University of Duisburg-Essen
Faculty of Engineering
Chair of Mechatronics

Highly-Dynamic Movements of a Humanoid Robot Using Whole-Body Trajectory Optimization

Master Thesis

Maschinenbau (M.Sc.)

Julian Eßer
Student ID: [redacted]

First examiner	Prof. Dr. Dr. h.c. Frank Kirchner (DFKI)
Second examiner	Dr.-Ing. Tobias Bruckmann (UDE)
Supervisor	Dr. rer. nat. Shivesh Kumar (DFKI)
Supervisor	Dr. Carlos Mastalli (University of Edinburgh)
Supervisor	Dr. Olivier Stasse (LAAS-CNRS)

September 18, 2020

UNIVERSITÄT DUISBURG ESSEN
Offen im Denken

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH
Robotics Innovation Center

Declaration

This master's thesis was carried out at the Robotics Innovation Center of the German Research Center for Artificial Intelligence (DFKI) in the Advanced AI Team on Mechanics & Control under the direction of Dr. Shivesh Kumar.

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

Bremen, September 18, 2020

Julian Eßer

Abstract

Motion planning for legged robots is a challenging problem and remains an open area of research. Particular difficulties arise from effective underactuation, the mechanism complexity, as well as nonlinear and hybrid dynamics. A common approach is to decompose this problem into smaller sub-problems that are solved sequentially. Recent research indicates that using a local optimal control solver, namely Differential Dynamic Programming (DDP), produces more efficient motions, with lower forces and impacts.

This master's thesis contributes in this direction by applying, evaluating and extending DDP-based whole-body trajectory optimization, pursuing three objectives. First, we develop a method for constraining DDP-like solvers in order to generate inherently balanced motion plans. Second, the proposed motion planning approach is evaluated for quasi-static and dynamic motions in a real-time physics simulation and in real-world experiments on the lightweight and biologically inspired RH5 humanoid robot. Finally, the limits of the approach and the system design are examined by solving highly-dynamic movements.

Keywords: Differential Dynamic Programming, Dynamic Bipedal Walking, Humanoid Robots, Motion Planning, Multi-Contact Optimal Control, Whole-Body Trajectory Optimization

Kurzfassung

Die Bewegungsplanung für Laufroboter ist eine anspruchsvolle Aufgabe und bleibt ein aktives Forschungsgebiet. Besondere Schwierigkeiten ergeben sich aus der effektiven Unteraktuierung, der Komplexität der Mechanismen sowie der nichtlinearen und hybriden Dynamik. Ein verbreiteter Ansatz besteht darin, das Problem in kleinere Teilprobleme zu zerlegen, die nacheinander gelöst werden. Neue Forschungsergebnisse zeigen, dass Algorithmen aus dem Bereich der optimalen Regelung, insbesondere der Differenziellen Dynamischen Programmierung (DDP), effizientere Bewegungen mit geringeren Kräften und Stößen erzeugen.

Diese Masterarbeit leistet einen Beitrag in diese Richtung, indem eine DDP-basierte Ganzkörper-Trajektorienoptimierung angewendet, bewertet und erweitert wird, wobei drei Ziele verfolgt werden. Erstens entwickeln wir eine Methode zur Beschränkung von DDP-artigen Algorithmen, um stabile Bewegungspläne zu erzeugen. Zweitens wird der vorgeschlagene Ansatz zur Bewegungsplanung für quasistatische und dynamische Bewegungen in einer physikalischen Echtzeitsimulation und in realen Experimenten mit dem leichten und biologisch inspirierten humanoiden Roboter RH5 evaluiert. Schließlich werden die Grenzen des Ansatzes und des Systemdesigns anhand hochdynamischer Bewegungen untersucht.

Stichworte: Bewegungsplanung, Differenzielle Dynamische Programmierung, Dynamisches Bipedales Laufen, Ganzkörper-Trajektorienoptimierung, Humanoide Roboter, Optimale Regelung

Acknowledgments

I would like to thank my first supervisor and institute director Prof. Frank Kirchner for giving me the opportunity to work on this interesting topic at the DFKI Robotics Innovation Center. Then, I would like to thank my second supervisor Dr. Tobias Bruckmann for providing his continuous guidance and constructive feedback during this thesis.

I would like to express my deepest gratitude to my main mentor and team leader Shivesh for his guidance, the numerous inspiring discussions we had throughout the course of this work, and his infectious enthusiasm for everything related to dynamics and many more surrounding topics. Special thanks also goes to Carlos and Olivier who supported me with many ideas and critical but constructive questions related to numerical optimization and humanoid robotics, respectively.

Finally, I want to also thank all my friends I met along the way, in the Lower Rhine area, Duisburg/Essen, Dortmund, Oberpfaffenhofen, Bremen and elsewhere. You have each made this journey an unforgettable experience. Last but not least, I want to thank my family (especially my mother) and my girlfriend Monique (one of the bravest person I ever met) for your supportive stronghold and I am forever grateful to have you in my life.

Julian

September 18, 2020

Contents

Acronyms

CoM	Center of Mass
CoP	Center of Pressure
DDP	Differential Dynamic Programming
DoF	Degrees of Freedom
DS	Double Support
EoM	Equations of Motion
FCoM	Floor Projection of Center of Mass
FD	Forward Dynamics
FDDP	Feasibility-driven Differential Dynamic Programming
ID	Inverse Dynamics
IK	Inverse Kinematics
KKT	Karush-Kuhn-Tucker
LF	Left Foot
LIP	Linear Inverted Pendulum
MPC	Model Predictive Control
OC	Optimal Control
RF	Right Foot
SP	Support Polygon
SQP	Sequential Quadratic Programming
TO	Trajectory Optimization
ZMP	Zero-Moment Point

CHAPTER 1

Introduction

This introduction guides the reader to the goal of the master's thesis. Beginning with a motivation on humanoid robots a general problem statement is derived. Thereupon, two approaches are presented for solving the motion planning problem as well as the used framework and experimental platform. Building upon this related work, the specific objectives of the thesis are defined and a brief overview of the structure is provided.

1.1. Motivation

Robotics research is highly motivated by the idea of creating machines with the ability to autonomously explore and interact with complex and dynamic environments. These intelligent agents can act in surroundings that are either inaccessible or dangerous to humans or support us in everyday life tasks.

The key promise of using legs for locomotion is the improved mobility over wheeled systems. Significant advantages are gained due to the ability of using isolated footholds and active suspension, which effectively decouples the main body from the roughness of the environment and allows e.g. to step over obstacles [1]. These benefits come at the expense of a significant increase in complexity since there is a need of ongoing, active balancing of the robot in order to avoid falling down [2].

Nature often plays a crucial rule and serves as source of inspiration in the design process of such systems. This is especially true for the research field of humanoid robots, which deals with robots that are generally inspired by human capabilities and share similar kinematics, sensing and behavior. Many of the objects that we interact with on a daily basis, are tailored to human form and human behavior, e.g. doors, stairs or tools. The same also applies to the environment in which we move around. Humans make use of their legs to climb stairs, lean towards difficult

Figure 1.1.: Humanoid robots can interact with humans in a very intuitive manner since they share similar kinematics, sensing and behavior. This opens up new opportunities for the cooperation needed, e.g. when assembling and installing infrastructure on foreign planets [4].

postures or traverse rough terrain [3]. These capabilities of humanoid robots have the potential to benefit mankind. Walking robot nurses would have the ability to freely move around and interact with the elderly. In disaster scenarios, humanoids could be sent to check for rescue persons. When exploiting foreign planets, humanoid robots have the ability to collaborate with humans in an intuitive and effective way (see Fig. 1.1).

We perform all these tasks seemingly effortless. But compared to current robots, the dynamic capabilities of humans and animals are still outstanding in terms of versatility, speed, efficiency and robustness [5]. In order to further close the gap between robots and their natural counterparts, current research is driving towards exploiting the natural dynamics of robots. This implies mutually dependent changes about how we think about the robots design [6] on the one hand and about ways of controlling them, on the other hand, in order to move in a more dynamic, efficient and natural way [7, 8, 9].

The specific difficulty of creating bipedal walking motions has several causes. The first difficulty is due to the mechanism complexity, i.e. dealing with high-dimensional degrees of freedom, leading to potentially expensive computations. Secondly, legged locomotion is subject to different contact situations and impact collisions, resulting in multi-phase models and hence hybrid dynamics. Finally, bipeds face the problem of effective underactuation, further restricting the applicable control approaches.

Dynamic bipedal locomotion adds additional complexity to this problem. A central characteristic of walking dynamically is that the Center of Mass (CoM) partially leaves the biped's support polygon. Furthermore, there is the need of generating feasible, controllable limit cycles. When considering running motions, difficulties are faced regarding the conservation of angular momentum, which implies restrictions on the controllability during flight phases [10].

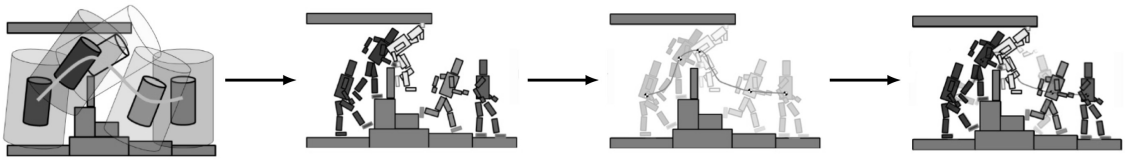


Figure 1.2.: Traditional Legged Locomotion Planning [12]. The motion planning problem is decomposed into subproblems that are solved sequentially: first, a contact planner solves for feasible foothold positions, then a feasible base motion is generated and finally a consistent whole-body trajectory is derived.

1.2. Related Work

There are existing different ways to generate feasible motion plans for robotic systems. Common approaches are relying on simplified dynamic models, while recent ones make use of numerical optimization for generating efficient motion plans.

1.2.1. Traditional Legged Locomotion Planning

Previously we already discovered, why locomotion synthesis for legged robots is a challenging problem. Solving this global problem typically is approached by splitting it up into successive subproblems that are solved sequentially: (i) contact planner, (ii) centroidal pattern generator, (iii) whole-body motion generator (Fig. 1.2) [11].

The first stage of traditional legged locomotion planning is a contact planner that selects appropriate foothold position and step timings. In other words, this component predefines *where* and *when* external forces can act on the system. Once these force locations have been predetermined, the goal is to generate a body motion, i.e. a CoM and end-effector trajectories, which can be created with the available forces. A common approach is to model the robot as a Linear Inverted Pendulum (LIP) and find a motion where the Center of Pressure (CoP) remains inside the Support Polygon (SP). The LIP can be solved analytically and efficiently and consequently has been used in a variety of approaches [13, 14, 15, 16]. From the contact sequence and the centroidal trajectory, a dynamic-physical whole-body trajectory has to be found. This often is done by solving the Inverse Kinematics (IK) [17] or operational-space Inverse Dynamics (ID) [18] of the system and produces appropriate joint positions or torques, respectively that can be applied to a physical system.

This approach is beneficial in terms of computation time but comes at the cost of limited motion complexity and energy efficiency. The first limitation comes from the underlying assumptions of the LIP model. More complex motions and terrains might require to place feet at different heights, reorientation of the base or jumping vertically [19]. Another limitation arises from the fact that whole-body planning produces more efficient motions than a simple program such as an IK solver [20].

1.2.2. Trajectory Optimization

Generating motion plans for dynamic systems is subject to Trajectory Optimization (TO) algorithms that belong to the broader research field of Optimal Control (OC).

TO is a numerical optimization technique with the goal of finding a state-control sequence, which locally minimizes a predefined cost function given a set of constraints. This approach allows to specify the behavior of a robot directly in the task space, e.g. a desired end-effector trajectory, while reducing the amount of hand-crafted components.

There are existing different methods to formulate a TO, namely *direct* and *indirect* methods. Unlike *direct* methods which explicitly represent the state, *indirect* methods only represent the control inputs while the state is obtained from forward simulation of the system. In direct methods, the OC problem is transcribed into a Sequential Quadratic Programming (SQP), which easily handles both equality and inequality constraints and is implemented in generic off-the-shelf solvers. Consequently, the direct approaches are forced to search in a constrained optimization space which is slower but finds better optima. The indirect approaches are more sensitive to local minima but are faster and better suited for warm-starting. For a comprehensive overview and further information on TO methods see [21, 22, 23].

TO based on reduced centroidal dynamics [24] has become a popular approach in the legged robotics community. In some approaches it is used after planning the contacts [25, 26, 27] while other approaches simultaneously optimize the centroidal trajectory and the contacts [28, 29, 30]. Either way, the transfer from centroidal to whole-body dynamics is only achieved by instantaneous feedback linearization, where typically quadratic programs with task-space dynamics are solved [31, 32, 33].

While TO based on reduced dynamics models has shown great experimental results, whole-body TO instead is proven to produce more efficient motions, with lower forces and impacts [20]. Hence, we will focus on *indirect methods*, namely Differential Dynamic Programming (DDP) to compute the whole-body motion. DDP allows to efficiently solve nonlinear OC problems due to its intrinsic sparse structure and is introduced in Sections 2.3 to 2.4 in more detail.

1.2.3. Crocoddyl Framework

Crocoddyl (Contact RObot COntrol by Differential Dynamic Library) is a recently presented open-source framework for efficient multi-contact optimal control [34], which is used within this thesis to plan whole-body motions.

The framework allows to efficiently compute optimal robot trajectories with predefined contact phases. Its solver is based on various efficient DDP-like algorithms. Along with Crocoddyl, a novel optimal control algorithm called Feasibility-driven Differential Dynamic Programming (FDDP) is introduced. The FDDP algorithm is an improved version of the classical DDP algorithm, which shows a greater globalization strategy. In the scope of this thesis all presented OC problems are solved with the FDDP algorithm with box-constraints on the control inputs.

Crocoddyl allows the computation of highly-dynamic movements (e.g. jumping, front-flip) within few milliseconds. However, these exemplary case studies do not produce balanced motions, leading to motion plans that are hard to stabilize on a real system [12]. To this end, we present a generic method for constraining DDP-like solvers in order to generate inherently balanced, dynamic motions that are applicable on real robots (see Chapter 3).

1.2.4. RH5 Humanoid Robot

The derived motion planning approach (see Chapter 3) has been tested both in simulation and real-world experiments on a full-size humanoid robot (see Chapters 4 to 6. RH5 is a lightweight and biologically inspired humanoid that has recently been developed at DFKI Robotics Innovation Center [35].

The RH5 humanoid robot (see Fig. 1.3) is designed to mimic the human anatomy with a total size of 200 cm, a weight of 62 kg and a total of 32 Degrees of Freedom (DoF). The two legs account for 12 DoF, the torso and neck kinematics each for three and the arms and grippers of the robot for 14 DoF. In order to achieve a dynamic walking speed of 1 m/s, the robot's design follows a series-parallel hybrid approach [36, Ch.2]. Consequently, linkages and parallel mechanisms are utilized in most of the robot's joints, e.g. the hip-flexion-extension, knee, ankle, torso and wrist. A comparison of RH5 with other state of the art humanoid robots revealed several advantages of this design approach, including increased maximum velocity and torque of the ankle as well as an advantageous weight of the lower leg [37]. Within the analysis of the motion planning approach, the maximum performance of the robot is evaluated in several case studies of highly-dynamic movements (see Chapter 5).

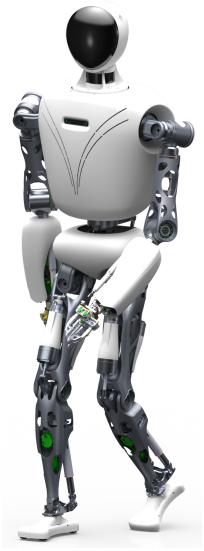


Figure 1.3.: The recently presented RH5 is a lightweight and biologically inspired humanoid used as experimental platform within this thesis.

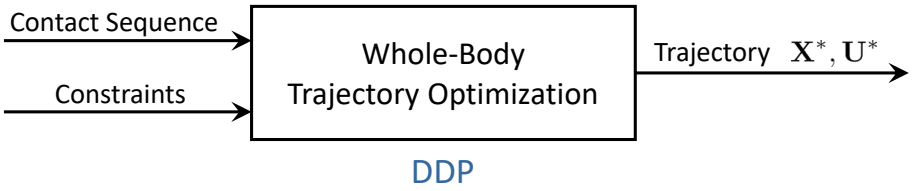


Figure 1.4.: The motion planning approach proposed within this thesis. Based on predefined foothold positions, step timings and stability constraints a DDP-based whole-body TO is used to generate inherently balanced motion plans.

1.3. Contributions

The overall goal of this master's thesis is to contribute to the research field of humanoid robotics by applying, evaluating and extending recently presented whole-body TO approaches on the RH5 humanoid robot.

The proposed motion planning approach (see Fig. 1.4) consists of a DDP-based whole-body TO. Beneath the contact position and timings, it also considers a set of contact stability constraints that allow computing inherently balanced motions. Output of the OC problem is a feasible state trajectory $\boldsymbol{X}^*$ with according optimal control inputs $\boldsymbol{U}^*$. In contrast to many other motion planning concepts followed by the legged robotics community, our approach (i) considers the full robot dynamics instead of using a reduced dynamics model such as the LIP model and (ii) simultaneously optimizes for the centroidal motion, contact stability and whole-body trajectory instead of a sequential optimization.

The specific contributions of this thesis are summarized below.

C1. A method for constraining DDP-like solvers in order to generate inherently balanced dynamic motions. The results are integrated into the open-source framework Crocoddyl.

C2. Evaluation of the stability of the proposed motion planning approach for bipedal walking gaits of increasing complexity in simulation.

C3. An experimental pipeline for executing optimization-based whole-body motions on a series-parallel hybrid robot.

C4. Identification of physical limitations for the RH5 humanoid by performing highly-dynamic movements as basis for future design iterations.

1.4. Structure

This thesis is organized in a total of 7 chapters (see Fig. 1.5). In the following, a summary of each chapter is provided which allows the readers to easily navigate through this document.

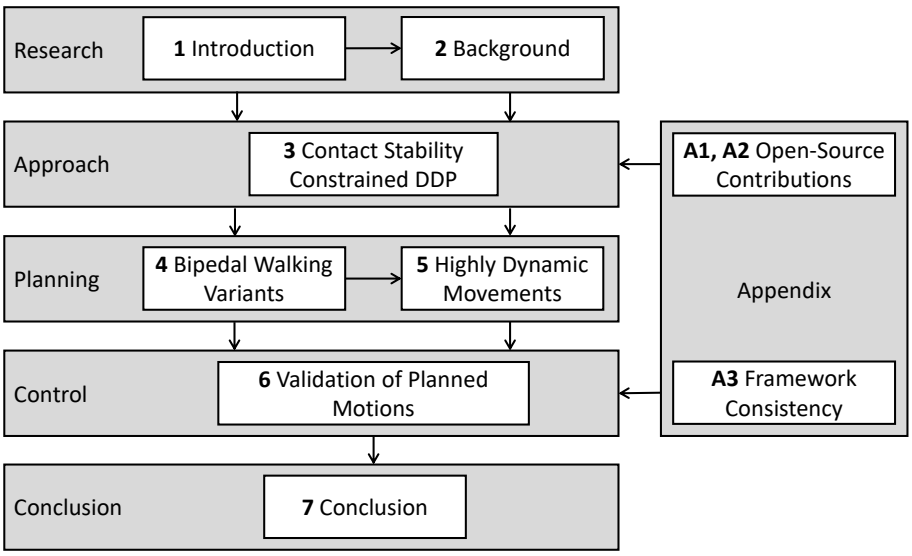


Figure 1.5.: Thesis structure and interconnection of chapters.

Chapter 1 (Introduction) motivates the problem and presents the related work and specific contributions. It also provides details on the structure of the thesis.

Chapter 2 (Mathematical Background) provides the reader with fundamental background in bipedal locomotion and stability analysis. Furthermore, it introduces the class of algorithms and relevant extensions used in this work.

Chapter 3 (Contact Stability Constrained DDP) presents a generic method for integrating stability constraints into DDP-like solvers in order to generate inherently balanced dynamic motions.

Chapter 4 (Bipedal Walking Variants) studies the proposed motion planning approach for bipedal walking gaits of increasing complexity and evaluates the resulting contact stability.

Chapter 5 (Highly-Dynamic Movements) studies the proposed motion planning approach for highly-dynamic movements and evaluates the performance limits of the RH5 humanoid robot.

Chapter 6 (Validation of Planned Motions) validates the physical correctness of the planned motions by stabilizing the OC trajectories with a simple control architecture in a real-time physics simulation and in real-world experiments.

Chapter 7 (Conclusion and Outlook) presents the summary of the thesis and identifies future research directions.

Appendix A.1, A.2 (Open-Source Contributions) contains the implementation of the CoP costs for contact and impulse dynamics, integrated into the open-source framework Crocoddyl within the scope of this thesis.

Appendix A.3 (Framework Consistency) provides a proof for consistency between the used frameworks Pinocchio, Crocoddyl and HyRoDyn.

CHAPTER 2

Mathematical Background: Optimal Bipedal Locomotion

The second chapter provides the reader with fundamentals regarding terminology, modeling and stability analysis in the context of humanoid robotics and presents the class of used algorithms with its relevant extensions used in the context of this thesis.

2.1. Foundations of Bipedal Locomotion

2.1.1. Terminology

In order to describe the locomotion of a humanoid robot, specific terms are required that are introduced within this section. Vukobratović et al. provide an extensive introduction to the terminology related to bipedal walking [38], concisely summarized by Dekker [39].

Walk
Walk can be defined as: "*Movement by putting forward each foot in turn, not having both feet off the ground at once*".

Run
Run is characterized by both feet partially leaving the ground at the same time.

Gait
The way each human walks and runs is unique, hence gait can be defined as: "*Manner of walking or running*".

Periodic gait
If a gait is realized by repeating each locomotion phase in an identical [1] way, the gait is referred to as *periodic*.

Symmetric gait
If the left and right leg move in an identical but time-shifted manner, the gait is referred to as *symmetric*.

Double Support
A situation where the humanoid has two isolated contact surfaces with the ground.

Single Support
A situation where the humanoid has only one contact surface with the ground.

Support Polygon
The support polygon is formed by the *convex hull* about the ground contact points.

Swing foot
This term refers to the leg that is performing a step, i.e. moving through the air.

Supporting foot
This term refers to the leg that is in contact with the ground, supporting all the weight of the humanoid.

2.1.2. Dynamic Modeling of Legged Robots

In the following, the dynamic model for floating base systems, such as legged robots, is derived based on a general formulation. A concise introduction to dynamic modeling is presented with [40], comprehensive studies can be found in [41, 42, 43].

General Formulation

Mathematical models of a robot's dynamics describe the motion as a function of time and control inputs. These models are the basis for both simulation and control of robotic systems. In an abstract form, the Equations of Motion (EoM) can be written as:

$$F(\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t), \ddot{\boldsymbol{q}}(t), \boldsymbol{u}(t), t) = 0, \tag{2.1}$$

where

- $\boldsymbol{t}$ is the time variable,

[1] The locomotion phase can be identical w.r.t. the step size or the duration, depending on the index.

- $\boldsymbol{q}$ is the vector of generalized coordinates,
- $\dot{\boldsymbol{q}}$ is the first time derivative (velocity) of $\mathbf{q}$,
- $\ddot{\boldsymbol{q}}$ is the second time derivative (acceleration) of $\mathbf{q}$and
- $\boldsymbol{u}$ is the vector of control inputs.

Consequently, the EoM provide a mapping between the control space on the one hand and the state space of robot on the other hand. Typical methods for computing the closed-form solution of the EoM are e.g. the classical *Newton-Euler* [44] method or the *Lagrange method* [45], where the former is based on principles for conservation of linear and angular momenta and the latter utilizes energy-based functions expressed in generalized coordinates.

Fixed Base Systems

For applications with fixed-based robots, e.g. a robotic manipulator, the multi-body dynamics can be formulated as

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \dot{\boldsymbol{q}}^T \boldsymbol{C}(\boldsymbol{q})\dot{\boldsymbol{q}} = \boldsymbol{\tau} + \boldsymbol{\tau}_g(\boldsymbol{q}), \tag{2.2}$$

where

- $\boldsymbol{M}(\boldsymbol{q})$ is the generalized inertia matrix,
- $\boldsymbol{C}(\boldsymbol{q})$ is the coriolis tensor,
- $\boldsymbol{\tau}$ is the vector of actuated joint torques and
- $\boldsymbol{\tau}_g(\boldsymbol{q})$ is the vector of external joint torques caused by gravity.

In contrast to the general formulation in Eq. (2.1), this expression is time-invariant. Hence, Eq. (2.2) can be used for computing the Forward Dynamics (FD), as well as the ID of a robotic system.

Implicit and Explicit Constraints

The motion of a robot is constrained by it's kinematics structure as well as additional constraints that may arise from the environment. Fig. 2.1 provides a classification of kinematic constraints, which are often found in the field of robotics [36, Ch.3]. While equality constraints arise from permanent physical contact between two bodies, inequality constraints arise due to phenomena such as collision, bouncing or loss of contact. Equality constraints can be further divided into holonomic and non-holonomic constraints. The former constrain the position (e.g. fixed or sliding contacts), the latter constrain the velocity (e.g. rolling contacts) of multi-body systems.

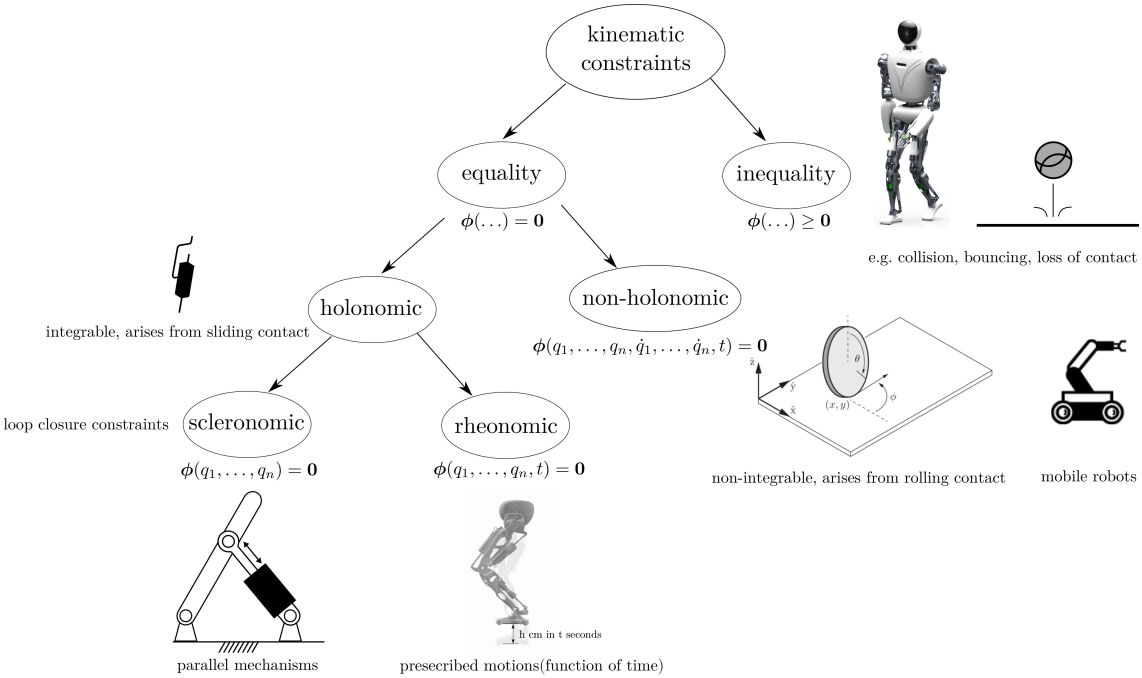


Figure 2.1.: Classification of kinematic constraints in multi-body systems [36].

In the scope of this thesis, we focus on mobile robots that contain parallel mechanisms. In contrast to serial manipulators, two additional scleronomic constraints have to be actively enforced for this type of robotic systems:

- Internal loop closure constraints,
- Contact constraints.

The contact constraints are modeled explicitly as holonomic scleronomic constraints in the robot dynamics (see Section 2.4.1). Since the dynamics solvers inside Crocoddyl do not allow computation for series-parallel mechanisms, a serialized robot model is used as basis of the OC problem. Hence, the closed loop constraints are only implicitly considered within the control architecture (see Section 6.2.1). Although the usage of this simplified model reduces the accuracy, it is proven to be sufficient for dynamic real-time control [46].

Floating Base Systems

As previously mentioned, the focus of this thesis is on mobile robots. These so-called floating base systems are characterized by having a base that is free to move, rather than being fixed in space. Consequently, the vector of generalized coordinates $\boldsymbol{q}$ not only contains the joint's angles, but also accounts for the position and orientation of the floating base. Legged robots belong to this category of rigid-body systems as they make and break contacts with their environment in order to move. Contrary to manipulators, contacts need to be actively enforced by holonomic constraints for

legged robots. There are namely two different types of contact constraints that can be applied: point contacts (3D) or surface contacts (6D).

For the case of point contacts, the dynamics of the floating base system become

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \dot{\boldsymbol{q}}^T \boldsymbol{C}(\boldsymbol{q})\dot{\boldsymbol{q}} = \boldsymbol{S}^T\boldsymbol{\tau} + \boldsymbol{\tau}_g(\boldsymbol{q}) + \sum_{i=1}^{k} \boldsymbol{J}_{C_i}^T \boldsymbol{f}_i,$$

where

- $\mathbf{S}$ is the selection matrix of actuated joints,
- $\boldsymbol{J}_{C_i}$ is the Jacobian at the location of a contact point C_i and
- $\boldsymbol{f}_i$ is the contact force acting at the contact point C_i.

For the case of surface contacts, such as a flat foot on a flat floor, modeling a point contact is not sufficient since it only constrains the translation. In order to also account for the rotational constraints enforced by the geometry one could take into account multiple point contacts. A non-redundant alternative is to model more general frame contact constraints as

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \dot{\boldsymbol{q}}^T \boldsymbol{C}(\boldsymbol{q})\dot{\boldsymbol{q}} = \boldsymbol{S}^T\boldsymbol{\tau} + \boldsymbol{\tau}_g(\boldsymbol{q}) + \sum_{i=1}^{k} \boldsymbol{J}_{C_i}^T \boldsymbol{w}_i, \tag{2.3}$$

where $\boldsymbol{w}_i$ is referred to as the *contact wrench* acting on the contact link i. This wrench stacks the resulting $\boldsymbol{f}_i$ of contact forces and the moment $\boldsymbol{\tau}_i$ exerted by these forces around the contact frame as

$$\boldsymbol{w}_i = (\boldsymbol{f}_i, \boldsymbol{\tau}_i)_{6\times 1}.$$

For more details on contact wrenches and spatial vector algebra in general, the interested reader is referred to e.g. [43, Ch.2].

2.2. Stability Analysis: Not Falling Down

Humanoid robots are high-dimensional, constrained and nonlinear dynamical systems. In this section, the most common criteria for analyzing the long-term stability behavior of such complex systems are presented. Exhaustive studies on stability criteria and their relation can be found in [39, 47, 48].

2.2.1. Static Stability Criteria

Floor Projection of the Center of Mass (FCoM)

Consider the case of a robot that is not moving, i.e. a humanoid in static double support. In that case, the only forces acting on the humanoid are the ones caused

by gravity. These forces can be represented by a virtual force acting on the CoM of the robot. The position of the CoM w.r.t. the base frame can be described by

$$\boldsymbol{p}_{\mathrm{CoM}} = \frac{\sum_{i=1}^{n} m_i \boldsymbol{p}_i}{\sum_{i=1}^{n} m_i},$$

where the robot has n links and $\boldsymbol{p}_i$ indicate the according link distances of the individual CoMs. The Floor Projection of Center of Mass (FCoM) equals the first two components of the CoM position vector $\boldsymbol{p}_{CoM}$ and the following relation holds:

$$\sum_{i=1}^{n}((\boldsymbol{p}_{\mathrm{FCoM}} - \boldsymbol{p}_i) \times m_i \boldsymbol{g}) = \boldsymbol{0}.$$

The FCoM can be used as a static stability margin, ensuring the motionless robot will not tip over or fall, if $\boldsymbol{p}_{\mathrm{FCoM}}$ always remains inside the SP. Note that this criteria is also applicable in so called *quasi-static* movements, where static forces are still dominating dynamic forces.

2.2.2. Dynamic Stability Criteria

In case of faster motions, dynamic forces will exceed the static forces and cannot be neglected anymore. The acting forces can be divided into contact forces and gravity/inertial forces, where the so called Zero-Moment Point (ZMP) is based on the former, and the CoP on the latter. In the following, both concepts are introduced according to the description in [49] with a nomenclature equivalent to [40].

Center of Pressure (CoP)

The CoP is defined as the point, where the field of pressure forces acting on the sole is equivalent to a single resulting force where the resulting moment is zero. Hence the CoP is a local quantity that is derived from the interaction forces at the contact surface.

Considering the case of a foot contacting a plane surface, the resulting contact force $\boldsymbol{f}^c$ is exerted by the environment onto the robot. This force consists of the resulting pressure force $\boldsymbol{f}^p = (\boldsymbol{f}^c \cdot \boldsymbol{n})\boldsymbol{n}$, as well as the resulting friction force $\boldsymbol{f}^f = \boldsymbol{f}^c - \boldsymbol{f}^p$. Hence, the following conditions hold:

$$\begin{aligned} \boldsymbol{\tau}_O^p &= \boldsymbol{0} \\ \boldsymbol{p}_{\mathrm{CoP}} \times (\boldsymbol{f}^p \cdot \boldsymbol{n})\boldsymbol{n} &= -\boldsymbol{\tau}_O^P \\ (\boldsymbol{f}^p \cdot \boldsymbol{n})\boldsymbol{n} \times \boldsymbol{p}_{\mathrm{CoP}} \times \boldsymbol{n} &= -\boldsymbol{n} \times \boldsymbol{\tau}_O^p \end{aligned}$$

Since both the sole point O and $\boldsymbol{p}_{\mathrm{CoP}}$ belong to the same plane, we get:

$$\boldsymbol{p}_{\mathrm{CoP}} = \frac{\boldsymbol{n} \times \boldsymbol{\tau}_O^p}{\boldsymbol{f}^p \cdot \boldsymbol{n}}.$$

Finally, friction forces are tangent to the contact surface and their moment is aligned with $\boldsymbol{n}$, so we equivalently can write this relationship as:

$$\boldsymbol{p}_{\text{CoP}} = \frac{\boldsymbol{n} \times \boldsymbol{\tau}_O^c}{\boldsymbol{f}^c \cdot \boldsymbol{n}}. \tag{2.4}$$

Equation (2.4) can be used to compute the CoP expressed in the local contact frame.

Zero-Moment Point (ZMP)

The ZMP is defined as a point on the ground where the *tipping moment* acting on the biped equals zero. This condition can be interpreted as a constraint on the contact moments, which contains at least the roll and pitch direction. Originally, the concept has been introduced in [50], it has been reviewed in [51] and made popular with [13].

The concept is build upon two key assumptions:

- There exists one planar contact surface (i.e. no multiple surfaces like on rough terrain)
- The friction is sufficiently high to prevent sliding of the feet

From the Newton-Euler equations, the motion of the biped can be written as

$$\begin{aligned} m\ddot{\boldsymbol{p}}_{\text{CoM}} &= m\boldsymbol{g} + \boldsymbol{f}^c \\ \dot{\boldsymbol{L}}_O &= \boldsymbol{p}_{\text{CoM}} \times m\boldsymbol{g} + \boldsymbol{\tau}^c_{\text{CoM}}, \end{aligned}$$

where m denotes the total mass of the robot, $\boldsymbol{g}$ is the gravity vector, $\ddot{\boldsymbol{p}}_{\text{CoM}}$ the centroidal acceleration, $\dot{\boldsymbol{L}}_O$ the change of the angular momentum. $\boldsymbol{w}^c_{\text{CoM}} = (\boldsymbol{\tau}^c_{\text{CoM}}, \boldsymbol{f}^c)_{6\times 1}$ denotes the sum of all contact wrenches in the CoM frame. The gravito-inertial wrench of the robot can be defined as

$$\begin{aligned} \boldsymbol{f}^{gi} &= m(\boldsymbol{g} - m\ddot{\boldsymbol{p}}_{\text{CoM}}) \\ \boldsymbol{\tau}^{gi}_O &= \boldsymbol{p}_{\text{CoM}} \times m\boldsymbol{g} - \dot{\boldsymbol{L}}_O. \end{aligned}$$

Using the wrench form of the Newton-Euler equations

$$\boldsymbol{w}^{gi} + \boldsymbol{w}^c = \boldsymbol{0}, \tag{2.5}$$

one can derive the ZMP, for the case of a planar surface, as

$$\boldsymbol{p}_{\text{CoM}} = \frac{\boldsymbol{n} \times \boldsymbol{\tau}_O^{gi}}{\boldsymbol{f}^{gi} \cdot \boldsymbol{n}}. \tag{2.6}$$

In practice, one can use this formula to compute the ZMP from force sensors or from an inertial measurement unit.

Coincidence of ZMP and CoP

As Sardain and Bessonnet outline, both the ZMP and the CoP yield the same point for the case of bipedal walking on a single plane surface [49]. Comparing Eq. (2.6) with Eq. (2.4), we recognize the only difference is that the former is applied to the (global) gravito-inertial wrench, while the latter is applied to the (local) contact wrench. If we recall the Newton-Euler equations from Eq. (2.5), it becomes clear why both points coincide when there is only one contact plane.

2.2.3. Stability Classification

There are existing several classifications of stability, which will be defined in the following according to [10, Sec.1.2.1] and [47]. See Vukobratović et al. for more details on differentiating the terms dynamic stability and dynamic balance [38].

Statically Stable Motion

The gait or movement of a humanoid is classified as *statically stable* if the FCoM does not leave the SP during the entire motion or gait. Consequently, the humanoid will remain in a stable position, whenever the movement is stopped. Typically, these kinds of stability are only obtained with very low walking velocities or quasi-static motions, where the static forces dominate the dynamic forces. To this end, the FCoM stability criteria is used for the generation of balanced static walking gaits (see Section 4.2).

Dynamically Stable Motion

If the FCoM partially leaves the SP at some point during the gait, but the CoP (or ZMP) always remains within the SP, the gait or movement is classified as *dynamically stable*. This stability margin is extremely useful for flat-foot dynamic walking since it prevents the foot from rotating around the boundary of the SP. The CoP is a central building block of the contact stability constrained DDP approach that will be discussed in Chapter 3.

2.3. Differential Dynamic Programming (DDP)

This section describes the basics of DDP, which is an OC algorithm that belongs to the TO class. The algorithm was introduced in 1966 by Mayne [52]. A modern description of the algorithm using the same notations as below can be found in [23, 53].

2.3.1. Finite Horizon Optimal Control

We consider a system with discrete-time dynamics, which can be modeled as a generic function $\boldsymbol{f}$

$$\boldsymbol{x}_{i+1} = \boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{u}_i), \tag{2.7}$$

that describes the evolution of the state $\boldsymbol{x} \in \boldsymbol{R}^n$ from time i to $i+1$, given the control $\boldsymbol{u} \in \boldsymbol{R}^m$. A complete trajectory $\{\boldsymbol{x}, \boldsymbol{u}\}$ is a sequence of states $\boldsymbol{x} = \{\boldsymbol{x}_0, \boldsymbol{x}_1, ..., \boldsymbol{x}_N\}$ and control inputs $\boldsymbol{u} = \{\boldsymbol{u}_0, \boldsymbol{u}_1, ..., \boldsymbol{u}_N\}$ satisfying Eq. (2.7). The *total cost* J of a trajectory can be written as the sum of running costs l and a final cost l_f starting from the initial state $\boldsymbol{x}_0$ and applying the control sequence $\boldsymbol{u}$ along the finite time-horizon:

$$J(\boldsymbol{x}_0, \boldsymbol{u}) = l_f(\boldsymbol{x}_N) + \sum_{i=0}^{N-1} l(\boldsymbol{x}_i, \boldsymbol{u}_i). \tag{2.8}$$

As discussed in Chapter 1, *indirect* methods such DDP represent the trajectory implicitly solely via the optimal control inputs $\boldsymbol{u}$. The states $\boldsymbol{x}$ are obtained from forward simulation of the system dynamics, i.e. integration Eq. (2.7). Consequently, the solution of the optimal control problem is the minimizing control sequence

$$\boldsymbol{u}^* = \underset{U}{\operatorname{argmin}}\, J(\boldsymbol{x}_0, \boldsymbol{u}).$$

2.3.2. Local Dynamic Programming

Let $\boldsymbol{u}_i \equiv \{\boldsymbol{u}_i, \boldsymbol{u}_{i+1}..., \boldsymbol{u}_{N-1}\}$ be the partial control sequence, the *cost-to-go* J_i is the partial sum of costs from i to N:

$$J_i(\boldsymbol{x}, \boldsymbol{u}_i) = l_f(\boldsymbol{x}_N) + \sum_{j=i}^{N-1} l(\boldsymbol{x}_j, \boldsymbol{u}_j). \tag{2.9}$$

The *Value function* at time i is the optimal cost-to-go starting at $\boldsymbol{x}$ given the minimizing control sequence

$$V_i(\boldsymbol{x}) = \min_{\boldsymbol{u}_i} J_i(\boldsymbol{x}, \boldsymbol{u}_i),$$

and the Value at the final time is defined as $V_N(\boldsymbol{x}) \equiv l_f(\boldsymbol{x}_N)$. The Dynamic Programming Principle [54] reduces the minimization over an entire sequence of control inputs to a sequence of minimizations over a single control, proceeding backwards in time:

$$V(\boldsymbol{x}) = \min_{\boldsymbol{u}}[l(\boldsymbol{x}, \boldsymbol{u}) + V'(\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}))]. \tag{2.10}$$

Note that Eq. (2.10) is referred to as the *Bellman equation* for *discrete-time* optimization problems [55]. For reasons of readability, the time index i is omitted and V' introduced to denote the Value at the next time step. The interested reader may note that the analogous equation for the case of *continuous-time* is a partial differential equation called the *Hamilton-Jacobi-Bellman equation* [56, 57].

2.3.3. Quadratic Approximation

DDP locally computes the optimal state and control sequences of the OC problem derived with Eq. (2.10) by iteratively performing a forward and backward pass. The *backward pass* on the trajectory generates a new control sequence and is followed by a *forward pass* to compute and evaluate the new trajectory.

Let $\boldsymbol{Q}(\delta\boldsymbol{x}, \delta\boldsymbol{u})$ be the variation in the argument on the right-hand side of Eq. (2.10) around the $i^{th}(\boldsymbol{x}, \boldsymbol{u})$ pair

$$\boldsymbol{Q}(\delta\boldsymbol{x}, \delta\boldsymbol{u}) = l(\boldsymbol{x} + \delta\boldsymbol{x}, \boldsymbol{u} + \delta\boldsymbol{u}) + V'(\boldsymbol{f}(\boldsymbol{x} + \delta\boldsymbol{x}, \boldsymbol{u} + \delta\boldsymbol{u})). \tag{2.11}$$

The DDP algorithm uses a quadratic approximation of this differential change. The quadratic Taylor expansion of $Q(\delta\boldsymbol{x}, \delta\boldsymbol{u})$ leads to

$$\boldsymbol{Q}(\delta\boldsymbol{x}, \delta\boldsymbol{u}) \approx \frac{1}{2} \begin{bmatrix} 1 \\ \delta\boldsymbol{x} \\ \delta\boldsymbol{u} \end{bmatrix}^T \begin{bmatrix} 0 & \boldsymbol{Q}_x^T & \boldsymbol{Q}_u^T \\ \boldsymbol{Q}_x & \boldsymbol{Q}_{xx} & \boldsymbol{Q}_{xu} \\ \boldsymbol{Q}_u & \boldsymbol{Q}_{ux} & \boldsymbol{Q}_{uu} \end{bmatrix} \begin{bmatrix} 1 \\ \delta\boldsymbol{x} \\ \delta\boldsymbol{u} \end{bmatrix}. \tag{2.12}$$

The coefficients can be computed as

$$\boldsymbol{Q}_x = l_x + \boldsymbol{f}_x^T \boldsymbol{V}'_x, \tag{2.13a}$$
$$\boldsymbol{Q}_u = l_u + \boldsymbol{f}_u^T \boldsymbol{V}'_x, \tag{2.13b}$$
$$\boldsymbol{Q}_{xx} = l_{xx} + \boldsymbol{f}_x^T \boldsymbol{V}'_{xx} \boldsymbol{f}_x + \boldsymbol{V}'_x \boldsymbol{f}_{xx}, \tag{2.13c}$$
$$\boldsymbol{Q}_{ux} = l_{ux} + \boldsymbol{f}_u^T \boldsymbol{V}'_{xx} \boldsymbol{f}_x + \boldsymbol{V}'_x \boldsymbol{f}_{ux}, \tag{2.13d}$$
$$\boldsymbol{Q}_{bu} = l_{uu} + \boldsymbol{f}_u^T \boldsymbol{V}'_{xx} \boldsymbol{f}_u + \boldsymbol{V}'_x \boldsymbol{f}_{uu}. \tag{2.13e}$$

where the primes denote the values at the next time-step.

2.3.4. Algorithmic Steps

The first algorithmic step of DDP, namely the backward pass, involves computing a new control sequence on the given trajectory and consequently determining the search direction of a a step in the numerical optimization. To this end, the quadratic approximation obtained from Eq. (2.12), minimized with respect to $\delta\boldsymbol{u}$ for some state perturbation $\delta\boldsymbol{x}$, results in

$$\delta\boldsymbol{u}^*(\delta\boldsymbol{x}) = \underset{\delta\boldsymbol{u}}{\operatorname{argmin}}\, \boldsymbol{Q}(\delta\boldsymbol{x}, \delta\boldsymbol{u}) = -\boldsymbol{Q}_{uu}^{-1}(\boldsymbol{Q}_u + \boldsymbol{Q}_{ux}\delta\boldsymbol{x}),$$

giving us an open-loop term $\boldsymbol{k}$ and a feedback gain term $\boldsymbol{k}$:

$$\boldsymbol{k} = -\boldsymbol{Q}_{uu}^{-1}\boldsymbol{Q}_u \quad \text{and} \quad \boldsymbol{k} = -\boldsymbol{Q}_{uu}^{-1}\boldsymbol{Q}_{ux}.$$

The resulting locally-linear feedback policy can be again inserted into Eq. (2.12) leading to a quadratic model of the Value at time i:

$$\begin{aligned}\Delta \boldsymbol{V} &= -\frac{1}{2}\boldsymbol{k}^T \boldsymbol{Q}_{uu}\boldsymbol{k} \\ \boldsymbol{V}_x &= \boldsymbol{Q}_x - \boldsymbol{k}^T \boldsymbol{Q}_{uu}\boldsymbol{k} \\ \boldsymbol{V}_{xx} &= \boldsymbol{Q}_{xx} - \boldsymbol{k}^T \boldsymbol{Q}_{uu}\boldsymbol{k}.\end{aligned}$$

After computing the feedback policy in the backward pass, the forward pass computes a corresponding trajectory by integrating the dynamics via

$$\begin{aligned}\hat{\boldsymbol{x}}_0 &= \boldsymbol{x}_0 \\ \hat{\boldsymbol{u}}_i &= \boldsymbol{u}_i + \alpha\boldsymbol{k}_i + \boldsymbol{k}_i(\hat{\boldsymbol{x}}_i - \boldsymbol{x}_i) \\ \hat{\boldsymbol{x}}_{i+1} &= \boldsymbol{f}(\hat{\boldsymbol{x}}_i, \hat{\boldsymbol{u}}_i),\end{aligned}$$

where $\hat{\boldsymbol{x}}_i, \hat{\boldsymbol{u}}_i$ are the new state-control sequences. The step size of the numerical optimization is described by the backtracking line search parameter α, which iteratively is reduced starting from 1. The backward and forward passes of the DDP algorithm are iterated until convergence to the (locally) optimal trajectory.

2.4. Handling Constraints With DDP

By nature, the DDP algorithm presented in Section 2.3 does not take into account constraints. Tassa et al. developed a control-limited DDP [23] that takes into account box inequality constraints on the control inputs allowing the consideration of torque limits on real robotic systems. Budhiraja et al. proposed a DDP version for the problem of multi-phase rigid contact dynamics by exploiting the Karush-Kuhn-Tucker (KKT) constraint of the rigid contact model [20].

To begin with, this section provides details on the above mentioned approach, since physically consistent bipedal locomotion is highly dependent on making contacts with the ground. Finally, the integration of robot tasks and physical consistency as additional constraints into the OC problem are explored.

2.4.1. DDP With Constrained Robot Dynamics

Contact Dynamics

In the case of rigid contact dynamics, DDP assumes a set of given contacts of the system with the environment. Then, an equality constrained dynamics can be incorporated by formulating rigid contacts as holonomic constraints to the robot dynamics. In other words, the contact points are assumed to have a fixed position on the ground.

The unconstrained robot dynamics can be represented as

$$\boldsymbol{M}\dot{\boldsymbol{v}}_{\text{free}} = \boldsymbol{S}\boldsymbol{\tau} - \boldsymbol{b} = \boldsymbol{\tau}_b, \tag{2.14}$$

with the joint-space inertia matrix $\boldsymbol{M} \in \boldsymbol{R}^{n\times n}$ and the unconstrained acceleration vector $\dot{\boldsymbol{v}}_{\text{free}}$. The right-hand side of Eq. (2.14) represents the n-dimensional force-bias vector accounting for the control $\boldsymbol{\tau}$, the Coriolis and gravitational effects $\boldsymbol{b}$ and the selection matrix $\boldsymbol{S}$ of actuated joints.

In order to incorporate the rigid contact constraints to the robot dynamics, one can apply the Gauss principle of least constraint [58]. The idea is to minimize the deviation in acceleration between the constrained and unconstrained motion:

$$\begin{aligned} \dot{\boldsymbol{v}} = \arg\min_{\boldsymbol{a}} \quad & \frac{1}{2}\left\|\dot{\boldsymbol{v}} - \dot{\boldsymbol{v}}_{\text{free}}\right\|_{\boldsymbol{M}} \\ \text{subject to} \quad & \boldsymbol{J}_c\dot{\boldsymbol{v}} + \dot{\boldsymbol{J}}_c\boldsymbol{v} = \boldsymbol{0}, \end{aligned} \tag{2.15}$$

where $\boldsymbol{M}$ formally represents the inertia tensor over the configuration manifold $\boldsymbol{q}$. In order to express the holonomic contact constraint $\phi(\boldsymbol{q})$ in the acceleration space, it needs to be differentiated twice. Consequently, the contact condition can be seen as a second-order kinematic constraints on the contact surface position where $\boldsymbol{J}_c = \begin{bmatrix}\boldsymbol{J}_{c_1} & \cdots & \boldsymbol{J}_c\end{bmatrix}$ is a stack of f contact Jacobians.

Karush-Kuhn-Tucker (KKT) Conditions

The Gauss minimization in Eq. (2.15) corresponds to an equality-constrained quadratic optimization problem. The optimal solutions $(\dot{\boldsymbol{v}}, \boldsymbol{\lambda})$ must satisfy the so-called KKT conditions given by

$$\begin{bmatrix}\boldsymbol{M} & \boldsymbol{J}_c^\top \\ \boldsymbol{J}_c & \boldsymbol{0}\end{bmatrix}\begin{bmatrix}\dot{\boldsymbol{v}} \\ -\boldsymbol{\lambda}\end{bmatrix} = \begin{bmatrix}\boldsymbol{\tau}_b \\ -\dot{\boldsymbol{J}}_c\boldsymbol{v}\end{bmatrix}. \tag{2.16}$$

These dual variables $\boldsymbol{\lambda}^k$ represent external wrenches at the contact level. For a given robot state and applied torques, Eq. (2.16) allows a direct computation of the contact forces. To this end, the contact constraints can be solved analytically at the level of dynamics instead of introducing additional constraints in the whole-body optimization [31].

2.4.2. KKT-Based DDP Algorithm

The KKT dynamics from Eq. (2.16) can be expressed as a function of the state $\boldsymbol{x}_i$ and the control $\boldsymbol{u}_i$:

$$\begin{aligned} \boldsymbol{x}_{i+1} &= \boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{u}_i), \\ \boldsymbol{\lambda}_i &= \boldsymbol{g}(\boldsymbol{x}_i, \boldsymbol{u}_i), \end{aligned} \tag{2.17}$$

where the concatenation of the configuration vector and its tangent velocity forms the state $\boldsymbol{x} = (\boldsymbol{q}, \boldsymbol{v})$, $\boldsymbol{u}$ is the input torque vector and $\boldsymbol{g}(\cdot)$ is the optimal solution of Eq. (2.16).

Supposing a sequence of predefined contacts, the cost-to-go of the DDP backward pass and its respective Hessians (compare Eq. (2.9) and Eq. (2.13)) turn into:

$$J_i(\boldsymbol{x}, \boldsymbol{u}_i) = l_f(\boldsymbol{x}_N) + \sum_{j=i}^{N-1} l(\boldsymbol{x}_j, \boldsymbol{u}_j, \boldsymbol{\lambda}_j)$$

with the control inputs $\boldsymbol{u}_i$ acting on the system dynamics at time i, and first-order approximation of $\boldsymbol{g}(\cdot)$ and $\boldsymbol{f}(\cdot)$ as

$$\begin{aligned}
\boldsymbol{Q}_{\boldsymbol{x}} &= \boldsymbol{l}_{\boldsymbol{x}} + \boldsymbol{g}_{\boldsymbol{x}}^T \boldsymbol{l}_{\boldsymbol{\lambda}} + \boldsymbol{f}_{\boldsymbol{x}}^T \boldsymbol{V}'_{\boldsymbol{x}}, \\
\boldsymbol{Q}_{\boldsymbol{u}} &= \boldsymbol{l}_{\boldsymbol{u}} + \boldsymbol{g}_{\boldsymbol{u}}^T \boldsymbol{l}_{\boldsymbol{\lambda}} + \boldsymbol{f}_{\boldsymbol{u}}^T \boldsymbol{V}'_{\boldsymbol{x}}, \\
\boldsymbol{Q}_{\boldsymbol{xx}} &\approx \boldsymbol{l}_{\boldsymbol{xx}} + \boldsymbol{g}_{\boldsymbol{x}}^T \boldsymbol{l}_{\boldsymbol{\lambda\lambda}} \boldsymbol{g}_{\boldsymbol{x}} + \boldsymbol{f}_{\boldsymbol{x}}^T \boldsymbol{V}'_{\boldsymbol{xx}} \boldsymbol{f}_{\boldsymbol{x}}, \\
\boldsymbol{Q}_{\boldsymbol{ux}} &\approx \boldsymbol{l}_{\boldsymbol{ux}} + \boldsymbol{g}_{\boldsymbol{u}}^T \boldsymbol{l}_{\boldsymbol{\lambda\lambda}} \boldsymbol{g}_{\boldsymbol{x}} + \boldsymbol{f}_{\boldsymbol{u}}^T \boldsymbol{V}'_{\boldsymbol{xx}} \boldsymbol{f}_{\boldsymbol{x}}, \\
\boldsymbol{Q}_{\boldsymbol{uu}} &\approx \boldsymbol{l}_{\boldsymbol{uu}} + \boldsymbol{g}_{\boldsymbol{u}}^T \boldsymbol{l}_{\boldsymbol{\lambda\lambda}} \boldsymbol{g}_{\boldsymbol{u}} + \boldsymbol{f}_{\boldsymbol{u}}^T \boldsymbol{V}'_{\boldsymbol{xx}} \boldsymbol{f}_{\boldsymbol{u}}.
\end{aligned} \tag{2.18}$$

Consequently, the KKT-based DDP algorithm utilizes the set of Eq. (2.18) inside the backward pass to incorporate the rigid contacts forces, while the updated system dynamics from Eq. (2.17) is utilized during the forward pass of the algorithm.

2.4.3. Task-Related Constraints

An important part of the motion generation is the execution of desired actions, e.g. grasping an object, moving the CoM or performing a robot step. For formulating these task-related constraints, we follow the notation used in [12].

An arbitrary task can be formulated as a regulator:

$$\boldsymbol{h}_{\text{task}_k}(\boldsymbol{x}_k, \boldsymbol{u}_k) = \boldsymbol{s}^d_{\text{task}} - \boldsymbol{s}_{\text{task}}(\boldsymbol{x}_k, \boldsymbol{u}_k),$$

where the task is defined as the difference between the desired and current feature vectors $\boldsymbol{s}^d_{\text{task}}$ and $\boldsymbol{s}_{\text{task}}(\boldsymbol{x}_k, \boldsymbol{u}_k)$, respectively. The task at each node can be added to the cost function via penalization as:

$$l_k(\boldsymbol{x}_k, \boldsymbol{u}_k) = \sum_{j \in \text{tasks}} \boldsymbol{w}_{j_k} \mid\mid \boldsymbol{h}_{j_k}(\boldsymbol{x}_k, \boldsymbol{u}_k) \mid\mid^2,$$

where $\boldsymbol{w}_{j_k}$ assigned to task j at corresponding time k. The DDP algorithm utilizes the derivatives of the regulator functions, namely computing the Jacobians and Hessians of the cost functions.

In the scope of this thesis, the following tasks are handled

$$\text{tasks} \subseteq \{CoM, LF_{SE(3)}, RF_{SE(3)}\},$$

namely the CoM tracking (CoM) and the tracking of the left- and right-foot pose $LF_{SE(3)}, RF_{SE(3)}$, respectively.

2.4.4. Inequality Constraints

Equally important for physically consistent motion planning is the consideration of boundaries, such as robot limits and stability constraints. These inequality constraints can be included into DDP-like solvers using i.e. penalization, active-set [59] and Augmented Lagrangian [60] strategies. In Crocoddyl, the penalization approach is used to consider inequality constraints in the OC formulation. The mathematical formulation is detailed in Section 3.3.

In the scope of this thesis, the following inequality constraints are utilized:

$$\text{inequalities} \subseteq \{\text{joint limits}, \text{friction cone}, CoP\}.$$

Further details on the constraints applied to the motion planning problems in the context of this thesis are provided in Section 4.1. In the next chapter, we explore an approach that combines multiple inequality constraints in order to embed contact stability into DDP-like solvers.

CHAPTER 3

Contact Stability Constrained DDP

This chapter presents a generic method for integrating contact stability constraints into DDP-like solvers. The key idea is to define inequality constraints for unilaterality, friction and the CoP of each contact surface with the goal of generating inherently balanced motions.

3.1. The Idea

Stability of the contacts is an essential objective of motion planning since prevents the robot from sliding and falling down. In Section 2.2 we have explored two different criteria for ensuring contact stability for dynamic systems, namely the ZMP and CoP. As outlined, the application of the ZMP is limited due to the assumptions of sufficiently high friction and the existence of one planar contact surface. Since we want to provide a *generic* method that can also be used for e.g. walking up stairs, these simplifying assumptions do not hold anymore.

Consequently, we decide to model a 6D surface contact, as introduced in Eq. (2.3) with dedicated constraints for (i) unilaterality of the contact forces (ii) Coulomb friction on the resultant force, and (iii) CoP inside the support area. For the sake of simplicity, we model a rectangular contact area. Nevertheless, this concept can be extended to arbitrary feet designs. This approach can be compared to the concept of

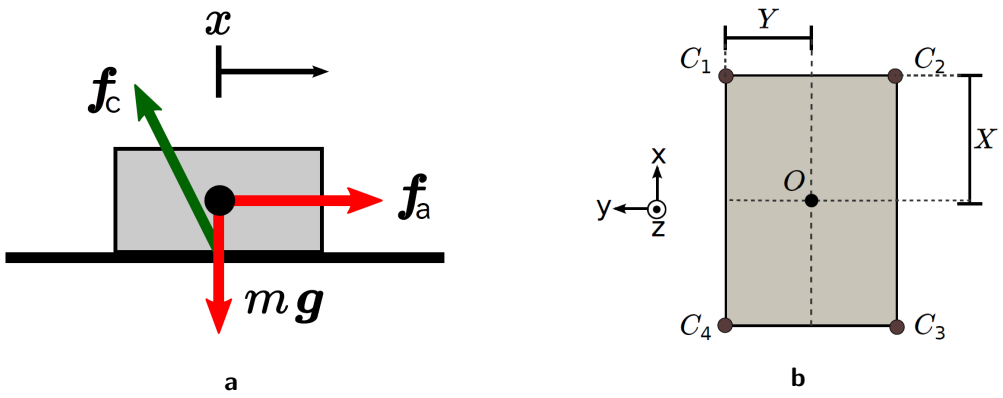


Figure 3.1.: (a) Visualization of acting forces on a simple rigid body and (b) notation used for the CoP definition in the contact surface plane [61].

contact wrench cone [61], without additionally enforcing the yaw torque constraint. These inequality constraints for surface contacts can compactly be summarized as

$$f_i^z > 0, \tag{3.1a}$$
$$| f_i^x | \leq \mu f_i^z, \tag{3.1b}$$
$$| f_i^y | \leq \mu f_i^z, \tag{3.1c}$$
$$| X | \geq C_x, \tag{3.1d}$$
$$| Y | \geq C_y. \tag{3.1e}$$

Let us now detail each line of the approach. The first inequality Eq. (3.1a) accounts for the unilaterality of the contact force. By nature, contact forces always have to be positive since the robot can only *push* from the ground, not *pull* to the ground (Fig. 3.1a). Inequality Eqs. (3.1b) and (3.1c) corresponds to the Coulomb friction, where μ denotes the static coefficient of friction. From a modeling perspective, this can be interpreted via the concept of spatial friction cones [62]. If, and only if the distributed contact forces lie inside their respective friction cones, these constraints are satisfied. Finally, inequality Eqs. (3.1d) and (3.1e) constrain the CoP to lie inside the rectangular contact area of each foot (see Fig. 3.1b). C_x and C_y denote the x and y position of $\boldsymbol{p}_{CoP}$, respectively. These CoP constraints prevent the robot from tilting around the edges of the rectangular surface contact. In particular, Eq. (3.1d) corresponds to a constraint of tilting around the pitch axis and Eq. (3.1e) prevents tilting around the roll axis.

Both, the unilaterality of the contact forces and the friction cone constraints, are already implemented inside Crocoddyl. However, the central component, bounding the CoP to lie inside the support area of each contact foot, is missing. Therefore, the rest of this chapter deals with the derivation of a set of implementable CoP constraints and describes the integration of these constraints as a cost function into the Crocoddyl framework.

3.2. Center of Pressure (CoP) Constraints

In this section we will derive a universal set of implementable constraints that bound the CoP to lie inside the rectangular contact area of each foot.

3.2.1. CoP Stability Conditions

Recapitulate the constraints from inequality Eqs. (3.1d) to (3.1e). Instead of using the absolute value of X and Y, one can also formulate the constraints as

$$\begin{aligned} -X \leq C_x \leq X, \\ -Y \leq C_y \leq Y. \end{aligned} \tag{3.2}$$

Based on this formulation it becomes evident that the CoP is constrained to lie inside the foot geometry visualized in Fig. 3.1b. In fact, these conditions can be represented via four single inequality equations as

$$\begin{aligned} X + C_x \geq 0, \\ X - C_x \geq 0, \\ Y + C_y \geq 0, \\ Y - C_y \geq 0. \end{aligned} \tag{3.3}$$

These four inequality equations will be used in the following to formulate the CoP constraints.

3.2.2. CoP Computation

Our goal is to determine explicit expressions for C_x and C_y for arbitrary floor orientations, including inclined ground. To this end, consider the computation routine for the CoP from Eq. (2.4)

$$\boldsymbol{p}_{CoP} = \frac{\boldsymbol{n} \times \boldsymbol{\tau}_{\boldsymbol{O}}^{\boldsymbol{c}}}{\boldsymbol{f}^c \cdot \boldsymbol{n}}.$$

For arbitrary orientations of the contact normal vector $\boldsymbol{n}$, we obtain

$$\boldsymbol{p}_{CoP} = \frac{\boldsymbol{n} \times \boldsymbol{\tau}_{\boldsymbol{O}}^{\boldsymbol{c}}}{\boldsymbol{f}^c \cdot \boldsymbol{n}} = \frac{\begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} \times \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}}{\begin{bmatrix} f_x \\ f_y \\ f_z \end{bmatrix} \cdot \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix}} = \begin{bmatrix} n_y t_z - n_z t_y \\ n_z t_x - n_x t_z \\ n_x t_y - n_y t_x \end{bmatrix} \cdot \frac{1}{f_x n_x + f_y n_y + f_z n_y}, \tag{3.4}$$

and solve for the desired position C_x and C_y of the CoP as

$$C_x = \frac{n_y t_z - n_z t_y}{f_x n_x + f_y n_y + f_z n_y}, \tag{3.5a}$$

$$C_y = \frac{n_z t_x - n_x t_z}{f_x n_x + f_y n_y + f_z n_y}. \tag{3.5b}$$

3.2.3. CoP Inequality Constraints

Now that we have found explicit expressions for computing the CoP (Eqs. (3.5a) to (3.5b)), we can insert them into Eq. (3.3), which gives a set of four CoP constraints as

$$\begin{aligned} X + \frac{n_y t_z - n_z t_y}{f_x n_x + f_y n_y + f_z n_y} &\geq 0, \\ X - \frac{n_y t_z - n_z t_y}{f_x n_x + f_y n_y + f_z n_y} &\geq 0, \\ Y + \frac{n_z t_x - n_x t_z}{f_x n_x + f_y n_y + f_z n_y} &\geq 0, \\ Y - \frac{n_z t_x - n_x t_z}{f_x n_x + f_y n_y + f_z n_y} &\geq 0. \end{aligned} \tag{3.6}$$

These conditions can be written in matrix form as:

$$\begin{bmatrix} Xn_0 & Xn_1 & Xn_2 & 0 & -n_2 & n_1 \\ Xn_0 & Xn_1 & Xn_2 & 0 & n_2 & -n_1 \\ Yn_0 & Yn_1 & Yn_2 & n_2 & 0 & -n_0 \\ Yn_0 & Yn_1 & Yn_2 & -n_2 & 0 & n_0 \end{bmatrix} \begin{bmatrix} f^x \\ f^y \\ f^z \\ \tau^x \\ \tau^y \\ \tau^z \end{bmatrix} \geq \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \tag{3.7}$$

and finally yield an implementable set of inequality equations for constraining the CoP to lie inside the rectangular contact area of each foot.

3.3. Integration Into the Crocoddyl Framework

This section presents the integration of the derived CoP inequality constraints from Eq. (3.7) into the Crocoddyl framework.

3.3.1. Inequality Constraints by Penalization

In Section 2.4 we have discussed possible ways of incorporating inequality constraints into DDP-like solvers. Crocoddyl handles inequality constraints, such as joint limits or friction cone, via penalization. In numerical optimization, the goal is to minimize

a given cost function. In Crocoddyl, an *action model* combines dynamics and cost model for each knot of the discretized OC problem from Eq. (2.8). The cost function for an action model at knot n can be written as:

$$l_n = \sum_{c=1}^{C} \alpha_c \Phi_c(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{\tau}), \tag{3.8}$$

where C different costs Φ_c are weighted by a respective coefficient $\alpha_c \in R$. The goal of the following two parts is to demonstrate how the inequality Eq. (3.7) is implemented inside a novel cost function into the framework.

3.3.2. Computation of Residual and Cost

In numerical analysis, the term *residual* corresponds to the error of a result [63]. For the sake of compactness, we abbreviate Eq. (3.7) as

$$\boldsymbol{A}\boldsymbol{w} \geq \boldsymbol{0}, \tag{3.9}$$

where $\boldsymbol{A}$ corresponds to a matrix of CoP inequality constraints and $\boldsymbol{w}$ is the contact wrench acting on the according foot. The residual $\boldsymbol{r} \in \boldsymbol{R}_{4\times 1}$ of the cost is retrieved by a simple matrix-vector multiplication:

$$\boldsymbol{r} = \boldsymbol{A}\boldsymbol{w}. \tag{3.10}$$

The residual vector depicted in Eq. (3.10) typically contains non-zero numbers. The resulting scalar CoP cost value Φ_{CoP} is computed via a bounded quadratic activation as

$$\Phi_{CoP} = \begin{cases} \frac{1}{2}\boldsymbol{r}^T\boldsymbol{r} & | \text{ lb} > \boldsymbol{r} > \text{ub} \\ 0 & | \text{ lb} \leq \boldsymbol{r} \leq \text{ub}. \end{cases} \tag{3.11}$$

In order to account for the positiveness of $\boldsymbol{r}$ (see Eq. (3.9)), the bounds are set to $\text{lb} = \boldsymbol{0}$ and $\text{ub} = \infty$, respectively. Finally, this bounded quadratic activation of the residual vector has the following implications:

- The CoP cost is zero, whenever $\boldsymbol{p}_{CoP}$ lies inside or on the border of the foot area spanned by X and Y.
- The CoP cost increases in a quadratic manner, when $\boldsymbol{p}_{CoP}$ exceeds the foot area spanned by X and Y.

Besides this formulation of the cost function also other designs are conceivable for the future. For example, the Euclidean distance from the CoP to the coordinate origin could be considered when computing the residual (see Eq. (3.11)).

3.3.3. Basic Usage and Contributions

The cost function is implemented in C++, but can be accessed via Python bindings for versatile and fast prototyping. In the following, a basic example is provided to demonstrate the interface of the CoP cost function to the interested reader.

```
# 1. Creating the cost model container
costModel = crocoddyl.CostModelSum(state, actuation.nu)
# 2. Defining the CoP cost
footGeometry = np.array([0.2, 0.08]) # dim [m] of the foot area
CoPCost = crocoddyl.CostModelContactCoPPosition(state,
crocoddyl.FrameCoPSupport(footId, footGeometry), actuation.nu)
# 3. Adding the CoP cost term with assigned weight to the cost model
costModel.addCost("LF_CoPCost", CoPCost, 1e3)
```

The contributions of this thesis to the open-source framework Crocoddyl are summarized in two main pull requests. The first one, #792 contains the basic formulation of the CoP cost function for contact dynamics action models (see Appendix A.1). With #830, an additional version of the cost function is implemented for impulse dynamics action models (see Appendix A.2). A functional unit test that checks the cost against numerical differentiation as well as accessible python bindings can be found in the according directory of [64].

CHAPTER 4

Bipedal Walking Variants

This chapter studies the proposed motion planning approach for bipedal walking gaits of the full-size humanoid RH5. It starts by describing the individual building blocks of the optimization problem, then discusses the simulation results obtained for increasing gait dynamics and finally provides an evaluation of the contact stability of the dynamic motions.

4.1. Formulation of the Optimization Problem

This section gives information about the adopted contact and impact modeling techniques and introduces the constraints used for generating physically consistent walking trajectories. The core formulation is based on the legged gaits described in [34] but contains various improvements necessary for application on real robots. All motions presented in this chapter are solved for a predefined sequence of contacts and step timings. Recapitulating Section 2.3, we formulate the optimization problem as

$$\boldsymbol{X}^*, \boldsymbol{U}^* = \arg\min_{\mathbf{X},\mathbf{U}} l_N(x_N) + \sum_{k=0}^{N-1} \int_{t_k}^{t_k+\Delta t} l(\mathbf{x}, \mathbf{u}) dt. \tag{4.1}$$

4.1.1. Contact and Impact Modeling

During a walking motion, the body is always in contact with the ground either in single support, or in double support. In Section 2.4 we have discovered, how rigid

contacts can be expressed as a kinematic constraint on the EoM (see Eq. (2.14)). Analogously, one can describe the impulse dynamics [1] of a multibody system as

$$\begin{bmatrix} \boldsymbol{M} & \boldsymbol{J}_c^\top \\ \boldsymbol{J}_c & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{v}^+ \\ -\boldsymbol{\Lambda} \end{bmatrix} = \begin{bmatrix} \boldsymbol{M}\boldsymbol{v}^- \\ -e\boldsymbol{J}_c\boldsymbol{v}^- \end{bmatrix}, \tag{4.2}$$

where $\boldsymbol{\Lambda}$ is the contact impulse, $\boldsymbol{v}^-$ and $\boldsymbol{v}^+$ are the generalized velocities before and after the impact and $e \in [0, 1]$ is the restitution coefficient that accounts for the elasticity of the collision. For all motions, we use this impulse model to account for the infinitesimal short change in the contact situation. To improve the numerical integration stability, terms defined by Baumgarte Stabilization [65] are used along with the rigid contact constraint described with Eq. (2.15). This numerical stabilization on the constraints can be expressed as:

$$a_0 = a_{\lambda(c)} - \alpha M^{\text{ref}}_{\lambda(c)} \ominus M_{\lambda(c)} - \beta v_{\lambda(c)}, \tag{4.3}$$

where a_0 is the desired acceleration in the constrained space, $v_{\lambda(c)}, a_{\lambda(c)}$ are the spatial velocity and acceleration of the contact $\lambda(c)$, respectively, $M^{\text{ref}}_{\lambda(c)} \ominus M_{\lambda(c)}$ is the inverse composition between the reference contact placement and the current one [34]. The PD gains α and β enable an adequate numerical stabilization of the constraints depending on the dynamics of the movement.

4.1.2. Robot Tasks

Robot tasks, such as grasping and object or performing a step, are an essential goal of motion planning. As outlined in Section 2.4, these task-related constraints are considered in the optimization process as regulator functions. In the context of this thesis two tasks are of specific interest, namely the foot tracking and CoM tracking.

Foot Tracking Cost In order to perform a symmetric gait (see Section 2.1), the design of dedicated foot trajectories is crucial. We use piecewise-linear functions to describe the swing foot reference trajectory. Deviation from this time-depended reference foot trajectory is highly penalized. The foot tracking cost can be formulated via the squared Euclidean norm (L2 norm) as

$$\Phi_{\text{foot}} = || \boldsymbol{f}(t) - \boldsymbol{f}^{\text{ref}}(t) ||_2^2,$$

where $\boldsymbol{f}(t)$ is the actual CoM position at time-step t and $\boldsymbol{f}^{\text{ref}}(t)$ is the according reference. Start and end position as well as timings are predefined and combined with a desired step height to shape the desired foot trajectory.

[1] Impulse dynamics account for the physical effects that occur at a switch from non-contact to contact condition [43].

CoM Tracking Cost In bipedal locomotion, the three-dimensional position of the CoM of the whole-body turned out to be crucial [66]. This is especially true for quasi-static motions, where the FCoM is used as static stability margin as explained in Section 2.1. Analogously to the foot cost, the CoM tracking cost is formulated as

$$\Phi_{\text{CoM}} = || \, \boldsymbol{c}(t) - \boldsymbol{c}^{\text{ref}}(t) \, ||_2^2 \, .$$

In order to account for the static stability in these motions, we perform a dedicated shifting of the FCoM to the foot center, before performing the swing-foot task.

4.1.3. Inequality Constraints for Physical Consistency

Essential demands on physically consistent motion planning are that (i) the robot limits (torque, joints) are considered and (ii) the generated trajectories are inherently balanced. To this end, we consider joint limits, friction cone and the novel CoP bound as inequality constraints in our formulation, while torque constraints are covered in the algorithm itself.

Contact Stability Constraints As detailed in Chapter 3 with the concept of contact stability constrained DDP, we constrain unilaterality, friction and CoP for each foot in contact. For the sake of clarity, Eq. (3.11) is again recalled as

$$\Phi_{\text{CoP}} = \begin{cases} \frac{1}{2}\boldsymbol{r}^T\boldsymbol{r} & | \, \text{lb} > \boldsymbol{r} > \text{ub} \\ 0 & | \, \text{lb} \leq \boldsymbol{r} \leq \text{ub}, \end{cases}$$

where the CoP position is bound to lie inside the foot contact area by the lower and upper bounds lb and ub, respectively. In the same manner friction cone constraints along with the unilaterality are considered as

$$\Phi_{\text{friction}} = \begin{cases} \frac{1}{2}\boldsymbol{r}^T\boldsymbol{r} & | \, \text{lb} > \boldsymbol{r} > \text{ub} \\ 0 & | \, \text{lb} \leq \boldsymbol{r} \leq \text{ub}, \end{cases}$$

where lb and ub bound the resulting contact force to lie inside a four-sided polygonal approximation of the spatial friction cone [62].

Joint Limits Physical boundaries of the joints must not be exceeded to avoid damage to the system. They are covered via a bounded quadratic activation as

$$\Phi_{\text{joints}} = \begin{cases} \frac{1}{2}\boldsymbol{r}^T\boldsymbol{r} & | \, \text{lb} > \boldsymbol{r} > \text{ub} \\ 0 & | \, \text{lb} \leq \boldsymbol{r} \leq \text{ub}, \end{cases}$$

where lb and ub correspond to the lower and upper bounds for joint position and velocities, respectively.

4.1.4. Further Regularization Terms

Additional to the described constraints for tasks and physical consistency, we optimize for minimization of the torques and regularize the robot posture.

Torque Minimization In order to improve the energy efficiency of the motions and maintain a human-like torque at the joints [67], we minimize the joint torques for realistic dynamic movements via

$$\Phi_{\text{torque}} = || \boldsymbol{\tau}(t) ||_2^2 .$$

Posture Regularization Finally, we deal with the redundancy of multi-body dynamics by applying a weighted least-squares cost function to regularize the state with respect to the nominal robot posture:

$$\Phi_{\text{posture}} = || \boldsymbol{q}(t) - \boldsymbol{q}^{\text{ref}}(t) ||_2^2 .$$

4.2. Simulation Results for Increasing Gait Dynamics

This section presents the simulation results for bipedal walking motions obtained by solving an optimization problem based on the described building blocks from the previous section. It studies both static and dynamic walking gaits, respectively.

4.2.1. Static Walking

The analysis of static walking gaits provides detailed insights on the optimization structure and allows a thorough experimental validation (see Chapter 7).

Focus of investigation is a slow two step walking motion. We assume a quasi-static motion and hence deploy a static stability criterion as introduced in Section 2.2 by following a dedicated FCoM trajectory. To this end, the optimization problem is composed of a total of five locomotion phases visualized in Fig. 4.1. From (a) an initial pose, (b) shift the FCoM above the Left Foot (LF), (c) perform a right step, (d) shift the FCoM above the Right Foot (RF), (e) perform a left step and (f) shift the FCoM to the half length of the line intersecting both feet center while returning to the initial pose.

Additionally, the CoM is optimized for a constant height over the whole gait. Table 4.1 gives a compact overview of the desired gait characteristics and the applied constraints of the optimization. In order to comply with the quasi-static assumption, the motion is performed about a time horizon of 15 s with a total stride length of 20 cm a robot model with fixed arms.

Table 4.1.: Static walking gait characteristics and applied optimization constraints.

Gait Characteristics		**Optimization Constraints**	
Step length:	10 cm	Tasks:	Φ_{foot}, Φ_{CoM}
Step height:	5 cm	Stability:	Φ_{friction}
Time:	3 s/phase	Limits:	Φ_{joint}, torques
Step size:	0.03 s	Regularization:	Φ_{posture}, Φ_{torque}

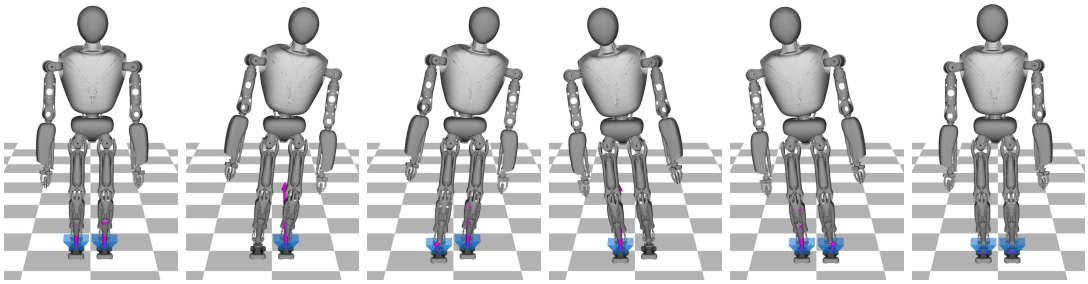


Figure 4.1.: Static walking gait based on dedicated FCoM motion, consisting of the locomotion phases (a) initial pose, (b) FCoM shift above LF, (c) right step, (d) FCoM shift above RF, (e) left step and (f) pose recovery. Significant lateral shifts of the FCoM are required to establish the static stability. [Video]

The results of the optimization problem Eq. (4.1) are shown in Fig. 4.2 and Fig. 4.3. Fig. 4.2 presents the resulting base and end-effector trajectories. As becomes clear, the FCoM tracking is sufficiently good and the CoM height remains in a reasonable range of $\pm$ 1 cm. Equally, the desired foot trajectory is tracked with adequate accuracy. Also, the foot velocities are reasonable with a maximum velocity of 0.1 m/s at the impact. The pursued step height is not reached exactly but is about one centimeter less. This effect can be explained by the immediately reverse direction at the vertex of the piecewise-linear trajectory, which is smoothened by the solver.

Fig. 4.3 shows the resulting joint trajectories for the torso, LF and RF. Both the joint position limits and the maximum permissible joint speeds remain far below the limits due to the slow nature of the motion. Interestingly, the pursued FCoM shifting turns out to be realized mostly based on a shift in the body roll activation rather than on a shift in the hips. Furthermore, it becomes evident that the posture regularization is effective since all joints end closely to the initial position.

4.2.2. Dynamic Walking

The analysis of a dynamic walking gait is concerned about generating efficient motions with higher velocities. This part forms the basis of the stability evaluation in the next section.

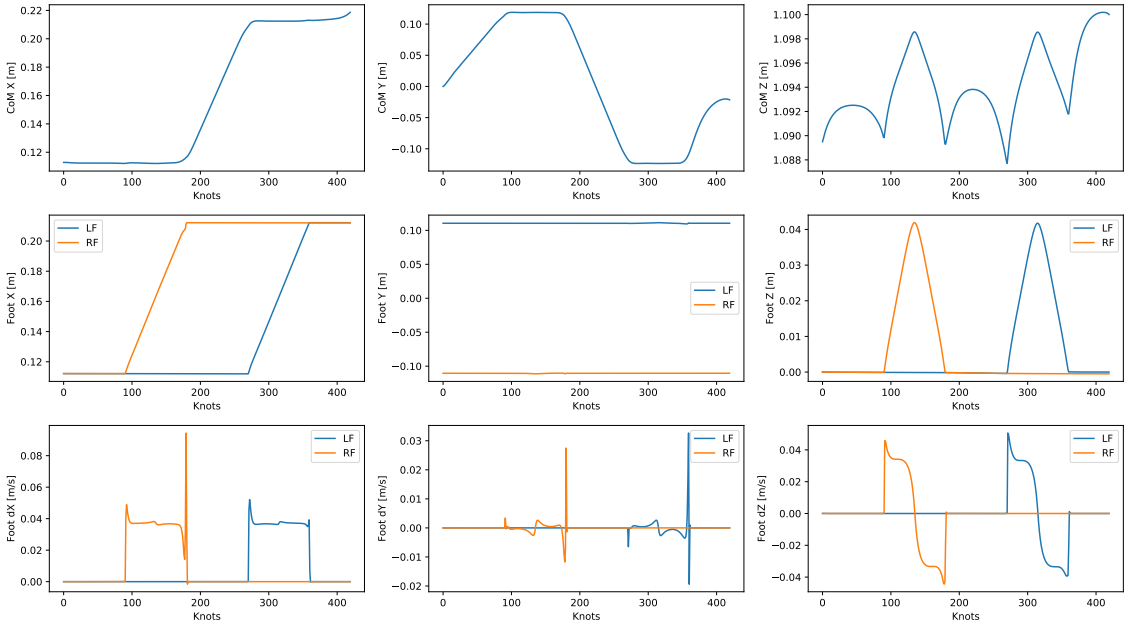


Figure 4.2.: Static walking gait solution in task space. Both the FCoM task and the swing foot task are satisfied with acceptable accuracy.

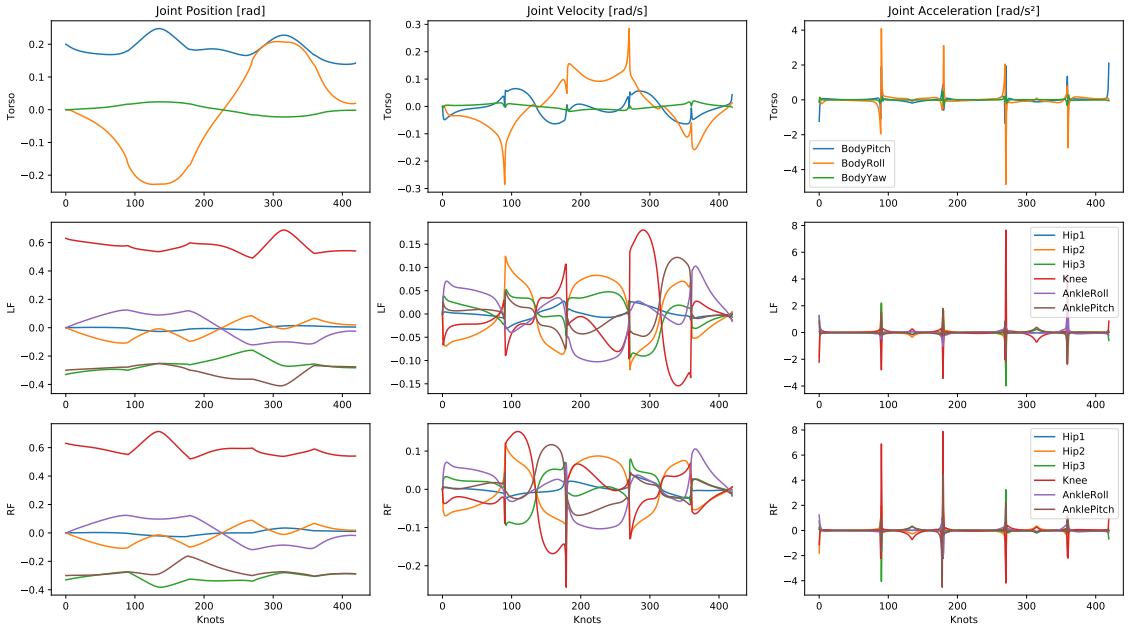


Figure 4.3.: Static walking gait solution of the joint states. Due to the slow nature of the motion, joint limits are far from being reached and velocity discontinuities remain within reasonable ranges.

Focus of investigation is a dynamic walking motion. Compared to the previously studied static walking gait these motions are characterized by higher velocities and

Table 4.2.: Dynamic walking gait characteristics and optimization constraints for a stance to swing ratio of one to three.

Gait Characteristics		**Optimization Constraints**	
Step length:	40 cm	Tasks:	Φ_{foot}
Step height:	5 cm	Stability:	Φ_{CoP}, Φ_{friction}
Time:	1.5 s/step	Limits:	Φ_{joint}, torques
Step size:	0.03 s	Regularization:	Φ_{posture}, Φ_{torque}

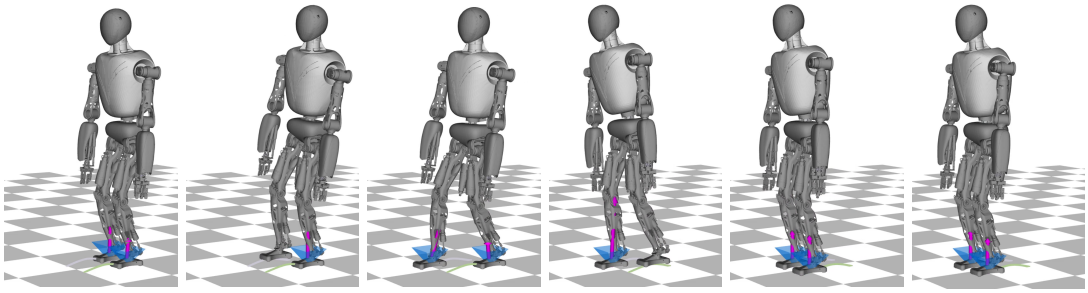


Figure 4.4.: Dynamic walking gait based on the contact stability constrained DDP approach, consisting of the locomotion phases (a) initial pose, (b,c) right step, (d,e) left step and (f) pose recovery. A natural CoM shifting emerges resulting from the inequality constraints for the CoP of each foot. [Video]

dynamic forces exceeding the static ones. These characteristics imply that dynamic stability criteria become necessary. To this end, we apply the proposed approach of contact stability constrained DDP described in Chapter 3. Consequently, the CoP of each foot is constrained instead of following a reference CoM trajectory. By this, the solver is enabled to find an optimal, dynamic CoM shifting along with the requested contact stability constraints.

Table 4.2 compactly summarizes the gait characteristics and applied optimization constraints. The optimization problem is composed of a total of five locomotion phases visualized in Fig. 4.4. From (a) an initial pose in Double Support (DS), (b,c) perform a right step, (d,e) perform a left step and (f) recover to the initial pose. In accordance with biomechanical findings [68], we choose a desired step length of 40 cm with 1.5 s per step and deliberately define a stance to swing ratio of one to three.

As becomes clear from Fig. 4.5, a natural CoM shifting to the sides emerges resulting from the inequality constraints for the CoP, which is about half of the amount as for the static walking case. Although the CoM height is not explicitly constrained, it stays in a reasonable range of about +3 cm, which may be caused from the final posture regularization. The end-effector velocities in z-direction are about twice as high as for the case of static walking, which is explained by the higher

walking speed. Fig. 4.6 shows the resulting joint states for the dynamic walking gait. The velocity and acceleration contain higher peaks and the joint deflections are stronger, which is reasonable due to the higher walking speed.

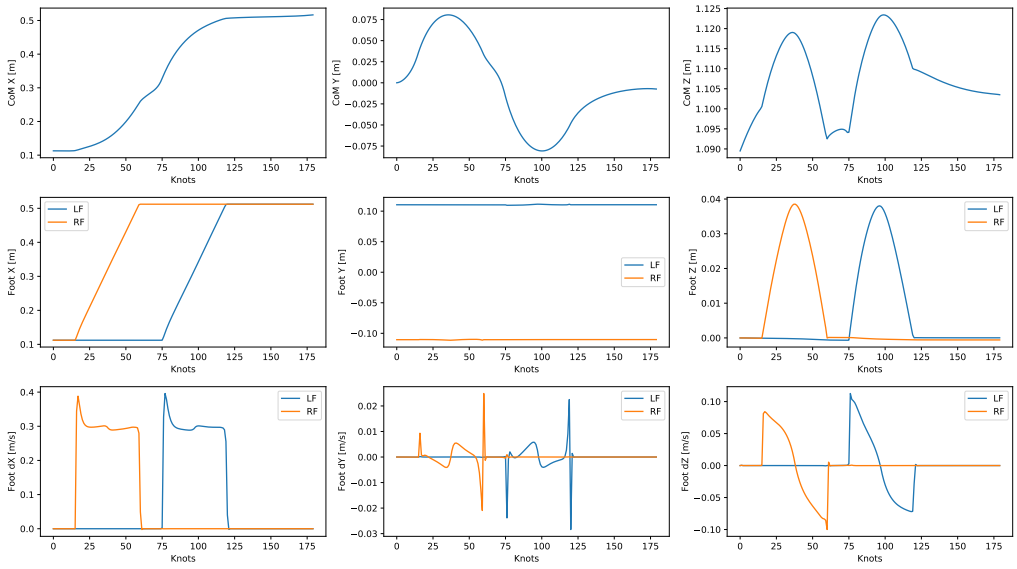


Figure 4.5.: Dynamic walking gait solution in task space. It becomes evident that a natural CoM shifting in y-direction emerges resulting from the inequality constraints for the CoP of each foot.

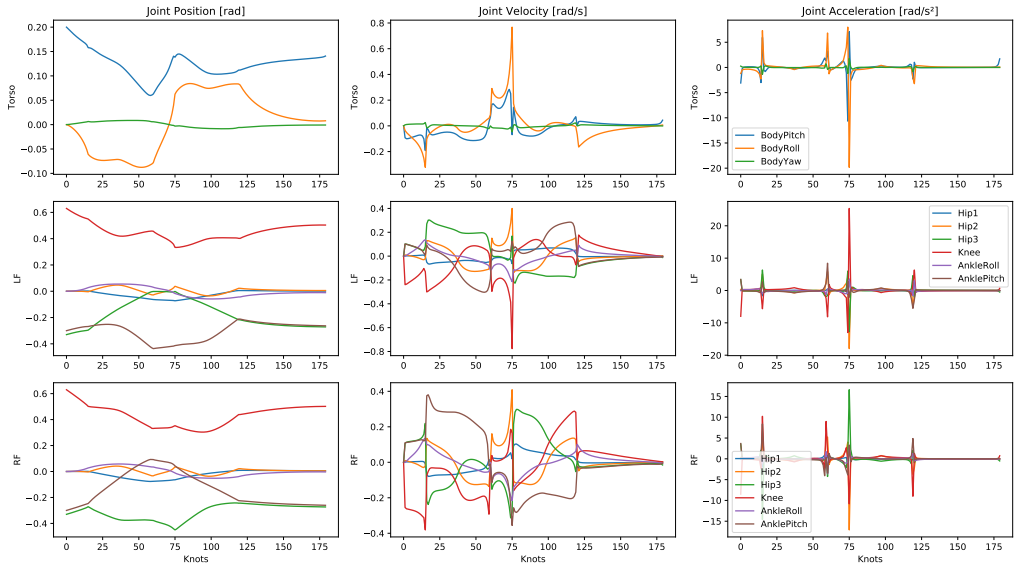


Figure 4.6.: Dynamic walking gait solution of the joint states. Both joint velocities and acceleration show higher peaks along with higher joint deflections compared to the static walking, which is reasonable due to the higher walking speed.

4.3. Evaluation of Contact Stability

This section evaluates, based on the presented dynamic walking gait from the previous section, the proposed approach of contact stability constrained DDP (Chapter 3).

4.3.1. Dynamically Balanced Walking Motion

As detailed in Section 2.2, the central characteristic for dynamically balanced motions is that the CoP or ZMP remains within the SP. The generic approach presented in Chapter 3 has been applied to the dynamic walking gait presented in the previous section. It utilizes the CoP criterion for each contact surface and hence the motion can be called dynamically balanced only if the CoP of each foot in contact stays within the according SP along the whole motion.

Fig. 4.7 shows the top view of the presented dynamic walking gait. The rectangles correspond to the true-to-scale dimensions of the robot feet. For clarity, only the first and last DS phase are visualized. The blue curve shows the time course of the resulting CoM trajectory, with relevant points in time marked separately. Since the CoM trajectory between lift-off and touch-down is largely outside the respective foot area, no static stability can be present, as expected. The orange and green crosses mark the time-dependent position of the CoPs for both feet. It is evident that both CoPs remain within the corresponding SP over the entire time course of the movement, which is why the movement can be classified as dynamically balanced.

4.3.2. Different Levels of CoP Restriction

Although the dynamic walking motion is inherently balanced, it becomes clear from Fig. 4.7 that the CoP partly lies *on* or *near* the border of the respective foot area. This effect can be attributed to the formulation of the CoP cost function (Eq. (3.11)), which is defined to be zero whenever the CoP lies within the given foot area and a quadratic penalization prevents the CoP to leave the SP. Theoretically, this formulation is sufficient to generate balanced motions. In practice however, it might be convenient for real-world experiments to consider a dedicated safety factor so that the CoPs maintain a certain distance from the edge. With the presented CoP cost function, this objective can be easily achieved by reducing the desired foot geometry. Fig. 4.8 shows the dynamic walking gait, where the CoP inequality constraints are active for a SP reduced to 50 percent of the original foot geometry. It becomes evident that also these more conservative contact stability constraints can be solved, which might be useful for the purpose of experimental validation.

In this section we have seen that the proposed contact stability constrained DDP produces motions that are dynamically balanced. In Chapter 6 we investigate if the generated motions can be tracked by a simple online stabilizer based on position control in joint space. Beforehand, in Chapter 5, we explore the effect of the motion planning approach and physical system limits on highly dynamic movements.

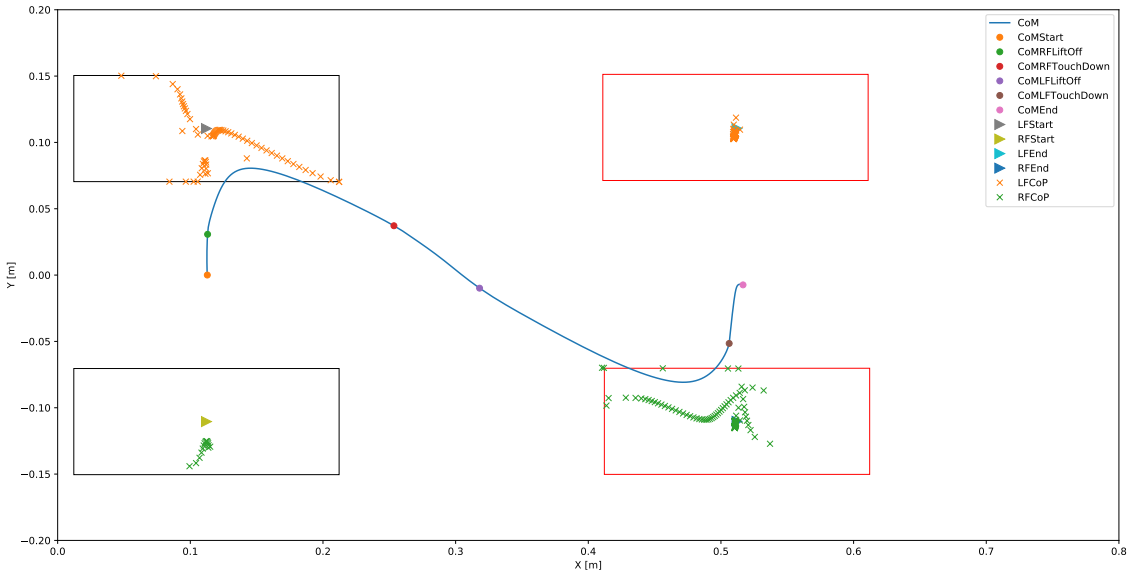


Figure 4.7.: Stability Analysis of the dynamic walking gait. As both CoPs remain within the corresponding SP, the movement can be classified as dynamically balanced. Hence, the functionality of the proposed contact stability constrained DDP approach is verified.

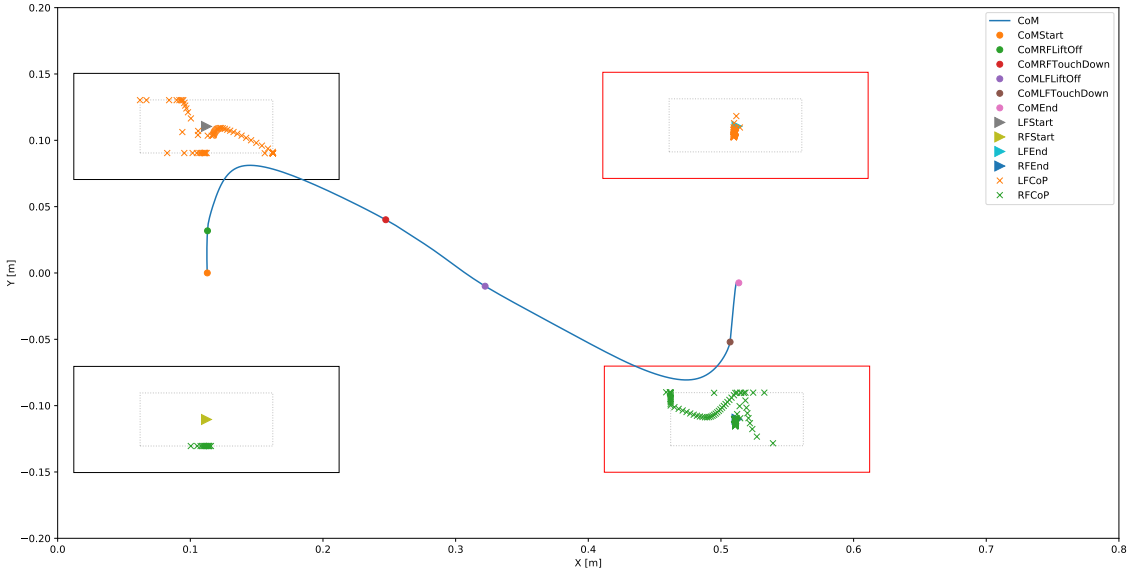


Figure 4.8.: Stability analysis of the dynamic walking gait with a CoP that is constrained in a SP reduced by 50 percent. By this, the stability is more conservative, which may be convenient for real-world experiments

CHAPTER 5

Highly-Dynamic Movements

This chapter presents an analysis of the proposed motion planning approach for highly-dynamic movements of the full-size humanoid RH5. To begin with, the central building blocks of the optimization problem are again briefly addressed. Then the simulation results for jumping tasks of increasing complexity are presented. Finally, an analysis of the maximal system performance based on the allowed joint and torque limits is presented.

5.1. Formulation of the Optimization Problem

This section concisely summarizes the constraints of the OC problem used to generate highly-dynamic movements as demonstrated in the following up section.

The vertical jump formulation follows an optimization problem of the form described in Eq. (4.1). For performing multiple forward jumps over obstacles instead, we successively solve P individual optimization problems of range N for each jump. To this end, we formulate a multi-phase OC problem as follows:

$$\boldsymbol{X}^*, \boldsymbol{U}^* = \arg\min_{\mathbf{X},\mathbf{U}} \sum_{p=0}^{P} \sum_{k=0}^{N} \int_{t_k}^{t_k+\Delta t} l_p(\mathbf{x}, \mathbf{u}) dt. \tag{5.1}$$

We constrain the optimization based on the building blocks introduced in Section 4.1. Precisely, we define a foot tracking cost (Φ_{foot}) based on piecewise-linear functions to incorporate the basic jumping height and length. We apply the contact stability constrained DDP (Chapter 3) with dedicated cost functions for CoP (Φ_{CoP}) and friction cone (Φ_{friction}). Physical consistency is ensured via the torque bounds of the solver and joint limits costs (Φ_{joints}). Additionally, torque minimization (Φ_{torque}) and posture regularization (Φ_{posture}) are optimized.

Table 5.1.: Vertical jump characteristics and applied optimization constraints.

Jump Characteristics		**Optimization Constraints**	
Jump length:	0 cm	Tasks:	Φ_{foot}
Jump height:	10 cm	Stability:	Φ_{CoP}, Φ_{friction}
Total time:	0.9 s	Limits:	Torques
Step size:	0.01 s	Regularization:	Φ_{posture}, Φ_{torque}

Biomechanical studies have shown that the effect of arm swinging is elementary for the performance in human jumping [69]. To this end, we also include the arms in the optimization for our jumping tasks. In order to account for the dynamic nature of the movements, it turns out to be useful to reduce the integration step size. We found a higher feasibility of the jumping tasks with an integration step size of 10 ms compared to 30 ms for the bipedal walking gaits (see Section 4.2). Since we do not use a contact planner along with this work, the contact timings had to be chosen to approximately match the physic of flight phases. Highly-dynamic movements are subject to higher impulse forces compared to walking movements. As described in Section 2.3, DDP relies on integration of the system dynamics in each step to obtain the states. The higher impulse forces then in turn cause numerical drifts of higher order in the contact constraints. This effect is visible as drifting of the support feet after touchdown. Through an extensive grid search, we found a set of valid Baumgarte gains, namely $\alpha, \beta = [0, 100]$, that reduce these numerical drifts for highly-dynamic movements (see Eq. (4.3)). In case the Baumgarte stabilization is not sufficient for stabilizing the contact positions, we found that further decreasing the integration step size also reduces the numerical drifts in the holonomic constraints.

5.2. Simulation Results for Increasing Task Complexity

This section presents the simulation results for three case studies of highly-dynamic movements obtained by solving an optimization problem based on the description in the previous section. First, we study the task of vertically jumping upwards, then we investigate forward jumping and finally we explore a challenging sequence of multiple forward jumps over obstacles.

5.2.1. A Simple Vertical Jump

The analysis of a simple vertical jump is a good example to understand the underlying challenges highly-dynamic movements bring along in the context of numerical optimization.

The vertical jumping task (see Table 5.1) consists of five phases as depicted in Fig. 5.1. From the initial position (a), a descending into the jump (b) takes place.

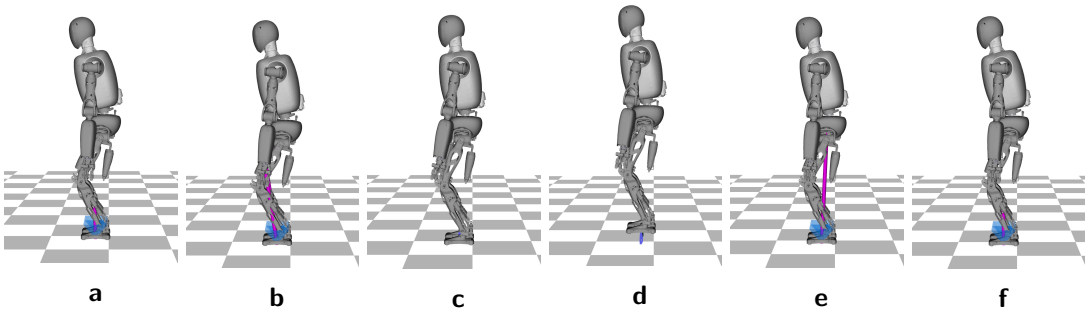


Figure 5.1.: A simple vertical jump consisting of the locomotion phases (a) initial pose, (b) descending, (c) take-off, (d) flight phase, (e) touchdown and (f) pose recovery. [Video]

Then, an upward motions accelerates the base until the take-off (c). The symmetrical flight phase (d) is ended with a touch down (e) followed by a pose recovery (f).

Fig. 5.2 provides insights in the dynamic nature of the motion. As can be seen, the joint positions stay within reasonable ranges. However, velocity peaks at the take-off exceed the maximum joint velocities of the body pitch and knee joints by a factor of two and four, respectively. This effect is plausible, since both the knee deflection as well as the torso swing are essential for a jump. Consequently, the short time horizon of the jump requires high peak velocities in these task-relevant joints.

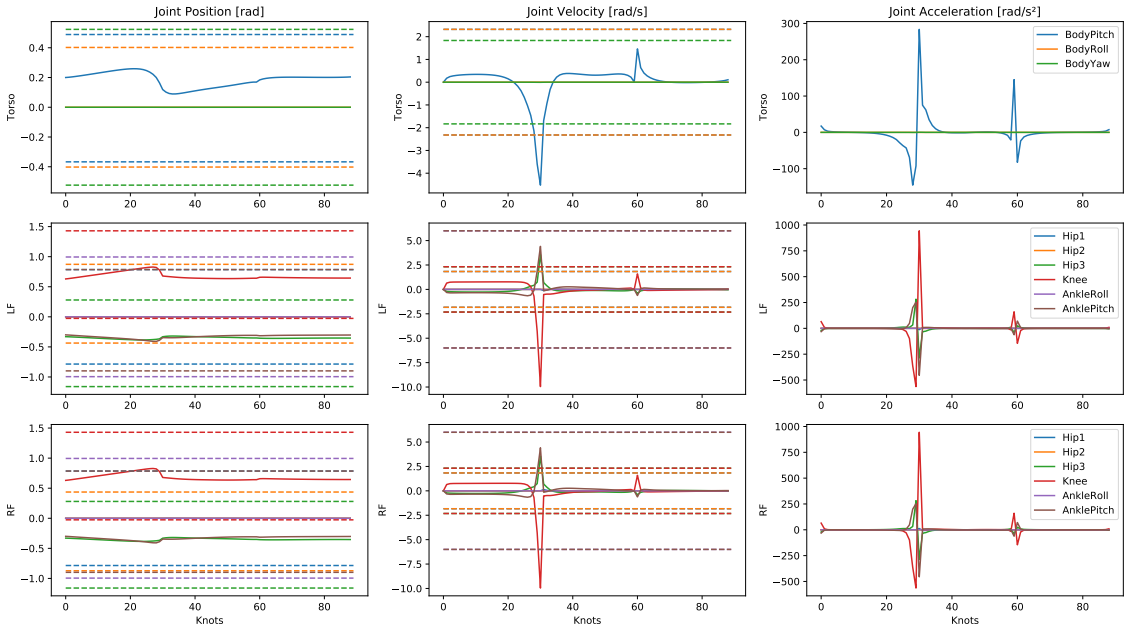


Figure 5.2.: Vertical jump solution of the joint states with according joint limits visualized as dashed lines. The maximum velocities of the body pitch and knee joints turn out to be insufficient for the highly-dynamic take-off.

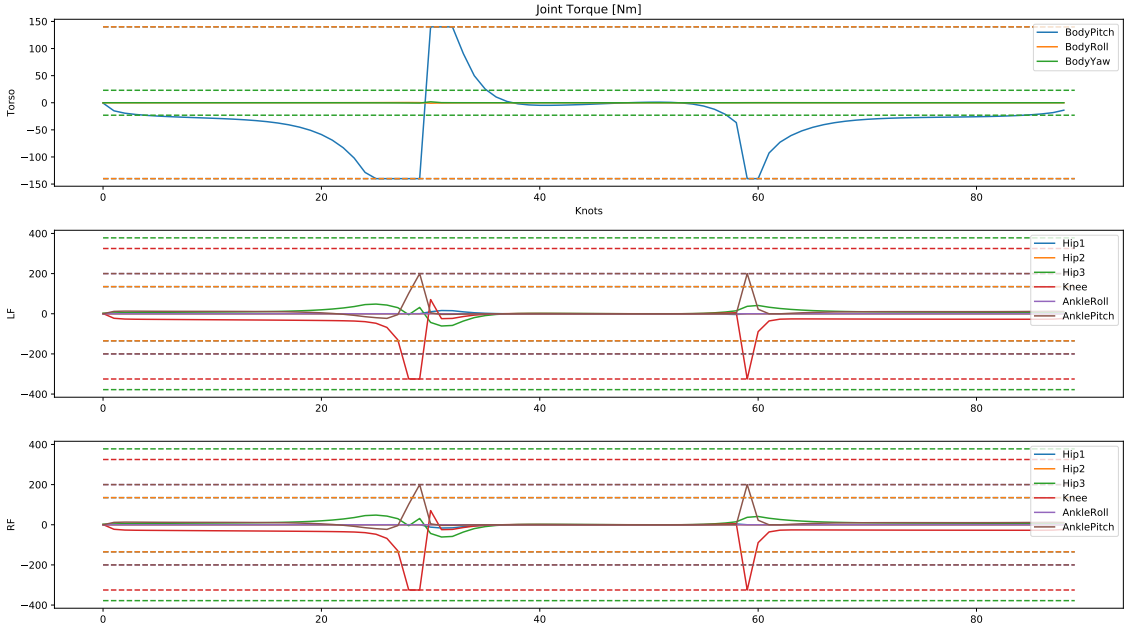


Figure 5.3.: Vertical jump solution of the joint torques with according motor limits visualized as dashed lines. The solver prevents the body pitch and knee torques to exceed the limits, but still maximum torques are required for these joints over longer time horizons.

A similar observation applies to the joint torques as shown in Fig. 5.3. Although the solver prevents the body pitch and knee torques from exceeding the limits in this case, it becomes evident that maximum torques for these joints are necessary over longer time horizons. Such continuous loads should be avoided as far as possible in order to guarantee the durability of the robotic system.

5.2.2. A Simple Forward Jump

Focus of investigation is a more dynamic forward jump to study the effect of increasing task complexity on the contact forces and stability.

Table 5.2 summarizes the characteristics and optimization constraints. The forward jumping problem consists of the same five locomotion phases as the vertical jump described previously (see Fig. 5.4). We can draw similar conclusions regarding the physical consistency of the system, namely appropriate joint position ranges and torque limits, but insufficient maximal velocities for the body pitch and knee joints.

Fig. 5.5 presents an overview about the solved optimal contact wrenches. When the robot does not move, the sum of vertical forces F_z for left and right foot equals the body's weight, which is about 600 N. It becomes evident that about 4000 N (7 x robot weight) are exerted to the ground at lift off and about 8000 N (14 x robot weight) at touchdown, which is reasonable according to biomechanical findings [70].

Table 5.2.: Forward jump characteristics and applied optimization constraints.

Jump Characteristics		**Optimization Constraints**	
Jump length:	30 cm	Tasks:	Φ_{foot}
Jump height:	10 cm	Stability:	Φ_{CoP}, Φ_{friction}
Total time:	0.9 s	Limits:	Torques
Step size:	0.01 s	Regularization:	Φ_{posture}, Φ_{torque}

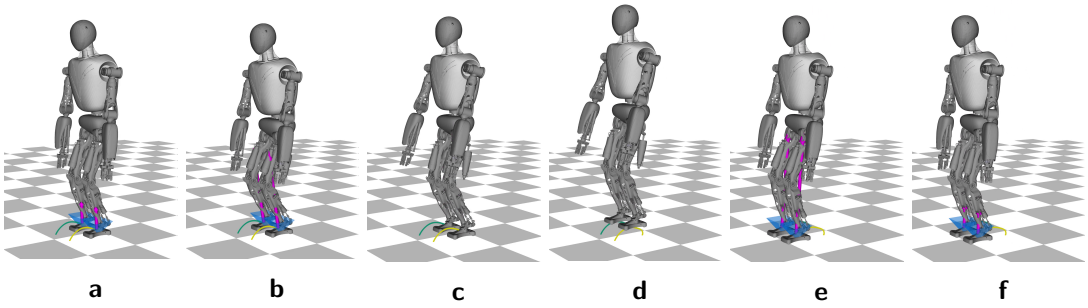


Figure 5.4.: A simple forward jump consisting of the locomotion phases (a) initial pose, (b) descending, (c) take-off, (d) flight phase, (e) touchdown and (f) pose recovery. In contrast to the vertical jump, a natural arm swinging effect emerges to gain momentum for the forward acceleration of the body. [Video]

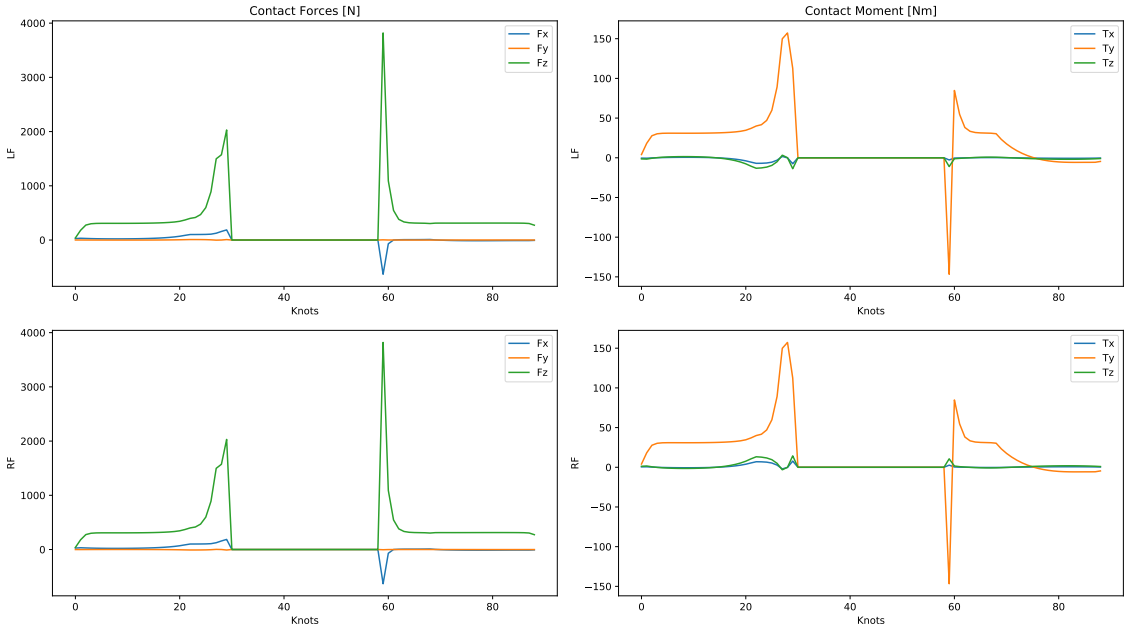


Figure 5.5.: Forward jump solution of the contact wrenches. The orders of magnitude are in accordance with biomechanical findings [70].

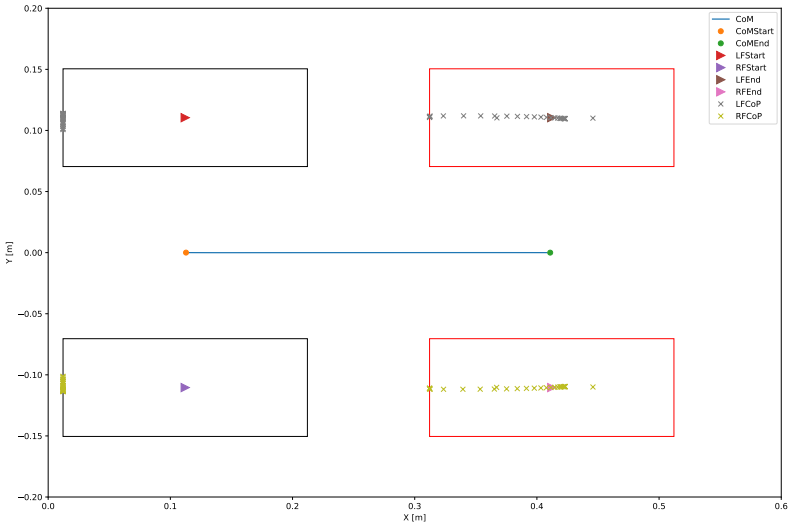


Figure 5.6.: Stability analysis of a simple forward jump shows that the motion is dynamically stable since the CoPs lie within the SP of the according foot. Consequently, the contact stability constrained DDP method also holds for highly-dynamic movements.

We have shown in Section 4.3 that the proposed motion planning approach allows computation of dynamically balanced walking gaits. In the following, we want to investigate the feasibility of contact stability for highly-dynamic movements. Fig. 5.6 shows the stability analysis for the forward jumping task. It becomes clear that the CoPs lie within the desired contact area of the left and right foot, respectively. Moreover, it can be observed that CoPs are located more in the rear area of the sole of the foot before the jump or in the front area after landing. The first effect can be explained due to the angular momentum gained during descending of the base. The latter observation accounts for the rapid deceleration of the base motion in x-direction resulting from the instantaneous impact after the flight phase.

5.2.3. Forward Jumping Over Multiple Obstacles

Finally, we want to further increase the task dynamics and investigate the proposed motion planning approach for a challenging sequence of multiple forward jumps over obstacles.

We build a sequence of OC problems, as introduced in Eq. (5.1), consisting of multiple forward jumps with increased jumping height and length (see Table 5.3), accordingly. Since the RH5 humanoid was not designed for tasks of this dynamics, neither joint nor torque limits can be satisfied. In contrast to the simple vertical and forward jumps, this case study is supposed to proof the flexibility of the proposed motion planning approach rather than analyzing the system design.

Table 5.3.: Multiple obstacles jumping characteristics and optimization constraints for a sequence of three successive jumps.

Jump Characteristics		**Optimization Constraints**	
Jump length:	60 cm	Tasks:	Φ_{foot}
Jump height:	25 cm	Stability:	Φ_{CoP}, Φ_{friction}
Total time:	0.9 s	Limits:	-
Step size:	0.01 s	Regularization:	Φ_{posture}, Φ_{torque}

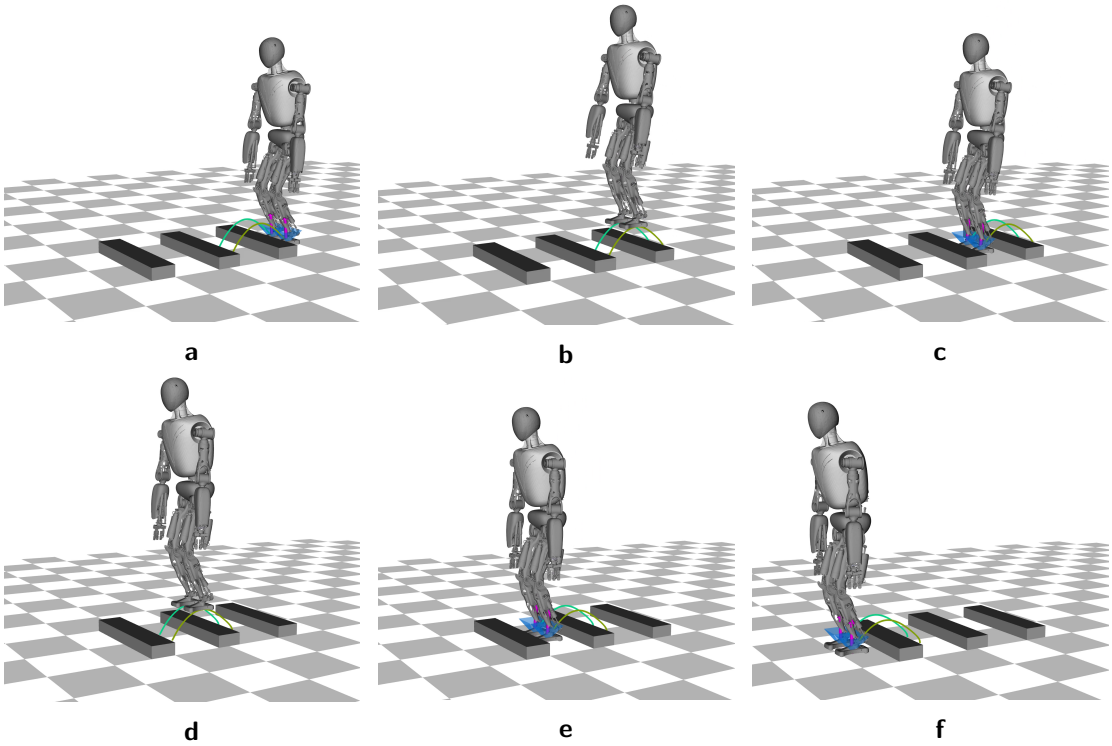


Figure 5.7.: Multi-phase OC problem of forward jumping over obstacles. Image order: row-wise, from top to bottom and left to right. [Video]

Previously, we demonstrated that the contact stability constrained DDP allows computation of a simple forward jump with dynamical balanced motions. Following up, we want to study if the concept also holds for multiple forward jumps.

Fig. 5.8 shows the obtained results for stability analysis for sequentially solving the sequence of optimization problems. The robot starts in an initial pose in DS, performs a jump and recovers to the initial pose again. This OC problem is repeated for three times with identical constraints and timings until the robot reaches the final DS pose marked by red rectangles. As becomes evident, the contact stability constrained DDP forces the CoPs of both feet to lie within the desired contact area, which proves that the motions are dynamically balanced.

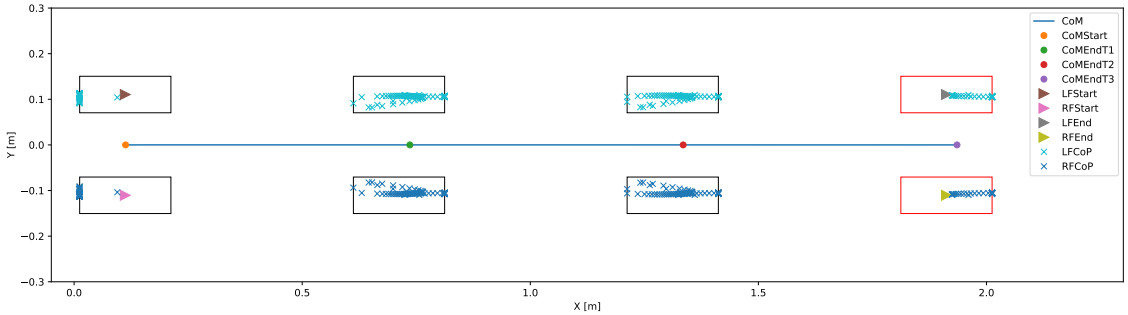


Figure 5.8.: Stability analysis of forward jumping over multiple obstacle. The results indicate that the motion is inherently balanced over the entire time horizon of the sequentially solved OC problem.

Recapitulating the results for the simple forward jump, we identified that the CoPs are located in the rear area of the foot sole during takeoff. We can draw similar conclusions for the first jump of the multi-phase OC problem. However, the CoPs for the second and third jump do not share this pattern. This effect can be attributed to the dynamic forces already acting on the robot while the second impact phase, respectively.

5.3. Evaluation of the System Design

This section presents an exhaustive study of the system limits of the RH5 humanoid robot based on the case studies introduced in the last section. The motivation is to form a basis of decision-making for future design iterations that allow the robot to perform highly-dynamic movements in real-world experiments.

The evaluation of the system design is performed by iteratively increasing the task complexity. We conduct multiple simulations with different jumping lengths l and heights h for the presented vertical jump ($h = 1 - 30$ cm), forward jump ($l = 10 - 50$ cm, $h = 10$ cm) and forward jumping task over multiple obstacles ($l = 60$ cm, $h = 25$ cm), respectively (see Table 5.4). Of particular interest are the system-relevant limits, namely the valid joint position and velocity ranges as well as the according maximum motor torques.

A few notes on the used notation in Table 5.4: checkmarks and crosses indicate whether the according limits are satisfied or not with the given movement. Checkmarks in brackets mean that a maximum torque needs to be applied for a period longer than 50 , which can not necessarily be provided by the real actuators. Underscored numbers along with the crosses depict how many joints the according limit is exceeded by.

In the following, the results of the study are presented in detail and design guidelines are derived based on the conducted experiments.

Table 5.4.: Capabilities of the RH5 humanoid to perform various highly-dynamic jumps (length l, height h) respecting the hardware limits, namely available motor torque and valid joint position and velocity ranges. While position limits are satisfied for all types of jumps, the velocity limits turn out to be insufficient. Furthermore, most jumps require a maximum torque applied for a period longer than 50 ms, which cannot necessarily be provided by the real actuators.

	Position Limits	Torque Limits	Velocity Limits
Vertical Jump ($l = 0$ cm)			
$h = 1$ cm	✓	✓	✓
$h = 5$ cm	✓	✓	$\times_3$
$h = 10$ cm	✓	(✓)	$\times_3$
$h = 20$ cm	✓	(✓)	$\times_5$
$h = 30$ cm	✓	(✓)	$\times_7$
Forward Jump ($h = 10$ cm)			
$l = 10$ cm	✓	(✓)	$\times_7$
$l = 20$ cm	✓	(✓)	$\times_7$
$l = 30$ cm	✓	(✓)	$\times_7$
$l = 40$ cm	✓	(✓)	$\times_7$
$l = 50$ cm	✓	(✓)	$\times_7$
Obstacle Jump ($h = 25$ cm)			
$l = 60$ cm	✓	$\times_5$	$\times_7$

5.3.1. Design Guidelines derived from Vertical Jumping

The vertical jumping tasks already gives good insights in the dynamic capabilities of the humanoid robot RH5. From Table 5.4 we can see that the smallest jump with a height of 1 cm can be realized with the system. This is remarkable, since the current design of the humanoid is not indented for the purpose of highly-dynamic movements.

For jumps with a height of 5 cm or above we can draw the conclusion that velocity limits cannot be satisfied anymore. In the first case, body pitch and knee velocities are exceeded, which is reasonable since they form the main part of the motion. For the highest jump on the other hand, additional limits for the ankle pitch joints and the shoulder roll joints turn out to be not sufficient anymore. These violations can be explained by the higher impact forces and increased arm swinging activities.

5.3.2. Design Guidelines derived from Forward Jumping

In comparison to the simple vertical jump, the forward jumping tasks involve higher dynamic forces acting on the base and hence require even faster motions. To this end, none of the forward jumping tasks turned out to be feasible with the currently

implemented velocity limits. Interestingly, the ankle pitch velocity turned out to be sufficient, while the hip joint velocity limits were exceeded. This effect may be explained by the reduced impact forces in z-direction and the increased angular momentum necessary to move the base forward.

Furthermore, we can draw similar conclusions for the torque feasibility from the tasks of vertical, forward and multiple jumping over obstacles. Although the maximum torque available with the currently embedded motors are high enough, they may not be feasible with the real actuators. For the body pitch, knee and ankle pitch joints, these maximal torques have to be applied for a period longer than 50 ms. To prevent long-term damage to the system, the corresponding actuators should be redesigned with an appropriate safety factor.

CHAPTER 6

Validation of Planned Motions

This chapter presents a two-step validation of the physical consistency of the planned motions where an online stabilization is applied to track the optimal trajectories. At first, we prove the stability of the motions in a real-time physics simulation. Following this, we explore the feasibility of the motions in real-world experiments on a full-size humanoid robot.

6.1. Validation in Real-Time Physics Simulation

This section investigates the stability of the planned motions in a physics simulator by applying an online stabilization based on joint space position control to track the trajectories obtained from OC. To begin with, an overview of the simulation setup is given, then the tracking results of the planned motions are discussed.

6.1.1. Simulation Setup

The optimal motions are tested in the dynamical simulation environment PyBullet [71]. PyBullet is an open-source framework for robotics simulation that allows fast computation of rigid-body dynamics along with collision detection. The motion tracking in the simulator is set up in a similar way the motions would be tested on a real robot, namely interpolating the trajectories and closing the loop on joint position level.

The control loop on the real system (see Section 6.2) is running at a frequency of 1 kHz. The generated bipedal walking variants and highly-dynamic movements presented in Chapters 4 to 5 are generated with a discrete OC formulation at 30 Hz and 10 Hz, respectively. To this end, the optimal trajectories are interpolated with a cubic spline in order to realize an up-scaling of the reference data to 1 kHz.

The control architecture consists of a simple PD-controller on joint space level. The controller is supposed to track both position and velocity reference trajectories with the standard PyBullet parameters in a real-time loop running at 1 kHz. In this real-time simulation, the same URDF robot model is used as in the Crocoddyl framework, including the same maximal motor torques. Furthermore, the parameters of the rigid contact models have been aligned between both frameworks to ensure comparability.

6.1.2. Results

Following up, we investigate the capabilities of the presented control architecture for tracking the planned motions. To this end, we study the control tracking performance for the dynamic walking gait (Section 4.2) and a highly-dynamic forward jump (Section 5.2).

Dynamic Walking

To begin with, we analyze the motion tracking for dynamic walking gait. Fig. 6.1 shows the tracking performance of the joint level control architecture. The reference trajectories from OC are visualized as solid lines, while the resulting trajectories of the real-time physics simulation are shown as dotted lines. It can be seen that the controller follows the optimal trajectories well. Small deviations can be seen around two seconds, which accounts for the lift off phase for the second robot step. This effect can be explained by the apparent abrupt change in the joint space but is found to be marginal for the overall tracking performance. As introduced previously, the control architecture is solely based on joint space position and velocity level. Although the tracking performance is good, this may not prove for the stability of the motions. Similar to the definition of robot tasks, the evaluation of the motions should be pursued in task space. To this end, Fig. 6.2 monitors the according motion of the floating base. As can be seen, the floating base deviates about $\pm$ 10 mm in x- and y-direction as well as + 5 mm in z-direction.

In this context, it is important to notice that no controller tracks these task space quantities. Instead, they are merely the result of the joint space control performance. The largest deviations in task space occur during the first step and the according impact in the first two seconds of the motions. It becomes evident that these task space errors do not correlate to the peaks discussed on joint space level. Furthermore, one can observe oscillations in the stabilization phase at the end of the motion. This effect can be attributed to the fact that the systems slightly starts to swing after the second impact. This behavior could be compensated e.g. with a dedicated control in task space instead of joint space.

Overall, it can be stated that the dynamic walking motion can be successfully stabilized by the proposed control approach.

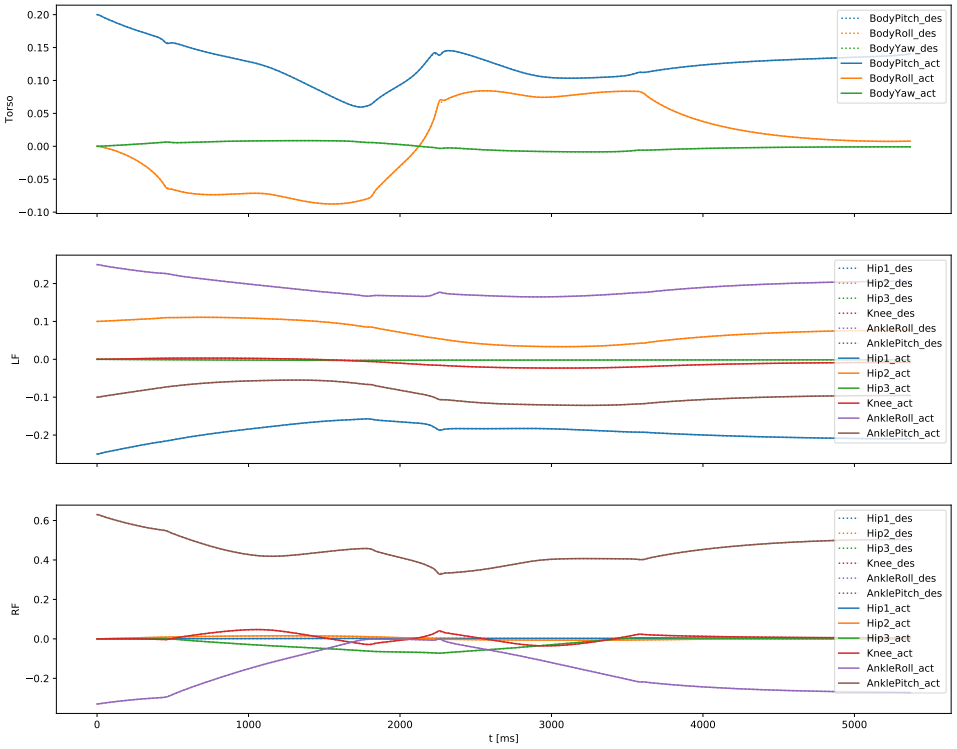


Figure 6.1.: Control tracking performance on joint level for dynamic walking. The results indicate that the control architecture follows the reference trajectories with sufficiently good accuracy.

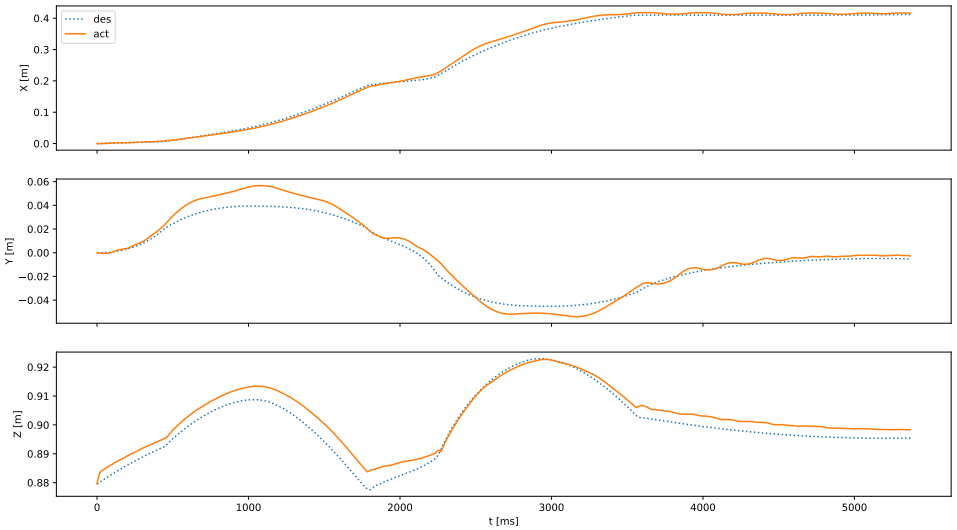


Figure 6.2.: Motion of the floating base resulting from joint space control for the dynamic walking gait. The results show a deviation of the floating base about $\pm$ 10 mm in x- and y-direction, as well as + 5 mm in z-direction.

Forward Jumping

To identify the limits of the control approach, we now study the online stabilization for a highly-dynamic forward jump (Section 5.2). In addition to the previously discussed dynamic walking, the jumping task introduces new challenges for the controller in terms of speed and robustness that will be investigated in the following.

Fig. 6.3 shows the tracking performance of the joint level control architecture for the forward jump. In contrast to the case of dynamic walking, the joint space controller reveals larger tracking deviations. This is especially true for the most dynamic part of the motion, namely the acceleration of the base and finally the takeoff around 300 to 400 ms. Large deviations can be especially seen for body pitch, and the knee joints, which turned out to be crucial for highly-dynamic movements.

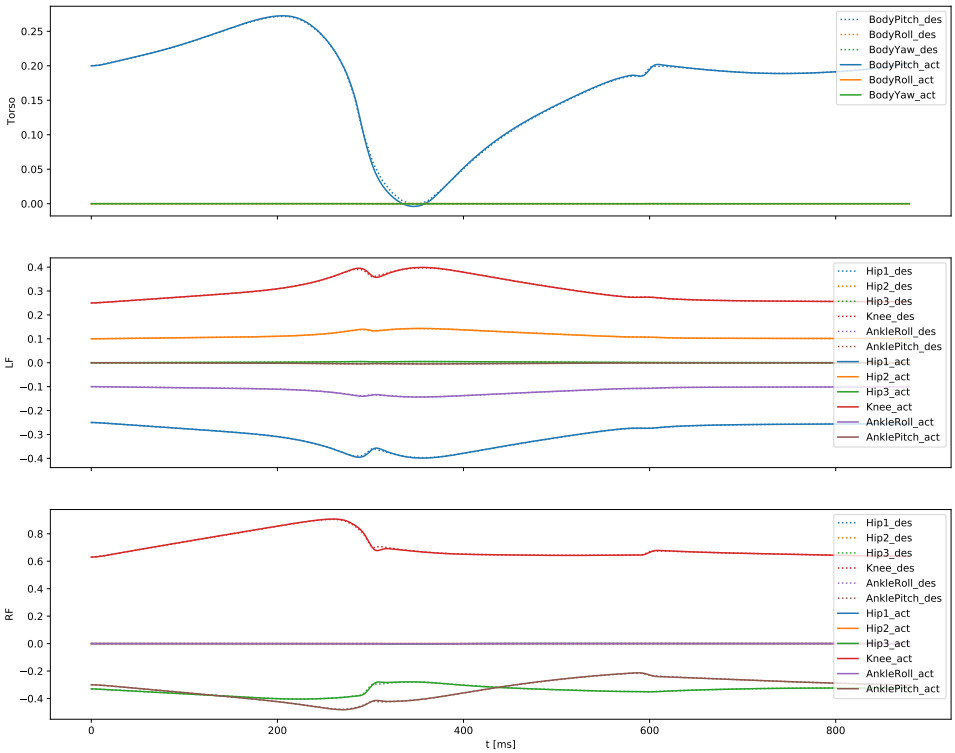


Figure 6.3.: Control tracking performance on joint level for a forward jump. The results indicate that the control architecture also is suitable to track the reference trajectories of highly-dynamic movements with sufficiently good accuracy.

In addition to these findings, Fig. 6.4 monitors the motion of the (uncontrolled) floating base resulting from the joint space control performance. As becomes evident, the deviations in task space are also much higher compared to the dynamics walking case. While the height of the floating base is reasonable, the x-position shows tracking errors of about $\pm$ 5 cm. Errors of this magnitude inevitably lead to instability of the movement to be performed. In this case, the strong deviation

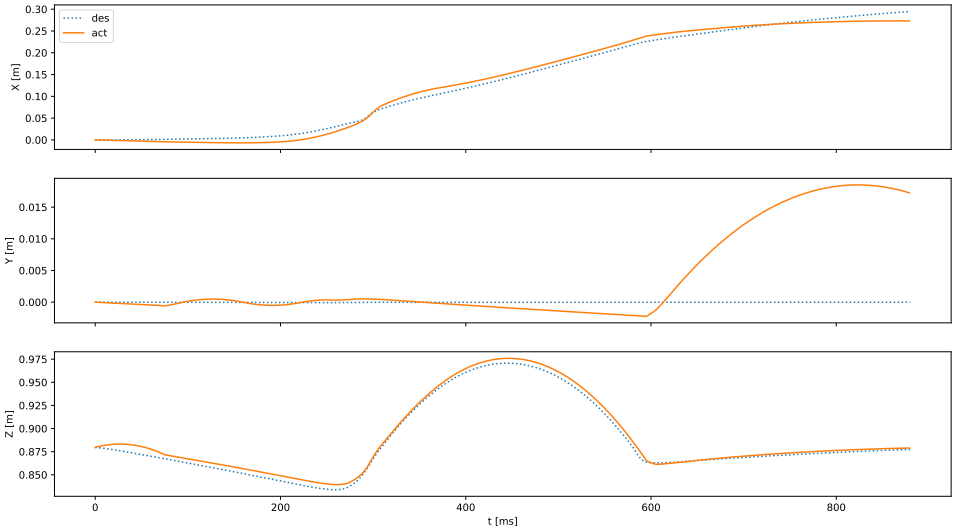


Figure 6.4.: Motion of the floating base resulting from joint space control for a forward jump. Although the tracking in x- and y-direction is appropriate, the y-direction of the floating base drifts about ± 10 mm apart.

in task space causes the robot to tilt around the rear edge of the foot after the touchdown.

Consequently, it turned out that a mere control on a joint space basis is not sufficient to track highly dynamic movements but is indeed appropriate to stabilize a dynamic walking gait in real-time.

6.2. Validation in Real-World Experiments

This section investigates the stability of the planned motions in real-world experiments on the full-size humanoid robot RH5 (see Section 1.2.4). Analogously to the validation in PyBullet, the goal is to track the OC trajectories with a joint space online stabilization on the real system.

A total of four experiments are conducted. The primary goal is to examine whether the presented control architecture is sufficient to follow the static and dynamic walking trajectories (see Chapters 4 to 5). For a dedicated root cause analysis, we test two additional motions on the real-system, namely one-legged balancing and a sequence of dynamic squat movements. The respective goal is to identify potential model deviations and to study the capabilities of the real-time controller.

Based on the simulation results from the last section, highly dynamic movements are not evaluated on the real system in the context of this thesis. As discussed in the last section, these motions definitely require more advanced control algorithms in task space due to the dynamic nature of the movement.

The rest of this section is structured as follows. First, an overview of the experimental setup with the involved components is presented. Following up, the

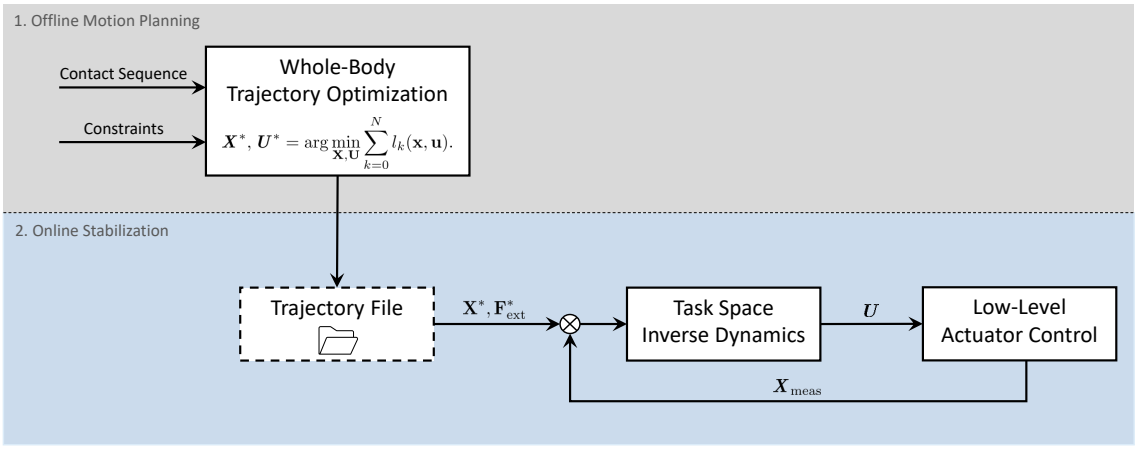


Figure 6.5.: Overview about the experimental pipeline. The offline planned motions are tracked in real-time with a joint space online stabilization on the real system.

experiments are presented and discussed with respect to the tracking performance and the stability of the motion.

6.2.1. Pipeline

The foundation for the experiments are the motion data generated offline with the proposed whole-body TO. The planned motions are then tracked in real-time with a joint space online stabilization on the real system. An overview of the experimental pipeline is given in Fig. 6.5. In the following, details on the involved components are provided. The presented motion planning approach computes inherently balanced motions that are concisely captured in an appropriate file. This trajectory file contains the optimal state trajectories $\boldsymbol{X}^* = [\boldsymbol{q}^*, \boldsymbol{v}^*]$, optimal control inputs $\boldsymbol{U}^*$ and the resulting contact wrenches $\boldsymbol{F}^*_{\text{ext}}$ acting on the feet. In order to minimize the computational effort in the real-time loop, the file already encompasses data discretized to the desired frequency of 1 kHz. As in the PyBullet validation, the trajectories are interpolated using cubic splines in order to ensure smoothness and differentiability.

As introduced in Section 1.2.4, the novel RH5 humanoid robot contains multiple parallel mechanisms in order to achieve a high dynamic performance, superior stiffness and payload-to-weight ratio. This leads to the presence of various closed loops and hence a series-parallel hybrid robotic system, which is difficult to model and control. In most multi-body dynamics libraries, e.g. RBDL [72] and OpenSim [73], these loop closure constraints are solved numerically. HyRoDyn (Hybrid Robot Dynamics), is a recently presented modular software framework for solving the kinematics and dynamics of these type of series-parallel hybrid robots analytically, leading to improved accuracy and computational performance [74].

The planned motions are computed based on a tree type robot model. For dynamic real-time control, this simplified model turns out to be sufficient, although

Table 6.1.: Overview about the difficulty of the conducted experiments. Three out of four planned motions could be successfully stabilized by the controller on the real system.

	Balancing	Static Walk	Fast Squats	Dynamic Walk
Surface contacts	✓	✓	✓	✓
Base motion	✓	✓	✓	✓
Swing foot motion	✓	✓	✗	✓
Step sequence	✗	✓	✗	✓
Impacts	✗	✓	✗	✓
Dynamic forces	✗	✗	✓	✓
Flight-phases	✗	✗	✗	✗
Success	✓	(✓)	✓	✗

the accuracy is reduced [46]. Nevertheless, the problem remains on transforming the results from the independent joint space, to the actuation space. In the context of this thesis, HyRoDyn is used to map the trajectories generated for the serialized robot model to compute the forces of the respective linear actuators.

Consistency of the frameworks involved in the motion planning and control pipeline is an indispensable prerequisite for the following experiments. Hence, Appendix A.3 provides a brief verification of the notability consistency between HyRoDyn and Pinocchio [75], which is used inside Crocoddyl for computation of robot dynamics and their analytical derivatives.

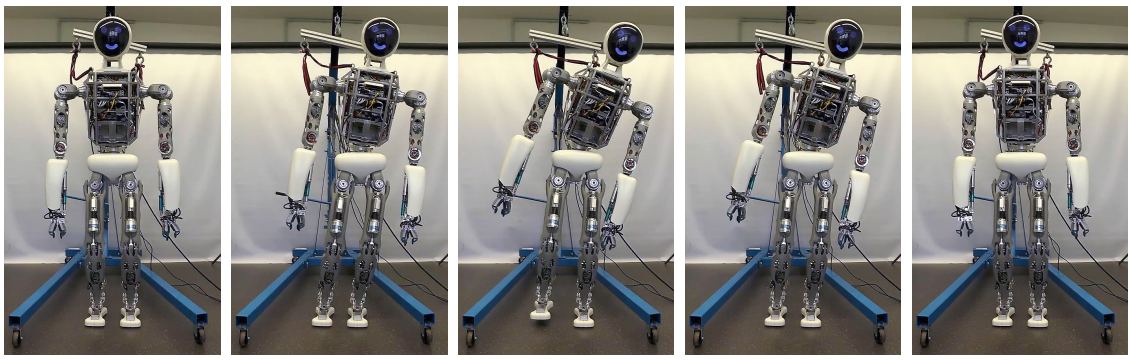
Low-level actuator controllers are utilized to compensate deviations from the reference trajectories. Analogously to the PyBullet validation, this control approach uses a cascaded feedback of position, velocity and an additional current control loop.

6.2.2. Experiments

We conduct a total of four experiments, gradually incorporating a new level of difficulty (see Table 6.1), that are introduced in the following.

Experiment I: One-Leg Balancing

The goal of this first experiment is to test the ability of the control architecture to track a slow balancing task. The quasi-static motion consists of five locomotion phases as visualized in Fig. 6.6. From (a) a stable pose in DS, (b) shifting the CoM above the LF (c) lifting the RF slightly up (d) and down and (e) returning to the initial pose.

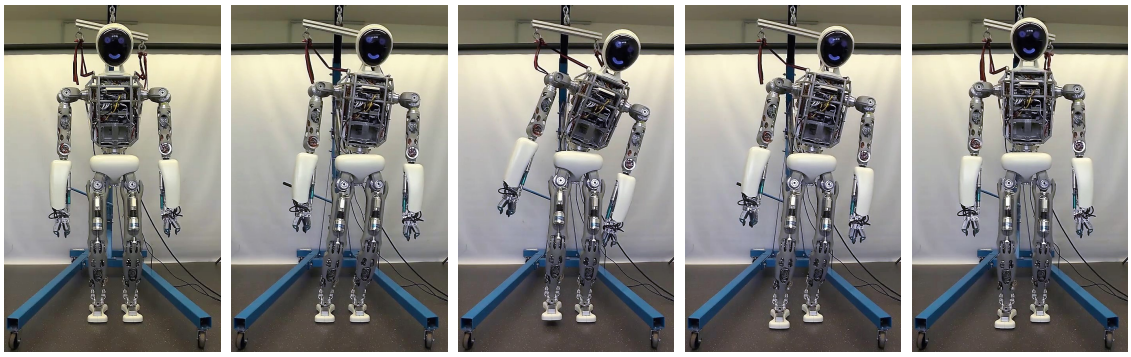


a b c d e

Figure 6.6.: Experiment I: One-leg balancing from (a) an initial pose, (b) CoM shift above the LF, (c) lifting the RF up and (d) down and (e) recovering to the initial pose. [Video]

Experiment II: Static Walking

The second experiment deals with a stabilization of the static walking pattern discussed in Section 4.2 (see Fig. 6.7). The objective of this test is to analyze the effect of more difficult swing-leg motions, a step sequence of two steps and the effect of impacts.



a b c d e

Figure 6.7.: Experiment II: Static walking from (a) an initial pose, (b) CoM shift above the LF, (c,d) performing a right step and (e) shifting the CoM to the center of the SP. [Video]

Experiment III: Fast Squats

The objective of this third experiment is to evaluate the tracking performance in the context of a dynamic motion. In contrast to the first two motions, the fast squatting experiment (see Fig. 6.8) involves dynamic forces acting on the robot resulting from a fast vertical base movement in the range of 15 cm within two seconds. Analogously to the multiple forward jumps (see Section 4.2), the motion

is composed of a sequence of three OC problems that are solved sequentially (see Eq. (5.1)). This experiment can be seen as preliminary test for dynamic walking following in the next experiment.

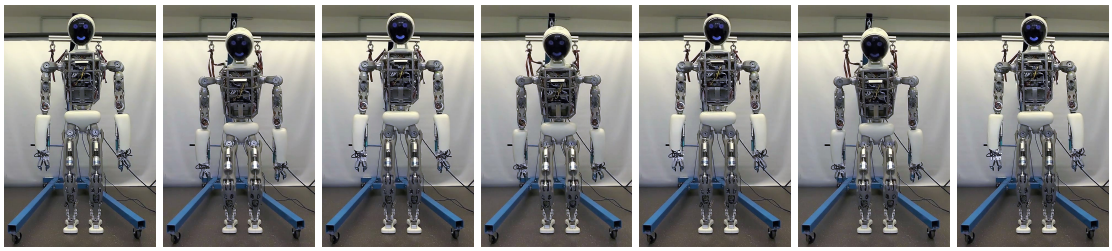


Figure 6.8.: Experiment III: Sequence of fast squats from (a) an initial pose over, (b,d,f) descending the CoM by 15 cm and (c,e,g) recovering to the initial pose. [Video]

Experiment IV: Dynamic Walking

This experiment investigates the capabilities of the control approach to follow the dynamic walking gait trajectories presented in Section 4.2 (see Fig. 6.9). From a perspective of complexity, it combines both the challenges of static walking, namely dedicated swing-leg motions, step sequence and impacts, with the difficulty of dynamic forces acting on the robot as explored with the fast squats experiment. eo

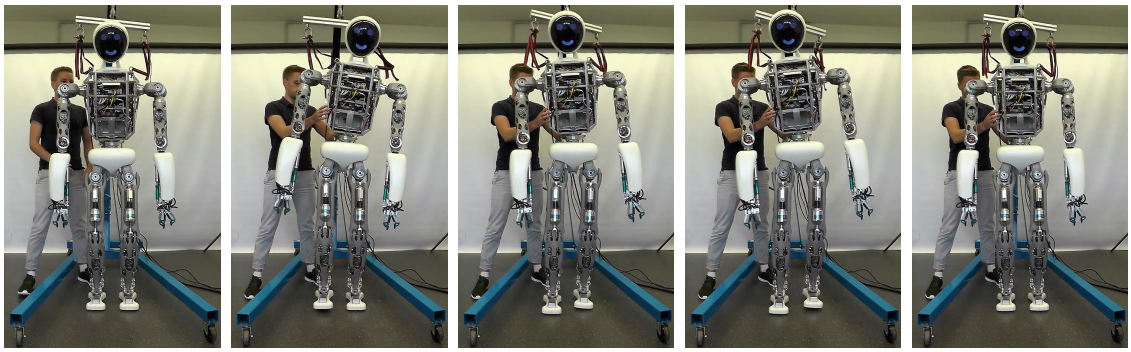


Figure 6.9.: Experiment IV: Dynamic walking from (a) an initial pose, (b,c) performing a right step and (e,f) a successive left step. [Video]

6.2.3. Discussion

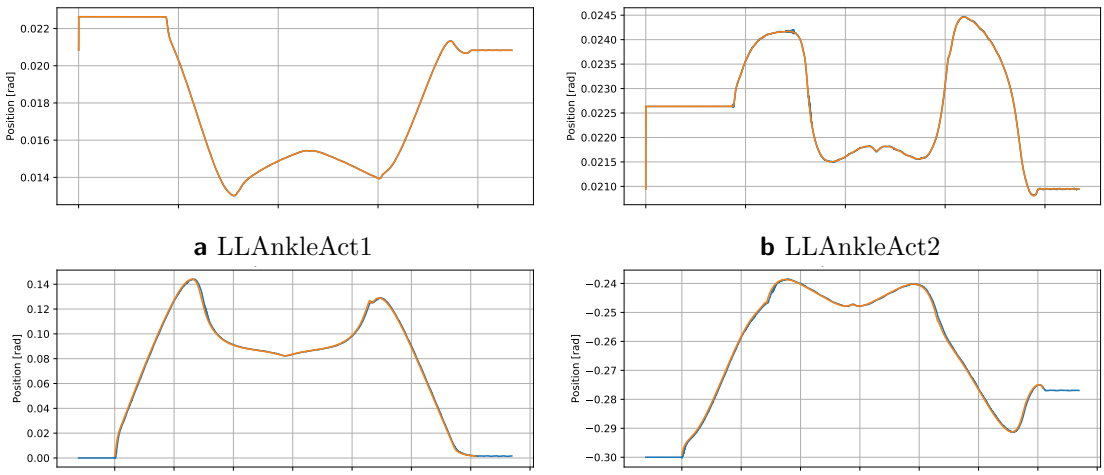
The following section analyzes the experimental results, highlighting limits of the utilized control architecture.

Stability of the Motions

All in all, three out of four planned motions could be successfully stabilized by the controller on the real system (see Table 6.1). Both the one-leg balancing experiment (I) and the fast squats experiment (III) were stabilized with good accuracy. For experiment II, static walking, only one step could be stabilized at a time. Furthermore, it could be shown that dynamic walking (IV) cannot be stabilized with the present control approach. Possible root causes are outlined in the following.

Tracking Performance

As detailed in Section 6.2.1, the online stabilizer solely works in joint space. The experimental results indicate that the overall joint space tracking for all four experiments is satisfying. Fig. 6.10 exemplary shows the tracking performance for the one-leg balancing experiment. It becomes evident that the control architecture allows following the computed reference trajectory in actuator space (a,b) closely. Furthermore, also the resulting tracking quality in the virtual joints (c,d) is appropriate. Similar conclusions hold for the other experiments. Even during critical impulse phases the joint space tracking shows no abnormalities. This precise tracking is achieved with high-gain joint space control, which allows a quick compensation of position differences that comes at the cost of lost compliance in the joints.



a LLAnkleAct1 **b** LLAnkleAct2

c LLAnkleRoll **d** LLAnklePitch

Figure 6.10.: Tracking performance for the one-leg balancing experiment in actuator space (a,b) and independent joint space (c,d). The reference position from OC in orange and the measured position in blue are plotted against the time. The results indicate that the control architecture tracks the reference trajectories with sufficient accuracy.

Handling Impulses

The impact phase turned out to be the main problem for the walking experiments. This is reasonable since the utilized control approach only compensates for errors in joint space, while errors in task space can arise quickly and are not compensated. Deviations of the reference base motion only effect the static stability if the FCoM leaves the support polygon. Differences from the foot position instead may cause instability with a respective error in the range of a few millimeters. The same holds for errors in the foot orientation. Consider for example the case in which the robot only touches the ground with the rear part of the swing foot's sole. Instead of the assumed surface contact between the foot and the ground, a line contact appears that causes the robot to fall in a combination with high-gain position control.

Model Discrepancies

The quasi-static experiments provide a good basis of analyzing the model accuracy. Due to the small SP, model deviations inevitably lead to instability during the single support phases. In the balancing and static walking experiments the motions could be stabilized only if the FCoM was adjusted by about $\pm$ 7.5 mm. The found model errors in the mass distribution, possibly combined with with errors in inertia distribution, may also pose part of the problem for walking gait stabilization. In order to compensate for these differences, an extensive system identification could be performed to check the dynamic model against the real system.

Mechanical Deficiencies

Beneath the identified limits of the control approach and the modeling errors, also mechanical weaknesses of the RH5 humanoid may contribute to the sim-to-real gap. In the scope of the experiments, several issues have been revealed, namely joint play, calibration errors and undesired structural flexibility in the ankle of the foot. Furthermore, the rigid contact points on the sole of the feet are found to be not the ideal solution, since small deviations can quickly lead to instability of the whole system. Instead, it may be advantageous for dynamic movements of the humanoid robot to use a planar sole, providing adequate damping properties.

CHAPTER 7

Conclusion and Outlook

7.1. Thesis Summary

This master's thesis was motivated by the generation of physically consistent, efficient motion plans for legged robots.

The premise of the investigation was that whole-body planning leads to more efficient motions than a simple program such as an IK solver. Therefore, the core algorithm of the proposed motion planning approach was a recently presented DDP-based whole-body TO. Building upon this, a generic method for constraining DDP-like solvers was presented to generate dynamically balanced motions. The results were integrated into the recently presented open-source framework Crocoddyl.

Following this, we investigated the CoP-based contact stability of the proposed motion planning approach for a wide range of motions with the biologically inspired RH5 humanoid robot. We were able to demonstrate that the resulting motion plans for both dynamic bipedal walking and various jumping tasks are inherently balanced. Additionally, the analysis of highly-dynamic movements allowed the derivation of useful guidelines for future design iterations of the humanoid robot.

Although the focus of this thesis was on motion planning, we evaluated the feasibility of the generated trajectories with a simple online stabilizer. We demonstrated in a real-time physics simulator that the motion plans can be stabilized by a simple control architecture solely based on joint-space position control. Furthermore, it could be shown that for real-world experiments, a control in the task space is indispensable to compensate for deviations between the model and reality.

The final result of this thesis is an efficient motion planning approach that produces inherently balanced motions. This algorithm efficiently generates highly-dynamic movements with flight-phases for various legged systems.

7.2. Future Directions

We see large potential in using DDP-based whole-body TO to generate motions for legged systems. Motion planning based on numerical optimization clearly reduces the number of hand-crafted components and allows the specification of high-level tasks directly in the operational space of the robot. This is likely to become even more important as legged robots tackle more difficult terrains that require higher dynamic motions.

From an algorithmic perspective, the formulation can still be improved in a number of ways. The goal of the algorithm is to efficiently and accurately generate physically consistent motions. Promising ways of simultaneously improving both measures are seen by directly embedding inequality constraints as strict bounds inside the DDP algorithm, instead of forcing them by penalization.

From a control perspective, two successive steps are of particular interest. First, working on an improved online stabilization is a worthwhile undertaking. As discussed in the last chapter, operational space control is inevitable to compensate for modeling errors directly in the task space. Following up, it would be interesting to embed the motion planning inside a Model Predictive Control (MPC) formulation. If the re-planning is quick enough, the robot will be enabled to act more robustly to unpredicted situations.

This direction for future research, namely improving the motion planning algorithm and embedding it in an MPC formulation on a real system, seems promising. Intelligently combining optimization-based planning and control may be the key to robots interacting with both the environment and humans in a more natural, dynamic and autonomous way. With some work in this direction, we may soon find legged robots crossing our daily paths or intuitively collaborating with us when assembling infrastructures and exploring foreign planets.

APPENDIX A

Appendix

A.1. CoP Cost Implementation for Contact Dynamics

This section contains the implementation of the CoP cost for contact dynamics action models, integrated into the open-source framework Crocoddyl within the context of this thesis.

For space-saving reasons, only the two core files are presented, each with shortened comments. The complete versions of these files, associated Python bindings, related files and a functional unit test can be traced in the associated pull request (#792).

A.1.1. contact-cop-position.hpp

```
///////////////////////////////////////////////////////
// BSD 3-Clause License
//
// Copyright (C) 2020, University of Duisburg-Essen,
// University of Edinburgh
// Copyright note valid unless otherwise stated in
// individual files.
// All rights reserved.
///////////////////////////////////////////////////////

#ifndef
   CROCODDYL_MULTIBODY_COSTS_CONTACT_COP_POSITION_HPP_
#define
   CROCODDYL_MULTIBODY_COSTS_CONTACT_COP_POSITION_HPP_
```

```
#include "crocoddyl/multibody/fwd.hpp"
#include "crocoddyl/multibody/cost-base.hpp"
#include "crocoddyl/multibody/contact-base.hpp"
#include "crocoddyl/multibody/contacts/contact-3d.hpp"
#include "crocoddyl/multibody/contacts/contact-6d.hpp"
#include "crocoddyl/multibody/data/contacts.hpp"
#include "crocoddyl/multibody/frames.hpp"
#include "crocoddyl/multibody/data/multibody.hpp"
#include "crocoddyl/core/activations/quadratic-barrier.
   hpp"
#include "crocoddyl/core/utils/exception.hpp"

namespace crocoddyl {

/**
 * @brief Define a center of pressure cost function
 */
template <typename _Scalar>
class CostModelContactCoPPositionTpl : public
   CostModelAbstractTpl<_Scalar> {
 public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW
  typedef _Scalar Scalar;
  typedef MathBaseTpl<Scalar> MathBase;
  typedef CostModelAbstractTpl<Scalar> Base;
  typedef CostDataContactCoPPositionTpl<Scalar> Data;
  typedef StateMultibodyTpl<Scalar> StateMultibody;
  typedef CostDataAbstractTpl<Scalar> CostDataAbstract;
  typedef ActivationModelAbstractTpl<Scalar>
     ActivationModelAbstract;
  typedef ActivationModelQuadraticBarrierTpl<Scalar>
     ActivationModelQuadraticBarrier;
  typedef ActivationBoundsTpl<Scalar> ActivationBounds;
  typedef DataCollectorAbstractTpl<Scalar>
     DataCollectorAbstract;
  typedef FrameCoPSupportTpl<Scalar> FrameCoPSupport;
  typedef typename MathBase::Vector2s Vector2s;
  typedef typename MathBase::Vector3s Vector3s;
  typedef typename MathBase::VectorXs VectorXs;
  typedef typename MathBase::MatrixXs MatrixXs;
  typedef typename MathBase::MatrixX3s MatrixX3s;
  typedef Eigen::Matrix<Scalar, 4, 6> Matrix46;

  /**
```

```
 * @brief Initialize the cop cost model
 */
CostModelContactCoPPositionTpl(boost::shared_ptr<
    StateMultibody> state, boost::shared_ptr<
    ActivationModelAbstract> activation, const
    FrameCoPSupport& cop_support, const std::size_t& nu)
    ;

/**
 * @brief Initialize the cop cost model
 */
CostModelContactCoPPositionTpl(boost::shared_ptr<
    StateMultibody> state, boost::shared_ptr<
    ActivationModelAbstract> activation, const
    FrameCoPSupport& cop_support);

/**
 * @brief Initialize the cop cost model
 */
CostModelContactCoPPositionTpl(boost::shared_ptr<
    StateMultibody> state, const FrameCoPSupport&
    cop_support, const std::size_t& nu);

/**
 * @brief Initialize the cop cost model
 */
CostModelContactCoPPositionTpl(boost::shared_ptr<
    StateMultibody> state, const FrameCoPSupport&
    cop_support);
virtual ~CostModelContactCoPPositionTpl();

/**
 * @brief Compute the cop cost
 */
virtual void calc(const boost::shared_ptr<
    CostDataAbstract>& data, const Eigen::Ref<const
    VectorXs>& x, const Eigen::Ref<const VectorXs>& u);

/**
 * @brief Compute the derivatives of the cop cost
 */
virtual void calcDiff(const boost::shared_ptr<
    CostDataAbstract>& data, const Eigen::Ref<const
    VectorXs>& x, const Eigen::Ref<const VectorXs>& u);
```

```
  /**
   * @brief Create the cop cost data
   */
  virtual boost::shared_ptr<CostDataAbstract> createData(
      DataCollectorAbstract* const data);

 protected:
  /**
   * @brief Return the cop
   */
  virtual void set_referenceImpl(const std::type_info& ti
      , const void* pv);

  /**
   * @brief Modify the cop
   */
  virtual void get_referenceImpl(const std::type_info& ti
      , void* pv) const;

  using Base::activation_;
  using Base::nu_;
  using Base::state_;
  using Base::unone_;

 private:
  FrameCoPSupport cop_support_;  //!< frame name of the
      contact foot and support region of the cop
};

template <typename _Scalar>
struct CostDataContactCoPPositionTpl : public
    CostDataAbstractTpl<_Scalar> {
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW
  typedef _Scalar Scalar;
  typedef MathBaseTpl<Scalar> MathBase;
  typedef CostDataAbstractTpl<Scalar> Base;
  typedef DataCollectorAbstractTpl<Scalar>
      DataCollectorAbstract;
  typedef FrameCoPSupportTpl<Scalar> FrameCoPSupport;
  typedef typename MathBase::Vector3s Vector3s;
  typedef typename MathBase::VectorXs VectorXs;
  typedef typename MathBase::MatrixXs MatrixXs;
  typedef typename MathBase::Matrix3s Matrix3s;
```

```
typedef typename MathBase::Matrix6xs Matrix6xs;
typedef typename MathBase::Matrix6s Matrix6s;
typedef typename MathBase::Vector6s Vector6s;

template <template <typename Scalar> class Model>
CostDataContactCoPPositionTpl(Model<Scalar>* const
   model, DataCollectorAbstract* const data)
    : Base(model, data), Arr_Ru(model->get_activation()
       ->get_nr(), model->get_state()->get_nv()) {
  Arr_Ru.setZero();

  // Check that proper shared data has been passed
  DataCollectorContactTpl<Scalar>* d = dynamic_cast<
     DataCollectorContactTpl<Scalar>*>(shared);
  if (d == NULL) {
    throw_pretty("Invalid␣argument:␣the␣shared␣data␣
       should␣be␣derived␣from␣DataCollectorContact");
  }

  // Get the active 6d contact (avoids data casting at
     runtime)
  FrameCoPSupport cop_support = model->template
     get_reference<FrameCoPSupport>();
  std::string frame_name = model->get_state()->
     get_pinocchio()->frames[cop_support.get_id()].name
     ;
  bool found_contact = false;
  for (typename ContactModelMultiple::
     ContactDataContainer::iterator it = d->contacts->
     contacts.begin();
       it != d->contacts->contacts.end(); ++it) {
    if (it->second->frame == cop_support.get_id()) {
      ContactData3DTpl<Scalar>* d3d = dynamic_cast<
         ContactData3DTpl<Scalar>*>(it->second.get());
      if (d3d != NULL) {
        throw_pretty("Domain␣error:␣a␣6d␣contact␣model␣
           is␣required␣in␣" + frame_name +
                     "in␣order␣to␣compute␣the␣CoP");
        break;
      }
      ContactData6DTpl<Scalar>* d6d = dynamic_cast<
         ContactData6DTpl<Scalar>*>(it->second.get());
      if (d6d != NULL) {
        found_contact = true;
```

```
            contact = it->second;
            break;
          }
        }
      }
      if (!found_contact) {
        throw_pretty("Domain error: there isn't defined
           contact data for " + frame_name);
      }
    }

    pinocchio::DataTpl<Scalar>* pinocchio;
    MatrixXs Arr_Ru;
    boost::shared_ptr<ContactDataAbstractTpl<Scalar> >
       contact;  //!< contact force
    using Base::activation;
    using Base::cost;
    using Base::Lu;
    using Base::Luu;
    using Base::Lx;
    using Base::Lxu;
    using Base::Lxx;
    using Base::r;
    using Base::Ru;
    using Base::Rx;
    using Base::shared;
};

}  // namespace crocoddyl

#include "crocoddyl/multibody/costs/contact-cop-position.
   hxx"
#endif  //
   CROCODDYL_MULTIBODY_COSTS_CONTACT_COP_POSITION_HPP_
```

A.1.2. contact-cop-position.hxx

```
///////////////////////////////////////////////////////
// BSD 3-Clause License
//
// Copyright (C) 2020, University of Duisburg-Essen,
// University of Edinburgh
// Copyright note valid unless otherwise stated in
// individual files.
```

```
// All rights reserved.
//////////////////////////////////////////////////////////

#include "crocoddyl/core/utils/exception.hpp"
#include "crocoddyl/multibody/costs/contact-cop-position.
   hpp"

namespace crocoddyl {

template <typename _Scalar>
CostModelContactCoPPositionTpl<_Scalar>::
   CostModelContactCoPPositionTpl(
    boost::shared_ptr<StateMultibody> state, boost::
       shared_ptr<ActivationModelAbstract> activation,
    const FrameCoPSupport& cop_support, const std::size_t
       & nu)
    : Base(state, activation, nu), cop_support_(
       cop_support) {}

template <typename _Scalar>
CostModelContactCoPPositionTpl<_Scalar>::
   CostModelContactCoPPositionTpl(
    boost::shared_ptr<StateMultibody> state, boost::
       shared_ptr<ActivationModelAbstract> activation,
    const FrameCoPSupport& cop_support)
    : Base(state, activation), cop_support_(cop_support)
       {}

template <typename _Scalar>
CostModelContactCoPPositionTpl<_Scalar>::
   CostModelContactCoPPositionTpl(boost::shared_ptr<
   StateMultibody> state, const FrameCoPSupport&
   cop_support, const std::size_t& nu)
: Base(state, boost::make_shared<
   ActivationModelQuadraticBarrier>(AtivationBounds(
   VectorXs::Zero(4), std::numeric_limits<_Scalar>::max()
    * VectorXs::Ones(4))), nu), cop_support_(cop_support)
    {}

template <typename _Scalar>
CostModelContactCoPPositionTpl<_Scalar>::
   CostModelContactCoPPositionTpl(boost::shared_ptr<
   StateMultibody> state, const FrameCoPSupport&
   cop_support)
```

```
: Base(state, boost::make_shared<
    ActivationModelQuadraticBarrier>(ActivationBounds(
    VectorXs::Zero(4), std::numeric_limits<_Scalar>::max()
     * VectorXs::Ones(4)))), cop_support_(cop_support) {}

template <typename Scalar>
CostModelContactCoPPositionTpl<Scalar>::~
    CostModelContactCoPPositionTpl() {}

template <typename Scalar>
void CostModelContactCoPPositionTpl<Scalar>::calc(const
    boost::shared_ptr<CostDataAbstract>& data, const Eigen
    ::Ref<const VectorXs>&, const Eigen::Ref<const
    VectorXs>&) {
  Data* d = static_cast<Data*>(data.get());

  // Compute the cost residual r =  A * f
  data->r.noalias() = cop_support_.get_A() * d->contact->
      jMf.actInv(d->contact->f).toVector();

  // Compute the cost
  activation_->calc(data->activation, data->r);
  data->cost = data->activation->a_value;
}

template <typename Scalar>
void CostModelContactCoPPositionTpl<Scalar>::calcDiff(
    const boost::shared_ptr<CostDataAbstract>& data, const
     Eigen::Ref<const VectorXs>&, const Eigen::Ref<const
    VectorXs>&) {
  // Update all data
  Data* d = static_cast<Data*>(data.get());

  // Get the derivatives of the local contact wrench
  const MatrixXs& df_dx = d->contact->df_dx;
  const MatrixXs& df_du = d->contact->df_du;
  const Matrix46& A = cop_support_.get_A();

  // Compute the derivatives of the activation function
  activation_->calcDiff(data->activation, data->r);

  // Compute the derivatives of the cost residual
  data->Rx.noalias() = A * df_dx;
  data->Ru.noalias() = A * df_du;
```

```
  d->Arr_Ru.noalias() = data->activation->Arr * data->Ru;

  // Compute the first order derivatives of the cost
     function
  data->Lx.noalias() = data->Rx.transpose() * data->
     activation->Ar;
  data->Lu.noalias() = data->Ru.transpose() * data->
     activation->Ar;

  // Compute the second order derivatives of the cost
     function
  data->Lxx.noalias() = data->Rx.transpose() * data->
     activation->Arr * data->Rx;
  data->Lxu.noalias() = data->Rx.transpose() * d->Arr_Ru;
  data->Luu.noalias() = data->Ru.transpose() * d->Arr_Ru;
}

template <typename Scalar>
boost::shared_ptr<CostDataAbstractTpl<Scalar> >
   CostModelContactCoPPositionTpl<Scalar>::createData(
   DataCollectorAbstract* const data) {
  return boost::allocate_shared<Data>(Eigen::
     aligned_allocator<Data>(), this, data);
}

template <typename Scalar>
void CostModelContactCoPPositionTpl<Scalar>::
   set_referenceImpl(const std::type_info& ti, const void
   * pv) {
  if (ti == typeid(FrameCoPSupport)) {
    cop_support_ = *static_cast<const FrameCoPSupport*>(
       pv);
  } else {
    throw_pretty("Invalid argument: incorrect type (it
       should be FrameCoPSupport)");
  }
}

template <typename Scalar>
void CostModelContactCoPPositionTpl<Scalar>::
   get_referenceImpl(const std::type_info& ti, void* pv)
   const {
  if (ti == typeid(FrameCoPSupport)) {
```

```
    FrameCoPSupport& ref_map = *static_cast<
      FrameCoPSupport*>(pv);
    ref_map = cop_support_;
  } else {
    throw_pretty("Invalid argument: incorrect type (it
      should be FrameCoPSupport)");
  }
}

}  // namespace crocoddyl
```

A.2. CoP Cost Implementation for Impulse Dynamics

This section contains the implementation of the CoP cost for impulse dynamics action models, integrated into the open-source framework Crocoddyl within the context of this thesis.

For space-saving reasons, only the two core files are presented, each with shortened comments. The complete versions of these files, associated Python bindings, related files and a functional unit test can be traced in the associated pull request (#830).

A.2.1. impulse-cop-position.hpp

```
///////////////////////////////////////////////////////
// BSD 3-Clause License
//
// Copyright (C) 2020, University of Duisburg-Essen,
// University of Edinburgh
// Copyright note valid unless otherwise stated in
// individual files.
// All rights reserved.
///////////////////////////////////////////////////////

#ifndef
   CROCODDYL_MULTIBODY_COSTS_IMPULSE_COP_POSITION_HPP_
#define
   CROCODDYL_MULTIBODY_COSTS_IMPULSE_COP_POSITION_HPP_

#include "crocoddyl/multibody/fwd.hpp"
#include "crocoddyl/multibody/cost-base.hpp"
#include "crocoddyl/multibody/impulse-base.hpp"
#include "crocoddyl/multibody/impulses/impulse-3d.hpp"
#include "crocoddyl/multibody/impulses/impulse-6d.hpp"
#include "crocoddyl/multibody/data/impulses.hpp"
#include "crocoddyl/multibody/frames.hpp"
```

```
#include "crocoddyl/multibody/data/multibody.hpp"
#include "crocoddyl/core/activations/quadratic-barrier.
   hpp"
#include "crocoddyl/core/utils/exception.hpp"

namespace crocoddyl {

template <typename _Scalar>
class CostModelImpulseCoPPositionTpl : public
   CostModelAbstractTpl<_Scalar> {
 public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW
  typedef _Scalar Scalar;
  typedef MathBaseTpl<Scalar> MathBase;
  typedef CostModelAbstractTpl<Scalar> Base;
  typedef CostDataImpulseCoPPositionTpl<Scalar> Data;
  typedef StateMultibodyTpl<Scalar> StateMultibody;
  typedef CostDataAbstractTpl<Scalar> CostDataAbstract;
  typedef ActivationModelAbstractTpl<Scalar>
     ActivationModelAbstract;
  typedef ActivationModelQuadraticBarrierTpl<Scalar>
     ActivationModelQuadraticBarrier;
  typedef ActivationBoundsTpl<Scalar> ActivationBounds;
  typedef DataCollectorAbstractTpl<Scalar>
     DataCollectorAbstract;
  typedef FrameCoPSupportTpl<Scalar> FrameCoPSupport;
  typedef typename MathBase::Vector2s Vector2s;
  typedef typename MathBase::Vector3s Vector3s;
  typedef typename MathBase::VectorXs VectorXs;
  typedef typename MathBase::MatrixXs MatrixXs;
  typedef typename MathBase::MatrixX3s MatrixX3s;
  typedef Eigen::Matrix<Scalar, 4, 6> Matrix46;

  CostModelImpulseCoPPositionTpl(boost::shared_ptr<
     StateMultibody> state,
                                 boost::shared_ptr<
                                    ActivationModelAbstract
                                    > activation,
                                 const FrameCoPSupport&
                                    cop_support);

  CostModelImpulseCoPPositionTpl(boost::shared_ptr<
     StateMultibody> state, const FrameCoPSupport&
     cop_support);
```

```
  virtual ~CostModelImpulseCoPPositionTpl();

  virtual void calc(const boost::shared_ptr<
     CostDataAbstract>& data, const Eigen::Ref<const
     VectorXs>& x, const Eigen::Ref<const VectorXs>& u);

  virtual void calcDiff(const boost::shared_ptr<
     CostDataAbstract>& data, const Eigen::Ref<const
     VectorXs>& x, const Eigen::Ref<const VectorXs>& u);

  virtual boost::shared_ptr<CostDataAbstract> createData(
     DataCollectorAbstract* const data);

 protected:
  virtual void set_referenceImpl(const std::type_info& ti
     , const void* pv);
  virtual void get_referenceImpl(const std::type_info& ti
     , void* pv) const;

  using Base::activation_;
  using Base::nu_;
  using Base::state_;
  // using Base::unone_;

 private:
  FrameCoPSupport cop_support_;  //!< frame name of the
     impulse foot and support region of the cop
};

template <typename _Scalar>
struct CostDataImpulseCoPPositionTpl : public
   CostDataAbstractTpl<_Scalar> {
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW
  typedef _Scalar Scalar;
  typedef MathBaseTpl<Scalar> MathBase;
  typedef CostDataAbstractTpl<Scalar> Base;
  typedef DataCollectorAbstractTpl<Scalar>
     DataCollectorAbstract;
  typedef ImpulseModelMultipleTpl<Scalar>
     ImpulseModelMultiple;
  typedef FrameCoPSupportTpl<Scalar> FrameCoPSupport;
  typedef typename MathBase::Vector3s Vector3s;
  typedef typename MathBase::VectorXs VectorXs;
  typedef typename MathBase::MatrixXs MatrixXs;
```

```
typedef typename MathBase::Matrix3s Matrix3s;
typedef typename MathBase::Matrix6xs Matrix6xs;
typedef typename MathBase::Matrix6s Matrix6s;
typedef typename MathBase::Vector6s Vector6s;

template <template <typename Scalar> class Model>
CostDataImpulseCoPPositionTpl(Model<Scalar>* const
   model, DataCollectorAbstract* const data)
    : Base(model, data), Arr_Ru(model->get_activation()
       ->get_nr(), model->get_state()->get_nv()) {
  Arr_Ru.setZero();

  // Check that proper shared data has been passed
  DataCollectorImpulseTpl<Scalar>* d = dynamic_cast<
     DataCollectorImpulseTpl<Scalar>*>(shared);
  if (d == NULL) {
    throw_pretty("Invalid␣argument:␣the␣shared␣data␣
       should␣be␣derived␣from␣DataCollectorImpulse");
  }

  // Get the active 6d impulse (avoids data casting at
     runtime)
  FrameCoPSupport cop_support = model->template
     get_reference<FrameCoPSupport>();
  std::string frame_name = model->get_state()->
     get_pinocchio()->frames[cop_support.get_id()].name
     ;
  bool found_impulse = false;
  for (typename ImpulseModelMultiple::
     ImpulseDataContainer::iterator it = d->impulses->
     impulses.begin();
       it != d->impulses->impulses.end(); ++it) {
    if (it->second->frame == cop_support.get_id()) {
      ImpulseData3DTpl<Scalar>* d3d = dynamic_cast<
         ImpulseData3DTpl<Scalar>*>(it->second.get());
      if (d3d != NULL) {
        throw_pretty("Domain␣error:␣a␣6d␣impulse␣model␣
           is␣required␣in␣" + frame_name +
                     "in␣order␣to␣compute␣the␣CoP");
        break;
      }
      ImpulseData6DTpl<Scalar>* d6d = dynamic_cast<
         ImpulseData6DTpl<Scalar>*>(it->second.get());
      if (d6d != NULL) {
```

```
            found_impulse = true;
            impulse = it->second;
            break;
          }
        }
      }
      if (!found_impulse) {
        throw_pretty("Domain error: there isn't defined
           impulse data for " + frame_name);
      }
    }

    pinocchio::DataTpl<Scalar>* pinocchio;
    MatrixXs Arr_Ru;
    boost::shared_ptr<ImpulseDataAbstractTpl<Scalar> >
       impulse;  //!< impulse force
    using Base::activation;
    using Base::cost;
    using Base::Lu;
    using Base::Luu;
    using Base::Lx;
    using Base::Lxu;
    using Base::Lxx;
    using Base::r;
    using Base::Ru;
    using Base::Rx;
    using Base::shared;
};

}  // namespace crocoddyl

#include "crocoddyl/multibody/costs/impulse-cop-position.
   hxx"

#endif  //
   CROCODDYL_MULTIBODY_COSTS_IMPULSE_COP_POSITION_HPP_
```

A.2.2. impulse-cop-position.hxx

```
///////////////////////////////////////////////////////
// BSD 3-Clause License
//
// Copyright (C) 2020, University of Duisburg-Essen,
// University of Edinburgh
```

```
// Copyright note valid unless otherwise stated in
// individual files.
// All rights reserved.
//////////////////////////////////////////////////////

#include "crocoddyl/core/utils/exception.hpp"
#include "crocoddyl/multibody/costs/impulse-cop-position.
   hpp"

namespace crocoddyl {

template <typename _Scalar>
CostModelImpulseCoPPositionTpl<_Scalar>::
   CostModelImpulseCoPPositionTpl(boost::shared_ptr<
   StateMultibody> state, boost::shared_ptr<
   ActivationModelAbstract> activation, const
   FrameCoPSupport& cop_support)
    : Base(state, activation, 0), cop_support_(
       cop_support) {}

template <typename _Scalar>
CostModelImpulseCoPPositionTpl<_Scalar>::
   CostModelImpulseCoPPositionTpl(boost::shared_ptr<
   StateMultibody> state, const FrameCoPSupport&
   cop_support)
    : Base(state,
           boost::make_shared<
              ActivationModelQuadraticBarrier>(
              ActivationBounds(VectorXs::Zero(4), std::
              numeric_limits<_Scalar>::max() * VectorXs::
              Ones(4))), 0), cop_support_(cop_support) {}

template <typename Scalar>
CostModelImpulseCoPPositionTpl<Scalar>::~
   CostModelImpulseCoPPositionTpl() {}

template <typename Scalar>
void CostModelImpulseCoPPositionTpl<Scalar>::calc(const
   boost::shared_ptr<CostDataAbstract>& data, const Eigen
   ::Ref<const VectorXs>&, const Eigen::Ref<const
   VectorXs>&) {
  Data* d = static_cast<Data*>(data.get());

  // Compute the cost residual r =  A * f
```

```
  data->r.noalias() = cop_support_.get_A() * d->impulse->
     jMf.actInv(d->impulse->f).toVector();

  // Compute the cost
  activation_->calc(data->activation, data->r);
  data->cost = data->activation->a_value;
}

template <typename Scalar>
void CostModelImpulseCoPPositionTpl<Scalar>::calcDiff(
   const boost::shared_ptr<CostDataAbstract>& data, const
    Eigen::Ref<const VectorXs>&, const Eigen::Ref<const
   VectorXs>&) {
  // Update all data
  Data* d = static_cast<Data*>(data.get());

  // Get the derivatives of the local impulse wrench
  const MatrixXs& df_dx = d->impulse->df_dx;
  const Matrix46& A = cop_support_.get_A();

  // Compute the derivatives of the activation function
  activation_->calcDiff(data->activation, data->r);

  // Compute the derivative of the cost residual
  data->Rx.noalias() = A * df_dx;

  // Compute the first order derivative of the cost
     function
  data->Lx.noalias() = data->Rx.transpose() * data->
     activation->Ar;

  // Compute the second order derivative of the cost
     function
  data->Lxx.noalias() = data->Rx.transpose() * data->
     activation->Arr * data->Rx;
}

template <typename Scalar>
boost::shared_ptr<CostDataAbstractTpl<Scalar> >
   CostModelImpulseCoPPositionTpl<Scalar>::createData(
    DataCollectorAbstract* const data) {
  return boost::allocate_shared<Data>(Eigen::
     aligned_allocator<Data>(), this, data);
}
```

```
template <typename Scalar>
void CostModelImpulseCoPPositionTpl<Scalar>::
   set_referenceImpl(const std::type_info& ti, const void
   * pv) {
  if (ti == typeid(FrameCoPSupport)) {
    cop_support_ = *static_cast<const FrameCoPSupport*>(
       pv);
  } else {
    throw_pretty("Invalid␣argument:␣incorrect␣type␣(it␣
       should␣be␣FrameCoPSupport)");
  }
}

template <typename Scalar>
void CostModelImpulseCoPPositionTpl<Scalar>::
   get_referenceImpl(const std::type_info& ti, void* pv)
   const {
  if (ti == typeid(FrameCoPSupport)) {
    FrameCoPSupport& ref_map = *static_cast<
       FrameCoPSupport*>(pv);
    ref_map = cop_support_;
  } else {
    throw_pretty("Invalid␣argument:␣incorrect␣type␣(it␣
       should␣be␣FrameCoPSupport)");
  }
}

}  // namespace crocoddyl
```

A.3. Consistency Between the Frameworks

This section contains a verification of the notability consistency between the novel frameworks HyRoDyn and Pinocchio, both of which are used for computing of the robot dynamics. To this end, both the IK and ID of the robot are recomputed with HyRoDyn and compared with the original reference trajectories and torques, respectively, of the OC solution obtained by Crocoddyl.

A.3.1. Inverse Kinematics

Fig. A.1 shows the comparison of the IK solution obtained from OC with the recomputation from HyRoDyn. As becomes evident, both trajectories match, which indicates that the kinematics notation between both frameworks is consistent.

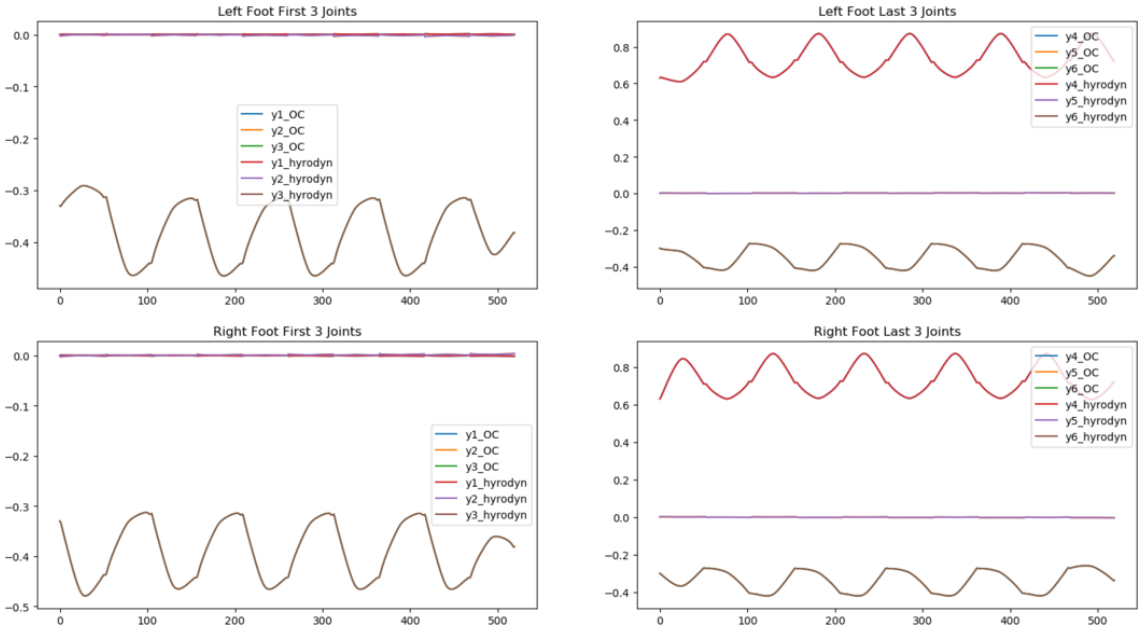


Figure A.1.: IK consistency check between the HyRoDyn and Pinocchio frameworks.

A.3.2. Inverse Dynamics

Fig. A.2 shows the comparison of the ID solution for optimal torque inputs from Crocoddyl with the according recomputation from HyRoDyn. As becomes evident, the torque inputs match, which proofs that also the dynamics notation between both novel frameworks is consistent.

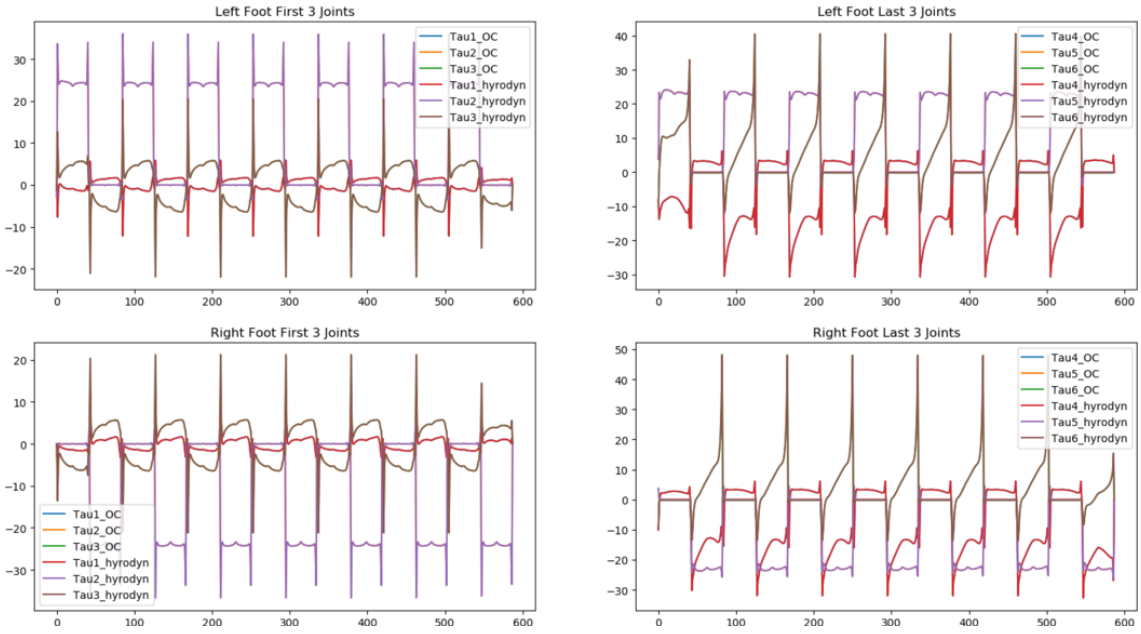


Figure A.2.: ID consistency check between the HyRoDyn and Pinocchio frameworks.

List of Figures

List of Tables

Bibliography

[1] Marc H Raibert. *Legged robots that balance*. MIT press, 1986.

[2] Pierre-Brice Wieber, Russ Tedrake, and Scott Kuindersma. Modeling and control of legged robots. In *Springer handbook of robotics*, pages 1203–1234. Springer, 2016.

[3] Paul Fitzpatrick, Kensuke Harada, Charles C Kemp, Yoshio Matsumoto, Kazuhito Yokoi, and Eiichi Yoshida. Humanoids. In *Springer handbook of robotics*, pages 1789–1818. Springer, 2016.

[4] Frank Kirchner, Elsa Kirchner, Manuel Meder, and Georg von Wichert. Transfit: Flexible interaction for infrastructures establishment by means of teleoperation and direct collaboration; transfer into industry 4.0. https://robotik.dfki-bremen.de/en/research/projects/transfit.html. Accessed: 2020-09-02.

[5] Marco Hutter, Christian Gehring, Michael Bloesch, Mark A Hoepflinger, C David Remy, and Roland Siegwart. Starleth: A compliant quadrupedal robot for fast, efficient, and versatile locomotion. In *Adaptive Mobile Robotics*, pages 483–490. World Scientific, 2012.

[6] Jerry E Pratt and Benjamin T Krupp. Series elastic actuators for legged robots. In *Unmanned Ground Vehicle Technology Vi*, volume 5422, pages 135–144. International Society for Optics and Photonics, 2004.

[7] Steve Collins, Andy Ruina, Russ Tedrake, and Martijn Wisse. Efficient bipedal robots based on passive-dynamic walkers. *Science*, 307(5712):1082–1085, 2005.

[8] Sami Haddadin, Felix Huber, and Alin Albu-Schäffer. Optimal control for exploiting the natural dynamics of variable stiffness robots. In *2012 IEEE International Conference on Robotics and Automation*, pages 3347–3354. IEEE, 2012.

[9] Jerry E Pratt. Exploiting inherent robustness and natural dynamics in the control of bipedal walking robots. Technical report, Massachusetts Institute of Technology Cambridge Departmend of Electrical Engineering, 2000.

[10] Eric R Westervelt, Jessy W Grizzle, Christine Chevallereau, Jun Ho Choi, and Benjamin Morris. *Feedback control of dynamic bipedal robot locomotion*. CRC press, 2018.

[11] Justin Carpentier, Andrea Del Prete, Steve Tonneau, Thomas Flayols, Florent Forget, Alexis Mifsud, Kevin Giraud, Dinesh Atchuthan, Pierre Fernbach, Rohan Budhiraja, et al. Multi-contact locomotion of legged robots in complex environments–the loco3d project. In *RSS Workshop on Challenges in Dynamic Legged Locomotion*, page 3p, 2017.

[12] Kevin Giraud, Pierre Fernbach, Gabriele Buondonno, Carlos Mastalli, Olivier Stasse, et al. Motion planning with multi-contact and visual servoing on humanoid robots. 2020.

[13] Shuuji Kajita, Fumio Kanehiro, Kenji Kaneko, Kiyoshi Fujiwara, Kensuke Harada, Kazuhito Yokoi, and Hirohisa Hirukawa. Biped walking pattern generation by using preview control of zero-moment point. In *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*, volume 2, pages 1620–1626. IEEE, 2003.

[14] Mrinal Kalakrishnan, Jonas Buchli, Peter Pastor, Michael Mistry, and Stefan Schaal. Fast, robust quadruped locomotion over challenging terrain. In *2010 IEEE International Conference on Robotics and Automation*, pages 2665–2670. IEEE, 2010.

[15] Alexander W Winkler, Carlos Mastalli, Ioannis Havoutis, Michele Focchi, Darwin G Caldwell, and Claudio Semini. Planning and execution of dynamic whole-body locomotion for a hydraulic quadruped on challenging terrain. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5148–5154. IEEE, 2015.

[16] C Dario Bellicoso, Fabian Jenelten, Péter Fankhauser, Christian Gehring, Jemin Hwangbo, and Marco Hutter. Dynamic locomotion and whole-body control for quadrupedal robots. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3359–3365. IEEE, 2017.

[17] Bernard Espiau, François Chaumette, and Patrick Rives. A new approach to visual servoing in robotics. *ieee Transactions on Robotics and Automation*, 8 (3):313–326, 1992.

[18] Oussama Khatib. A unified approach for motion and force control of robot manipulators: The operational space formulation. *IEEE Journal on Robotics and Automation*, 3(1):43–53, 1987.

[19] Alexander W Winkler. *Optimization-based motion planning for legged robots*. PhD thesis, ETH Zurich, 2018.

[20] Rohan Budhiraja, Justin Carpentier, Carlos Mastalli, and Nicolas Mansard. Differential dynamic programming for multi-phase rigid contact dynamics. In *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*, pages 1–9. IEEE, 2018.

[21] John T Betts. Survey of numerical methods for trajectory optimization. *Journal of guidance, control, and dynamics*, 21(2):193–207, 1998.

[22] Matthew P Kelly. Transcription methods for trajectory optimization: a beginners tutorial. *arXiv preprint arXiv:1707.00284*, 2017.

[23] Yuval Tassa, Nicolas Mansard, and Emo Todorov. Control-limited differential dynamic programming. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1168–1175. IEEE, 2014.

[24] David E Orin, Ambarish Goswami, and Sung-Hee Lee. Centroidal dynamics of a humanoid robot. *Autonomous robots*, 35(2-3):161–176, 2013.

[25] Hongkai Dai, Andrés Valenzuela, and Russ Tedrake. Whole-body motion planning with centroidal dynamics and full kinematics. In *2014 IEEE-RAS International Conference on Humanoid Robots*, pages 295–302. IEEE, 2014.

[26] Justin Carpentier, Steve Tonneau, Maximilien Naveau, Olivier Stasse, and Nicolas Mansard. A versatile and efficient pattern generator for generalized legged locomotion. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3555–3561. IEEE, 2016.

[27] Alexander Herzog, Nicholas Rotella, Stefan Schaal, and Ludovic Righetti. Trajectory generation for multi-contact momentum control. In *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pages 874–880. IEEE, 2015.

[28] Carlos Mastalli, Michele Focchi, Ioannis Havoutis, Andreea Radulescu, Sylvain Calinon, Jonas Buchli, Darwin G Caldwell, and Claudio Semini. Trajectory and foothold optimization using low-dimensional models for rough terrain locomotion. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1096–1103. IEEE, 2017.

[29] Alexander W Winkler, C Dario Bellicoso, Marco Hutter, and Jonas Buchli. Gait and trajectory optimization for legged systems through phase-based end-effector parameterization. *IEEE Robotics and Automation Letters*, 3(3):1560–1567, 2018.

[30] Bernardo Aceituno-Cabezas, Carlos Mastalli, Hongkai Dai, Michele Focchi, Andreea Radulescu, Darwin G Caldwell, José Cappelletto, Juan C Grieco, Gerardo Fernández-López, and Claudio Semini. Simultaneous contact, gait, and motion planning for robust multilegged locomotion via mixed-integer convex optimization. *IEEE Robotics and Automation Letters*, 3(3):2531–2538, 2017.

[31] Layale Saab, Oscar E Ramos, François Keith, Nicolas Mansard, Philippe Soueres, and Jean-Yves Fourquet. Dynamic whole-body motion generation under rigid contacts and other unilateral constraints. *IEEE Transactions on Robotics*, 29(2):346–362, 2013.

[32] Alexander Herzog, Nicholas Rotella, Sean Mason, Felix Grimminger, Stefan Schaal, and Ludovic Righetti. Momentum control with hierarchical inverse dynamics on a torque-controlled humanoid. *Autonomous Robots*, 40(3):473–491, 2016.

[33] Joris Vaillant, Abderrahmane Kheddar, Hervé Audren, François Keith, Stanislas Brossette, Adrien Escande, Karim Bouyarmane, Kenji Kaneko, Mitsuharu Morisawa, Pierre Gergondet, et al. Multi-contact vertical ladder climbing with an hrp-2 humanoid. *Autonomous Robots*, 40(3):561–580, 2016.

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

[35] H Peters, P Kampmann, and M Simnofske. Konstruktion eines zweibeinigen humanoiden roboters. *Proceedings of the 2. VDI Fachkonferenz Humanoide Roboter, VDI Fachkonferenz Humanoide Roboter*, 2017.

[36] Shivesh Kumar. *Modular and Analytical Methods for Solving Kinematics and Dynamics of Series-Parallel Hybrid Robots*. PhD thesis, Universität Bremen, 2019.

[37] Shivesh Kumar, Hendrik Wöhrle, José de Gea Fernández, Andreas Müller, and Frank Kirchner. A survey on modularity and distributivity in series-parallel hybrid robots. *Mechatronics*, 68:102367, 2020.

[38] M Vukobratović, Branislav Borovac, and Veljko Potkonjak. Towards a unified understanding of basic notions and terms in humanoid robotics. *Robotica*, 25 (1):87–101, 2007.

[39] MHP Dekker. Zero-moment point method for stable biped walking. *Eindhoven University of Technology*, 2009.

[40] Stephane Caron. Legged locomotion teaching material. https://scaron.info/category/teaching.html. Accessed: 2020-06-22.

[41] Friedrich Pfeiffer and Christoph Glocker. *Multibody dynamics with unilateral contacts*. John Wiley & Sons, 1996.

[42] Abhinandan Jain. *Robot and multibody dynamics: analysis and algorithms*. Springer Science & Business Media, 2010.

[43] Roy Featherstone. *Rigid body dynamics algorithms*. Springer, 2014.

[44] John YS Luh, Michael W Walker, and Richard PC Paul. On-line computational scheme for mechanical manipulators. 1980.

[45] John M Hollerbach. A recursive lagrangian formulation of maniputator dynamics and a comparative study of dynamics formulation complexity. *IEEE Transactions on Systems, Man, and Cybernetics*, 10(11):730–736, 1980.

[46] Shivesh Kumar, Julius Martensen, Andreas Mueller, and Frank Kirchner. Model simplification for dynamic control of series-parallel hybrid robots-a representative study on the effects of neglected dynamics shivesh. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5701–5708. IEEE, 2019.

[47] Elena Garcia, Joaquin Estremera, and Pablo Gonzalez-de Santos. A classification of stability margins for walking robots. *Robotica*, 20(6):595–606, 2002.

[48] Bruno Siciliano and Oussama Khatib. *Springer handbook of robotics*. Springer, 2016.

[49] Philippe Sardain and Guy Bessonnet. Forces acting on a biped robot. center of pressure-zero moment point. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 34(5):630–637, 2004.

[50] Miomir Vukobratović and J Stepanenko. On the stability of anthropomorphic systems. *Mathematical biosciences*, 15(1-2):1–37, 1972.

[51] Miomir Vukobratović and Branislav Borovac. Zero-moment pointâ€"thirty five years of its life. *International journal of humanoid robotics*, 1(01):157–173, 2004.

[52] David Mayne. A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems. *International Journal of Control*, 3 (1):85–95, jan 1966. doi: 10.1080/00207176608921369.

[53] Yuval Tassa, Tom Erez, and Emanuel Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4906–4913. IEEE, 2012.

[54] Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

[55] Donald E Kirk. *Optimal control theory: an introduction*. Courier Corporation, 2004.

[56] Russ Tedrake. Underactuated robotics: Algorithms for walking, running, swimming, flying, and manipulation (course notes for mit 6.832). http://underactuated.mit.edu/. Accessed: 2020-06-01.

[57] Morton I Kamien and Nancy Lou Schwartz. *Dynamic optimization: the calculus of variations and optimal control in economics and management*. Courier Corporation, 2012.

[58] Firdaus E Udwadia and Robert E Kalaba. A new perspective on constrained motion. *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, 439(1906):407–410, 1992.

[59] Zhaoming Xie, C Karen Liu, and Kris Hauser. Differential dynamic programming with nonlinear constraints. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 695–702. IEEE, 2017.

[60] Taylor A Howell, Brian E Jackson, and Zachary Manchester. Altro: A fast solver for constrained trajectory optimization. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7674–7679. IEEE, 2019.

[61] Stéphane Caron, Quang-Cuong Pham, and Yoshihiko Nakamura. Stability of surface contacts for humanoid robots: Closed-form formulae of the contact wrench cone for rectangular support areas. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5107–5112. IEEE, 2015.

[62] Imin Kao, Kevin M Lynch, and Joel W Burdick. Contact modeling and manipulation. In *Springer Handbook of Robotics*, pages 931–954. Springer, 2016.

[63] Jonathan Richard Shewchuk et al. An introduction to the conjugate gradient method without the agonizing pain, 1994.

[64] Rohan Budhiraja Carlos Mastalli, Nicolas Mansard, et al. Crocoddyl: a fast and flexible optimal control library for robot control under contact sequence. https://github.com/loco-3d/crocoddyl/wikis/home, 2020.

[65] Joachim Baumgarte. Stabilization of constraints and integrals of motion in dynamical systems. *Computer methods in applied mechanics and engineering*, 1(1):1–16, 1972.

[66] Justin Carpentier, Mehdi Benallegue, and Jean-Paul Laumond. On the centre of mass motion in human walking. *International Journal of Automation and Computing*, 14(5):542–551, 2017.

[67] Wangdo Kim, Arkady S Voloshin, and Stanley H Johnson. Modeling of heel strike transients during running. *Human Movement Science*, 13(2):221–244, 1994.

[68] Arthur D Kuo. A simple model of bipedal walking predicts the preferred speed–step length relationship. *J. Biomech. Eng.*, 123(3):264–269, 2001.

[69] Everett A Harman, Michael T Rosenstein, Peter N Frykman, Richard M ROSenStein, et al. The effects of arms and countermovement on vertical jumping. *Med Sci Sports Exerc*, 22(6):825–833, 1990.

[70] Ali Meghdari and M Aryanpour. Dynamical modeling and analysis of the human jumping process. In *ASME International Mechanical Engineering Congress and Exposition*, volume 36290, pages 453–461, 2002.

[71] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. 2016.

[72] Martin L Felis. Rbdl: an efficient rigid-body dynamics library using recursive algorithms. *Autonomous Robots*, 41(2):495–511, 2017.

[73] Scott L Delp, Frank C Anderson, Allison S Arnold, Peter Loan, Ayman Habib, Chand T John, Eran Guendelman, and Darryl G Thelen. Opensim: open-source software to create and analyze dynamic simulations of movement. *IEEE transactions on biomedical engineering*, 54(11):1940–1950, 2007.

[74] Shivesh Kumar, Kai Alexander von Szadkowski, Andreas Mueller, and Frank Kirchner. An analytical and modular software workbench for solving kinematics and dynamics of series-parallel hybrid robots. *Journal of Mechanisms and Robotics*, 12(2), 2020.

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

DOI: 10.17185/duepublico/73499
URN: urn:nbn:de:hbz:464-20201130-165211-5