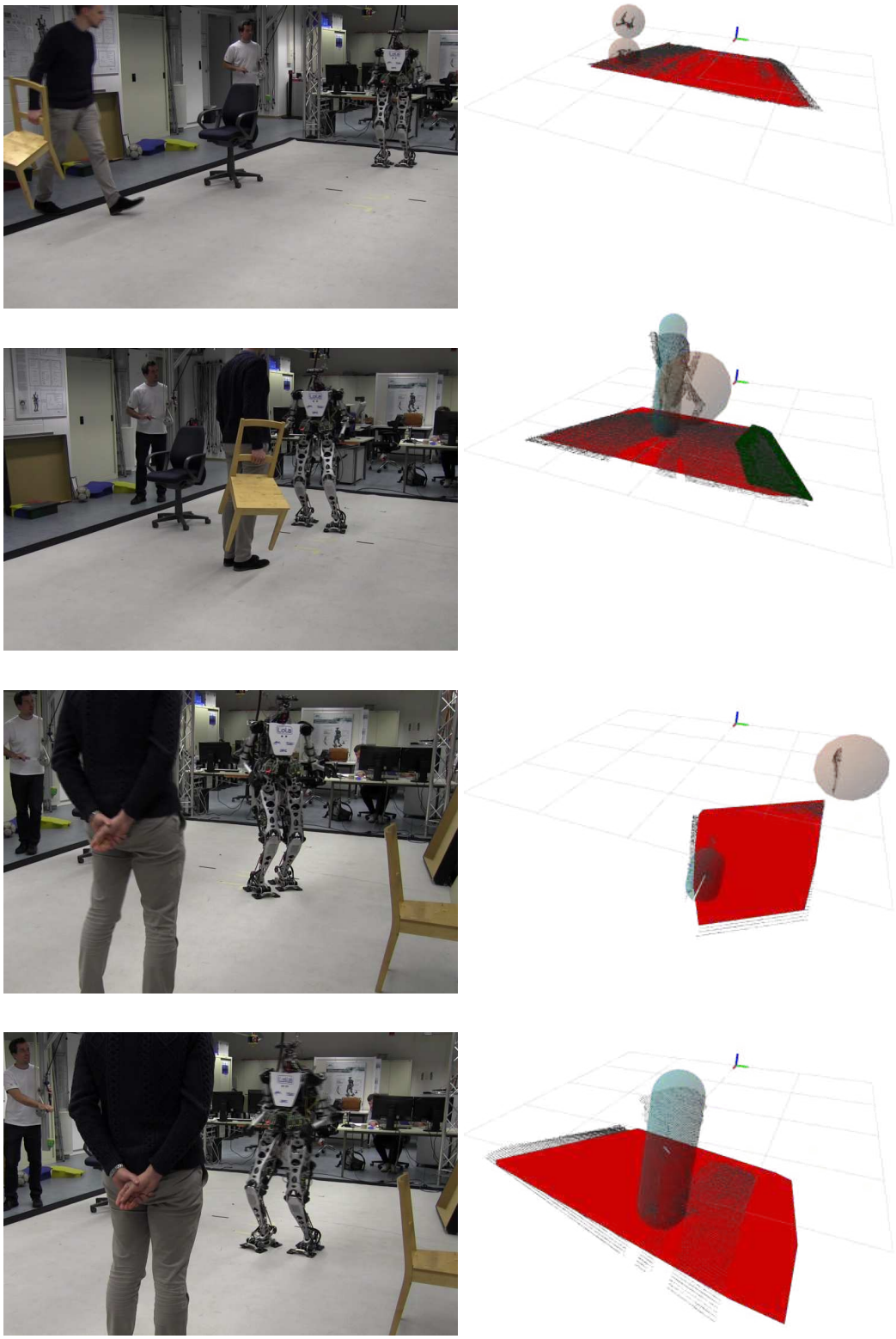


Figure 8.5: Experiment in dynamic scenarios. Left: the robot tries to reach a goal position in the room despite a person repeatedly blocking its way. Right: the results of the perception system. The ground is approximated with a polygon (red) while obstacles are approximated with SSVs (cyan); the point cloud is cropped to the walking area and further-away points are removed to evaluate limited reaction times.

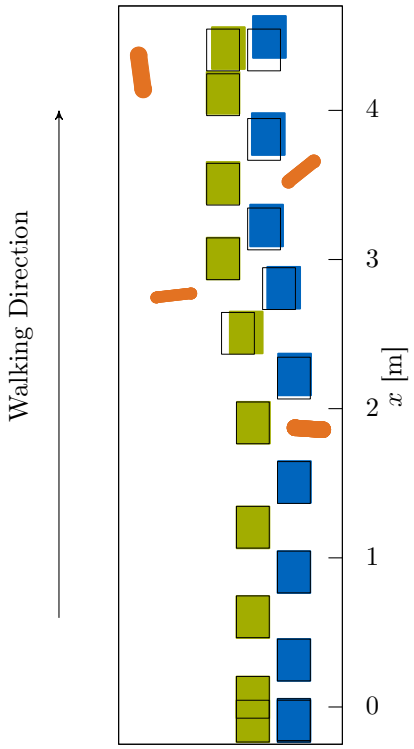


Figure 8.6: Experiment 2: Ideal step sequence is represented as black boxes. Modified and executed steps are represented as filled boxes. Relevant obstacles in orange (see fig. 8.7).

(0.8 s).
Unfortunately, open source perception systems for humanoids are extremely uncommon. Most relevant publications in the field don't release their source code and often omit thorough performance results, which makes it difficult to perform an objective comparison. Nevertheless, a few examples are mentioned for reference. The framework presented by Nishiwaki et al. [137] requires a 1 second sensor sweep with the robot still to acquire and process perception information to create 2.5D maps. Using a similar sensor and resolution as this work, the framework presented by Maier et al. [114] for the *Nao* robot runs at a frequency of 6 Hz and uses a 3D voxelization to model environments, without performing surface segmentation. More recently, Fallon et al. [47] presented an approach for sensor fusion and plane detection. It is based on 2.5D maps which are reduced by removing points belonging to the ground. In its actual configuration, the segmentation process takes 615 ms in average. Even though the algorithm runs during motion, each walking step takes 4 s, which is five times slower than the presented experiments with *Lola*. Moreover, all these frameworks assume static environments in their application. A notable exception is the one presented by Karkowski et al. [92], which achieved fast segmentation times of only a few milliseconds in dynamic environments although the final implementation runs at 10 Hz. However, it is based on a limited height-map representation and only performs 2D collision checking. Compared to the results mentioned above, the framework presented in this work proved to be faster as well as more flexible and generally applicable.

Figure 8.7: External Disturbances. From 1 to 6, the robot finds his way through cluttered terrain while being repeatedly disturbed by a human.

Spatial Resolution	Plane Segmentation max	min	mean	Surface Modeling max	min	mean	Obstacle Modeling max	min	mean
$1\,\mathrm{cm}^3$ ($\sim$ 65000 points)	45	6	**13**	373	1	**92**	353	8	**107**
$2\,\mathrm{cm}^3$ ($\sim$ 21000 points)	19	2	**6**	123	1	**21**	284	3	**57**

Table 8.1: Performance of Perception System. Runtime (in ms) for different point cloud resolutions of the separate parallel processes of the perception system (maximum, minimum and mean values) for highly complex and dynamic scenarios spanning 500 frames approximately.

Chapter 9

Conclusions

Contributions of this Thesis

Humanoid robots have great potential to be used as human replacements where the human form is an important factor. However, they are still a long way from being directly applicable. The main reasons behind this are their high number of degrees of freedom and possible motions as well as their state of underactuation and potential instability. For the present time, exploration missions will surely continue to be realized with wheeled robots, flying robots (such as quadcopters) or a combination of both. The first real world applications of legged robots will most probably use quadrupeds due to their superior stability (see chapter 1). Nevertheless, humanoid robots still have countless of applications, from the development of medical prostheses to the testing of human accessories (such as protective clothing) and safe exploration of human-made environments.
Before reaching that point, humanoid robots will have to (among other things) achieve a level of flexibility and robustness in locomotion that is at least comparable to humans. The work presented in this thesis was developed as part of a project intended to advance the state of the art in that direction. The result is a general framework for autonomous navigation with special modules for perception, motion planning and robot stabilization. With it, the robot is able to autonomously navigate in unknown dynamic environments by avoiding obstacles or stepping over platforms while resisting external disturbances. Moreover, it is able to do it in real-time while walking at speeds of around 0.5 m/s, which is faster than any other published works, as far as the author knows.
Particularly, this thesis deals with the acquisition and modeling of perception information as well as with the integration of perception inaccuracies into the robot's control system. Its most important contributions are:

- A review and analysis of perception sensing technology (chapter 3).
- A new procedure for integrating and calibrating such a sensor into a robotic system (chapter 3).
- A strategy for the combination of environment recognition and modeling algorithms, together with a new environment modeling strategy for unknown environments (chapter 4).
- An efficient, open source perception system that is capable of processing dynamic environments in 3D during the robot's motion. It can track and model unknown obstacles and surfaces faster than any other published system, as far as the author knows (chapter 5).
- A new walking control strategy for humanoid robots. It includes methods to adapt the robot's walking based on the inaccuracies of the perception system and a modifi-

cation to ZMP-based controllers to increase their robustness against these inaccuracies (chapter 6).

- An set of open source augmented reality systems for online visualization of perception and planning information (chapter 7).

Many of the presented systems are released as open source for the benefit of the community. They can be of help for other researchers who may use them to test and improve their systems or build on top of them. The results presented in chapter 8 should serve as a demonstration of the application possibilities of such robotic systems as well as a motivation to develop them further. In the following, the limitations of the presented work are discussed and future research directions are suggested.

Shortcomings and Open Issues

For this project, focus was laid on two-legged locomotion. *Lola* does not even posses actuated hands, its arms are used for the compensation of angular momentum and center of gravity trajectories [164]. The development of the methods and algorithms presented in this thesis takes mostly the robot's legs into consideration.
One area in which this is evident is the environment modeling strategy and perception system. The classification of the environment into walkable and non-walkable components does not take grasping or arm contact possibilities into consideration. In order to do this, further classification of the environment and new modeling strategies would have to be implemented. Another limitation of this modeling strategy is that it doesn't perform well in enclosed spaces, as walls are identified as obstacles due to their high inclination. Their approximation with SSVs is sometimes a problem as it restricts the available walking area considerably.
One of the main objectives of this work was to achieve real-time performance in dynamic environments. This was accomplished at the sake of strong simplifications in the environment modeling strategy and perception system as well as the walking controller, including the planning system and stabilization strategies. The sum of these approximations results in a system performance that is not ideal, sometimes failing to find safe paths or stabilize the robot in time. As processing power increases, all of these systems could be improved resulting in a better performance.
Another important assumption taken throughout this work is the rigidity of the environment. Walkable surfaces are assumed to be clearly defined and rigid, serving as firm ground for the robot's walking. Non-walkable obstacles are considered rigid and fatal upon contact, which restricts the navigation possibilities of the robot. Naturally, there are countless of cases where these assumptions fail. Humans are able to walk over non-rigid terrain, adapting their walking to it. They are also capable of recognizing and traversing flexible obstacles, such as curtains or tree leaves. In the future, artificial intelligence techniques could help identify these cases in order to correctly deal with them.

Potential Directions of Future Research

As explained above, the presented framework is still considerably constrained in its applicability and is far away from a real world application. Some of the topics on which future research could be done are mentioned in this section.

- *Environment Modeling and Perception System*. As discussed before, the strategy presented in this work fails to perform adequately in some cases and does not take actions as arm-environment contact or manipulation into account. One interesting direction of research would be to apply the concept discussed in chapter 4 to this problem, finding an environment classification and modeling strategy that can be efficiently processed by a corresponding planning framework. Another interesting problem is the correct modeling and processing of enclosed areas in a way that can be taken into account for real-time navigation.

- *Machine Learning and Artificial Intelligence*. As was repeatedly mentioned, this work assumes completely unknown environments and focuses on general environment modeling strategies. Nevertheless, for tasks such as manipulation or direct interaction with the environment learning methods will become increasingly relevant in the near future. Moreover, in order to deal with non-standard environments, such as traversable objects or non-rigid terrain they may be the only effective alternative. Artificial Intelligence can be potentially applied to the walking planning and control modules as well: they depend on a large number of factors, which are presently set by the user but might be automatically adapted to the task and walking scenario by such methods.

- *Control*. Even assuming a perfect environment representation and ideal motion planning, humanoid robots are not nearly as capable of maintaining stability on irregular terrain or in the presence of disturbances as human beings. One of the ways to improve this is through faster, more capable force control and whole-body control strategies. By including additional or more precise sensors, properties about the environment could be directly considered by a control framework. In the author's opinion, this is an area that is fundamental to achieve real world applicability.

- *Planning in Dynamic Environments*. Even though dynamic obstacles can be tracked by the perception system (chapter 5) and their velocity can be estimated, the motion planning module considers the environment static: fast reactions are achieved through high update-rates. This is done because of the limited processing power available in a real-time application. An interesting direction of research would be to include time as an additional variable in the planning system to take dynamic components of the environment directly into consideration.

Naturally, these are just a few suggestions based on considerations taken during this project. The field of humanoid navigation is still an open one and there is not even a clear consensus on which control principles are most effective for such a system. One sure thing is that the technology will continue to advance in the near future and its performance may soon be surprising compared to the results presented in this work. It may still be that this generation will see the first real world applications of such robotic systems.

Appendix A

Visualization Library

Naturally, one of the key components for easier development and debugging of perception algorithms is a powerful, comprehensive visualization tool. Due to the nature of the presented application, such a tool would have to be able to display not only point clouds but also the geometries used for approximating the different components of the environments. In order to be usable during experiments, it also should be efficient in its use of computational resources.
There are a few visualizatin tools available, such as the RViz for ROS [156] or PCL's Visualizer [144]. However, the PCL Visualizer is slow and not easily customizable and RViz requires the complete ROS overhead to work which makes it impractical for other purposes outside ROS-specific applications. Besides, none of these tools provides interfaces that allow to take control of the visualizer's camera parameters, which is necessary for augmented reality applications (see chapter 7).
With these issues in mind, a new open source visualization tool was developed. It is available in the institute's repository[1] and consists of a C++ library based on OpenGL [138] which can be called from any C++ application. Its structure can be seen in fig. A.1. In order to implement it, the user deals with one interface called "ARVisualizer" in which characteristics such as background and camera parameters (for augmented reality applications), graphical user interface windows, objects and actions can be set. Each instance of the ARVisualizer starts a separate window that can be used for different kinds of applications, such as camera calibration (chapter 3), visualization of point clouds and geometric approximations (chapter 5) or augmented reality (chapter 7). Various examples of visualization applications are provided in the repository.

[1] https://github.com/am-lola/ARVisualizer

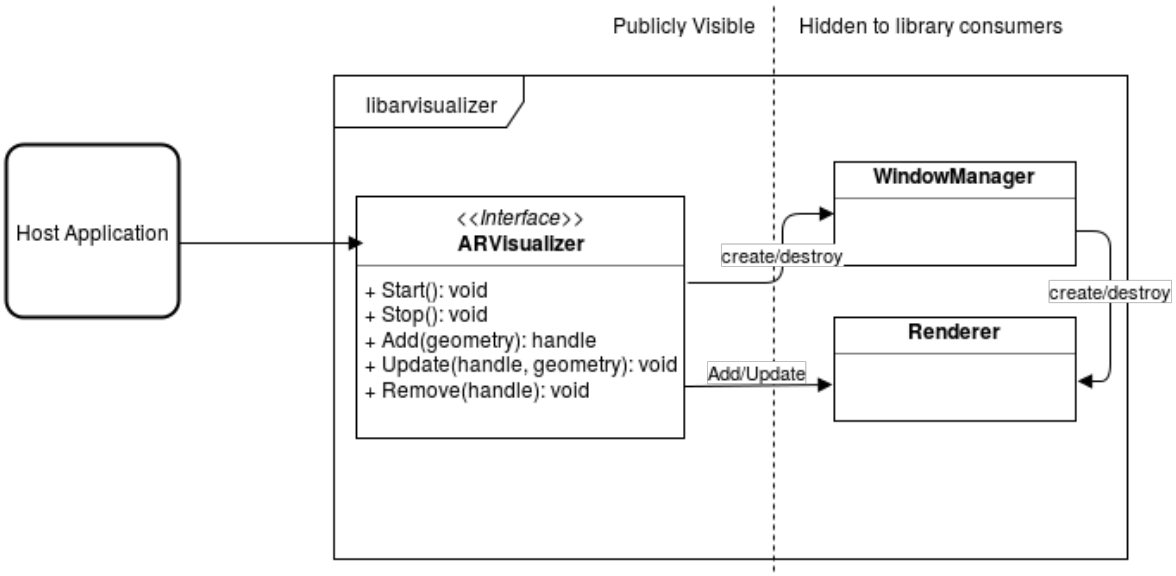


Figure A.1: High-level diagram of library architecture

Appendix B

Surface Clustering

The clustering of data into different objects is a very common problem in computer science. There are numerous algorithms that deal with this issue which have different advantages and disadvantages. For the surface approximation process, 3D points have to be clustered into different surfaces that the robot may step onto. The problem seems at first to be fit for a 3D clustering algorithm. However, if information about the point cloud's planes is available, it can be simplified considerably: by projecting the points to their corresponding surface planes the problem is reduced to a 2D clustering (chapter 5). For this work, several algorithms were evaluated:

- *K-means* [107]. This algorithm assigns points to different clusters and iteratively minimizes the total squared error between each point and its cluster center. It is additionally necessary to detect the number of clusters, using an algorithm such as the one described by Pham et al. [143]. One of its main disadvantages is that, as a result of assigning every single point to one cluster, it is highly sensible to sensor noise.
- *Gaussian Mixture Models*. This algorithm works by iteratively estimating probabilistic distributions of points [128] and is popular among machine learning applications. These probabilistic algorithms often fail to separate non-uniformly distributed clusters and, due to their iterative nature and long runtime, are not well suited for dynamic scenarios (chapter 5).
- *2D/3D Euclidean Clustering* [139, 145]. It separates points by iteratively evaluating their distance to their neighbors.
- *DBSCAN* [43]. This algorithm starts from a single point, a given neighbor distance and a minimum number of neighbors. It first joins the initial point together with all neighbors within the given distance. Every subsequent point that has the minimum number of neighbors within the given distance belongs to the cluster. When no more points comply, a new cluster is started. By providing the right parameters, it can obtain very accurate results.
- *2D Grid*. By projecting the points to a discrete grid, clustering can be performed by simple connectivity check between grid cells. Its runtime and accuracy depend on the grid cell size.

The performance of the local implementation of the 2D grid was compared against the DBSCAN and 2D/3D Euclidean Clustering algorithms for a set of examples. In terms of quality of results, the difference number of points belonging to the separate clusters is consistently below 1% between algorithms, and usually corresponds to noisy data around the borders. The most notable differences can be found in runtime, where the 2D Grid consistently outperforms the other algorithms by a factor of 50 or more. In table B.1, the results for an example scenario can be seen.

Plane	3D Euclidean	2D Euclidean	DBSCAN	2D Grid
1	157.4	1155.9	425.1	1.1
2	62.8	289.1	160.0	0.6
3	20.0	35.3	50.8	0.2
4	12.0	17.0	26.7	0.2

Table B.1: Performance of Clustering Algorithms. Runtime (in ms) of different algorithms for the four existing planes on a lab scenario. Adapted from Floeren [50].

References

[1] Ackerman, E. “Lidar that will make self-driving cars affordable”. In: *IEEE Spectrum* (Oct. 2016), p. 14. ISSN: 0018-9235.

[2] Ackerman, E. “Hail, Robotaxi!” In: *IEEE Spectrum* (Jan. 2017), pp. 22–25. ISSN: 0018-9235.

[3] Ackerman, E. et al. “SCHAFT Unveils Awesome New Bipedal Robot at Japan Conference”. In: *IEEE Spectrum* (Apr. 2016). URL: https://spectrum.ieee.org/automaton/robotics/humanoids/shaft-demos-new-bipedal-robot-in-japan.

[4] Aikawa, H. et al. “Development of a new humanoid robot WABIAN-2”. In: *IEEE International Conference on Robotics and Automation*. 2006, pp. 76–81. ISBN: 0-7803-9505-0.

[5] Asimov, I. “Liar!” In: *Astounding Science Fiction* (May 1941). ISSN: 1059-2113.

[6] Asimov, I. “Galley Slave”. In: *Galaxy - ASIN: B0018ZIATY* (Dec. 1957).

[7] *Asus Xtion PRO LIVE*. 2017. URL: https://www.asus.com/3D-Sensor/Xtion_PRO_LIVE/ (visited on 09/09/2017).

[8] Bae, H. et al. “Walking-Wheeling Dual Mode Strategy for Humanoid Robot, DRC-HUBO+”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2016, pp. 1342–1348. ISBN: 9781509037612.

[10] Bedini, S. A. “The Role of Automata in the History of Technology”. In: *Technology and Culture* 5.1 (1965). DOI: 10.2307/3101120.

[11] Bontsema, J. et al. “CROPS: high tech agricultural robots”. In: *International Conference of Agricultural Engineering*. Zurich, 2014, pp. 6–10. ISBN: 978-0-9930236-0-6.

[13] Buschmann, T. “Simulation and Control of Biped Walking Robots”. Dissertation. Technische Universität München, 2010. ISBN: 978-3-86853-804-5.

[14] Buschmann, T. et al. “A Collocation Method for Real-Time Walking Pattern Generation”. In: *IEEE-RAS International Conference on Humanoid Robots*. 2007, pp. 1–6. ISBN: 978-1-4244-1861-9.

[15] Buschmann, T. et al. “Biped Walking Control Based on Hybrid Position/Force Control”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2009, pp. 3019–3024. ISBN: 978-1-4244-3803-7.

[16] Buschmann, T. et al. “Walking in Unknown Environments — a Step Towards More Autonomy”. In: *IEEE-RAS International Conference on Humanoid Robots*. 2010, pp. 237–244. ISBN: 978-1-4244-8688-5. DOI: 10.1109/ICHR.2010.5686338.

[17] Buschmann, T. et al. “Experiments in Fast Biped Walking”. In: *IEEE International Conference on Mechatronics*. 2011, pp. 863–868. ISBN: 978-1-61284-982-9. DOI: 10.1109/ICMECH.2011.5971235.

[18] Buschmann, T. et al. “Event-Based Walking Control - From Neurobiology to Biped Robots”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012, pp. 1793–1800. ISBN: 978-1-4673-1736-8.

[19] Buschmann, T. et al. “Controlling legs for locomotion-insights from robotics and neurobiology”. In: *Bioinspiration & Biomimetics* 10.4 (2015), pp. 01–38. ISSN: 1748-3190. DOI: 10.1088/1748-3190/10/4/041001.

[21] Bykat, A. “Convex hull of a finite set of points in two dimensions”. In: *Information Processing Letters* 7.6 (1978), pp. 296–298. ISSN: 00200190. DOI: 10.1016/0020-0190(78)90021-2.

[22] Čapek, K. *R.U.R. (Rossum’s Universal Robots)*. Dover Publications, 2001, p. 58. ISBN: 0486419266.

[23] Chestnutt, J. et al. “Footstep planning for the Honda ASIMO humanoid”. In: *IEEE International Conference on Robotics and Automation*. 2005, pp. 629–634. ISBN: 078038914X. DOI: 10.1109/ROBOT.2005.1570188.

[24] Chestnutt, J. et al. “Biped Navigation in Rough Environments using On-board Sensing”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2009, pp. 3543–3548. ISBN: 978-1-4244-3803-7. DOI: 10.1109/IROS.2009.5354575.

[25] Christensen, H. I. et al. “Sensing and Estimation”. In: *Springer Handbook of Robotics*. Ed. by Siciliano, B. et al. Springer Berlin Heidelberg, 2008. Chap. 4, pp. 87–108. ISBN: 978-3-540-23957-4.

[26] Craig, J. J. *Introduction to robotics: mechanics and control*. Vol. 3. Pearson Prentice Hall Upper Saddle River, 2005. ISBN: 978-0201543612.

[27] Cree, M. J. et al. “Analysis of the SoftKinetic DepthSense for Range Imaging”. In: *International Conference on Image Analysis and Recognition*. Ed. by Kamel, M. et al. Berlin, Heidelberg: Springer, 2013, pp. 668–675. ISBN: 978-3-642-39094-4. DOI: 10.1007/978-3-642-39094-4_76.

[28] Cupec, R. et al. “An Approach to Environment Modelling for Biped Walking Robots”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2005, pp. 424–429. ISBN: 0-7803-8912-3. DOI: 10.1109/IROS.2005.1545446.

[29] Dai, H. et al. “L2 -Gain Optimization for Robust Bipedal Walking on Unknown Terrain”. In: *IEEE International Conference on Robotics and Automation*. 2013, pp. 3101–3108. ISBN: 9781467356428.

[30] Daniilidis, K. et al. “3-D Vision and Recognition”. In: *Springer Handbook of Robotics*. Ed. by Siciliano, B. et al. Springer Berlin Heidelberg, 2008. Chap. 23, pp. 543–562. ISBN: 978-3-540-23957-4.

[31] *DARPA Robotics Challenge*. 2017. URL: http://archive.darpa.mil/roboticschallenge/ (visited on 09/09/2017).

[32] Deits, R. et al. “Footstep Planning on Uneven Terrain with Mixed-Integer Convex Optimization”. In: *IEEE-RAS International Conference on Humanoid Robots*. 2014, pp. 279–286. ISBN: 978-1-4799-7174-9. DOI: 10.1109/HUMANOIDS.2014.7041373.

[33] Dimitrov, D. et al. “On the implementation of model predictive control for on-line walking pattern generation”. In: *IEEE International Conference on Robotics and Automation*. 2008, pp. 2685–2690. ISBN: 978-1-4244-1646-2.

[34] Ditrich, F. et al. “A new efficient algorithm for fitting rectangular boxes and cubes in 3D”. In: *IEEE International Conference on Image Processing*. Vol. 18. 2005, pp. 1–4. ISBN: 0780391349. DOI: 10.1109/ICIP.2005.1530260.

[35] Ditrich, F. et al. “Robust Fitting of 3D Objects by Affinely Transformed Superellipsoids Using Normalization”. In: *International Conference on Computer Analysis of Images and Patterns*. 2007, pp. 490–497. ISBN: 9783540742715. DOI: 10.1007/978-3-540-74272-2_61.

[36] Due Trier, Ø. et al. “Feature extraction methods for character recognition-A survey”. In: *Pattern Recognition* 29.4 (1996), pp. 641–662. ISSN: 00313203. DOI: 10.1016/0031-3203(95)00118-2.

[37] Durrant-Whyte, H. et al. “Multisensor Data Fusion”. In: *Springer Handbook of Robotics*. Ed. by Siciliano, B. et al. Springer Berlin Heidelberg, 2008. Chap. 25, pp. 585–614. ISBN: 978-3-540-23957-4.

[38] Eddy, W. F. “A New Convex Hull Algorthm for Planar Sets”. In: *ACM Transactions on Mathematical Software* 3.4 (1977), pp. 393–403. ISSN: 00983500. DOI: 10.1145/355759.355766.

[39] El Khoury, A. et al. “Optimal motion planning for humanoid robots”. In: *IEEE International Conference on Robotics and Automation*. 2013, pp. 3136–3141. ISBN: 978-1-4673-5643-5. DOI: 10.1109/ICRA.2013.6631013.

[40] Engel, J. et al. “LSD-SLAM: Direct Monocular SLAM”. In: *European Conference on Computer Vision*. 2014, pp. 834–849. ISBN: 9783319106045. DOI: 10.1007/978-3-319-10605-2_54.

[41] Englsberger, J. et al. “Overview of the torque-controlled humanoid robot TORO”. In: *IEEE-RAS International Conference on Humanoid Robots*. 2015, pp. 916–923. ISBN: 9781479971749. DOI: 10.1109/HUMANOIDS.2014.7041473.

[42] Ess, A. et al. “Object Detection and Tracking for Autonomous Navigation in Dynamic Environments”. In: *The International Journal of Robotics Research* 29.14 (2010), pp. 1707–1725. ISSN: 0278-3649. DOI: 10.1177/0278364910365417.

[43] Ester, M. et al. “A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise”. In: *2nd International Conference on Knowledge Discovery and Data Mining*. 1996, pp. 226–231. ISBN: 1577350049. DOI: 10.1.1.71.1980.

[44] *European Robotics Challenges*. 2017. URL: http://www.euroc-project.eu/ (visited on 09/09/2017).

[45] Evans, J. “Driverless cars: Kangaroos throwing off animal detection software”. In: *ABC Australia* (June 2017). URL: https://goo.gl/cGrLqe.

[46] Ewald, A. et al. “Generating Smooth Trajectories Free from Overshoot for Humanoid Robot Walking Pattern Replanning”. In: *Autonomous Mobile Systems 2012* (2012), pp. 89–97. DOI: 10.1007/978-3-642-32217-4_10.

[47] Fallon, M. F. et al. “Continuous Humanoid Locomotion over Uneven Terrain using Stereo Fusion”. In: *IEEE-RAS International Conference on Humanoid Robots*. 2015, pp. 881–888. ISBN: 9781479968855. DOI: 10.1109/HUMANOIDS.2015.7363465.

[48] Fischler, M. a. et al. “Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography”. In: *Communications of the ACM* 24 (1981), pp. 381–395. ISSN: 00010782. DOI: 10.1145/358669.358692.

[49] Fisher, R. B. et al. “Range Sensors”. In: *Springer Handbook of Robotics*. Ed. by Siciliano, B. et al. Springer Berlin Heidelberg, 2008. Chap. 22, pp. 521–542. ISBN: 978-3-540-23957-4.

[51] Frenkel, L. et al. “Recursive Expectation-Maximization (EM) Algorithms for Time-Varying Parameters with Applications to Multiple Target Tracking”. In: *IEEE Transactions on Signal Processing* 47.2 (1999), pp. 306–320. ISSN: 1053587X. DOI: 10.1109/78.740104.

[52] Gao, J. H. et al. “A smartphone-based laser distance sensor for outdoor environments”. In: *IEEE International Conference on Robotics and Automation*. 2016. DOI: 10.1109/ICRA.2016.7487457.

[53] Garrido-Jurado, S. et al. “Automatic generation and detection of highly reliable fiducial markers under occlusion”. In: *Pattern Recognition* 47.6 (2014), pp. 2280–2292. DOI: 10.1016/j.patcog.2014.01.005.

[54] *Gazebo Simulation*. 2016. URL: https://www.gazebosim.org (visited on 11/20/2016).

[55] Goswami, A. et al. “Rate of change of angular momentum and balance maintenance of biped robots”. In: *IEEE International Conference on Robotics and Automation*. 2004, pp. 3785–3790. ISBN: 0-7803-8232-3.

[56] Grendelkhan. *A Waymo self-driving car on the road in Mountain View, making a left turn. Distributed under the CC BY-SA 4.0 licence.* 2017. URL: https://creativecommons.org/.

[59] Gutmann, J. S. et al. “3D Perception and Environment Map Generation for Humanoid Robot Navigation”. In: *The International Journal of Robotics Research* 27.10 (2008), pp. 1117–1134. ISSN: 0278-3649. DOI: 10.1177/0278364908096316.

[60] Gutmann, J. et al. “A Floor and Obstacle Height Map for 3D Navigation of a Humanoid Robot”. In: *IEEE International Conference on Robotics and Automation*. 2005, pp. 1066–1071. ISBN: 0-7803-8914-X. DOI: 10.1109/ROBOT.2005.1570257.

[61] Harada, K. et al. “An Analytical Method on Real-time Gait Planning for a Humanoid Robot”. In: *IEEE/RAS International Conference on Humanoid Robots*. 2004, pp. 640–655. ISBN: 0-7803-8863-1.

[62] Hart, P. et al. “A Formal Basis for the Heuristic Determination of Minimum Cost Paths”. In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107. ISSN: 0536-1567. DOI: 10.1109/TSSC.1968.300136.

[63] Herbert, M. et al. “Terrain mapping for a roving planetary explorer”. In: *IEEE International Conference on Robotics and Automation*. 1989. ISBN: 0-8186-1938-4. DOI: 10.1109/ROBOT.1989.100111.

[64] Hesch, J. A. et al. “Descending-stair detection, approach, and traversal with an autonomous tracked vehicle”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2010, pp. 5525–5531. ISBN: 978-1-4244-6674-0. DOI: 10.1109/IROS.2010.5649411.

[66] Hildebrandt, A.-C. “Autonomous Robots in Unknown and Dynamic Scenarios”. Dissertation. Technische Universität München, 2018.

[72] Hinterstoisser, S. et al. “Model Based Training, Detection and Pose Estimation of Texture-Less 3D Objects in Heavily Cluttered cenes”. In: *Asian Conference on Computer Vision*. 2012, pp. 548–562. ISBN: 978-3-642-37330-5. DOI: 10.1007/978-3-642-37331-2_42.

[73] Hirose, M. et al. “Honda Humanoid Robots Development”. In: *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* 365.1850 (2007), pp. 11–19. DOI: 10.1098/rsta.2006.1917.

[74] Hodgins, J. et al. “Adjusting Step Length for Rough Terrain Locomotion”. In: *IEEE Transactions on Robotics and Automation* 7.3 (June 1991), pp. 289–298. ISSN: 1042-296X. DOI: 10.1109/70.88138.

[75] Hornung, A. et al. “Anytime search-based footstep planning with suboptimality bounds”. In: *IEEE-RAS International Conference on Humanoid Robots*. 2012, pp. 674–679. ISBN: 978-1-4673-1369-8. DOI: 10.1109/HUMANOIDS.2012.6651592.

[76] Hornung, A. et al. “OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees”. In: *Autonomous Robots* 34 (2013), pp. 189–206. DOI: 10.1007/s10514-012-9321-0.

[77] Huang, A. S. et al. “A high-rate, heterogeneous data set from the DARPA Urban Challenge”. In: *The International Journal of Robotics Research* 29.13 (2010), pp. 1595–1601. ISSN: 0278-3649.

[78] Hubicki, C. et al. “ATRIAS: Design and validation of a tether-free 3D-capable spring-mass bipedal robot”. In: *The International Journal of Robotics Research* 35.12 (2016), pp. 1497–1521. ISSN: 0278-3649. DOI: 10.1177/0278364916648388.

[79] *Intel® RealSense™ Technology*. URL: https://www.intel.com/content/www/us/en/architecture-and-technology/realsense-overview.html (visited on 09/11/2017).

[80] Jalobeanu, M. et al. “Reliable Kinect-based Navigation in Large Indoor Environments”. In: *IEEE International Conference on Robotics and Automation*. 2015. DOI: 10.1109/ICRA.2015.7139225.

[81] Jamaluddin, A. Z. et al. “Design and calibration of an Omni-RGB+D Camera”. In: *International Conference on Ubiquitous Robots and Ambient Intelligence*. IEEE, 2016, pp. 386–387. DOI: 10.1109/URAI.2016.7734066.

[82] Kagami, S. et al. “Vision-based 2.5 D terrain modeling for humanoid locomotion”. In: *IEEE International Conference on Robotics and Automation*. 2003, pp. 2141–2146. ISBN: 0780377362. DOI: 10.1109/ROBOT.2003.1241910.

[83] Kagami, S. et al. “Humanoid HRP2-DHRC for autonomous and interactive behavior”. In: *Robotics Research*. Springer, Berlin, Heidelberg, 2007, pp. 103–117. ISBN: 1610-7438. DOI: 10.1007/978-3-540-48113-3_10.

[84] Kajita, S. et al. “Dynamic walking control of a biped robot along a potential energy conserving orbit”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 1990, pp. 789–794. DOI: 10.1109/70.149940.

[85] Kajita, S. et al. “Study of Dynamic Biped Locomotion on Rugged Terrain - Derivation and Application of the Linear Inverted Pendulum Mode”. In: *IEEE International Conference on Robotics and Automation*. 1991, pp. 1405–1411. ISBN: 0-8186-2163-X. DOI: 10.1109/ROBOT.1991.131811.

[86] Kajita, S. et al. “The 3D Linear Inverted Pendulum Mode: A simple modeling for a biped walking pattern generation”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2001, pp. 239–246. ISBN: 0780366123.

[87] Kajita, S. et al. “A Running Controller of Humanoid Biped HRP-2LR”. In: *IEEE International Conference on Robotics and Automation*. 2005, pp. 616–622. ISBN: 0-7803-8914-X.

[88] Kajita, S. et al. *Introduction to Humanoid Robotics*. Springer Publishing Company, Incorporated, 2014. ISBN: 978-3-642-54535-1. DOI: 10.1007/978-3-642-54536-8.

[89] Kaneko, K. et al. “Design of prototype humanoid robotics platform for HRP”. In: *IEEE/RSJ International Conference on Intelligent Robots and System*. 2002, pp. 2431–2436. ISBN: 0-7803-7398-7.

[90] Kaneko, K. et al. “Humanoid robot HRP-2”. In: *IEEE International Conference on Robotics and Automation*. 2004, pp. 1083–1090. ISBN: 0-7803-8232-3.

[91] Karkowski, P. et al. “Real-time footstep planning in 3D environments”. In: *IEEE-RAS International Conference on Humanoid Robots*. 2016, pp. 69–74. ISBN: 9781509047185. DOI: 10.1109/HUMANOIDS.2016.7803256.

[92] Karkowski, P. et al. “Real-time footstep planning using a geometric approach”. In: *IEEE International Conference on Robotics and Automation*. 2016, pp. 1782–1787. ISBN: 978-1-4673-8026-3. DOI: 10.1109/ICRA.2016.7487323.

[93] Keeley, B. “Nonhuman Animal Senses”. In: *The Oxford Handbook of the Philosophy of Perception*. Oxford University Press, 2014. DOI: 10.1093/oxfordhb/9780199600472.013.039.

[94] Kehl, W. et al. “Deep Learning of Local RGB-D Patches for 3D Object Detection and 6D Pose Estimation”. In: *European Conference on Computer Vision*. 2016. DOI: 10.1007/978-3-319-46487-9_13.

[95] Khoshelham, K. et al. “Accuracy and resolution of Kinect depth data for indoor mapping applications”. In: *Sensors* 12.2 (2012), pp. 1437–54. ISSN: 1424-8220. DOI: 10.3390/s120201437.

[96] Kleeman, L. et al. “Sonar Sensing”. In: *Springer Handbook of Robotics*. Ed. by Siciliano, B. et al. Springer Berlin Heidelberg, 2008. Chap. 21, pp. 491–520. ISBN: 978-3-540-23957-4.

[98] Koo, S. et al. “Incremental object learning and robust tracking of multiple objects from RGB-D point set data”. In: *Journal of Visual Communication and Image Representation* 25.1 (2014), pp. 108–121. ISSN: 10473203. DOI: 10.1016/j.jvcir.2013.03.020.

[99] Kugi, a. et al. “On the Passivity-Based Impedance Control of Flexible Joint Robots”. In: *IEEE Transactions on Robotics* 24.2 (Apr. 2008), pp. 416–429. ISSN: 1552-3098.

[100] Kuindersma, S. et al. “An efficiently solvable quadratic program for stabilizing dynamic locomotion”. In: *IEEE International Conference on Robotics and Automation*. 2014, pp. 2589–2594. ISBN: 978-1-4799-3685-4. DOI: 10.1109/ICRA.2014.6907230.

[101] *KUKA youbot*. 2017. URL: http://www.youbot-store.com/ (visited on 09/09/2017).

[102] El-laithy, R. A. et al. “Study on the use of Microsoft Kinect for robotics applications”. In: *IEEE/ION Position, Location and Navigation Symposium*. 2012, pp. 1280–1288. ISBN: 978-1-4673-0387-3. DOI: 10.1109/PLANS.2012.6236985.

[104] Lazaros, N. et al. “Review of Stereo Vision Algorithms: From Software to Hardware”. In: *International Journal of Optomechatronics* 2.4 (2008), pp. 435–462. DOI: 10.1080/15599610802438680.

[105] Li, S. et al. “Real-time and model-free object tracking using particle filter with Joint Color-Spatial Descriptor”. In: *IEEE International Conference on Intelligent Robots and Systems* (2015), pp. 6079–6085. ISSN: 21530866. DOI: 10.1109/IROS.2015.7354243.

[106] Li, Z. et al. “Stabilization for the compliant humanoid robot COMAN exploiting intrinsic and controlled compliance”. In: *IEEE International Conference on Robotics and Automation*. 2012, pp. 2000–2006. ISBN: 978-1-4673-1405-3. DOI: 10.1109/ICRA.2012.6224705.

[107] Lloyd, S. P. “Least Squares Quantization in PCM”. In: *IEEE Transactions on Information Theory* 28.2 (1982), pp. 129–137. ISSN: 15579654. DOI: 10.1109/TIT.1982.1056489.

[108] Loffler, K. et al. “Model Based Control of a Biped Robot”. In: *7th International Workshop on Advanced Motion Control*. IEEE, 2002, pp. 443–448. ISBN: 0-7803-7479-7.

[109] Logan, M. “The coolest stuff you didn’t know google photos could do”. In: *Wired* (Aug. 2015), pp. 1–7. URL: https://www.wired.com/2015/06/coolest-stuff-didnt-know-google-photos/.

[110] Lohmeier, S. “Design and Realization of a Humanoid Robot for Fast and Autonomous Bipedal Locomotion”. PhD thesis. TU München, 2010. ISBN: 9783868537345.

[111] Lohmeier, S. et al. “System design and control of anthropomorphic walking robot LOLA”. In: *IEEE/ASME Transactions on Mechatronics* 14.6 (2009), pp. 658–666. ISSN: 1083-4435. DOI: 10.1109/TMECH.2009.2032079.

[112] Luber, M. et al. “People Tracking in RGB-D Data With On-line Boosted Target Models”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2011, pp. 3844–3849. ISBN: 978-1-61284-455-8.

[113] Madrigal, A. C. *Inside Waymo’s Secret World for Training Self-Driving Cars - The Atlantic*. Aug. 2017. URL: https://www.theatlantic.com/amp/article/537648/.

[114] Maier, D. et al. “Real-time navigation in 3D environments based on depth camera data”. In: *IEEE International Conference on Humanoid Robots*. 2012, pp. 692–697. ISBN: 978-1-4673-1369-8. DOI: 10.1109/HUMANOIDS.2012.6651595.

[115] Maier, D. et al. “Integrated perception, mapping, and footstep planning for humanoid navigation among 3D obstacles”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2013, pp. 2658–2664. ISBN: 9781467363587. DOI: 10.1109/IROS.2013.6696731.

[116] Maier, D. et al. “Vision-Based Humanoid Navigation Using Self-Supervized Obstacle Detection”. In: *International Journal of Humanoid Robotics* 10 (2013), p. 1350016. ISSN: 0219-8436. DOI: 10.1142/S0219843613500163.

[118] McKinsey & Company. *The Future of the North American Automotive Supplier Industry: Evolution of Component Costs, Penetration, and Value Creation Potential Through 2020*. Tech. rep. March. 2012. URL: https://goo.gl/MKx1NF.

[119] Michel, P. et al. “GPU-accelerated Real-Time 3D Tracking for Humanoid Locomotion and Stair Climbing”. In: *IEEE International Conference on Intelligent Robots and Systems*. 2007, pp. 463–469. ISBN: 1424409128. DOI: 10.1109/IROS.2007.4399104.

[120] *Microsoft*. URL: https://www.microsoft.com/ (visited on 09/12/2017).

[121] *Microsoft HoloLens*. URL: https://www.microsoft.com/en-us/hololens (visited on 09/28/2017).

[122] Moravec, H. et al. “High resolution maps from wide angle sonar”. In: *IEEE International Conference on Robotics and Automation*. Vol. 2. 1985. DOI: 10.1109/ROBOT.1985.1087316.

[123] Morio. *Honda prototype robots; P1, P2, P3, and P4 (from left to right). Distributed under the CC BY-SA 3.0 licence.* 2017. URL: https://creativecommons.org/.

[124] Morisawa, M. et al. “A Biped Pattern Generation Allowing Immediate Modification of Foot Placement in Real-time”. In: *IEEE-RAS International Conference on Humanoid Robots*. 2006, pp. 581–586. ISBN: 1-4244-0199-2.

[125] Morisawa, M. et al. “Reactive Biped Walking Control for a Collision of a Swinging Foot on Uneven Terrain”. In: *IEEE-RAS International Conference on Humanoid Robotics*. 2011, pp. 768–773. ISBN: 9781612848686.

[126] Mund, J. et al. “Introducing LiDAR Point Cloud-based Object Classification for Safer Apron Operations”. In: *International Symposium on Enhanced Solutions for Aircraft and Vehicle Surveillance Applications*. Berlin, 2016. URL: https://goo.gl/28Yoqh.

[127] Mur-Artal, R. et al. “ORB-SLAM: A Versatile and Accurate Monocular SLAM System”. In: *IEEE Transactions on Robotics* 31.5 (2015), pp. 1147–1163. ISSN: 15523098. DOI: 10.1109/TRO.2015.2463671.

[128] Murphy, K. P. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012. ISBN: 0262018020.

[129] Nagatani, K. et al. “Emergency response to the nuclear accident at the Fukushima Daiichi nuclear power plants using mobile rescue robots”. In: *Journal of Field Robotics* 30.1 (2013), pp. 44–63. DOI: 10.1002/rob.21439.

[130] Naveau, M. et al. “A Reactive Walking Pattern Generator Based on Nonlinear Model Predictive Control”. In: *IEEE Robotics and Automation Letters* 2.1 (2017), pp. 10–17. DOI: 10.1109/LRA.2016.2518739.

[131] Nelson, G. et al. “PETMAN: A Humanoid Robot for Testing Chemical Protective Clothing”. In: *Journal of the Robotics Society of Japan* 30.4 (2012), pp. 372–377. ISSN: 0289-1824. DOI: 10.7210/jrsj.30.372.

[132] Newcombe, R. A. et al. “DTAM: Dense tracking and mapping in real-time”. In: *IEEE International Conference on Computer Vision*. 2011, pp. 2320–2327. ISBN: 9781457711015. DOI: 10.1109/ICCV.2011.6126513.

[133] Nishiwaki, K. et al. “High Frequency Walking Pattern Generation based on Preview Control of ZMP”. In: *IEEE-RAS International Conference on Humanoid Robots*. 2006, pp. 2667–2672. ISBN: 0-7803-9505-0.

[134] Nishiwaki, K. et al. “Walking control on uneven terrain with short cycle pattern generation”. In: *IEEE/RAS International Conference on Humanoid Robots*. 2007, pp. 447–453. ISBN: 978-1-4244-1861-9.

[135] Nishiwaki, K. et al. “Frequent walking pattern generation that uses estimated actual posture for robust walking control”. In: *IEEE-RAS International Conference on Humanoid Robots*. 2009, pp. 535–541. ISBN: 978-1-4244-4597-4.

[136] Nishiwaki, K. et al. “Strategies for Adjusting the ZMP Reference Trajectory for Maintaining Balance in Humanoid Walking”. In: *IEEE International Conference on Robotics and Automation*. 2010, pp. 4230–4236. ISBN: 978-1-4244-5038-1. DOI: 10.1109/ROBOT.2010.5510002.

[137] Nishiwaki, K. et al. “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. ISSN: 0278-3649. DOI: 10.1177/0278364912455720.

[138] *Open Graphics Library*. 2017. URL: https://www.opengl.org (visited on 09/20/2017).

[139] *Open Source Computer Vision Library*. 2017. URL: http://opencv.org/ (visited on 09/09/2017).

[140] Oßwald, S. et al. “From 3D Point Clouds to Climbing Stairs: A Comparison of Plane Segmentation Approaches for Humanoids”. In: *IEEE-RAS International Conference on Humanoid Robots*. 2011, pp. 93–98. ISBN: 9781612848679. DOI: 10.1109/Humanoids.2011.6100836.

[141] Park, I.-w. et al. “Mechanical design of humanoid robot platform KHR-3 (KAIST humanoid robot - 3: HUBO)”. In: *IEEE-RAS International Conference on Humanoid Robots*. 2005, pp. 321–326. ISBN: 0-7803-9320-1.

[142] Park, Y. et al. “Texture-Less Object Tracking with Online Training using An RGB-D Camera”. In: *IEEE International Symposium on Mixed and Augmented Reality*. 2011, pp. 121–126. ISBN: 9781457721830. DOI: 10.1109/ISMAR.2011.6092377.

[143] Pham, D. et al. “Selection of K in K -means clustering”. In: *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science* 219.1 (2005), pp. 103–119. ISSN: 0954-4062. DOI: 10.1243/095440605X8298.

[144] *Point Cloud Library*. 2017. URL: http : / / www . pointclouds . org (visited on 09/09/2017).

[145] *Point Cloud Library (PCL) Documentation*. 2017. URL: http://docs.pointclouds.org/trunk/ (visited on 09/09/2017).

[146] Poppinga, J. et al. “Fast Plane Detection and Polygonalization in Noisy 3D Range Images”. In: *IEEE International Conference on Intelligent Robots and Systems*. 2008, pp. 22–26. ISBN: 9781424420582.

[147] Pratt, J. E. et al. “Exploiting Natural Dynamics in the Control of a Planar Bipedal Walking Robot”. In: *36th Annual Allerton Conference on Communication, Control, and Computing*. 1998, pp. 253–262. ISBN: 978-3-540-76133-4. DOI: 10 . 1007 / BFb0035216.

[148] Pratt, J. E. et al. “Velocity-Based Stability Margins for Fast Bipedal Walking”. In: *Fast Motions in Biomechanics and Robotics: Optimization and Feedback Control*. Springer Berlin Heidelberg, 2006, pp. 299–324. ISBN: 978-3-540-36119-0. DOI: 10.1007/978-3-540-36119-0_14.

[149] Pratt, J. et al. “Virtual Model Control: An Intuitive Approach for Bipedal Locomotion”. In: *The International Journal of Robotics Research* 20.2 (2001), pp. 129–143. ISSN: 0278-3649. DOI: 10.1177/02783640122067309.

[150] Pratt, J. et al. “Capture Point: A Step toward Humanoid Push Recovery”. In: *IEEE-RAS International Conference on Humanoid Robots*. 2006, pp. 200–207. ISBN: 1-4244-0199-2.

[151] Quain, J. R. *What Self-Driving Cars See (The New York Times)*. May 2017. URL: https://goo.gl/VtKRe5.

[152] Quigley, M. et al. “ROS: an open-source Robot Operating System”. In: *ICRA Workshop on Open Source Software*. 2009. URL: https://goo.gl/e6QkWP.

[153] Raibert, M. *Legged Robots that Balance*. Cambridge, Massachusetts: The MIT Press series in artificial intelligence, 1986. ISBN: 0262181177.

[154] Raibert, M. et al. “Bigdog, the rough-terrain quadruped robot”. In: *17th World Congress of The International Federation of Automatic Control*. 2008, pp. 10822–10825. ISBN: 9781123478.

[155] Ramos, O. E. et al. “Toward Reactive Vision-Guided Walking on Rough Terrain: An Inverse-Dynamics Based Approach”. In: *International Journal of Humanoid Robotics* 11.02 (June 2014), p. 1441004. ISSN: 0219-8436.

[156] *Robot Operating System*. 2017. URL: http://www.ros.org/ (visited on 09/20/2017).

[157] Royer, E. et al. "Monocular Vision for Mobile Robot Localization and Autonomous Navigation". In: *International Journal of Computer Vision* 74.3 (Sept. 2007), pp. 237–260. ISSN: 1573-1405. DOI: 10.1007/s11263-006-0023-y.

[158] Rusu, R. B. "Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments". PhD thesis. Technische Universität München, 2009. DOI: 10.1007/s13218-010-0059-6.

[159] Rusu, R. B. et al. "Fast 3D Recognition and Pose Using the Viewpoint Feature Histogram". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2010, pp. 2155–2162. ISBN: 9781424466757. DOI: 10.1109/IROS.2010.5651280.

[160] Rusu, R. B. et al. "3D is here: Point Cloud Library (PCL)". In: *IEEE International Conference on Robotics and Automation*. 2011, pp. 1–4. ISBN: 9781612843865. DOI: 10.1109/ICRA.2011.5980567.

[161] Sakagami, Y. et al. "The intelligent ASIMO: System overview and integration". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2002, pp. 2478–2483. ISBN: 0-7803-7398-7.

[162] Schmidhuber, J. "Deep learning in neural networks: An overview". In: *Neural Networks* 61 (2015), pp. 85–117. DOI: 10.1016/j.neunet.2014.09.003.

[163] Schütz, C. et al. "A modular robot system for agricultural applications". In: *International Conference of Agricultural Engineering*. Zurich, 2014, pp. 6–10.

[164] Schwienbacher, M. "Efficient Algorithms for Biped Robots". PhD thesis. Technische Universität München, 2014.

[165] Schwienbacher, M. et al. "Self-collision avoidance and angular momentum compensation for a biped humanoid robot". In: *IEEE International Conference on Robotics and Automation*. 2011, pp. 581–586. ISBN: 978-1-61284-386-5. DOI: 10.1109/ICRA.2011.5980350.

[166] Shim, I. et al. "Vision System and Depth Processing for DRC-HUBO +". In: *IEEE International Conference on Robotics and Automation*. 2016, pp. 2456–2463. ISBN: 9781467380256.

[167] Siciliano, B. et al. *Springer Handbook of Robotics*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007. ISBN: 354023957X.

[168] SoftBank. *Boston Dynamics*. 2017. URL: https://www.bostondynamics.com/ (visited on 10/25/2017).

[169] SoftBank. *Boston Dynamics' Spot Navigation*. 2017. URL: https://youtu.be/R8mHEELQmZ8 (visited on 08/08/2017).

[170] SparkFunElectronics. *Published under the CC BY 2.0 licence.* URL: https://www.sparkfun.com/ (visited on 09/12/2017).

[171] Sreenath, K. et al. "A Compliant Hybrid Zero Dynamics Controller for Stable, Efficient and Fast Bipedal Walking on MABEL". In: *The International Journal of Robotics Research* 30.9 (2011), pp. 1170–1193. ISSN: 0278-3649. DOI: 10.1177/0278364910379882.

[172] Stumpf, A. et al. "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, pp. 287–294. ISBN: 978-1-4799-7174-9. DOI: 10.1109/HUMANOIDS.2014.7041374.

[173] Sugihara, T. “Standing stabilizability and stepping maneuver in planar bipedalism based on the best COM-ZMP regulator”. In: *IEEE International Conference on Robotics and Automation*. 2009, pp. 1966–1971. ISBN: 978-1-4244-2788-8.

[177] Takenaka, T. et al. “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, pp. 1084–1091. ISBN: 978-1-4244-3803-7.

[178] Takenaka, T. et al. “Real Time Motion Generation and Control for Biped Robot -2nd Report: Running Gait Pattern Generation-”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2009, pp. 1092–1099. ISBN: 978-1-4244-3803-7.

[179] Takenaka, T. et al. “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, pp. 1594–1600. ISBN: 978-1-4244-3803-7.

[180] Takenaka, T. et al. “Real Time Motion Generation and Control for Biped Robot - 4th report: Integrated Balance Control-”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2009, pp. 1601–1608. ISBN: 978-1-4244-3803-7.

[181] Tateno, K. et al. “CNN-SLAM: Real-time dense monocular SLAM with learned depth prediction”. In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2017. arXiv: 1704.03489.

[182] Tesla Motors Inc. *Tesla Self-Driving Demonstration*. 2016. URL: https://www.tesla.com/videos/autopilot-self-driving-hardware-neighborhood-long.

[183] Thaler, L. et al. “Mouth-clicks used by blind expert human echolocators – signal description and model based signal synthesis”. In: *PLOS Computational Biology* 13.08 (2017), pp. 1–17. DOI: 10.1371/journal.pcbi.1005670.

[184] Ulbrich, H. et al. “Design and Realization of a Redundant Modular Multipurpose Agricultural Robot”. In: *Proceedings of the XVII International Symposium on Dynamic Problems of Mechanics - ISSN 2316-9567*. Natal, Brazil, 2015.

[186] Vanillase. *Asimo at a Honda factory. Distributed under the CC BY-SA 3.0 licence.* 2017. URL: https://creativecommons.org/.

[187] Vukobratovic, M. et al. “On The Stability of Anthropomorphic Systems”. In: *Mathematical Bioscience* 15 (1972), pp. 1–37. DOI: 10.1016/0025-5564(72)90061-2.

[193] Wang, C.-C. “Extrinsic calibration of a vision sensor mounted on a robot”. In: *IEEE Transactions on Robotics and Automation* 8.2 (Apr. 1992), pp. 161–175. ISSN: 1042296X. DOI: 10.1109/70.134271.

[194] *Waymo*. URL: https://waymo.com/ (visited on 09/12/2017).

[195] Westervelt, E. et al. “Hybrid zero dynamics of planar biped walkers”. In: *IEEE Transactions on Automatic Control* 48.1 (2003), pp. 42–56. ISSN: 0018-9286. DOI: 10.1109/TAC.2002.806653.

[196] Whitney, D. “Resolved Motion Rate Control of Manipulators and Human Prostheses”. In: *IEEE Transactions on Man Machine Systems* 10.2 (June 1969), pp. 47–53. ISSN: 0536-1540.

[197] Wieber, P.-B. “Trajectory Free Linear Model Predictive Control for Stable Walking in the Presence of Strong Perturbations”. In: *IEEE-RAS International Conference on Humanoid Robots*. 2006, pp. 137–142. ISBN: 1-4244-0199-2.

[198] *Willow Garage PR2*. 2016. URL: https://www.willowgarage.com/pages/pr2/overview (visited on 11/20/2016).

[200] Wittmann, R. “Robust Walking Robots in Unknown Environments”. Dissertation. Technische Universität München, 2017.

[204] Wöhler, C. *3D Computer Vision: Efficient Methods and Applications*. Springer London, 2013, pp. 357–362. ISBN: 978-1-4471-4150-1.

[206] Yoshida, E. et al. “Task-driven Support Polygon Reshaping for Humanoids”. In: *IEEE-RAS International Conference on Humanoid Robots*. 2006, pp. 208–213. ISBN: 1-4244-0199-2. DOI: 10.1109/ICHR.2006.321386.

[207] Yoshiike, T. et al. “Development of Experimental Legged Robot for Inspection and Disaster Response in Plants”. In: *IEEE International Conference on Intelligent Robots and Systems*. 2017, pp. 4869–4876. ISBN: 9781538626818. DOI: 978-1-5386-2681-8/17.

[209] *ZED - Depth Sensing and Camera Tracking*. 2017. URL: https://www.stereolabs.com/zed/specs/ (visited on 09/07/2017).

[210] Zhang, Z. “Microsoft Kinect Sensor and Its Effect”. In: *IEEE Multimedia* 19.2 (Feb. 2012), pp. 4–10. ISSN: 1070-986X. DOI: 10.1109/MMUL.2012.24.

Publications by the Author

[65] Hildebrandt, A. C. et al. “Real-Time Predictive Kinematic Evaluation and Optimization for Biped Robots”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2016, pp. 5789–5796. ISBN: 9781509037629. DOI: 10.1109/IROS.2016.7759852.

[67] Hildebrandt, A.-C. et al. “Real-time 3D collision avoidance for biped robots”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2014, pp. 4184–4190. ISBN: 9781479969340. DOI: 10.1109/IROS.2014.6943152.

[68] Hildebrandt, A.-C. et al. “Real-Time Pattern Generation Among Obstacles for Biped Robots”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2015, pp. 2780–2786. ISBN: 9781479999941. DOI: 10.1109/IROS.2015.7353759.

[69] Hildebrandt, A.-C. et al. “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, pp. 53–59. ISBN: 978-1-5090-2255-7. DOI: 10.1109/ICARSC.2016.15.

[70] Hildebrandt, A.-C. et al. “Real-Time Path Planning in Unknown Environments for Bipedal Robots”. In: *IEEE Robotics and Automation Letters* 2.4 (2017), pp. 1856–1863. DOI: 10.1109/LRA.2017.2712650.

[71] Hildebrandt, A.-C. et al. “Versatile and Robust Bipedal Walking in Unknown Environments (submitted)”. In: *Autonomous Robots* (2018).

[174] Sygulla, F. et al. “A Flexible and Low-Cost Tactile Sensor for Robotic Applications”. In: *IEEE International Conference on Advanced Intelligent Mechatronics*. 2017. ISBN: 9781509059980. DOI: 10.1109/AIM.2017.8013995.

[175] Sygulla, F. et al. “Hybrid Position / Force Control for Biped Robot Stabilization with Integrated Center of Mass Dynamics”. In: *IEEE-RAS International Conference on Humanoid Robots*. 2017. ISBN: 9781538646779. DOI: 10.1109/HUMANOIDS.2017.8246955.

[176] Sygulla, F. et al. “An EtherCAT-Based Real-Time Control System Architecture for Humanoid Robots”. In: *IEEE International Conference on Automation Science and Engineering*. 2018. URL: https://mediatum.ub.tum.de/1394924.

[188] Wahrmann, D. et al. “Fast Object Approximation for Real-Time 3D Obstacle Avoidance with Biped Robots”. In: *IEEE International Conference on Advanced Intelligent Mechatronics*. 2016, pp. 38–45. ISBN: 9781509020645. DOI: 10.1109/AIM.2016.7576740.

[189] Wahrmann, D. et al. “An Autonomous and Flexible Robotic Framework for Logistics Applications”. In: *Journal of Intelligent & Robotic Systems* (Dec. 2017). ISSN: 1573-0409. DOI: 10.1007/s10846-017-0746-8.

[190] Wahrmann, D. et al. “Modifying the Estimated Ground Height to Mitigate Error Effects on Bipedal Robot Walking”. In: *IEEE International Conference on Advanced Intelligent Mechatronics*. 2017. DOI: 10.1109/AIM.2017.8014226.

[191] Wahrmann, D. et al. “Time-variable, event-based walking control for biped robots”. In: *International Journal of Advanced Robotic Systems* 15.2 (2018). DOI: 10.1177/1729881418768918.

[192] Wahrmann, D. et al. “Vision-Based 3D Modeling of Unknown Dynamic Environments for Real-Time Humanoid Navigation (submitted)”. In: *International Journal of Humanoid Robotics* (2018). URL: https://mediatum.ub.tum.de/1435457.

[201] Wittmann, R. et al. “Real-Time Nonlinear Model Predictive Footstep Optimization for Biped Robots”. In: *IEEE-RAS International Conference on Humanoid Robots*. 2015, pp. 711–717. ISBN: 9781479968855. DOI: 10.1109/HUMANOIDS.2015.7363432.

[202] Wittmann, R. et al. “State Estimation for Biped Robots Using Multibody Dynamics”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2015, pp. 2166–2172. ISBN: 9781479999934. DOI: 10.1109/IROS.2015.7353667.

[203] Wittmann, R. et al. “Model-Based Predictive Bipedal Walking Stabilization”. In: *IEEE-RAS International Conference on Humanoid Robots*. 2016. ISBN: 9781509047178. DOI: 10.1109/HUMANOIDS.2016.7803353.

Supervised Student Theses

[9] Bates, T. “Augmented Reality for Humanoid Robots”. Interdisciplinary Project. Technische Universität München, 2016.

[12] Bräu, F. “Onlineberechnung von Kollisionsgeometrien”. Semester Thesis. Technische Universität München, 2014.

[20] Buttner, C. “Position and Velocity Estimation of Obstacles for Humanoid Robots”. Interdisciplinary Project. Technische Universität München, 2016.

[50] Floeren, S. “Real-time environment recognition for biped walking”. Interdisciplinary Project. Technische Universität München, 2017.

[57] Grundner, C. “Teleoperation eines Service-Satelliten mit einem Roboterarm”. Bachelor’s Thesis. Technische Universität München, 2016.

[58] Gutermuth, D. et al. “Vision-based environment classification for humanoid robotic navigation”. Interdisciplinary Project. Technische Universität München, 2016.

[97] Knopp, T. “Auswirkungen von unsicherer Bodenerkennung auf zweibeinige Roboter”. Bachelor’s Thesis. Technische Universität München, 2016.

[103] Lalić, M. “Vision-based Fine-grain Approximation of Objects for Humanoid Robotic Navigation”. Interdisciplinary Project. Technische Universität München, 2015.

[117] Makhani, A. “Simulated point clouds for the evaluation of 3D Computer vision algorithms”. Interdisciplinary Project. Technische Universität München, 2016.

[185] Uygur, I. “Vision Based Detection of Surfaces for Humanoid Robotic Navigation”. Interdisciplinary Project. Technische Universität München, 2015.

[199] Wittmann, J. “Autonome Robotik - Reaktive Pfadplanung in der Landwirtschaftsrobotik”. Semester Thesis. Technische Universität München, 2017.

[205] Wu, Y. “Control of a humanoid robot over unexpected floor heights”. Semester Thesis. Technische Universität München, 2017.

[208] Yousefpour, S. “3D Image Processing for a Harvesting Manipulator”. Interdisciplinary Project. Technische Universität München, 2015.