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. Olivier Stasse (LAAS-CNRS)

September 22, 2020

UNIVERSITÄT DUISBURG ESSEN
Offen im Denken

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. Modeling and Control of Legged Robots

2.1.1. Terminology

Three Dimensions of Motion

Gait Analysis (Double Support, Gait Phases ..)

2.1.2. Dynamics

2.1.3. Stability Analysis

2.1.4. Motion Generation

2.1.5. Motion Control

Kinematic Control (High-gain joint position trajectory tracking)

Impedance Control with Joint Space Inverse Dynamics (Low-gain joint control with model compensation)

Task Space Inverse Dynamics Control (Directly regulating in Â«task spaceÂ»)

Virtual Model Control (Dynamic control of a quasistatic system)

2.1.6. Efficient Walking

2.2. Trajectory Optimization

2.3. Differential Dynamic Programming (DDP)

This section describes the basics of DDP, which is an Optimal Control (OC) algorithm that belongs to the Trajectory Optimization (TO) class. The algorithm was introduced in 1966 by **?** [**?**].

2.3.1. Local Dynamic Programming

The discrete time dynamics of a system can be modeled as a generic function $\boldsymbol{f}$

$$\boldsymbol{x}_{i+1} = \boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{u}_i), \tag{2.1}$$

which describes the evolution of the state $\boldsymbol{x} \in \boldsymbol{R}^n$ from time i to $i+1$, given the control $u \in \boldsymbol{R}^m$. A valid 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. (2.1).

$$J(\boldsymbol{x}_0, \boldsymbol{U}) = \sum_{i=0}^{N-1} l(\boldsymbol{x}_i, \boldsymbol{u}_i) + l_f(\boldsymbol{x}_N) \tag{2.2}$$

$$\boldsymbol{U}^* = \underset{U}{\operatorname{argmin}}\, J(\boldsymbol{x}_0, \boldsymbol{U}) \tag{2.3}$$

$$J_i(\boldsymbol{x}, \boldsymbol{U}_i) = \sum_{j=i}^{N-1} l(\boldsymbol{x}_j, \boldsymbol{u}_j) + l_f(\boldsymbol{x}_N) \tag{2.4}$$

$$V_i(\boldsymbol{x}) = \min_{\boldsymbol{U}_i} J_i(\boldsymbol{x}, \boldsymbol{U}_i) \tag{2.5}$$

$$V(\boldsymbol{x}) = \min_{\boldsymbol{u}}[l(\boldsymbol{x}, \boldsymbol{u}) + V'(\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}))] \tag{2.6}$$

2.3.2. Quadratic Approximation

$$Q(\delta\boldsymbol{x}, \delta\boldsymbol{u}) = l(\boldsymbol{x}+\delta\boldsymbol{x}, \boldsymbol{u}+\delta\boldsymbol{u}) + V'(\boldsymbol{f}(\boldsymbol{x}+\delta\boldsymbol{x}, \boldsymbol{u}+\delta\boldsymbol{u})) \tag{2.7}$$

$$Q_{\boldsymbol{x}} = l_{\boldsymbol{x}} + \boldsymbol{f}_{\boldsymbol{x}}^T V'_{\boldsymbol{x}} \tag{2.8a}$$

$$Q_{\boldsymbol{u}} = l_{\boldsymbol{u}} + \boldsymbol{f}_{\boldsymbol{u}}^T V'_{\boldsymbol{x}} \tag{2.8b}$$

$$Q_{\boldsymbol{xx}} = l_{\boldsymbol{xx}} + \boldsymbol{f}_{\boldsymbol{x}}^T V'_{\boldsymbol{xx}} \boldsymbol{f}_{\boldsymbol{x}} + V'_{\boldsymbol{x}} \cdot \boldsymbol{f}_{\boldsymbol{xx}} \tag{2.8c}$$

$$Q_{\boldsymbol{ux}} = l_{\boldsymbol{ux}} + \boldsymbol{f}_{\boldsymbol{u}}^T V'_{\boldsymbol{xx}} \boldsymbol{f}_{\boldsymbol{x}} + V'_{\boldsymbol{x}} \cdot \boldsymbol{f}_{\boldsymbol{ux}} \tag{2.8d}$$

$$Q_{\boldsymbol{uu}} = l_{\boldsymbol{uu}} + \boldsymbol{f}_{\boldsymbol{u}}^T V'_{\boldsymbol{xx}} \boldsymbol{f}_{\boldsymbol{u}} + V'_{\boldsymbol{x}} \cdot \boldsymbol{f}_{\boldsymbol{uu}} \tag{2.8e}$$

$$\delta\boldsymbol{u}^*(\delta\boldsymbol{x}) = \operatorname*{argmin}_{\delta\boldsymbol{u}} Q(\delta\boldsymbol{x}, \delta\boldsymbol{u}) = \boldsymbol{k} + \boldsymbol{K} \tag{2.9}$$

$$\boldsymbol{k} = -Q_{\boldsymbol{uu}}^{-1} Q_{\boldsymbol{u}} \quad \textit{and} \quad \boldsymbol{K} = -Q_{\boldsymbol{uu}}^{-1} Q_{\boldsymbol{ux}} \tag{2.10}$$

$$\Delta V = -\frac{1}{2} \boldsymbol{k}^T Q_{\boldsymbol{uu}} \boldsymbol{k} \tag{2.11a}$$

$$V_{\boldsymbol{x}} = Q_{\boldsymbol{x}} - \boldsymbol{K}^T Q_{\boldsymbol{uu}} \boldsymbol{k} \tag{2.11b}$$

$$V_{\boldsymbol{xx}} = Q_{\boldsymbol{xx}} - \boldsymbol{K}^T Q_{\boldsymbol{uu}} \boldsymbol{K} \tag{2.11c}$$

2.3.3. Line Search and Regularization

2.4. DDP With Constrained Robot Dynamics

2.4.1. Handling Tasks

2.4.2. Contact Dynamics

2.4.3. Karush-Kuhn-Tucker (KKT) Conditions

2.4.4. KKT-Based DDP Algorithm

Rigid contacts can be formulated as holonomic scleronomic constraints to the robot dynamics.

2.5. The RH5 Humanoid Robot

CHAPTER 3

Dynamic Bipedal Walking

CHAPTER 4

Highly-Dynamic Movements

CHAPTER 5

Experimental Validation

5.1. Validation in Real-Time Physics Simulation

5.2. Validation on the RH5 Humanoid Robot

CHAPTER 6

Conclusion and Outlook

6.1. Summary

6.2. 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)

$$\boldsymbol{f_x}, \boldsymbol{f_u} \quad (dynamics)$$

$$\boldsymbol{l_x}, \boldsymbol{l_u}, \boldsymbol{l_{xx}}, \boldsymbol{l_{ux}}, \boldsymbol{l_{uu}} \quad (cost)$$

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, $\boldsymbol{\tau}_b = \boldsymbol{S\tau} - \boldsymbol{b} \in \boldsymbol{R}^{nv}$ 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**:

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

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

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