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: 3015459

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 22, 2020

UNIVERSITÄT
DUISBURG
ESSEN

Robotics Innovation Center

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Declaration

This study was carried out at the Robotics Innovation Center of the German Research Center for Artificial Intelligence in the Advanced AI Team on Mechanics & Control.

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 22, 2020

Julian Eßer

Abstract

Keywords: Humanoids, Dynamic Bipedal Walking, Motion Planning, Multi-Contact Optimal Control, Differential Dynamic Programming, Trajectory Optimization

Kurzfassung

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1. Motivation

1.2. Related Work

1.3. Contribution

1.4. Structure

CHAPTER 2

Background: Optimal Bipedal Locomotion

The second chapter provides the reader with fundamentals on the mathematical modeling of legged robots, outlines how motion generation can be formulated as optimization problem and introduces the class of algorithms used within this thesis.

2.1. Foundations of Bipedal Locomotion

2.1.1. Terminology

In order to describe the locomotion of a humanoid robot, some specific terms are required that are introduced within this section. **?** provide an extensive introduction to the terminology related to bipedal walking, concisely summarized by **?** [**?** **?**].

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 in turn is characterized by a movement where partially both feet 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 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 this section we will introduce the equations of motion for a manipulator as well as for floating base systems. A concise introduction to this topic is presented with [?], more comprehensive studies can be found in [? ?].

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) are written as:

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

where

- $\mathbf{t}$ is the time variable,
- $\mathbf{q}$ 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
- $\mathbf{u}$ 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* method or the *Lagrange method*, 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.

Manipulator Equations

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 eq. (2.1), this expression is time-invariant. Hence, eq. (2.2) can be used to calculate the acceleration $\ddot{\boldsymbol{q}}$ based on the given state $(\boldsymbol{q}, \dot{\boldsymbol{q}})$, also known as Forward Dynamics (FD), as well as the Inverse Dynamics (ID).

Floating Base Systems (Legged Robots)

A floating base system is 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 joints 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 brake contacts with their environment in order to move. Contrary to manipulators, contacts need to be actively enforced by kinematic contact 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 the 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

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

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

$$\boldsymbol{w}_i = \{\boldsymbol{f}_i, \boldsymbol{m}_i\}.$$

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

2.2. Contact Stability Analysis

Overview of stability margins in [?] and for Handbook of Robotics see main reference from tedrake [?]; CoM and CoP calculation and coincidence described in [?].

2.2.1. Stability Criteria

Floor Projection of the Center of Mass (FCoM)

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

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

Zero-Moment Point (ZMP)

Introduction in [?], reviewed in [?], made popular with [?]. Assumptions:

- One planar contact are (i.e. no multiple surfaces like on rough terrain)
- Sufficiently high friction to prevent sliding of the feet

Center of Pressure (CoP)

Computation via

$$p_{CoP} = \frac{\boldsymbol{n} \times \boldsymbol{M}}{\boldsymbol{F} \cdot \boldsymbol{n}} \tag{2.3}$$

Coincidence of ZMP and CoP

2.2.2. Stability Classification

There are existing several classifications on stability, which will be defined in the following according to [**?** , Sec.1.2.1] and [**?**]. See **?** for more details on differentiating the terms dynamic stability and dynamic balance [**?**].

Statically Stable Gait

The gait or movement of a humanoid is classified as *statically stable*, if the Floor Projection of Center of Mass (FCoM) does not leave the Support Polygon (SP) during the entire motion or gait. Consequently, the humanoid will remain in a stable position, whenever the movement is stopped. Typically, these kind of stability are only obtained with very low walking velocities or quasi-static motions, where the static forces dominate the dynamic forces.

Dynamically Stable Gait

If the FCoM partially leaves the SP at some point during the gait, but the Center of Pressure (CoP) (or Zero-Moment Point (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.

2.3. Differential Dynamic Programming (DDP)

This section describes the basics of Differential Dynamic Programming (DDP), which is an Optimal Control (OC) algorithm that belongs to the Trajectory Optimization (TO) class. The algorithm was introduced in 1966 by **?** [**?**]. A modern description of the algorithm using the same notations as below can be found in [**?** **?**].

2.3.1. Finite Horizon Optimal Control

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

$$\boldsymbol{x}_{i+1} = \boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{u}_i), \tag{2.4}$$

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. (2.4). 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:

$$J(\boldsymbol{x}_0, \boldsymbol{U}) = l_f(\boldsymbol{x}_N) + \sum_{i=0}^{N-1} l(\boldsymbol{x}_i, \boldsymbol{u}_i). \tag{2.5}$$

As dicussed in chapter 1, *indirect* methods such DDP represent the trajectory implicitly solely via the optimal controls $\boldsymbol{U}$. The states $\boldsymbol{X}$ are obtained from forward simulation of the system dynamics, i.e. integration eq. (2.4). Consequently, the solution of the optimal control problem is the minimizing control sequence

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

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:

$$J_i(\boldsymbol{x}, \boldsymbol{U}_i) = l_f(\boldsymbol{x}_N) + \sum_{j=i}^{N-1} l(\boldsymbol{x}_j, \boldsymbol{u}_j). \tag{2.6}$$

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 [?] reduces the minimization over an entire sequence of controls to a sequence of minimizations over a single control, proceeding backwards in time:

$$V(\boldsymbol{x}) = \min_{\boldsymbol{u}} [l(\boldsymbol{x}, \boldsymbol{u}) + V'(\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}))]. \tag{2.7}$$

Note that eq. (2.7) is referred to as the *Bellman equation* for *discrete-time* optimization problems [?]. 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* [? ?].

2.3.3. Quadratic Approximation

DDP locally computes the optimal state and control sequences of the OC problem derived with eq. (2.7) 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. (2.7) around the $i-th(\boldsymbol{x}, \boldsymbol{u})$ pair

$$\boldsymbol{Q}(\delta\boldsymbol{x}, \delta\boldsymbol{u}) = l(\boldsymbol{x} + \delta\boldsymbol{x}, \boldsymbol{u} + \delta\boldsymbol{u}) + V'(\boldsymbol{f}(\boldsymbol{x} + \delta\boldsymbol{x}, \boldsymbol{u} + \delta\boldsymbol{u})). \tag{2.8}$$

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

$$\boldsymbol{Q}(\delta\boldsymbol{x}, \delta\boldsymbol{u}) \approx \frac{1}{2} \begin{bmatrix} 1 \\ \delta\boldsymbol{x} \\ \delta\boldsymbol{u} \end{bmatrix}^T \begin{bmatrix} 0 & \boldsymbol{Q}_x^T & \boldsymbol{Q}_u^T \\ \boldsymbol{Q}_x & \boldsymbol{Q}_{xx} & \boldsymbol{Q}_{xu} \\ \boldsymbol{Q}_u & \boldsymbol{Q}_{ux} & \boldsymbol{Q}_{uu} \end{bmatrix} \begin{bmatrix} 1 \\ \delta\boldsymbol{x} \\ \delta\boldsymbol{u} \end{bmatrix}, \tag{2.9}$$

where the coefficients can be computed to

$$\boldsymbol{Q}_x = l_x + \boldsymbol{f}_x^T \boldsymbol{V}'_x, \tag{2.10a}$$
$$\boldsymbol{Q}_u = l_u + \boldsymbol{f}_u^T \boldsymbol{V}'_x, \tag{2.10b}$$
$$\boldsymbol{Q}_{xx} = l_{xx} + \boldsymbol{f}_x^T \boldsymbol{V}'_{xx}\boldsymbol{f}_x + \boldsymbol{V}'_x \cdot \boldsymbol{f}_{xx}, \tag{2.10c}$$
$$\boldsymbol{Q}_{ux} = l_{ux} + \boldsymbol{f}_u^T \boldsymbol{V}'_{xx}\boldsymbol{f}_x + \boldsymbol{V}'_x \cdot \boldsymbol{f}_{ux}, \tag{2.10d}$$
$$\boldsymbol{Q}_{uu} = l_{uu} + \boldsymbol{f}_u^T \boldsymbol{V}'_{xx}\boldsymbol{f}_u + \boldsymbol{V}'_x \cdot \boldsymbol{f}_{uu}. \tag{2.10e}$$

The last terms of eqs. (2.10c) to (2.10e) denote the product of a vector with a tensor.

2.3.4. Backward Pass

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. (2.9), minimized with respect to $\delta\boldsymbol{u}$ for some state perturbation $\delta\boldsymbol{x}$, results in

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

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

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

The resulting locally-linear feedback policy can be again inserted into eq. (2.9) leading to a quadratic model of the Value at time i:

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

2.3.5. Forward Pass

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.

2.3.6. Numerical Characteristics

Like Newton's method, DDP is a second-order algorithm [**?**] and consequently takes large steps towards the minimum. With these types of algorithms, regularization and line-search often are required to achieve convergence [**?**].

Line-search is one of the basic iterative approaches from numerical optimization in order to find a local minimum of an objective function. Backtracking line-search especially determines the step length, namely the control modification, by some search parameter.

Regularization uses ##### F I L L #####

The interested reader can find a more extensive introduction to numerical optimization in e.g. [**?**] and **?** provide details and extension on these characteristics in the context of the DDP algorithm.

2.4. Handling Constraints with DDP

By nature, the DDP algorithm presented in section 2.3 does not take into account constraints. **?** developed a control-limited DDP [**?**] that takes into account box inequality constraints on the controls allowing the consideration of torque limits on real robotic systems. **?** proposed a DDP version for the problem of multi-phase rigid contact dynamics by exploiting the Karush-Kuhn-Tucker constraint of the rigid contact model [**?**]. Since physically consistent bipedal locomotion is highly dependent on making contacts with the ground, this section provides details on the above mentioned approach.

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

$$\boldsymbol{M}\dot{\boldsymbol{v}}_{free} = \boldsymbol{S}\boldsymbol{\tau} - \boldsymbol{b}, \tag{2.11}$$

with the joint-space intertia matrix $\boldsymbol{M} \in \boldsymbol{R}^{nxn}$ and the unconstrained acceleration vector $\dot{\boldsymbol{v}}_{free}$. The right-hand side of eq. (2.11) 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 [**?**]. The idea is to minimize the deviation in acceleration between the constrained and unconstrained motion:

$$\begin{aligned} \dot{\boldsymbol{v}} = \arg\min_{\boldsymbol{a}} \quad & \frac{1}{2}\,\|\dot{\boldsymbol{v}} - \dot{\boldsymbol{v}}_{free}\|_{\boldsymbol{M}} \\ \text{subject to} \quad & \boldsymbol{J}_c\dot{\boldsymbol{v}} + \dot{\boldsymbol{J}}_c\boldsymbol{v} = \boldsymbol{0}, \end{aligned} \tag{2.12}$$

where $\boldsymbol{M}$ formally represents the metric 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. (2.12) corresponds to an equality-constrained quadratic optimization problem. The optimal solutions $(\dot{\boldsymbol{v}}, \boldsymbol{\lambda})$ must satisfy the so-called Karush-Kuhn-Tucker (KKT) conditions given by

$$\begin{bmatrix}\boldsymbol{M} & \boldsymbol{J}_c^\top \\ \boldsymbol{J}_c & \boldsymbol{0}\end{bmatrix}\begin{bmatrix}\dot{\boldsymbol{v}} \\ -\boldsymbol{\lambda}\end{bmatrix} = \begin{bmatrix}\boldsymbol{\tau}_b \\ -\dot{\boldsymbol{J}}_c\boldsymbol{v}\end{bmatrix}. \tag{2.13}$$

These dual variables $\boldsymbol{\lambda}^k$ can be seen as external wrenches at the contact level. For a given robot state and applied torques, eq. (2.13) 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 [**?**].

2.4.2. KKT-Based DDP Algorithm

The KKT dynamics from eq. (2.13) can be expressed as a function of the state $\boldsymbol{x}_i$ and the control $\boldsymbol{u}_i$:

$$\begin{aligned} \boldsymbol{x}_{i+1} &= \boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{u}_i), \\ \boldsymbol{\lambda}_i &= \boldsymbol{g}(\boldsymbol{x}_i, \boldsymbol{u}_i), \end{aligned} \tag{2.14}$$

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. (2.13).

Supposing a sequence of predefined contacts, the cost-to-go of the DDP backward-pass and its respective Hessians (compare eq. (2.6) and 2.10) 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

$$\begin{aligned} \boldsymbol{Q}_{\boldsymbol{x}} &= \boldsymbol{l}_{\boldsymbol{x}} + \boldsymbol{g}_{\boldsymbol{x}}^T \boldsymbol{l}_{\boldsymbol{\lambda}} + \boldsymbol{f}_{\boldsymbol{x}}^T \boldsymbol{V}'_{\boldsymbol{x}}, \\ \boldsymbol{Q}_{\boldsymbol{u}} &= \boldsymbol{l}_{\boldsymbol{u}} + \boldsymbol{g}_{\boldsymbol{u}}^T \boldsymbol{l}_{\boldsymbol{\lambda}} + \boldsymbol{f}_{\boldsymbol{u}}^T \boldsymbol{V}'_{\boldsymbol{x}}, \\ \boldsymbol{Q}_{\boldsymbol{xx}} &\approx \boldsymbol{l}_{\boldsymbol{xx}} + \boldsymbol{g}_{\boldsymbol{x}}^T \boldsymbol{l}_{\boldsymbol{\lambda\lambda}} \boldsymbol{g}_{\boldsymbol{x}} + \boldsymbol{f}_{\boldsymbol{x}}^T \boldsymbol{V}'_{\boldsymbol{xx}} \boldsymbol{f}_{\boldsymbol{x}}, \\ \boldsymbol{Q}_{\boldsymbol{ux}} &\approx \boldsymbol{l}_{\boldsymbol{ux}} + \boldsymbol{g}_{\boldsymbol{u}}^T \boldsymbol{l}_{\boldsymbol{\lambda\lambda}} \boldsymbol{g}_{\boldsymbol{x}} + \boldsymbol{f}_{\boldsymbol{u}}^T \boldsymbol{V}'_{\boldsymbol{xx}} \boldsymbol{f}_{\boldsymbol{x}}, \\ \boldsymbol{Q}_{\boldsymbol{uu}} &\approx \boldsymbol{l}_{\boldsymbol{uu}} + \boldsymbol{g}_{\boldsymbol{u}}^T \boldsymbol{l}_{\boldsymbol{\lambda\lambda}} \boldsymbol{g}_{\boldsymbol{u}} + \boldsymbol{f}_{\boldsymbol{u}}^T \boldsymbol{V}'_{\boldsymbol{xx}} \boldsymbol{f}_{\boldsymbol{u}}. \end{aligned} \tag{2.15}$$

Consequently, the KKT-based DDP algorithm utilizes the set of eq. (2.15) inside the backward-pass to incorporate the rigid contacts forces, while the updated system dynamics from eq. (2.14) 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 Center of Mass (CoM) or performing a robot step. For formulating these task-related constraints, we follow the notation used in [**?**].

An arbitrary task can be formulated as a regulator:

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

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

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

where $\boldsymbol{w}_{j_k}$ assigned to task j at corresponding time k. The DDP algorithm utilized the derivatives of the regulators functions, namely computing the Jacobians and Hessians of the cost functions. In the scope of this thesis, the following tasks are handled

$$tasks \subseteq \{CoM, LH_{SE(3)}, RH_{SE(3)}, LF_{SE(3)}, RH_{SE(3)}\} : \tag{2.16}$$

1) the CoM tracking (CoM), 2) the tracking of the left- and right-hand pose ($LH_{SE(3)}$, $RH_{SE(3)}$) and 3) the tracking of the left- and right-feet pose ($LF_{SE(3)}$, $RH_{SE(3)}$).

2.5. RH5 Humanoid Robot

The derived approaches for constrained DDP have been tested both in simulation and real-world experiments on a full-size humanoid robot. RH5 is a lightweight and biologically inspired humanoid that has recently been developed at DFKI Robotics Innovation Center[**?**].

The RH5 humanoid robot (see fig. 2.1) is designed to mimic the human anatomy with a total size of 200cm, a weight of 62kg 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 16 DoF. In order to achieve a high dynamic performance, the robot's design follows a series-parallel hybrid approach. Consequently,linkages and parallel mechanisms are utilized in most of the robots 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 better maximum velocity and torque of the ankle as well as an advantageous weight of the lower leg [**?**]. The interested reader can find a comprehensive introduction on series-parallel hybrid robots in [**?** , Ch.2].

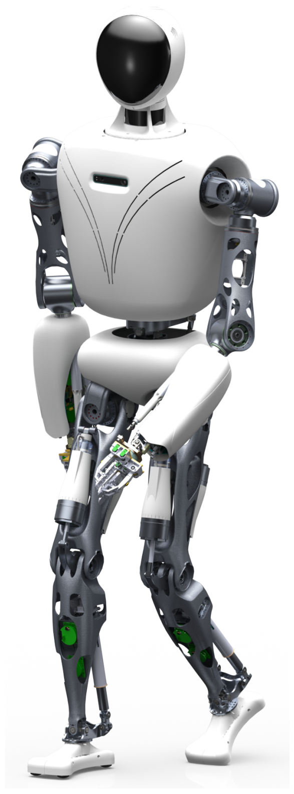

Figure 2.1.: The recently presented RH5 humanoid is used as experimental platform within this thesis.

CHAPTER 3

Contact Stability Constrained DDP

3.1. Idea: Constraining the CoP of each Contact Surface

3.2. Center of Pressure Constraints

3.2.1. Center of Pressure (CoP) Conditions

Conditions for CoP inside convex hull:

$$\begin{aligned} Pitch: -X &\leq C_x \leq X \\ Roll: -Y &\leq C_y \leq Y \end{aligned} \tag{3.1}$$

Represented by the four inequality conditions:

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

3.2.2. CoP Computation

$$\boldsymbol{p}_{CoP} = \frac{\boldsymbol{n} \times \boldsymbol{M}}{\boldsymbol{F} \cdot \boldsymbol{n}} = \frac{\begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} \times \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}}{\begin{bmatrix} f_x \\ f_y \\ f_z \end{bmatrix} \cdot \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix}} = \begin{bmatrix} n_y t_z - n_z t_y \\ n_z t_x - n_x t_z \\ n_x t_y - n_y t_x \end{bmatrix} \cdot \frac{1}{f_x n_x + f_y n_y + f_z n_y} \tag{3.3}$$

3.2.3. CoP Constraints: General Case

These conditions are be implemented via:

$$\begin{bmatrix} -Yn_0 & -Yn_1 & -Yn_2 & -n_2 & 0 & n_0 \\ -Yn_0 & -Yn_1 & -Yn_2 & n_2 & 0 & -n_0 \\ -Xn_0 & -Xn_1 & -Xn_2 & 0 & n_2 & -n_1 \\ -Xn_0 & -Xn_1 & -Xn_2 & 0 & -n_2 & n_1 \end{bmatrix} \cdot \begin{bmatrix} f^x \\ f^y \\ f^z \\ \tau^x \\ \tau^y \\ \tau^z \end{bmatrix} \leq \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{3.4}$$

3.2.4. CoP Constraints: Special Case of Horizontal Floor

For the case of horizontal floor, with the according normal vector $\boldsymbol{n} = [0, 0, 1]$, the CoP can be computed to:

$$\boldsymbol{p}_{CoP} = \begin{bmatrix} -t_y/f_z \\ t_x/f_z \\ 0 \end{bmatrix} \tag{3.5}$$

Represented by the four inequality conditions:

$$\begin{aligned} \tau^x &\leq Yf^z \\ -\tau^x &\leq Yf^z \\ \tau^y &\leq Yf^z \\ -\tau^y &\leq Yf^z \end{aligned} \tag{3.6}$$

Briefly summarized as:

$$\begin{aligned} || \tau^x || &\leq Yf^z \\ || \tau^y || &\leq Xf^z \end{aligned} \tag{3.7}$$

These conditions are be implemented via:

$$\begin{bmatrix} 0 & 0 & -Y & 1 & 0 & 0 \\ 0 & 0 & -Y & -1 & 0 & 0 \\ 0 & 0 & -X & 0 & 1 & 0 \\ 0 & 0 & -X & 0 & -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} f^x \\ f^y \\ f^z \\ \tau^x \\ \tau^y \\ \tau^z \end{bmatrix} \leq \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{3.8}$$

3.2.5. Backup: Friction Cone constraints

$$\begin{aligned} || \; f^x \; || &\leq \mu f^z \\ || \; f^y \; || &\leq \mu f^z \\ f^z &> 0 \end{aligned} \tag{3.9}$$

For the case of four edges of the linear approximation of the friction cone, the equations become:

$$\begin{bmatrix} 1 & 0 & -\mu \\ -1 & 0 & -\mu \\ 0 & 1 & -\mu \\ 0 & -1 & -\mu \\ 0 & 0 & -\mu \end{bmatrix} \cdot \begin{bmatrix} f^x \\ f^y \\ f^z \end{bmatrix} \leq \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{3.10}$$

3.3. Integration into the Crocoddyl Framework

3.3.1. Inequality Constraints by Penalization

3.3.2. Computing the Residual

3.3.3. Unittesting Numerical Differentiation

CHAPTER 4

Bipedal Walking Variants

CHAPTER 5

Highly-Dynamic Movements

CHAPTER 6

Experimental Validation on the RH5 Humanoid

6.1. Preliminaries

6.1.1. Experimental Setup

6.1.2. Control Architecture

6.1.3. Notational Compatibility of the Frameworks

6.2. Quasi-Static Movements

6.2.1. Squatting

6.2.2. Balancing

6.3. Static Bipedal Walking

6.4. Dynamic Bipedal Walking

CHAPTER 7

Conclusion and Outlook

7.1. Summary

7.2. Future Directions

APPENDIX A

Appendix

A.0.1. Carlos Talk: Essentials

In the end we want to solve a bilevel (nested) optimization

$$\begin{aligned}\boldsymbol{X}^*, \boldsymbol{U}^* &= \arg\min_{\boldsymbol{X},\boldsymbol{U}} \sum_{k=0}^{N-1} task(x_k, u_k) \\ x_k &= \arg\min physics(x_k, u_k), \\ s.t. \quad & constraints(x_k, u_k)\end{aligned} \tag{A.1}$$

Which more formally looks like

$$\begin{aligned}\boldsymbol{X}^*, \boldsymbol{U}^* = \begin{Bmatrix} \boldsymbol{x}_0^*, \cdots, \boldsymbol{x}_N^* \\ \boldsymbol{u}_0^*, \cdots, \boldsymbol{u}_N^* \end{Bmatrix} &= \arg\min_{\boldsymbol{X},\boldsymbol{U}} l_N(x_N) + \sum_{k=0}^{N-1} \int_{t_k}^{t_k+\Delta t} l(\boldsymbol{x}, \boldsymbol{u}) dt \\ s.t. \quad \dot{\boldsymbol{v}}, \boldsymbol{\lambda} &= \arg\min_{\dot{\boldsymbol{v}},\boldsymbol{\lambda}} ||\dot{\boldsymbol{v}} - \dot{\boldsymbol{v}}_{free}||_M, \\ \boldsymbol{x} &\in \mathcal{X}, \boldsymbol{u} \in \mathcal{U}\end{aligned}$$

KKT Matrix:

$$\begin{aligned}\boldsymbol{X}^*, \boldsymbol{U}^* &= \arg\min_{\boldsymbol{X},\boldsymbol{U}} \sum_{k=0}^{N-1} task(x_k, u_k) \\ & KKT - Dynamics(x_k, u_k)\end{aligned}$$

Multi-contact dynamics as holonomic constraints:

$$\begin{bmatrix} \dot{\boldsymbol{v}} \\ -\boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \boldsymbol{M} & \boldsymbol{J}_c^\top \\ \boldsymbol{J}_c & \boldsymbol{0} \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{\tau}_b \\ -\boldsymbol{a_0} \end{bmatrix}$$

A.1. Crocoddyl: Contact RObot COntrol by Differential DYnamic programming Library (Wiki Home)

A.1.1. Welcome to Crocoddyl

Crocoddyl is an optimal control library for robot control under contact sequence. Its solver is based on an efficient Differential Dynamic Programming (DDP) algorithm. Crocoddyl computes optimal trajectories along to optimal feedback gains. It uses Pinocchio for fast computation of robot dynamics and its analytical derivatives. Crocoddyl is focused on multi-contact optimal control problem (MCOP) which as the form:

$$\boldsymbol{X}^*, \boldsymbol{U}^* = \begin{Bmatrix} \boldsymbol{x}_0^*, \cdots, \boldsymbol{x}_N^* \\ \boldsymbol{u}_0^*, \cdots, \boldsymbol{u}_N^* \end{Bmatrix} = \arg\min_{\boldsymbol{X},\boldsymbol{U}} \sum_{k=1}^{N} \int_{t_k}^{t_k+\Delta t} l(\boldsymbol{x}, \boldsymbol{u}) dt$$

subject to

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}),$$

$$\boldsymbol{x} \in \mathcal{X}, \boldsymbol{u} \in \mathcal{U}, \boldsymbol{\lambda} \in \mathcal{K}.$$

where

- the state $\boldsymbol{x} = (\boldsymbol{q}, \boldsymbol{v})$ lies in a manifold, e.g. Lie manifold $\boldsymbol{q} \in SE(3) \times \boldsymbol{R}^{n_j}$,
- the system has underactuacted dynamics, i.e. $\boldsymbol{u} = (\boldsymbol{0}, \boldsymbol{\tau})$,
- $\mathcal{X}, \mathcal{U}$ are the state and control admissible sets, and
- $\mathcal{K}$ represents the contact constraints.

Note that $\boldsymbol{\lambda} = \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u})$ denotes the contact force, and is dependent on the state and control.

Let's start by understanding the concept behind crocoddyl design.

A.1.2. Action Models

In crocoddyl, an action model combines dynamics and cost models. Each node, in our optimal control problem, is described through an action model. Every time that we want describe a problem, we need to provide ways of computing the dynamics, cost functions and their derivatives. All these is described inside the action model.

To understand the mathematical aspects behind an action model, let's first get a locally linearize version of our optimal control problem as:

$$\boldsymbol{X}^*(\boldsymbol{x}_0),\, \boldsymbol{U}^*(\boldsymbol{x}_0) = \arg\min_{\boldsymbol{X},\boldsymbol{U}} = cost_T(\delta\boldsymbol{x}_N) + \sum_{k=1}^{N} cost_t(\delta\boldsymbol{x}_k, \delta\boldsymbol{u}_k)$$

subject to

$$dynamics(\delta\boldsymbol{x}_{k+1}, \delta\boldsymbol{x}_k, \delta\boldsymbol{u}_k) = \boldsymbol{0},$$

where

$$cost_T(\delta\boldsymbol{x}_k) = \frac{1}{2}\begin{bmatrix}1\\ \delta\boldsymbol{x}_k\end{bmatrix}^\top \begin{bmatrix}0 & \boldsymbol{l}_{\boldsymbol{x}k}^\top\\ \boldsymbol{l}_{\boldsymbol{x}k} & \boldsymbol{l}_{\boldsymbol{xx}k}\end{bmatrix} \begin{bmatrix}1\\ \delta\boldsymbol{x}_k\end{bmatrix},$$

$$cost_t(\delta\boldsymbol{x}_k, \delta\boldsymbol{u}_k) = \frac{1}{2}\begin{bmatrix}1\\ \delta\boldsymbol{x}_k\\ \delta\boldsymbol{u}_k\end{bmatrix}^\top \begin{bmatrix}0 & \boldsymbol{l}_{\boldsymbol{x}k}^\top & \boldsymbol{l}_{\boldsymbol{u}k}^\top\\ \boldsymbol{l}_{\boldsymbol{x}k} & \boldsymbol{l}_{\boldsymbol{xx}k} & \boldsymbol{l}_{\boldsymbol{ux}k}^\top\\ \boldsymbol{l}_{\boldsymbol{u}k} & \boldsymbol{l}_{\boldsymbol{ux}k} & \boldsymbol{l}_{\boldsymbol{uu}k}\end{bmatrix} \begin{bmatrix}1\\ \delta\boldsymbol{x}_k\\ \delta\boldsymbol{u}_k\end{bmatrix}$$

$$dynamics(\delta\boldsymbol{x}_{k+1}, \delta\boldsymbol{x}_k, \delta\boldsymbol{u}_k) = \delta\boldsymbol{x}_{k+1} - (\boldsymbol{f}_{\boldsymbol{x}k}\delta\boldsymbol{x}_k + \boldsymbol{f}_{\boldsymbol{u}k}\delta\boldsymbol{u}_k)$$

Notes

- An action model describes the dynamics and cost functions for a node in our optimal control problem.
- Action models lie in the discrete time space.
- For debugging and prototyping, we have also implemented NumDiff abstractions. These computations depend only in the defining of the dynamics equation and cost functions. However to asses efficiency, crocoddyl uses analytical derivatives computed from Pinocchio.

Differential and Integrated Action Models

It's often convenient to implement action models in continuous time. In crocoddyl, this continuous-time action models are called Differential Action Model (DAM). And together with predefined Integrated Action Models (IAM), it possible to retrieve the time-discrete action model needed by the solver. At the moment, we have the following integration rules:

- simpletic Euler and
- Runge-Kutta 4.

Add On from Introduction.jpnb

Optimal control solvers often need to compute a quadratic approximation of the action model (as previously described); this provides a search direction (computeDirection). Then it's needed to try the step along this direction (tryStep).

Typically calc and calcDiff do the precomputations that are required before computeDirection and tryStep respectively (inside the solver). These functions update the information of:

- **calc**: update the next state and its cost value

$$\delta\dot{\boldsymbol{x}}_{k+1} = \boldsymbol{f}(\delta\boldsymbol{x}_k, \boldsymbol{u}_k)$$

- **calcDiff**: update the derivatives of the dynamics and cost (quadratic approximation)

$$\boldsymbol{f}_{\boldsymbol{x}}, \boldsymbol{f}_{\boldsymbol{u}} \quad (dynamics)$$

$$\boldsymbol{l}_{\boldsymbol{x}}, \boldsymbol{l}_{\boldsymbol{u}}, \boldsymbol{l}_{\boldsymbol{xx}}, \boldsymbol{l}_{\boldsymbol{ux}}, \boldsymbol{l}_{\boldsymbol{uu}} \quad (cost)$$

A.1.3. State and its Integrate and Difference Rules

General speaking, the system's state can lie in a manifold M where the state rate of change lies in its tangent space $T_{\mathbf{x}}M$. There are few operators that needs to be defined for different routines inside our solvers:

$$\boldsymbol{x}_{k+1} = integrate(\boldsymbol{x}_k, \delta\boldsymbol{x}_k) = \boldsymbol{x}_k \oplus \delta\boldsymbol{x}_k$$

$$\delta\boldsymbol{x}_k = difference(\boldsymbol{x}_{k+1}, \boldsymbol{x}_k) = \boldsymbol{x}_{k+1} \ominus \boldsymbol{x}_k$$

where $\mathbf{x} \in M$ and $\delta\mathbf{x} \in T_{\mathbf{x}}M$. And we also need to defined the Jacobians of these operators with respect to the first and second arguments:

$$\frac{\partial\boldsymbol{x} \oplus \delta\boldsymbol{x}}{\partial\boldsymbol{x}}, \frac{\partial\boldsymbol{x} \oplus \delta\boldsymbol{x}}{\partial\delta\boldsymbol{x}} = Jintegrante(\boldsymbol{x}, \delta\boldsymbol{x})$$

$$\frac{\partial\boldsymbol{x}_2 \ominus \boldsymbol{x}_2}{\partial\boldsymbol{x}_1}, \frac{\partial\boldsymbol{x}_2 \ominus \boldsymbol{x}_1}{\partial\boldsymbol{x}_1} = Jdifference(\boldsymbol{x}_2, \boldsymbol{x}_1)$$

For instance, a state that lies in the Euclidean space will the typical operators:

$$integrate(\boldsymbol{x}, \delta\boldsymbol{x}) = \boldsymbol{x} + \delta\boldsymbol{x}$$

$$difference(\boldsymbol{x}_2, \boldsymbol{x}_1) = \boldsymbol{x}_2 - \boldsymbol{x}_1$$

$$Jintegrate(\cdot, \cdot) = Jdifference(\cdot, \cdot) = \boldsymbol{I}$$

All these functions are encapsulate inside the State class. For Pinocchio models, we have implemented the StateMultibody class which can be used for any robot model.

A.2. Crocoddyl Wiki: Differential Action Model for Floating in Contact Systems (DAMFIC)

A.2.1. System Dynamics

As you might know, a differential action model describes the systems dynamics and cost function in continuous-time. For multi-contact locomotion, we account for the rigid contact by applying the Gauss principle over holonomic constraints in a set of predefined contact placements, i.e.:

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

This is equality-constrained quadratic problem with an analytical solution of the form:

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

in which

$$(\dot{\boldsymbol{v}}, \boldsymbol{\lambda}) \in (\boldsymbol{R}^{nv}, \boldsymbol{R}^{nf})$$

are the primal and dual solutions,

$$\boldsymbol{M} \in \boldsymbol{R}^{nv\times nv}$$

is formally the metric tensor over the configuration manifold $\boldsymbol{q} \in \boldsymbol{R}^{nq}$,

$$\boldsymbol{J}_c = \begin{bmatrix} \boldsymbol{J}_{c_1} & \cdots & \boldsymbol{J}_{c_f} \end{bmatrix} \in \boldsymbol{R}^{nf\times nv}$$

is a stack of f contact Jacobians, $\boldsymbol{\tau}_b = \boldsymbol{S}\boldsymbol{\tau} - \boldsymbol{b} \in \boldsymbol{R}^{nv}$ is the force-bias vector that accounts for the control $\boldsymbol{\tau} \in \boldsymbol{R}^{nu}$, the Coriolis and gravitational effects $\boldsymbol{b}$, and $\boldsymbol{S}$ is the selection matrix of the actuated joint coordinates, and nq, nv, nu and nf are the number of coordinates used to describe the configuration manifold, its tangent-space dimension, control commands and contact forces, respectively.

And this equality-constrained forward dynamics can be formulated using state space representation, i.e.:

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})$$

where $\boldsymbol{x} = (\boldsymbol{q}, \boldsymbol{v}) \in \boldsymbol{R}^{nq+nv}$ and $\boldsymbol{u} = \boldsymbol{\tau} \in \boldsymbol{R}^{nu}$ are the state and control vectors, respectively. Note that $\dot{\boldsymbol{x}}$ lies in the tangent-space of $\boldsymbol{x}$, and their dimension are not the same.

A.2.2. Add On from Introduction.jpnb

A.2.3. Solving the Optimal Control Problem

Our optimal control solver interacts with a defined ShootingProblem. A **shooting problem** represents a **stack of action models** in which an action model defines a specific node along the OC problem.

First we need to create an action model from DifferentialFwdDynamics. We use it for building terminal and running action models. In this example, we employ an simpletic Euler integration rule.

Next we define the set of cost functions for this problem. One could formulate

- Running costs (related to individual states)
- Terminal costs (related to the final state)

in order to penalize, for example, the state error, control error, or end-effector pose error.

Onces we have defined our shooting problem, we create a DDP solver object and pass some callback functions for analysing its performance.

Application to Bipedal Walking

In crocoddyl, we can describe the multi-contact dynamics through holonomic constraints for the support legs. From the Gauss principle, we have derived the model as:

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

.

This DAM is defined in "DifferentialActionModelFloatingInContact" class. Given a predefined contact sequence and timings, we build per each phase a specific multi-contact dynamics. Indeed we need to describe **multi-phase optimal control problem**. One can formulate the multi-contact optimal control problem (MCOP) as follows:

$$\boldsymbol{X}^*, \boldsymbol{U}^* = \begin{Bmatrix} \boldsymbol{x}_0^*, \cdots, \boldsymbol{x}_N^* \\ \boldsymbol{u}_0^*, \cdots, \boldsymbol{u}_N^* \end{Bmatrix} = \arg \min_{\boldsymbol{X}, \boldsymbol{U}} \sum_{p=0}^{P} \sum_{k=1}^{N(p)} \int_{t_k}^{t_k + \Delta t} l_p(\boldsymbol{x}, \boldsymbol{u}) dt$$

subject to

$$\dot{\boldsymbol{x}} = \boldsymbol{f}_p(\boldsymbol{x}, \boldsymbol{u}), \text{for } t \in [\tau_p, \tau_{p+1}]$$

$$\boldsymbol{g}(\boldsymbol{v}^{p+1}, \boldsymbol{v}^p) = \boldsymbol{0}$$

$$\boldsymbol{x} \in \mathcal{X}_p, \boldsymbol{u} \in \mathcal{U}_p, \boldsymbol{\lambda} \in \mathcal{K}_p.$$

where $\boldsymbol{g}(\cdot,\cdot,\cdot)$ describes the contact dynamics, and they represents terminal constraints in each walking phase. In this example we use the following **impact model**:

$$\boldsymbol{M}(\boldsymbol{v}_{next} - \boldsymbol{v}) = \boldsymbol{J}_{impulse}^{T}$$

$$\boldsymbol{J}_{impulse}\boldsymbol{v}_{next} = \boldsymbol{0}$$

$$\boldsymbol{J}_{c}\boldsymbol{v}_{next} = \boldsymbol{J}_{c}\boldsymbol{v}$$