NATIONAL TECHNICAL UNIVERSITY OF ATHENS
SCHOOL OF MECHANICAL ENGINEERING
DEPARTMENT OF M.D & C.S.
Control Systems Laboratory

Diploma Thesis

Optimization-based motion planning for quadruped robots

Iro Papagiannaki

Supervisor: E.G. Papadopoulos

ATHENS, 2022

Περίληψη

Τα τετράποδα ρομπότ έχουν γίνει δημοφιλή λόγω της ικανότητάς τους να διασχίζουν εδάφη που τα τροχοφόρα ρομπότ δεν μπορούν. Τα ρομπότ με πόδια μπορούν να ανεβαίνουν σκάλες, να περπατούν σε χωράφια που μπορεί να έχουν ανώμαλο έδαφος με λακκούβες και ακόμη να έχουν πρόσβαση σε σκηνές καταστροφής γεμάτες συντρίμμια.

Στην παρούσα διπλωματική εργασία αναπτύσσονται αλγόριθμοι που σχετίζονται με τον σχεδιασμό κίνησης του τετράποδου ρομπότ "Άργος". Το "Άργος" έχει σχεδιαστεί από την ομάδα "Legged Robots" του Εργαστηρίου Αυτόματου Ελέγχου και Ρυθμίσεως Μηχανών & Εγκαταστάσεων του ΕΜΠ και σκοπός του είναι η επιθεώρηση αμπελώνων. Έτσι, στο πλαίσιο αυτής της εργασίας, μία εργασία υψηλού επιπέδου όπως η μετακίνηση του ρομπότ με σταθερή ταχύτητα, μεταφράζεται σε ένα επιθυμητό σχέδιο κίνησης που μπορεί να εκτελέσει το τετράποδο. Οι ιδιοδεκτικοί αισθητήρες του "Άργους" παρέχουν τις απαραίτητες μετρήσεις ανάδρασης.

Η κίνηση των συστημάτων με πόδια είναι δύσκολη λόγω της υποεπενέργησης τους. Η κίνηση του σώματος προς τα εμπρός δεν μπορεί να δημιουργηθεί άμεσα, αλλά προκύπτει από τις δυνάμεις επαφής μεταξύ των ποδιών και του περιβάλλοντος. Διερευνώνται τρεις διαφορετικοί αλγόριθμοι σχεδιασμού κίνησης, καθένας από τους οποίους έχει διαφορετικό επίπεδο πολυπλοκότητας και δυνατότητες κίνησης.

Στο πρώτο σχήμα ελέγχου, τα πόδια ακολουθούν μια προκαθορισμένη συνεχή τροχιά στον Καρτεσιανό χώρο και η κίνηση του σώματος προκύπτει από τις δυνάμεις επαφής που δημιουργούνται κατά τη φάση της στάσης των ποδιών. Βασίζεται σε έναν προηγούμενο αλγόριθμο σχεδίου κίνησης που αναπτύχθηκε από την ομάδα "Legged Robots" [41], με τη διαφορά ότι εδώ η τροχιά του ποδιού ορίζεται από πολυωνυμικές καμπύλες.

Στο δεύτερο σχήμα ελέγχου [23], η κίνηση του ρομπότ καθορίζεται από εντολές υψηλού επιπέδου που σχετίζονται με τη γραμμική και γωνιακή ταχύτητα του κύριου σώματος. Τα βήματα υπολογίζονται με βάση αυτές τις εντολές και από προβλέψεις που παρέχονται από απλοποιημένα δυναμικά μοντέλα. Η θέση του κυρίως σώματος ορίζεται σε σχέση με τη θέση των ποδιών και είναι επιθυμητό να γέρνει προς τα πόδια που βρίσκονται σε φάση στάσης. Υπολογίζονται οι εικονικές δυνάμεις που θα έπρεπε ιδανικά να ασκούνται στο κύριο σώμα και διαμορφώνεται ένα πρόβλημα βελτιστοποίησης για τη βέλτιστη κατανομή αυτών των δυνάμεων στα πόδια που είναι σε στάση.

Η κίνηση του σώματος, οι κινήσεις των ποδιών και οι δυνάμεις επαφής είναι αλληλεξαρτώμενα μεγέθη, επομένως, η παραγωγή έγκυρων τροχιών για καθένα από αυτά μπορεί να είναι επίπονη ή ακόμα και ανέφικτη όταν η εργασία είναι πολύπλοκη. Για το λόγο αυτό, ο τελευταίος αλγόριθμος σχεδιασμού κίνησης [63] εστιάζει στην εύρεση αυτών των τροχιών με βέλτιστο και αυτοματοποιημένο τρόπο. Η βελτιστοποίηση τροχιάς είναι μια μέθοδος που παράγει βέλτιστα σχέδια κίνησης για κάθε εργασία υψηλού επιπέδου. Μόλις το πρόβλημα μοντελοποιηθεί σωστά και ληφθούν υπόψη οι φυσικοί περιορισμοί, μπορούν να παραχθούν οι τροχιές κάθε υποσυστήματος, έτσι ώστε ολόκληρο το σύστημα να οδηγηθεί στον επιθυμητό στόχο.

Αυτή η Διπλωματική παρουσιάζει προσεγγίσεις για τη μετατροπή του φυσικού προβλήματος κίνησης σε ένα πρόβλημα μαθηματικής βελτιστοποίησης, το οποίο μπορεί να λυθεί με διαθέσιμο λογισμικό. Οι μεταβλητές απόφασης αποτελούνται από την εξαδιάστατη κίνηση του σώματος, τις θέσεις που θα πατήσουν τα πόδια, τις κινήσεις ποδιού στη φάση αιώρησης και τις δυνάμεις επαφής. Το τετράποδο διαμορφώνεται ως ένα ενιαίο άκαμπτο

σώμα που ελέγχεται από τις δυνάμεις επαφής. Η ακολουθία με την οποία κινούνται τα πόδια καθορίζεται από τον τύπο του βηματισμού. Έχει προταθεί μια παραλλαγή του βηματισμού «τροχασμού» (ή “trotting”), η οποία περιλαμβάνει επιπλέον φάσεις στάσης με όλα τα πόδια στο έδαφος, για την αποφυγή απότομων μεταβολών στις δυνάμεις επαφής όταν ένα πόδι έρχεται σε επαφή με το έδαφος.

Οι τρεις αλγόριθμοι σχεδιασμού κίνησης που αναπτύχθηκαν επαληθεύτηκαν στο περιβάλλον προσομοίωσης Simscape Multibody. Τα προβλήματα βελτιστοποίησης ρυθμίστηκαν μέσω της CasADi, η οποία είναι μια εξωτερική βιβλιοθήκη που επιλύει με πολύ αποτελεσματικό τρόπο μη γραμμικά προβλήματα βελτιστοποίησης.

Abstract

Quadruped robots have become popular due to their capability to traverse terrain that wheeled robots cannot. Legged robots can climb stairs, walk across fields that may have uneven terrain with puddles and even access debris-filled disaster scenes.

In this thesis, algorithms that are related to the motion-planning of the quadruped Argos are developed. Argos is designed by the Legged Robots Team of the Control Systems Lab in NTUA, and its purpose is the inspection of vineyards. Thus, in the context of this work, a high-level task which may be the locomotion of the robot with a constant speed, is translated into a desired motion-plan that the quadruped can execute. Argos's proprioceptive sensors provide the necessary feedback measurements.

Legged locomotion is challenging due to the system being underactuated. A forward body movement cannot be directly generated but results from contact forces between the feet and the environment. Three different motion planning algorithms are investigated, each of which has a different level of complexity and motion capabilities.

In the first framework, the feet follow a predefined continuous trajectory in Cartesian space and the body motion arises from the contact forces generated during the feet's stance phase. It is based on a prior motion plan algorithm that has been developed by the Legged Robots Team [41], but the main difference is that the foot trajectory is defined by polynomial splines.

In the second control framework [23], the robot's locomotion is specified by high-level commands related to the main body's linear and angular velocity. The footsteps are calculated based on these commands and by predictions provided by simplified dynamic models. The main body's position is defined relative to the feet's position, and it is desirable that it leans towards the legs that are in stance phase. The virtual forces that should ideally act on the main body are calculated and an optimization problem is formulated for the optimal distribution of these forces to the stance legs.

The body motion, the feet motions and the contact forces are interdependent quantities, thus, producing valid trajectories for each of them may be tedious or even infeasible when the task is complex. For this reason, the last motion planning algorithm [63] focuses on finding these trajectories in an optimal and automated way. Trajectory optimization is a method that produces optimal motion plans for any high-level task. Once the problem is properly modeled and the physical constraints are set up, the trajectories of each subsystem can be generated, so that the whole system is driven to the desired goal.

This thesis presents approaches to transcribe the physical locomotion problem into a mathematical optimization problem, which can be solved by off-the-shelf software. The decision variables consist of the six-dimensional body motion, the footholds, the swing-leg motions, and the contact forces. The quadruped is modeled as a single rigid body controlled by the contact forces. The sequence in which the feet are moving is determined by the Gait pattern. A variation of the trotting gait, which include extra stance phases with all the feet on the ground, has been proposed to avoid abrupt changes in contact forces when a foot hits the ground.

The three developed motion-planning algorithms have been verified in the Simscape Multibody simulation environment. The optimization problems are set up in CasADi, which is an external library that solves nonlinear optimization problems in a very efficient way.

To my best friend, Alex

Acknowledgements

First, I would like to express my gratitude and appreciation to Professor E. Papadopoulos whose support, guidance and overall insights in the field of robotics have made this an inspiring experience for me. A special thanks to the Ph.D. candidates of the CSL-EP laboratory, Konstantinos Koutsoukis, Athanasios Mastrogeorgiou and especially Konstantinos Machairas, not only for the thoughtful comments and recommendations on this thesis, but also for inspiring me to think from multiple perspectives to form a comprehensive and objective critique. I would also like to thank them and the rest of my friends and lab mates, Katerina Smyrli, Marianna Sotiriou, Aris Geladaris, Antonis Aggouridis and Aristotelis Papatheodorou, for a cherished time spent together in the lab and in social settings.

Getting through my thesis required more than academic support, and I have many people to thank for their encouragement and support all through my studies. I cannot begin to express my gratitude and appreciation for their friendship. Alexandra Papatheodorou, Georgia Nikolaou, Alexandros Konstantinidis and John Valvis have been unwavering in their personal and professional support during the time I spent at NTUA.

I would like to extend my sincere thanks to my dear friends Ioanna Pouliezou, Christina Pouliezou, Elina Dosta and my teammates Afroditi Belli and Sofia Syrma for their tremendous understanding, continuous encouragement through the process of researching and writing this thesis and for our unforgettable moments over the last thirteen years. Finally, I must express my very profound gratitude to my parents for all the unconditional support all through my studies.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

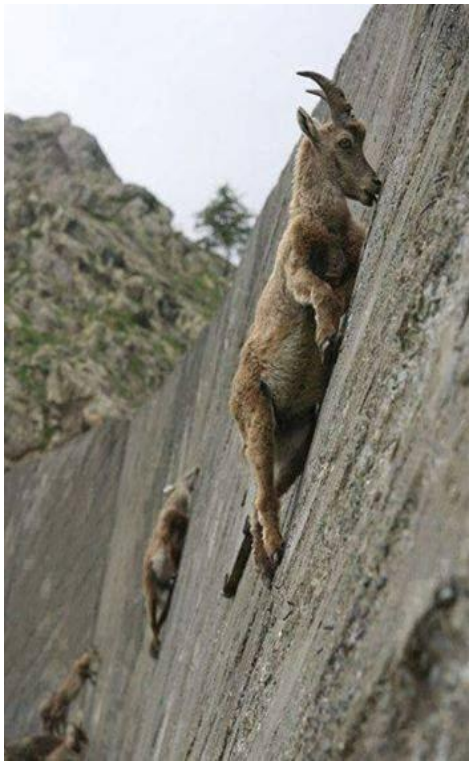
CD	Centroidal Dynamics
CoM	Center of Mass
CoP	Center of Pressure
DAE	Differential Algebraic Equation
DDP	Differential Dynamic Programming
DoF	Degrees of Freedom
FPA	Foot Placement Algorithm
iLQG	Iterative Linear Quadratic Gaussian Regulator
IMU	Inertial Measurement Unit
LCP	Linear Complementary Problem
LP	Linear Program
LIPM	Linear Inverted Pendulum Model
MPC	Model Predictive Control
NLP	Nonlinear Programming Problem
NMPC	Nonlinear Model Predictive Control
OCP	Optimal Control Problem
ODE	Ordinary Differential Equation
PR-MPC	Policy Regularized Model Predictive Control
QP	Quadratic Program
RBD	Rigid Body Dynamics
RK4	Runge-Kutta 4th-order
SLIP	Spring Loaded Inverted Pendulum
SLQ	Sequential Linear Quadratic Programming
SQP	Sequential Quadratic Programming
SRBD	Single Rigid Body Dynamics
TO	Trajectory Optimization
ZMP	Zero Moment Point

1 Introduction

Robotics has gained much attention in recent years. The goal is the construction of systems that can assist humans and replicate human actions in an automated way. For years, robots have been used mostly in the industry, as they can substitute humans in repetitive and tedious tasks, e.g., manufacturing, assembling, packaging, etc. Robotic systems are now intended to be used in disaster scenarios, such as in an area following an earthquake, or places where hazardous waste exists, which are risky to be approached by humans. Figure 1-1 shows pictures of animals performing highly dynamic motions and balancing on an almost vertical rock wall reliably. Legged animals possess natural abilities that currently outperform robotic systems. Compared to humans or animals, robots are not yet capable of fully autonomous movement in an uncontrolled environment. This challenge is addressed in this thesis. Three different motion planning algorithms are studied and proposed to yield alternative ways for the locomotion of a quadruped from an initial standing pose to a final one.

(a) Cheetah running

(b) Cheetah running



(c) Goat climbing

Figure 1-1. Legged animals performing highly dynamic motions.

1.1 Motivation

A major motivation for investigating legged locomotion is its usefulness in unstructured environments. Legged robots can traverse non-smooth, unstable terrain such as rocky terrain that wheeled robots or tracks cannot access. Quadrotors or gliders cannot carry heavy loads or manipulate objects the same way a ground vehicle can. Legged designs are powerful due to their versatility in accomplishing complex tasks. Both the natural and the urban environment

is mostly accessible by legged systems owing to their ability to take discrete steps. They are capable of crossing gaps to reach the opposite side, jump over obstacles, climbing stairs, and adapting to surface changes. No matter what the ground is, they can adjust their behavior to walk on roads, woods, grass or even mud without getting stuck.

This thesis concerns the locomotion of the quadruped Argos, which is designed by the Legged Robots Team at Control Systems Lab – NTUA. Its main purpose is the inspection of vineyards, removing the need for daily human field presence. The motion planning algorithms studied herein use proprioceptive sensors feedback, and produce joint torques to force a quadruped move on flat terrain. The motion plans that are produced are based on optimization methods; an appropriate criterion is minimized while enforcing physical constraints.

1.2 Literature Review

At an early stage, trajectory generation arose from observations of animal gaits, leading to the investigation of efficient heuristics. In [50], the foot placement algorithm (FPA), proposed by Raibert, achieved balancing control of a hopping robot in terms of speed, posture, and height control. Another common physics-based heuristic that has proven to be highly influential in legged systems is the capture point heuristic which can be used when the robot is treated like a linear inverted pendulum [49]. Similar approaches approximate the full dynamics with the Spring-Loaded Inverted Pendulum (SLIP) model [12], which simplifies the high dimensional legged system problem to a small number of well-understood states. Regarding the earliest successful control schemes, the hydraulically actuated quadruped HyQ from IIT performs a trotting gait by employing a simple virtual model control approach that emulates the dynamic characteristics of linear springs for each leg [28]. Accordingly, dynamic quadruped trotting at various speeds is achieved using an active compliance controller to drive each leg [41].

The capabilities of legged robots can be further enhanced if the motion plans are determined by optimization methods. In a recent work, the Legged Robots Team has implemented CoM trajectory and footfalls optimization using MATLAB fmincon [16] and after that, a gait optimization algorithm was formulated using the TOWR optimization software [38]. Furthermore, the optimization algorithms that were developed in [15] led to life-like motions and dynamic gaits. It is worth mentioning that the second control framework presented in this thesis, is primarily based on [23], in which the robot's posture was controlled through the optimal distribution of virtual forces, that should ideally act on the main body, to the stance legs. Finally, a really useful implementation of optimization is demonstrated in [46], where a Linear Program (LP) determines the online foothold selection and body motion generation while ensuring that the realization of the generated motion plans will not exceed the actuation limits.

Approaches based on Model Predictive Control (MPC) have also shown promising results. The MIT Cheetah 2 used MPC to place optimally its feet and adjust its pose before jumping over unexpected obstacles [48]. Its descendant, MIT Cheetah 3 stabilizes a diverse set of quadrupedal gaits by implementing a policy-regularized model-predictive control (PR-MPC) [11], which biases the solution of the MPC towards common heuristics from the literature to accelerate the computations. The MPC formulation incorporates the system dynamics which can be approximated by common simplified models. MIT Cheetah 3 [10], [17], and Mini Cheetah [34] use Single Rigid Body Dynamics (SRBD) model, which is a good approximation of the real model and speeded up the computations greatly. The same dynamic model was

used in [39], where the trajectory optimization problem was formulated using a multiple shooting method, enabling a humanoid robot to climb stairs.

A wide variety of methods can solve the optimal control problem. Differential Dynamic Programming (DDP), which is based on dynamic programming [40], is an optimal control algorithm in the class of trajectory optimization that has been applied successfully in legged systems. Iterative Linear Quadratic Gaussian Regulator (iLQG) [59] and Sequential Linear Quadratic Programming (SLQ) [55] are iterative methods that can handle nonlinear optimization problems. Another trajectory optimization formulation that generates plans for the body motion and the footholds using direct collocation with an SQP solver was proposed in [21]. It produces a coarse plan in a few seconds and due to the integration of simple state-feedback laws and a hierarchical whole-body controller in the motion execution, the robot can successfully follow the motion plans and it is robust to perturbations.

Some recent approaches overcome the restrictions that are applied to the achievable motions by the predefined contact schedule. The algorithms developed in [19], [42], [63] address only the contact sequence, but allow the optimization to eliminate specific phases by setting their duration to zero. A whole-body Nonlinear Model Predictive Control approach for Rigid Body Dynamics (RBD) systems subject to contacts, in which the contact locations, sequences and timings are optimized, is presented in [43]. The nonlinear optimal control problem was solved at a rate of 150 Hz for a time horizon of 0,5 s.

1.3 Structure

The current thesis is organized as follows. Chapter 2 presents the system parameters of Argos, which include the physical properties and the definition of the three degrees of freedom in each leg. Then, the forward and inverse kinematics are analyzed to relate the foot position to the body position through the angular positions of the joints. Next, the dynamic models that approximate the physical model of the quadruped are discussed. A static analysis follows that gives an estimation of the foot forces and the respective joint torques when the quadruped is in stance. Additionally, the simulation environment of Simscape, and how the foot-ground interaction is modelled, is described. Finally, the results of simulating the robot in stance are compared with those of the analytical solution.

Chapter 3 describes the theory behind the optimal trajectory planning. The commonly used direct methods for the numerical solution of the trajectory optimization problem are analyzed and evaluated. Several gradient-based nonlinear programming solvers are listed, and the two most used solvers are compared. A two-link manipulator is stabilized at the upper position using the direct methods and two different optimization libraries which are also evaluated based on their computational speed.

Chapter 4 can be separated in three main parts. The first part demonstrates a simple but effective control framework that is used for the locomotion of Argos. It consists of the gait scheduling, the definition of foot trajectories in Cartesian space which are mapped to joint angles through inverse kinematics and the implementation of a PD controller that tracks the desired motion plans. The second part describes a second motion planning approach, in which the robot tracks velocity and turn rate commands. Initially, a gait pattern is proposed and then the desired footholds and the foot trajectories are defined. The desired body pose is controlled by adjusting the virtual foot forces through the formulation of an optimization problem. A PD controller is also designed to track the desired foot trajectories. The last part develops the

implementation of a trajectory optimization algorithm using the Hermite-Simpson direct collocation method. Firstly, the techniques that can handle the discontinuous dynamics of the legged systems are explored and then the trajectory optimization problem is formulated step by step. The outputs are the optimal motions plans for the body pose, the foot trajectories and the foot forces that minimize the given objective function. In addition, the physical constraints are discussed analytically. A section that presents the simulation results follows at the end of each motion planning approach.

Finally, Chapter 5 discusses the importance of the results and the implications of the work developed. Then, several suggestions for future work are listed.

Appendices contain helpful supplementary information Appendix A includes the code related to forward kinematics that was developed using Mathematica. Accordingly, Appendix B includes the code for the calculation of the geometric Jacobian. Appendix C provides supplementary information about the Simpson's rule of integration. Appendix D and Appendix E describe the implementation of the two control frameworks in MATLAB-Simscape simulation environment and finally Appendix F refers to the code of the Trajectory Optimization formulation.

2 System Modelling

2.1 Robot Model

Argos is a quadruped robot (Figure 2-1, Figure 2-2) designed by the Legged Robots Team at Control Systems Lab – NTUA. Its main purpose is the inspection of vineyards. This is a complex task, but this work focuses on the robot's motion planning. Argos is about 0.8 m tall and weighs 53 kg approximately. Each of the robot's four legs has three torque-controlled joints: hip, knee, and abduction/adduction. The quadruped has 12 actuated degrees of freedom (DoFs) and 6 unactuated DoFs which describe its pose. To control the system, measurements/ estimations of all the joint angles $\mathbf{q}_j \in \mathbb{R}^{12}$, the joint velocities $\dot{\mathbf{q}}_j$, as well as the pose $\mathbf{q}_b \in \mathbb{R}^6$ and the velocities $\dot{\mathbf{q}}_b$ of the main body are required. The position of each hip and foot at every time instance are obtained using the sensors' measurements and the system kinematics. There are also pressure sensors installed on the feet that provide information whether a leg is in contact with the ground (ground force $F_n \geq 0$).

Table 2-1, Table 2-2, and Table 2-3 list the robot's parameters. M is the mass of the body, and m_1, m_2, m_3 are the masses of upper segment, lower segment, and leg roll segment. The I_{xx}, I_{yy}, I_{zz} are the body and the four leg roll segments moments of inertia. The lengths listed in Table 2-3 are shown in Figure 2-1 and Figure 2-2.

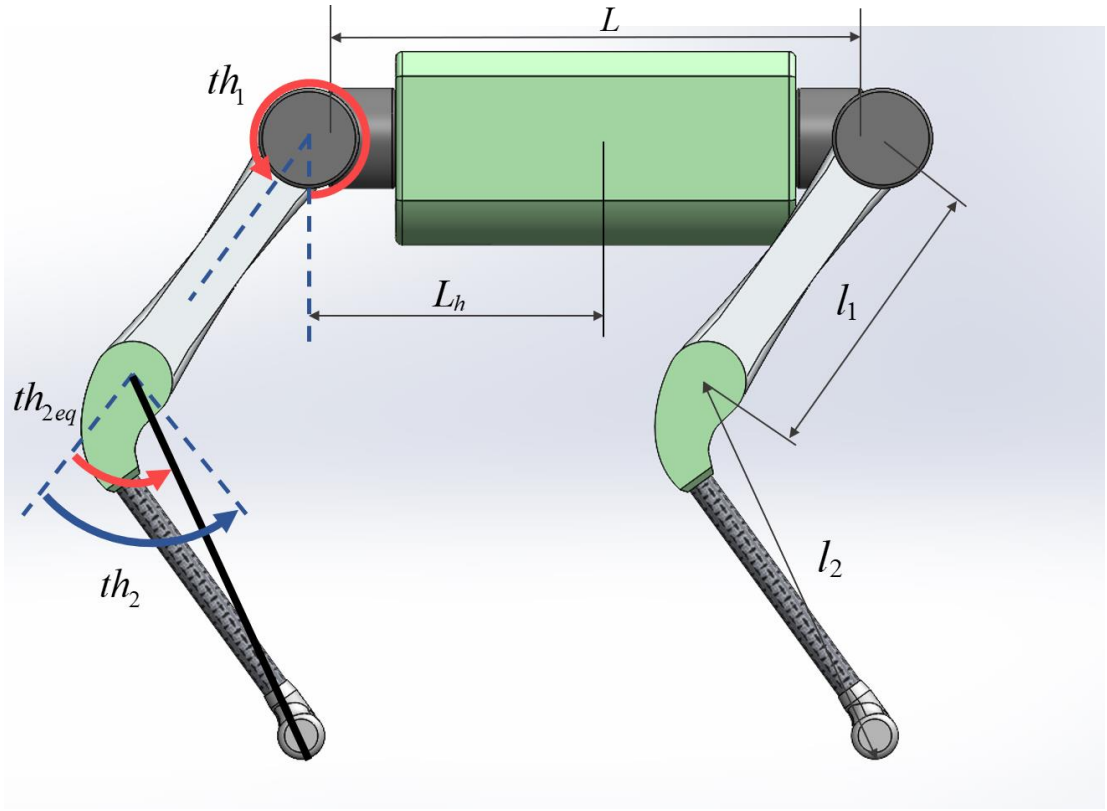


Figure 2-1. Geometrical properties of Argos - Left view.

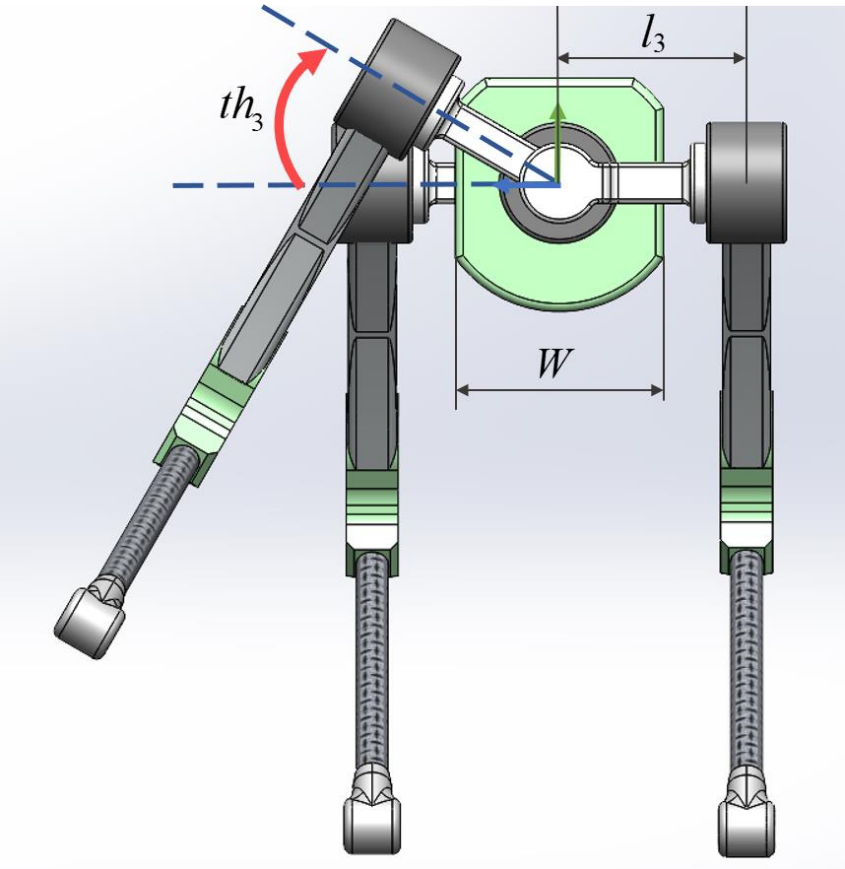


Figure 2-2. Geometrical properties of Argos - Front view.

Table 2-1. Robot's mass properties.

Body	M	25 kg
Leg upper segment	m_1	2.91 kg
Leg lower segment	m_2	1.74 kg
Leg roll segment	m_3	2.34 kg
Total Mass	M_{total}	52.96 kg

Table 2-2. Robot's moments of Inertia.

Main body (x axis)	I_{xx}	0.26 kgm^2
Main body (y axis)	I_{yy}	1.05 kgm^2
Main body (z axis)	I_{zz}	1.07 kgm^2

Table 2-3. Robot's geometrical properties.

Body Length (x axis)	L	0.8 m
CoM to Hip distance (x axis)	L_h	0.43 m
Body Width (z axis)	W	0.26 m
Leg upper segment	l_1	0.45 m
Leg lower segment	l_2	0.62 m
Leg roll segment	l_3	0.23 m

2.2 Kinematics

2.2.1 Forward Kinematics

The quadruped can have different configurations depending on the leg coordinates. As it is easier to determine the motion of the robot in the Cartesian space, equations that map the joint angles to the desired motions in the Cartesian space are presented in this and the following section. Initially, to determine the position and orientation of the robot's CoM, the transformation matrix that expresses the base frame B relative to the inertial frame I (Figure 2-3) is obtained in (2-4) using the rotation matrices (2-1), (2-2), (2-3) in an Euler angle z-y-x sequence. Euler angles are used to describe the orientation of the floating body with respect to a fixed inertial frame. This is required when the motion plans expressed in the inertial frame need to be mapped to the base frame and then to a leg motion. The rotation about the three principal axis x, y, z is given by (2-1), (2-2), (2-3) respectively. The x_m, y_m, z_m are the coordinates of the CoM's position relative to the inertial frame and the q_x, q_y, q_z denote the rotation angle of the CoM about the x, y, and z axis respectively.

$$\mathbf{T_x} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(q_x) & -\sin(q_x) & 0 \\ 0 & \sin(q_x) & \cos(q_x) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2-1}$$

$$\mathbf{T_y} = \begin{bmatrix} \cos(q_y) & 0 & \sin(q_y) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(q_y) & 0 & \cos(q_y) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2-2}$$

$$\mathbf{T_z} = \begin{bmatrix} \cos(q_z) & -\sin(q_z) & 0 & 0 \\ \sin(q_z) & \cos(q_z) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2-3}$$

$$\mathbf{T_M} = \mathbf{T_x}\mathbf{T_y}\mathbf{T_z} = \begin{bmatrix} 1 & 0 & 0 & x_m \\ 0 & 1 & 0 & y_m \\ 0 & 0 & 1 & z_m \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2-4}$$

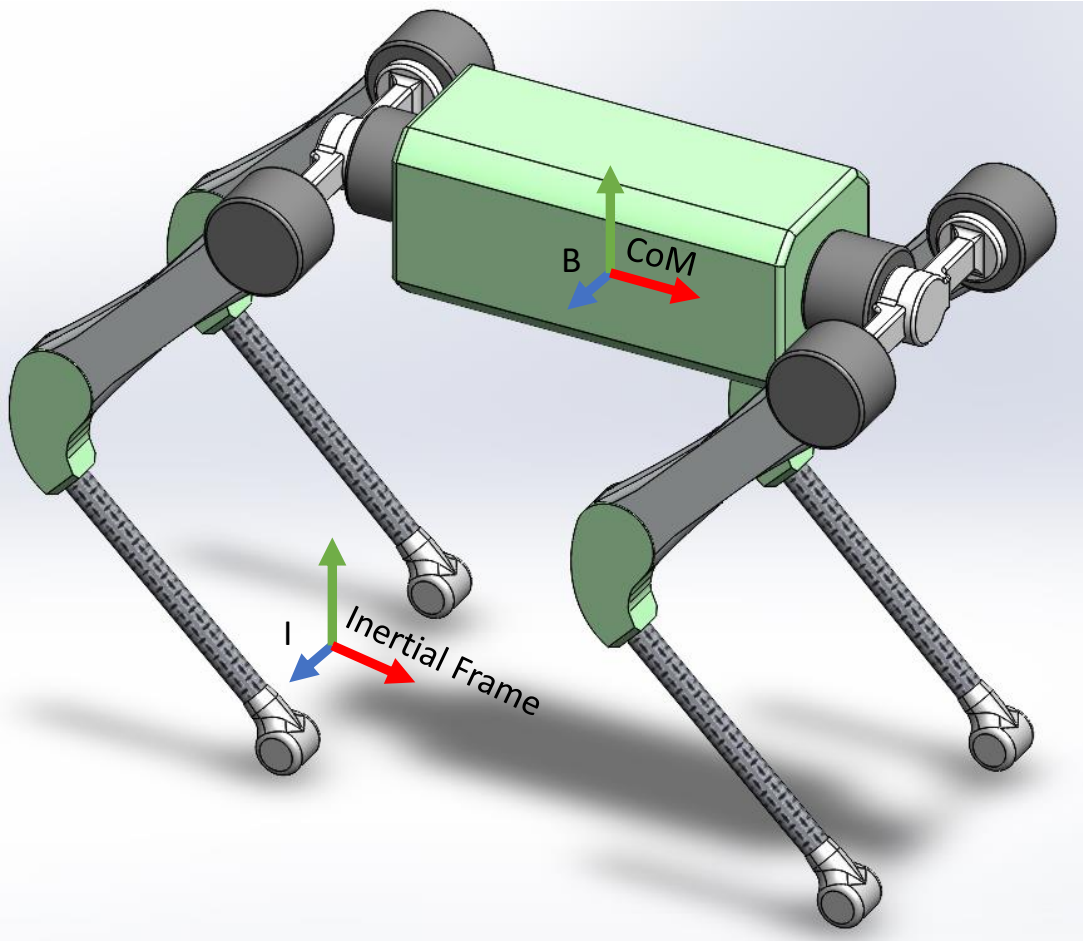


Figure 2-3. Inertial frame I - Body fixed frame B.

Table 2-4. Coordinate systems.

The CS of the center of the body	CS_B	$[x_B \quad y_B \quad z_B]$
The main CS of each leg	CS_0	$[x_0 \quad y_0 \quad z_0]$
The CS of the abd/add joint	CS_1	$[x_1 \quad y_1 \quad z_1]$
The CS of the hip joint	CS_2	$[x_2 \quad y_2 \quad z_2]$
The CS of the knee joint	CS_3	$[x_3 \quad y_3 \quad z_3]$
The CS of the foot	CS_E	$[x_E \quad y_E \quad z_E]$

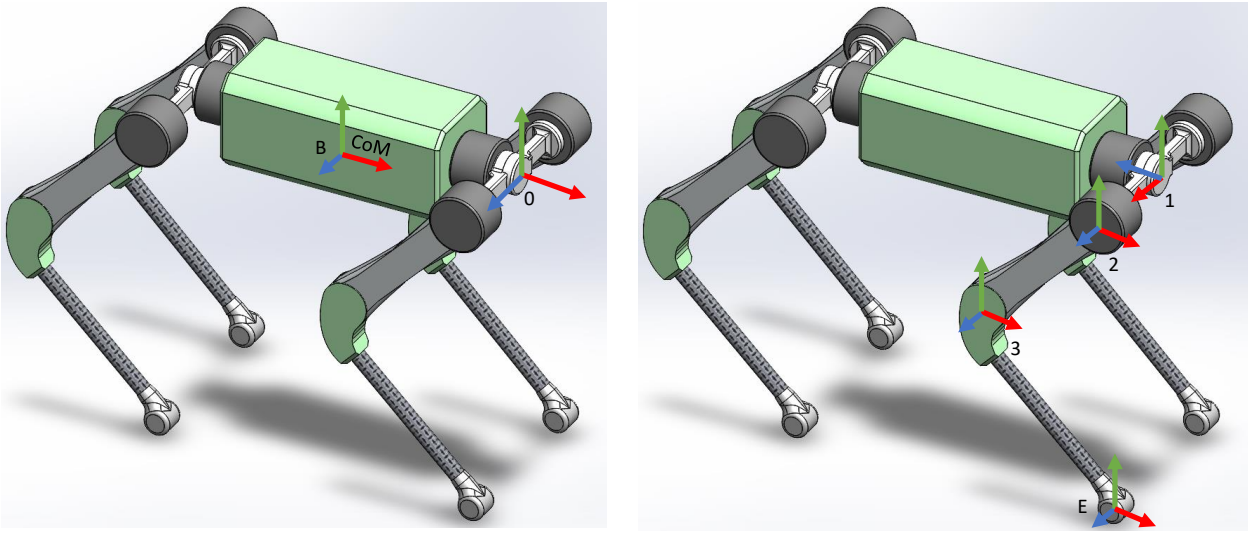


Figure 2-4. Coordinate Systems. The X, Y, Z axes are denoted by red, green and blue color respectively.

Table 2-5. Transformation matrices.

	Front Right	Front Left
$CS_B \rightarrow CS_0$[1]	$TRANS_x\left(\frac{L}{2}\right)$	$TRANS_x\left(\frac{L}{2}\right)$
$CS_0 \rightarrow CS_1$	$ROT_y\left(-\frac{\pi}{2}\right)ROT_z\left(th_3\right)$	$ROT_y\left(\frac{\pi}{2}\right)ROT_z\left(th_3\right)$
$CS_1 \rightarrow CS_2$	$TRANS_x(l_3)ROT_y\left(\frac{\pi}{2}\right)ROT_z\left(th_1\right)$	$TRANS_x(l_3)ROT_y\left(-\frac{\pi}{2}\right)ROT_z\left(th_1\right)$
$CS_2 \rightarrow CS_3$	$TRANS_y(-l_1)ROT_z\left(th_{2eq}\right)$	$TRANS_y(-l_1)ROT_z\left(th_{2eq}\right)$
$CS_3 \rightarrow CS_E$	$TRANS_y(-l_2)$	$TRANS_y(-l_2)$

The variables th_1, th_2, th_{2eq} and th_3 are the joint angles, which are shown in Figure 2-1 and Figure 2-2. The joint angle th_2 is defined as the angle between link 1 and 2. However, the purpose of this analysis is the derivation of the expression that relates the position of the contact point with the CS_0. Thus, in the rest of this work, the joint angle th_{2eq} will be used. The forward kinematic matrix given in (2-5) is derived from the transformations listed in Table 2-5 and each coordinate system is represented in Figure 2-4. Equation (2-5) is expressed in symbolic form using Wolfram Mathematica (see Appendix A).

[1] The Transformation matrix for the rear legs gets a negative sign as $TRANS_x\left(-\frac{L}{2}\right)$.

$$ {}^{0}T_E = {}^{0}T_1\,{}^{1}T_2\,{}^{2}T_3\,{}^{3}T_E \tag{2-5} $$

The positions of the left and right feet in Cartesian space are given by:

$$ {}^{0}x_E = l_1 \sin\left(th_1\right) + l_2 \sin\left(th_1 + th_{2eq}\right) \tag{2-6} $$

$$ {}^{0}y_E = l_3 \sin\left(th_3\right) - \cos\left(th_3\right)\left(l_1 \cos\left(th_1\right) + l_2 \cos\left(th_1 + th_{2eq}\right)\right) \tag{2-7} $$

Left: $$ {}^{0}z_E = -l_3 \cos\left(th_3\right) - \sin\left(th_3\right)\left(l_1 \cos\left(th_1\right) + l_2 \cos\left(th_1 + th_{2eq}\right)\right) \tag{2-8} $$

Right: $$ {}^{0}z_E = l_3 \cos\left(th_3\right) + \sin\left(th_3\right)\left(l_1 \cos\left(th_1\right) + l_2 \cos\left(th_1 + th_{2eq}\right)\right) \tag{2-9} $$

The foot velocity in Cartesian space (2-10) is related to the joint angular velocities (2-13) by differentiating (2-6)-(2-9). The foot acceleration (2-14) is obtained in a similar way by differentiating (2-10).

$$ \mathbf{v}_E = \mathbf{J}\dot{\mathbf{q}} \tag{2-10} $$

$$ \mathbf{v}_E = \begin{bmatrix} {}^{0}\dot{X}_E \\ {}^{0}\dot{Y}_E \\ {}^{0}\dot{Z}_E \end{bmatrix} \tag{2-11} $$

$$ \mathbf{J} = \begin{bmatrix} \frac{\partial\,{}^{0}X_E}{\partial th_1} & \frac{\partial\,{}^{0}X_E}{\partial th_{2eq}} & \frac{\partial\,{}^{0}X_E}{\partial th_3} \\ \frac{\partial\,{}^{0}Y_E}{\partial th_1} & \frac{\partial\,{}^{0}Y_E}{\partial th_{2eq}} & \frac{\partial\,{}^{0}Y_E}{\partial th_3} \\ \frac{\partial\,{}^{0}Z_E}{\partial th_1} & \frac{\partial\,{}^{0}Z_E}{\partial th_{2eq}} & \frac{\partial\,{}^{0}Z_E}{\partial th_3} \end{bmatrix} \tag{2-12} $$

$$ \dot{\mathbf{q}} = \begin{bmatrix} \dot{th}_1 \\ \dot{th}_{2eq} \\ \dot{th}_3 \end{bmatrix} \tag{2-13} $$

$$ \dot{\mathbf{v}}_E = \dot{\mathbf{J}}\dot{\mathbf{q}} + \mathbf{J}\ddot{\mathbf{q}} \tag{2-14} $$

The geometric Jacobian $\mathbf{J}$ is expressed in symbolic form using Wolfram Mathematica (see Appendix B).

2.2.2 Inverse Kinematics

In the inverse kinematic analysis, the position of the foot is provided, and the unknown variables are the joint angles that bring the foot to the desired position. Having already defined the forward kinematics, each joint angle can be uncoupled using simple geometry and trigonometry [1]. The variables used in (2-15)-(2-18) are displayed in Figure 2-5. The problem has two solutions (elbow up, elbow down), but due to the kinematic constraints, one of those is acceptable.

$$th_{2eq} = \arccos\left(\frac{x_E^2 + y_E^2 + z_E^2 - l_1^2 - l_2^2 - l_3^2}{2 l_1 l_2}\right) \tag{2-15}$$

Right Legs: $th_3 = \psi_3 - \varphi_3 = \arctan 2\left(\sqrt{y_E^2 + z_E^2 - l_3^2}, l_3\right) - \arctan 2\left(-y_E, z_E\right)$ (2-16)

Left Legs: $th_3 = \psi_3 - \varphi_3 = \arctan 2\left(\sqrt{y_E^2 + z_E^2 - l_3^2}, l_3\right) - \arctan 2\left(-y_E, -z_E\right)$ (2-17)

$$th_1 = \varphi_1 - \psi_1 = \arctan 2\left(x_E, \sqrt{y_E^2 + z_E^2 - l_3^2}\right) - \arctan 2\left(l_2 \sin(th_{2eq}), l_1 + l_2 \cos(th_{2eq})\right) \tag{2-18}$$

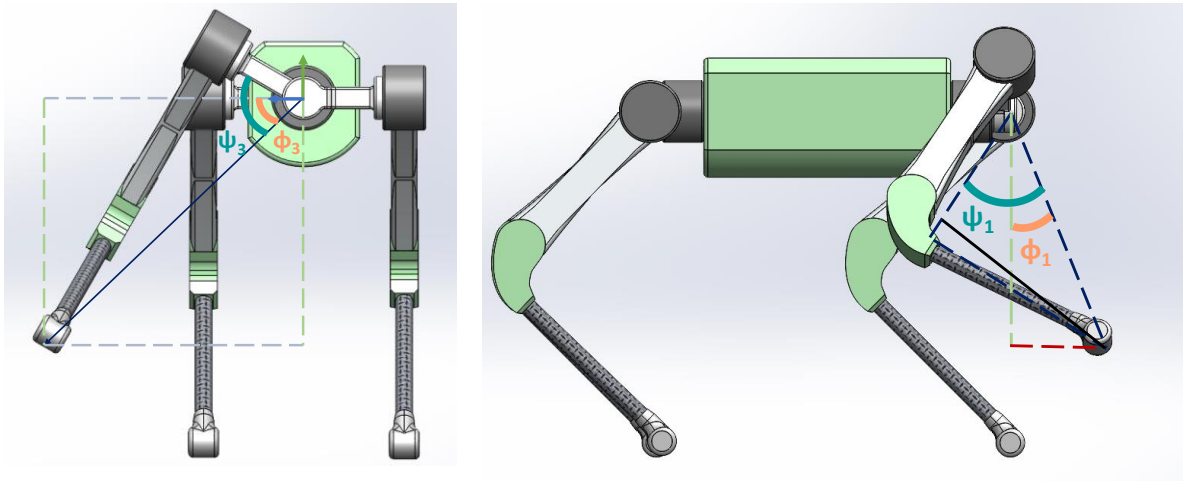


Figure 2-5. Inverse Kinematics - Geometrical Analysis.

2.3 Dynamic Models

There are several dynamic models in which the states, the controls, and the relationship between them differ significantly. These representations are approximations of the physical model with assumptions to simplify it and make it handy. A motion plan is acceptable if the modelling equations are always fulfilled. In general, the dynamics of a system is represented by Ordinary Differential Equations (ODEs) (2-19). Some of the most common dynamical models will be described next [62], starting from the most complex model, and ending up to the one including the most assumptions.

$$\dot{x} = \mathrm{F}\left(x(t), u(t)\right) \tag{2-19}$$

2.3.1 Rigid Body Dynamics

The quadruped robot studied herein consists of multiple rigid bodies (the main body and the four legs) (Figure 2-6). Each leg has three degrees of freedom. The links of the legs are connected to each other with rotary joints. It is assumed that bodies do not deform when external forces are applied. The states $\mathbf{q} = [\mathbf{q}_b^T \quad \mathbf{q}_j^T]^T \in \mathbb{R}^{18\times 1}$ comprise the position and orientation of the CoM ($q_b \in \mathbb{R}^{6\times 1}$) and the angle of each actuated joint ($q_j \in \mathbb{R}^{12\times 1}$). The model is described by:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{h}(\mathbf{q}, \dot{\mathbf{q}}) = \mathbf{S}^T\boldsymbol{\tau} + \mathbf{J}(\mathbf{q})^T\mathbf{f} \tag{2-20}$$

where $\mathbf{M} \in \mathbb{R}^{18\times18}$ is the joint-space inertia matrix, $\mathbf{h} \in \mathbb{R}^{18\times1}$ is the effect of Centrifugal, Coriolis and gravity terms, $\mathbf{S}^T = [\mathbf{0}_{12\times6} \quad \mathbf{I}_{12\times12}]^T$ is a Selection Matrix which applies the torque $\tau \in \mathbb{R}^{12\times1}$ to only the n joint rows and the Jacobian $\mathbf{J}$ maps forces $\mathbf{f} = [\mathbf{f}_1^T \quad \dots \quad \mathbf{f}_4^T]^T \in \mathbb{R}^{12\times1}$ at the 4 end-effectors to 6+12 dimensional generalized forces. Equation (2-20) can be separated into two groups; the one that is related to the underactuated part of the robot (2-21) and the rest that is fully-actuated since there is a torque-generating motor for each robot joint (2-22).

$$\mathbf{M}_u(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{h}_u(\mathbf{q},\dot{\mathbf{q}}) = \mathbf{J}_u(\mathbf{q})^T\mathbf{f} \tag{2-21}$$

$$\mathbf{M}_a(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{h}_a(\mathbf{q},\dot{\mathbf{q}}) = \boldsymbol{\tau} + \mathbf{J}_a(\mathbf{q})^T\mathbf{f} \tag{2-22}$$

The joint torques cannot directly influence the body motion, as the latter is determined by the contact forces. These contact forces are generated through the torques that the actuators apply. Consequently, the actuation system affects the body movement indirectly.

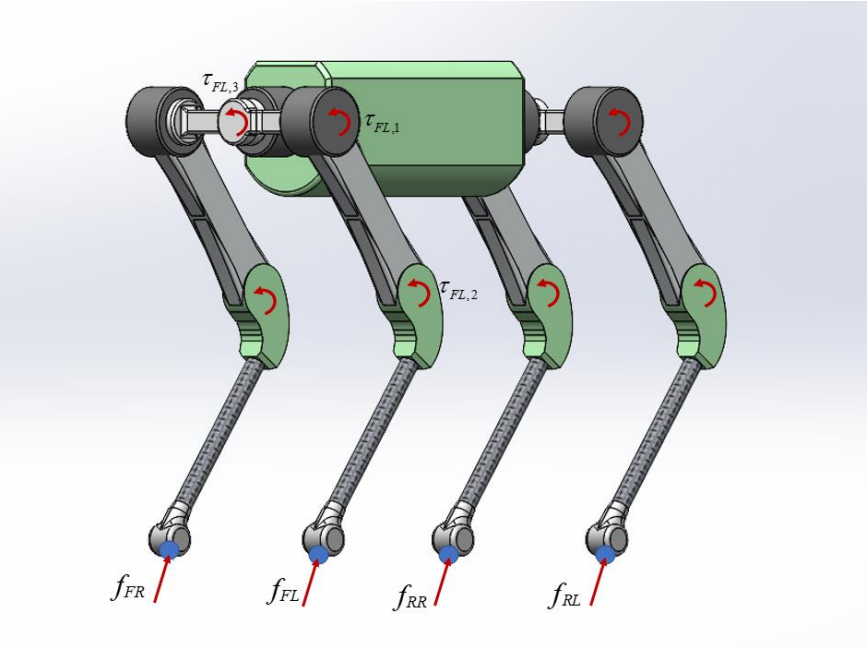


Figure 2-6. Visualization of Rigid Body Dynamics (RBD).

2.3.2 Centroidal Dynamics

As before the only assumption is that bodies do not deform, but in this model, the change of momentum is expressed at a body fixed frame at the body CoM. The dynamics are written as

$$\frac{d\left(\mathbf{A}\dot{\mathbf{q}}\right)}{dt} = \mathbf{A}(\mathbf{q})\ddot{\mathbf{q}} + \dot{\mathbf{A}}(\mathbf{q})\dot{\mathbf{q}} = \begin{bmatrix} m\mathbf{g} + \sum_{i=1}^{i=4}\mathbf{f}_i \\ \sum_{i=1}^{i=4}\mathbf{f}_i \times \left(\mathbf{r}(\mathbf{q}) - \mathbf{p}_i(\mathbf{q})\right) \end{bmatrix} \tag{2-23}$$

$$\mathbf{q} = [\mathbf{q}_b^T \quad \mathbf{q}_j^T]^T \in \mathbb{R}^{18\times1} \tag{2-24}$$

where $\mathbf{g}$ is the gravity acceleration, m is the entire body mass, $\mathbf{r}$ is the position of the main body CoM in Cartesian space, $\mathbf{p}_i$ denotes the position of each foot i in Cartesian space, and $\mathbf{f}_i$ is the force acting on each foot. The Centroidal Momentum Matrix (CMM) $\mathbf{A} \in \mathbb{R}^{6\times18}$ [45] maps the momentum of each rigid body to the CoM. In this model, the input to the system is the vector containing the external forces $\mathbf{f}_i$. The first term of the right-hand side denotes the sum of the forces that are applied to the CoM, while the second term denotes the sum of

torques applied to the CoM. The left-hand side expresses the effect of linear and angular change of momentum of all rigid bodies on the CoM. As shown in the right-hand side, the positions of the feet depend on the joint configuration, thus these are included to the states vector.

2.3.3 Single Rigid Body Dynamics

The single rigid body dynamics (SRBD) model (Figure 2-7) is described by (2-25)-(2-27), which neglect the effect of joint velocities at the total momentum and it is assumed that the full-body inertia does not deviate significantly as the feet change their position. Due to the additional assumptions compared to the Centroidal Dynamics model, the joint dependency is eliminated. The dynamics of the robot is independent of the joint space which makes it easy to enforce the constraints in Cartesian space. As shown in (2-25)-(2-27), $\ddot{\mathbf{r}}$ represents the base acceleration, $\boldsymbol{\omega}$ is angular velocity of the base calculated from Euler angles $\boldsymbol{\theta}$ and its rates of change $\dot{\boldsymbol{\theta}}$, m is the robot's total mass, $\mathbf{I} \in \mathbb{R}^{3\times3}$ is the constant inertia matrix calculated at the nominal position with respect to a frame anchored at the CoM and whose axis are parallel to the inertial frame, $\mathbf{g}$ is the gravity acceleration, $\mathbf{p}_i$ is the three dimensional position of the feet and $\mathbf{f}_i$ are the contact forces. The dynamics of the system includes the angular velocity $\boldsymbol{\omega}$ which is written as a function of $\dot{\boldsymbol{\theta}}$ and $\boldsymbol{\omega}$. The relation between the Euler angle rates (order of application: yaw, pitch, roll) and the angular velocity is presented in (2-29) . Both quantities are expressed relative to the inertial frame. The extraction of the transformation matrix is described analytically in [22].

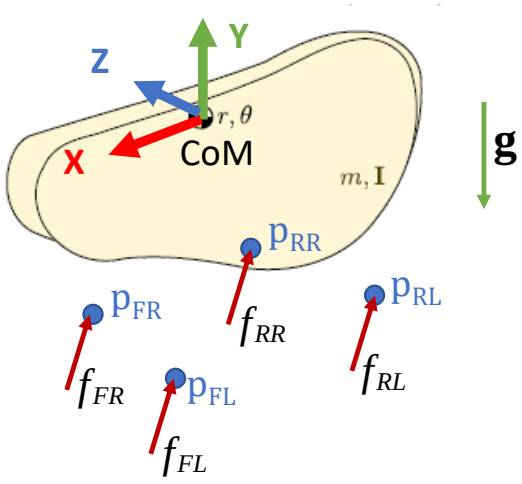


Figure 2-7. Visualization of the Single Rigid Body Dynamics (SRBD) model.

$$m\ddot{\mathbf{r}}(t) = \sum_{i=1}^{i=4} \mathbf{f}_i(t) + m\mathbf{g} \tag{2-25}$$

$$\mathbf{g} = \begin{bmatrix} 0 & -9.81 & 0 \end{bmatrix}^T \tag{2-26}$$

$$\mathbf{I}\dot{\boldsymbol{\omega}}(t) + \boldsymbol{\omega}(t) \times \mathbf{I}\boldsymbol{\omega}(t) = \sum_{i=1}^{i=4} \mathbf{f}_i(t) \times \left(\mathbf{r}(t) - \mathbf{p}_i(t)\right) \tag{2-27}$$

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_x \\ \theta_y \\ \theta_z \end{bmatrix} \tag{2-28}$$

$$\dot{\boldsymbol{\theta}} = \begin{bmatrix} \cos(\theta_z)\sec(\theta_y) & \sin(\theta_z)\sec(\theta_y) & 0 \\ -\sin(\theta_z) & \cos(\theta_z) & 0 \\ \cos(\theta_z)\tan(\theta_y) & \sin(\theta_z)\tan(\theta_y) & 1 \end{bmatrix} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \tag{2-29}$$

2.3.4 Linear Inverted Pendulum Model

The Linear Inverted Pendulum Model (LIPM) is a simplified form of the SRBD, since more assumptions are added to make it simpler and remove the nonlinearities of the cross product. The ODE equation that describes the model is given by (2-30) where the forward acceleration of the base $\ddot{r}_x$ depends on the position of the center of pressure (CoP) $p_{c,x}$. The CoM's position in x direction is denoted by r_x and the gravitational acceleration is denoted by g. The CoP refers to the point where the net torque caused by the ground-reaction forces is zero [33]. Replacing x indices by z in (2-30) determines the motion of the robot in z direction. The CoM's height h is constant, the footholds are at constant height, and the angular velocity and acceleration of the base are constrained to be zero. The input to this system is the CoP calculated by the vertical components of each individual contact force and the feet positions. Apparently, the contribution of the tangential components of the contact forces is omitted. This model cannot be used to generate complex motions and it is inappropriate for uneven terrain. A representation of LIPM is shown in Figure 2-8.

$$\ddot{r}_x = \frac{g}{h}(r_x - p_{c,x}) \tag{2-30}$$

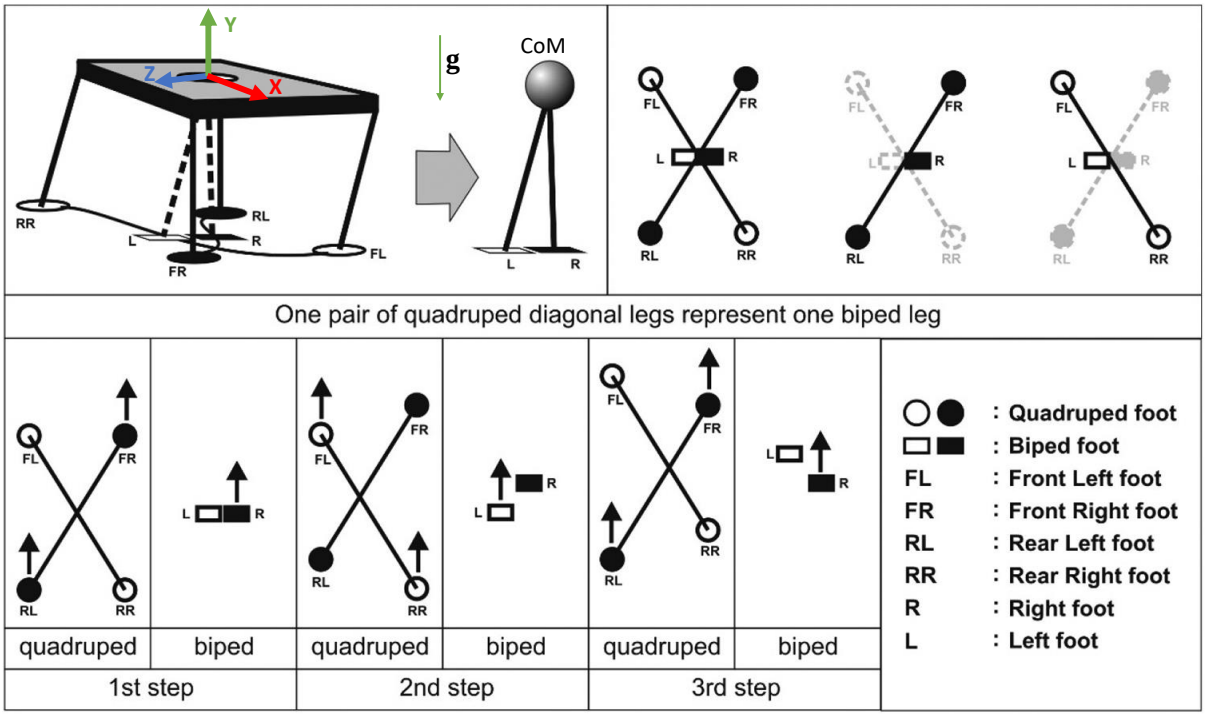


Figure 2-8. Linear Inverted Pendulum Model (LIPM) [58].

2.4 Statics

Under the assumptions of sagittal (longitudinal) plane symmetry and that the frictional forces are zero, the ground reaction forces that act on the stance feet can be computed explicitly when the robot is in static equilibrium. The goal of this analysis is a rough estimation of the torques required, how the configuration of the leg affects the magnitude of the torque at each joint and the examination of the differences between front and rear leg's loads. The results can be utilized as an estimation of the expected torque requirements and can be compared with those of the simulation.

According to the first assumption, half of the body mass is supported by the front right and rear right leg. As shown in Figure 2-9, gravitational forces for each member are applied at their centers of mass (CoM) and the ground reaction forces are applied at the contact points.

The equations of static equilibrium are given in (2-31)-(2-32). The problem has multiple solutions regarding the direction in which the contact forces are applied. The case where the contact forces are vertical to the ground and the friction forces that are exerted on the foot in the x and z direction are zero is analyzed next. Solving the system of equations (2-31)-(2-32) , the foot forces on the front and the rear leg are equal to $F_{y_{front}} = 106.2N$ and $F_{y_{rear}} = 153.6N$. The torques that are computed counterbalance the robot's weight. Greater joint torques do not imply that the foot will slip; it will start slipping if the force that is generated on the foot exceeds the friction force. The condition (2-33) where μ is the static friction coefficient of the ground, ensures that the foot does not slip. The pivot point around which the torques are generated in (2-32) is defined to be the torso's CoM. The distances l_{i_k} , where $i = 1,2,3$ symbolize upper, lower and roll segments respectively, and *k* refers to front or rear leg, are depicted in Figure 2-10.

$$\sum F_y = 0 \Rightarrow F_{y_{rear}} + F_{y_{front}} = \left(\frac{1}{2}M + 2m_1 + 2m_2 + 2m_3\right)g \tag{2-31}$$

$$\begin{aligned}\sum \tau = 0 \Rightarrow\ & m_1 g l_{1_{rear}} + m_2 g l_{2_{rear}} - F_{y_{rear}}\left(L_h + l_1 \sin\left(-th_1\right) - l_2 \sin\left(th_1 + th_{2_{eq}}\right)\right) \\ & -m_1 g l_{1_{front}} - m_2 g l_{2_{front}} + F_{y_{front}}\left(L_h - l_1 \sin\left(-th_1\right) + l_2 \sin\left(th_1 + th_{2_{eq}}\right)\right)\end{aligned} \tag{2-32}$$

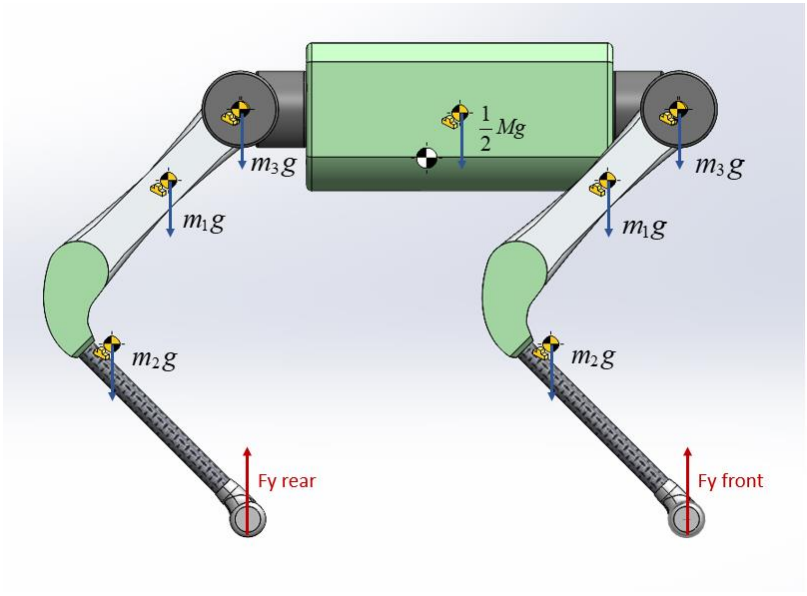


Figure 2-9. Gravitational forces and ground reaction forces.

$$F_x, F_z < \mu F_y \tag{2-33}$$

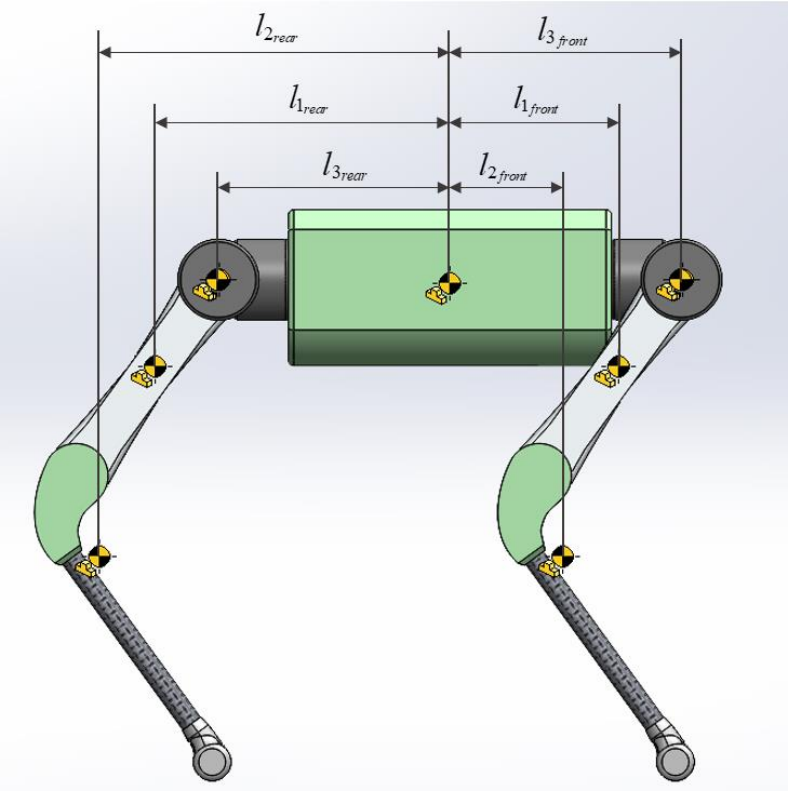


Figure 2-10. Distances between links' CoM and main body's CoM.

The knee torque is calculated by the equation of torque equilibrium at the isolated lower segment Figure 2-11 (left). Then, hip torque is calculated by the equation of torque equilibrium applied at the upper and the lower segments as depicted in Figure 2-11 (middle). Finally, the abduction torque is calculated by the equation of torque equilibrium applied at the whole leg Figure 2-11 (right). The geometrical parameters that are used in (2-34)-(2-39) are shown in Figure 2-12.

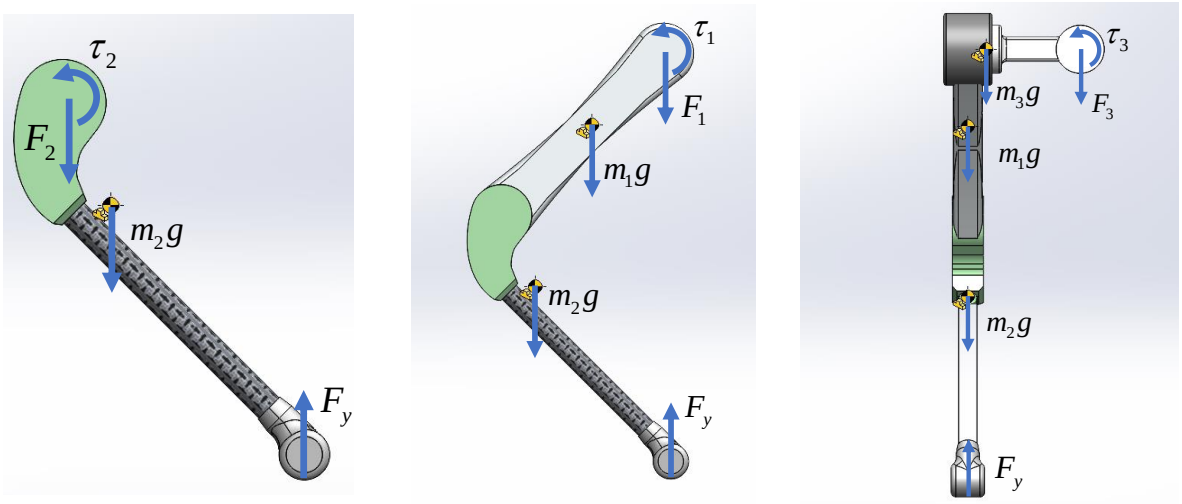


Figure 2-11. Left: Forces and torques at leg lower segment. Middle: Forces and torques at leg lower and upper segments. Right: Forces and torques at the whole leg.

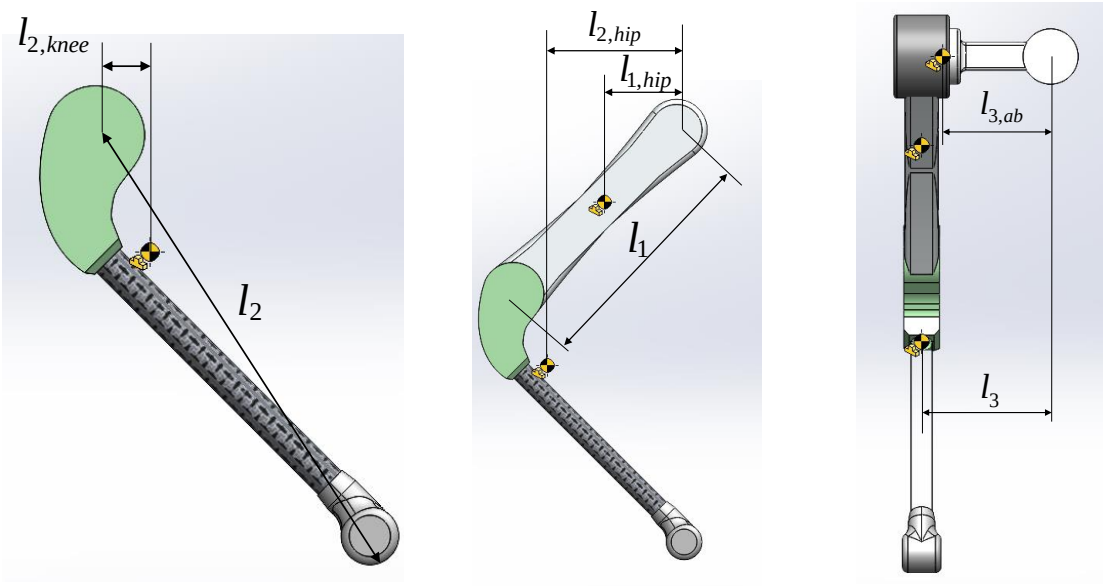


Figure 2-12. Geometrical parameters that relate the CoM of each link to the joints.

$$\tau_{1_{rear}} = -F_{y_{rear}}\left(l_1 \sin\left(th_1\right) + l_2 \sin\left(th_1 + th_2\right)\right) - m_1 g l_{1,hip} - m_2 g l_{2,hip} \quad (2\text{-}34)$$

$$\tau_{2_{rear}} = -F_{y_{rear}} l_2 \sin\left(th_1 + th_2\right) + m_2 g l_{2,knee} \quad (2\text{-}35)$$

$$\tau_{3_{rear}} = \left(\left(m_1 + m_2\right) g - F_{y_{rear}}\right) l_3 + m_3 g l_{3,ab} \quad (2\text{-}36)$$

$$\tau_{1_{front}} = -F_{y_{front}}\left(l_1 \sin\left(th_1\right) + l_2 \sin\left(th_1 + th_2\right)\right) - m_1 g l_{1,hip} - m_2 g l_{2,hip} \quad (2\text{-}37)$$

$$\tau_{2_{front}} = -F_{y_{front}} l_2 \sin\left(th_1 + th_2\right) + m_2 g l_{2,knee} \quad (2\text{-}38)$$

$$\tau_{3_{front}} = \left(\left(m_1 + m_2\right) g - F_{y_{front}}\right) l_3 + m_3 g l_{3,ab} \quad (2\text{-}39)$$

2.5 Simulation Environment

2.5.1 Plant description

The quadruped was designed in Solidworks, and it is composed of the main body and three links at each leg that are connected together through rotary joints. These individual parts are imported into Simscape, and their position is specified by rigid transforms and joints that connect the world frame to the robot. The world frame is placed on the ground, and it is connected to the body-fixed frame through a bushing joint. The bushing joint ensures that the main body has six degrees of freedom (three translational and three rotational). The initial position of the robot relative to the world frame is specified by setting the state target options in the bushing joint block.

2.5.2 Foot – Ground Interaction Model

The quadruped interacts with the environment while walking. Choosing an appropriate force model and ground parameters that approximate the actual conditions is a challenging task. However, MATLAB provides a library, namely Simscape Multibody Contact Forces Library

[52] that can be used to model the contact with the ground. A 'sphere to plane' contact model is selected (Figure 2-13) and the ground is modeled as a spring-damper system. The force model can be linear, nonlinear, or even custom made. A linear spring-damper resists a toe's penetration. According to that, a force is applied only along the direction of penetration and damping force is zero as penetration decreases (2-40), (2-41).

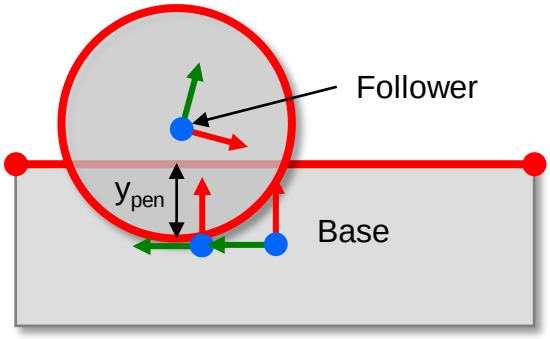


Figure 2-13. Foot - Ground Interaction model.

$$F_x = \begin{Bmatrix} k\,y_{pen} + b\,v_{pen} & y_{pen} > 0, v_{pen} > 0 \\ k\,y_{pen} & y_{pen} > 0, v_{pen} < 0 \\ 0 & y_{pen} \le 0 \end{Bmatrix} \quad (2\text{-}40)$$

$$F_y = 0 \quad (2\text{-}41)$$

The documentation of the contact forces library suggests starting with stiffness of $1e^4\ N/m$ and damping of $1e^2\ Ns/m$ and adjust from there. We assume flat ground with ground stiffness (k) and damping (b) equal to $1e^4\ N/m$ and $5e^2\ Ns/m$ respectively. These parameters are used for the walking simulation experiments. In the statics experiments presented in Section 2.6, stiffness and damping are set equal to $1e^5\ N/m$ and $1e^4\ Ns/m$ to avoid the oscillations in the transient state. When a leg is in stance phase, it must stay still in its position. For this reason, a stick-slip continuous friction model is chosen from the library (Figure 2-14), where the friction coefficients must be tuned appropriately as they greatly affect toe's slipping.

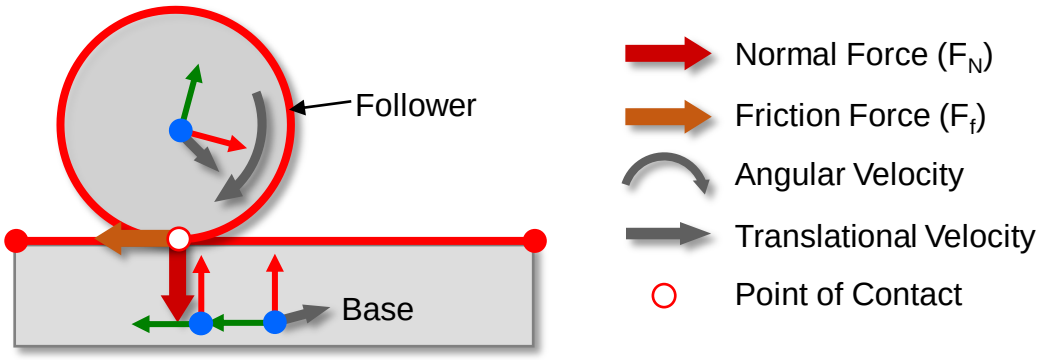


Figure 2-14. Friction model indicating forces applied to the ground.

The friction force is described by a stick-slip continuous model given by (2-42). It is equal to the normal force F_N multiplied by the friction coefficient μ which is a function of the relative velocity at the point of contact v_{poc} and the velocity threshold v_{th}.

$$F_f = F_N \mu \tag{2-42}$$

$$\mu = \begin{cases} v_{poc} \dfrac{\mu_s}{v_{th}} & v_{poc} < v_{th} \\ \mu_s - v_{poc} \dfrac{(\mu_s - \mu_k)}{0.5 v_{th}} & v_{th} \le v_{poc} \le 1.5 v_{th} \\ \mu_k & v_{poc} > v_{th} \end{cases} \tag{2-43}$$

The values of the different friction coefficients, illustrated in Figure 2-15, were set as $\mu_s = 0.9$, $\mu_k = 0.88$ and $v_{th} = 0.001 m/s$.

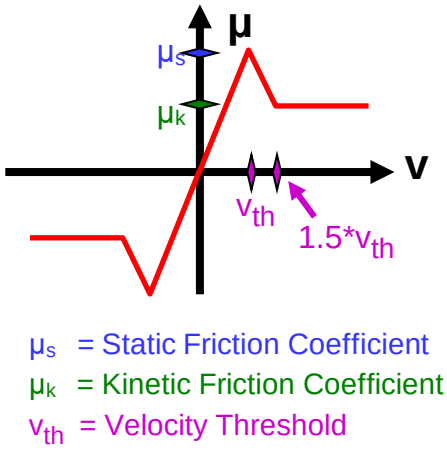


Figure 2-15. Friction coefficients.

2.5.3 Solver selection

The contact forces library that is imported recommends using the variable step solvers ode23t or ode15s. It was found that, ode15s needs almost half the time compared to the other. The library also recommends setting relative and absolute tolerances equal to 1e-4 and 1e-5 respectively and the maximum step size equal to 1e-2. However, these parameters can be automatically adjusted, so these settings can be also set to 'auto'. In summary, the solver ode15s with tolerances 1e-4 and 1e-5 and maximum step size 1e-2. was used in all the experiments presented in this work.

2.6 Statics Experiments for Simulation Environment Verification

Robot in stance phase - Inverse dynamics

In this simulation, the quadruped stands on the ground and keeps a specified leg configuration given in (2-46). The joint primitive is actuated in inverse dynamics mode by providing motion as input while having actuation torque automatically computed. The goal is to compare the simulation results to those from the static equilibrium analysis. Figure 2-16, Figure 2-17 represent the joint torques of the front and rear right leg.

$$q_{hip} = -\frac{\pi}{4} rad \tag{2-44}$$

$$q_{knee} = \frac{\pi}{2} \tag{2-45}$$

$$q_{ab} = 0 \tag{2-46}$$

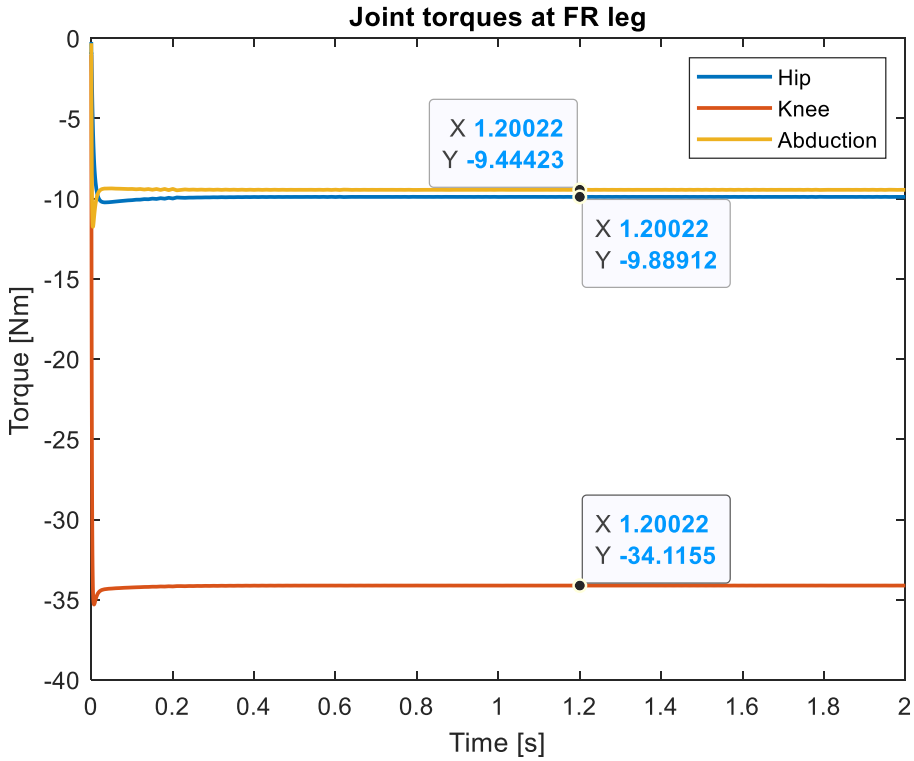


Figure 2-16. Joint torques at stance - Inverse dynamics - Front Right Leg.

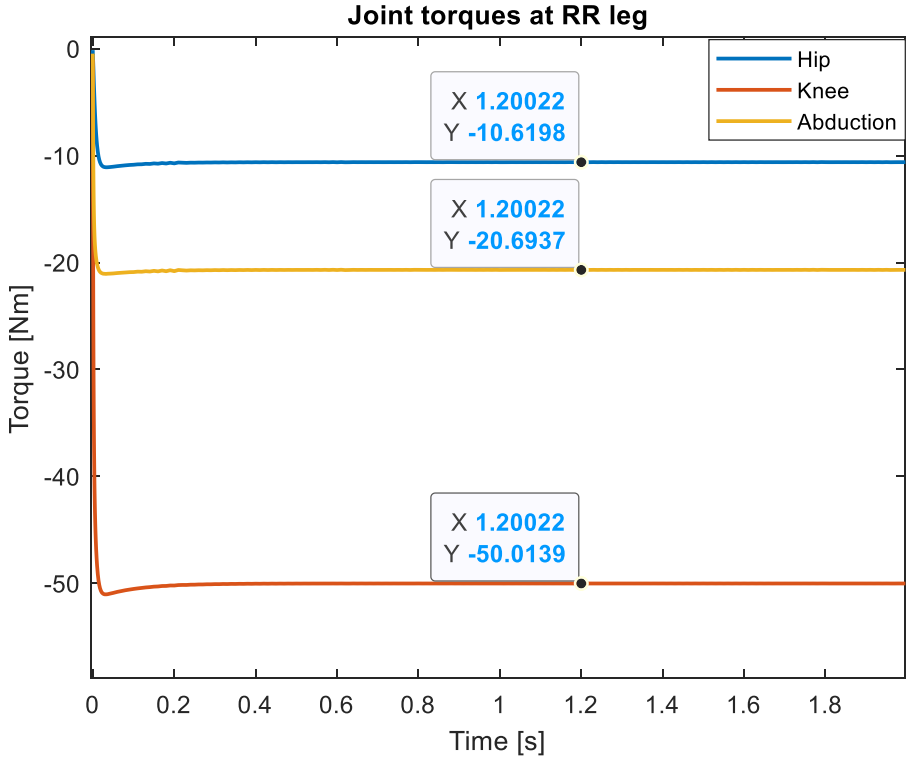


Figure 2-17. Joint torques at stance - Inverse dynamics - Rear Right Leg.

Robot in stance phase - Forward dynamics

If the joint primitive in the aforementioned case is actuated in forward dynamics mode, then the input is the actuation torque. The goal is to keep the legs still in their initial position. To handle this problem, a stabilizing PD controller regulates the torque needed, so that the quadruped stands at its nominal position.

$$\tau_j = K_P\left(q_j^d - q_j^a\right) + K_D\left(\dot{q}_j^d - \dot{q}_j^a\right) \tag{2-47}$$

The subscript j refers to the hip, knee or ab/ad joint, while the superscripts d and a refer to the desired and the actual angles respectively. Table 2-6 summarizes the PD controller gains. The gain values are quite large to ensure that the desired joint angular position and angular velocity are tracked. The term K_D is high to avoid the oscillations at the transient state. The steady state appears in just a few seconds. Figure 2-18, Figure 2-19 represent the joint torques of the front and rear right leg

Table 2-6. PD controller gains - Quadruped in stance.

Gains	Hip	Knee	Abduction
K_P	$1e^6$	$1e^6$	$1e^6$
K_D	$1e^4$	$1e^4$	$1e^4$

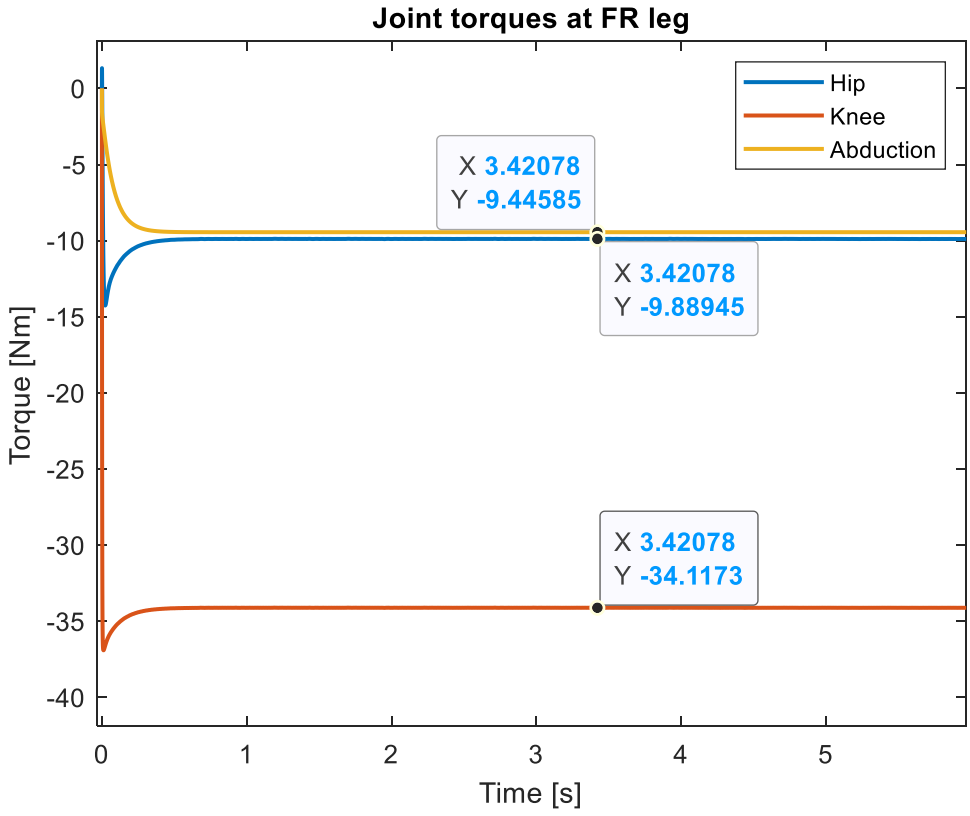


Figure 2-18. Joint torques at stance - Forward dynamics - Front Right Leg.

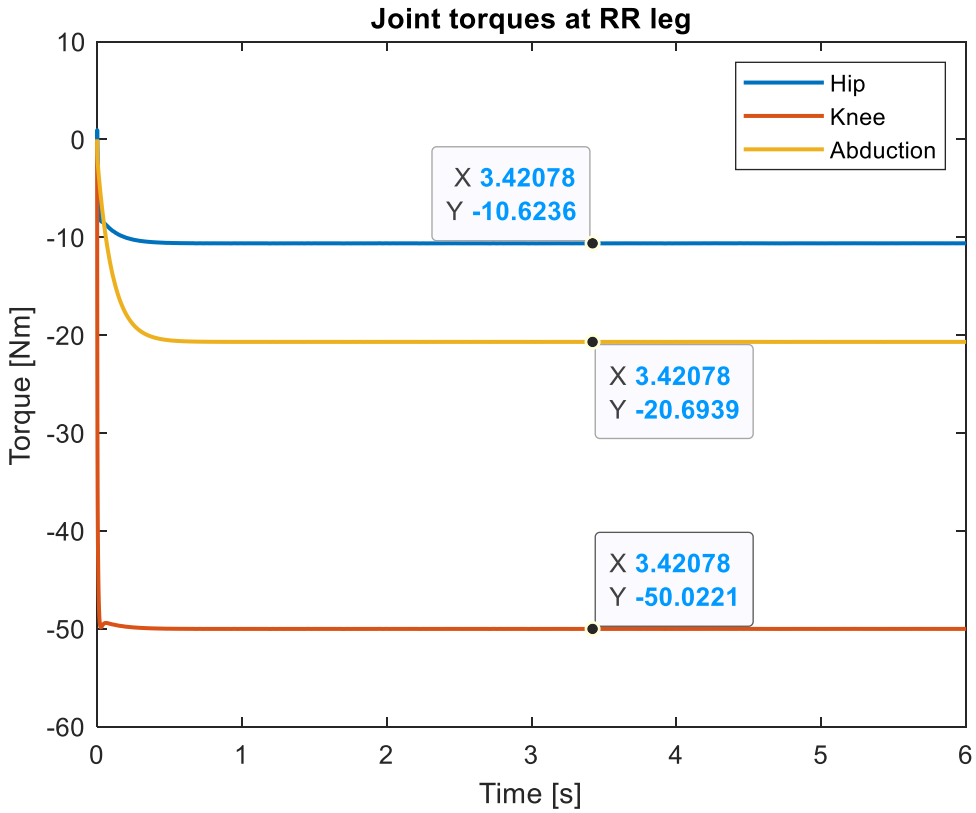


Figure 2-19. Joint torques at stance - Forward dynamics - Rear Right Leg.

Comparison with analytical solutions

Table 2-7 summarizes the results of the analytical solution, which are compared with those of the two aforementioned simulations. The simulation results are identical and their max deviation from the analytical solution is $0.3Nm$. In the simulation, the contact force consists of the force component which is normal to the ground and the friction forces along x and z axis. However, the friction forces have a little impact, since their magnitude is almost $0.01N$ in steady-state. The deviation may be caused by the assumption that the foot forces are exerted on a single point. Given that the cylindrical geometry of the foot penetrates the ground slightly, the forces are exerted on a contact area.

Table 2-7. Comparison between the analytical solution and the simulation results.

Torque [Nm]	Leg	Analytical Solution	Simulation (Inverse Dynamics)	Simulation (Forward Dynamics)
Hip	Front	-9.8	-9.9	-9.9
	Rear	-10.5	-10.6	-10.6
Knee	Front	-34.2	-34.1	-34.1
	Rear	-50.0	-50.0	-50.0
Abduction	Front	-9.7	-9.4	-9.4
	Rear	-20.9	-20.7	-20.7

3 Optimal Trajectory Planning

3.1 Introduction

This chapter focuses on the theory behind trajectory optimization (TO). Trajectory optimization is a process that outputs trajectories for a dynamical system while satisfying a set of physical constraints. The quadruped Argos is a dynamical system in which TO can be used to generate motion plans both for the unactuated main body and the actuated legs. The implementation of TO in Argos is investigated in Section 4.4. This chapter is organized as follows. Firstly, the term trajectory optimization is distinguished from the term optimal control and then some of the methods that can handle the TO problem are appraised. A simple TO example of a two-link manipulator is formulated in two optimization libraries, which are evaluated in terms of computation speed.

Optimal control is a mathematical optimization method which finds the control trajectories that drive a system to a desired state while minimizing an objective function and satisfying the constraints on the motion of the system. Before identifying the implementation details, the terms trajectory optimization and optimal control need to be distinguished. In general, trajectory optimization is a technique for computing an open-loop solution to an optimal control problem. These two terms must not be confused. The first one refers to the case of a function optimization problem that optimizes a given performance index $J(\mathbf{x})$, where the vector $\mathbf{x}$ represents the optimization variables and it consists of static parameters (or real numbers). This type of problem is called parameter optimization problem [35]. On the other hand, in the case of a functional optimization problem that optimize a given performance index $J(\mathbf{f}(t))$ where the optimization variables are vector functions $\mathbf{f}(t)$, the appropriate term is “optimal control” [51].

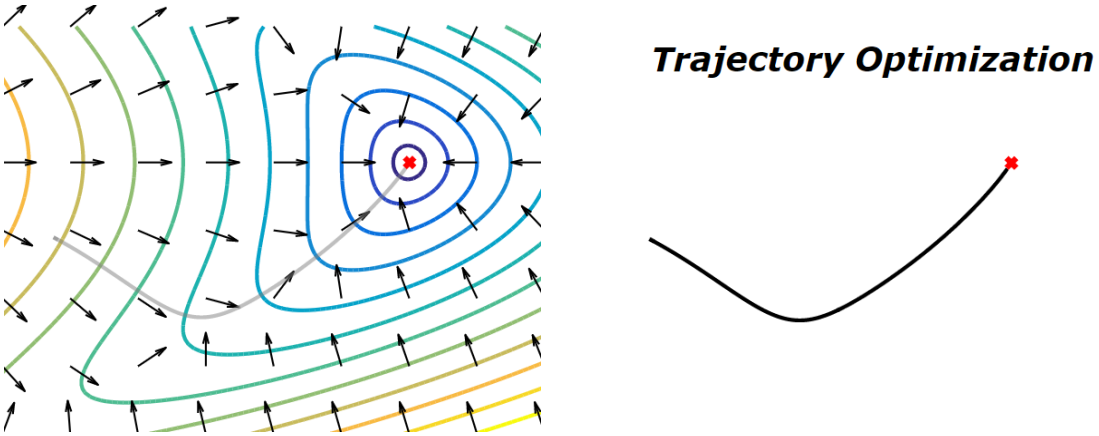


Figure 3-1. Left: Closed-loop solution - Right: open-loop solution [1].

The most general formulation of optimal control finds a closed loop solution which describes the ideal action that should be taken at every point in the entire space to reach the goal (Figure 3-1). In this case, the control is in general a function of state and time, $u = u(x,t)$. While this method ensures that the global solution is found, it is limited to lower-dimension problems since the complexity of the problem increases linearly with the number of states. On the other hand, using a local method and finding an open-loop solution is less difficult to compute and it is appropriate for higher-dimensional problems [35]. The goal of trajectory

optimization is to find an optimal open-loop control that is valid from only a single initial condition. This open-loop control is a function of time $u = u(t)$ and under its influence, the system follows an optimal trajectory from an initial to a final point. Since the method provides an open-loop feed-forward control input (Figure 3-1), it must be combined with a stabilizing controller when applied to a real system.

3.2 Approaches

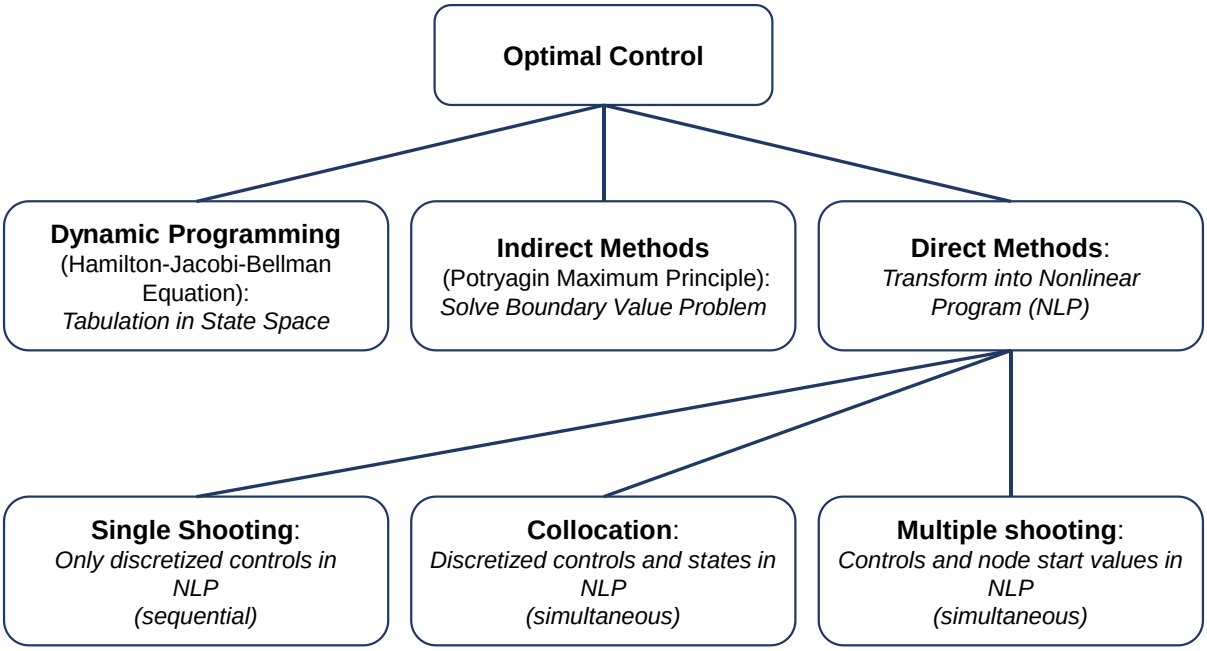


Figure 3-2. Overview of numerical methods for optimal control [18].

The optimal control problem could be handled with three types of algorithms, namely *Dynamic Programming* (Hamilton-Jacobi-Bellman Equations), *Indirect Methods* and *Direct Methods* (Figure 3-2). “Dynamic programming uses the principle of optimality of subarcs to compute recursively a feedback control for all times t and all x_0 ” [18]. However, it requires a discretization of the full state space, so it is preferred for unconstrained low-dimensional systems. Indirect methods refer to boundary value problems which are first optimized and then discretized. This class includes the calculus of variations, Euler-Lagrange differential equations and the Pontryagin’s Maximum Principle [18]. The main disadvantages are the difficulty in solving the differential equations due to the strong nonlinearity and instability and because changes in the control structure may require a completely new problem setup. Direct methods transcribe the original infinite horizon optimal control problem, where the decision variables are vector functions, to a finite dimensional nonlinear program (NLP). The NLP is a constrained parameter optimization where either the objective function or the constraints are nonlinear. Direct methods find the optimal solution of the discretized problem, so they are less accurate, but they are easier to pose and solve. The direct methods could be further classified as sequential or simultaneous methods. Their difference is that sequential methods parameterize only controls, while the simultaneous methods parameterize both states and controls. These parameterized variables are referred as optimization or decision variables. The trajectory optimization problem formulated in Section 4.4 is handled by one of the direct methods, so this category will be further explored next.

3.3 Direct methods

Transcription converts each continuous aspect of the problem into a discrete approximation. The methods that are used for the approximation of the objective function, the dynamics, which is expressed in the form $\dot{\mathbf{x}} = \mathrm{f}(\mathbf{x}, \mathbf{u})$ and the constraints are investigated in the following publications [35][37][18]. Also, the output of the NLP is a vector that contains the points that compose the discretized optimal trajectory. Using all these points, a continuous polynomial spline of a predetermined degree can be constructed.

3.3.1 Single shooting

The simplest sequential method is single-shooting, where the state trajectory $\mathbf{X} \in \mathbb{R}^{n_s \times N+1}$ is regarded a function of the controls $\mathbf{U} \in \mathbb{R}^{n_u \times N}$ and of the initial value $\mathbf{x}_0 \in \mathbb{R}^{1 \times n_s}$ (3-1), where n_s is the number of states and n_u is the number of the control inputs. Assuming that the controls and the states are discretized in N segments, then the number of points is $N+1$. The optimization variables are the control inputs, while the states result by a forward integration method. The most widely used integration method that finds approximate values at the discretized trajectory, is the Runge–Kutta 4th order (RK4) method. Each state, except the initial one, is described as the current state plus the weighted average of the slopes k_1, k_2, k_3, k_4 multiplied by the time interval of each segment h (3-2).

$$\mathbf{X} = \begin{bmatrix} \mathbf{x_0} \\ \mathrm{F}(\mathbf{x_0}, \mathbf{u_0}) \\ \vdots \\ \mathrm{F}(\mathbf{x}_{N-1}, \mathbf{u}_{N-1}) = \mathrm{F}(\mathrm{F}(\mathbf{x}_{N-2}, \mathbf{u}_{N-2}), \mathbf{u}_{N-1}) \end{bmatrix}^T \in \mathbb{R}^{n_s \times N+1} \tag{3-1}$$

$$\mathbf{x}_{n+1}^{RK} = \mathrm{F}(\mathbf{x}_n^{RK}, \mathbf{u}_n) = \mathbf{x}_n^{RK} + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4), \qquad n = 0, \ldots, N \tag{3-2}$$

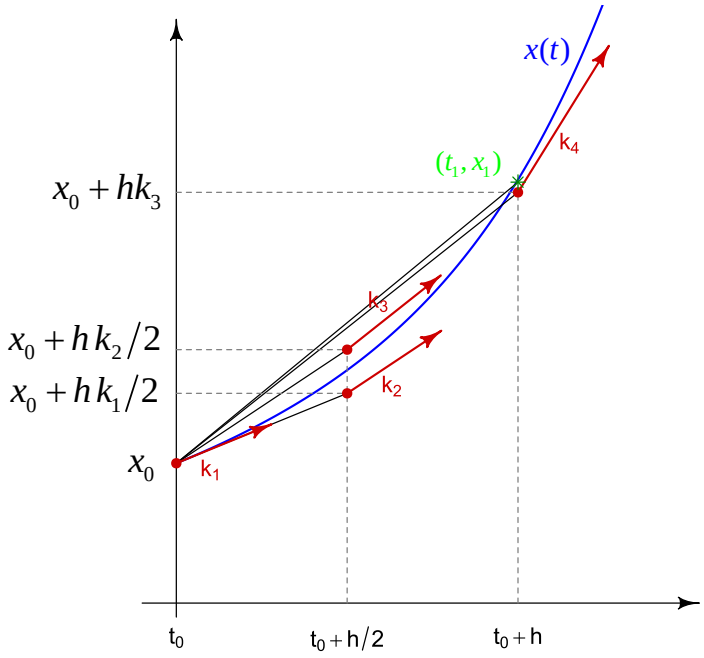


Figure 3-3. Slopes used by the Runge-Kutta 4th order method [29].

The formulas for the four slopes (derivatives) shown in Figure 3-3 are given by:

$$
\begin{aligned}
k_1 &= \mathrm{f}\left(\mathbf{x}_n^{RK}, \mathbf{u}_n\right) \\
k_2 &= \mathrm{f}\left(\mathbf{x}_n^{RK} + \frac{h}{2}k_1, \mathbf{u}_n\right) \\
k_3 &= \mathrm{f}\left(\mathbf{x}_n^{RK}, + \frac{h}{2}k_2, \mathbf{u}_n\right) \\
k_4 &= \mathrm{f}\left(\mathbf{x}_n^{RK} + hk_3, \mathbf{u}_n\right)
\end{aligned}
\tag{3-3}
$$

The entire trajectory is represented as a single segment, with a single constraint at the last point, known as a defect constraint, requiring that the final state of the simulation (last row of (3-1)) matches the desired final state of the system. In this method, the decision variables are the control inputs, thus the problem remains relatively small. The control input can be chosen as a piecewise continuous arbitrary function such as a zero-order-hold, a linear, a cubic, or an orthogonal polynomial. The input is applied to the dynamic system and the states $x(t)$ are forward simulated using RK4. The state trajectory cannot be initialized directly. Unstable systems can be difficult to be handled, since small modifications of the input can have large effects on the state $\mathbf{x}$ later in the trajectory. A block diagram that illustrates the steps of the single shooting method solved by an NLP solver, IPOPT, is given in Figure 3-4. IPOPT is investigated at the NLP solvers section.

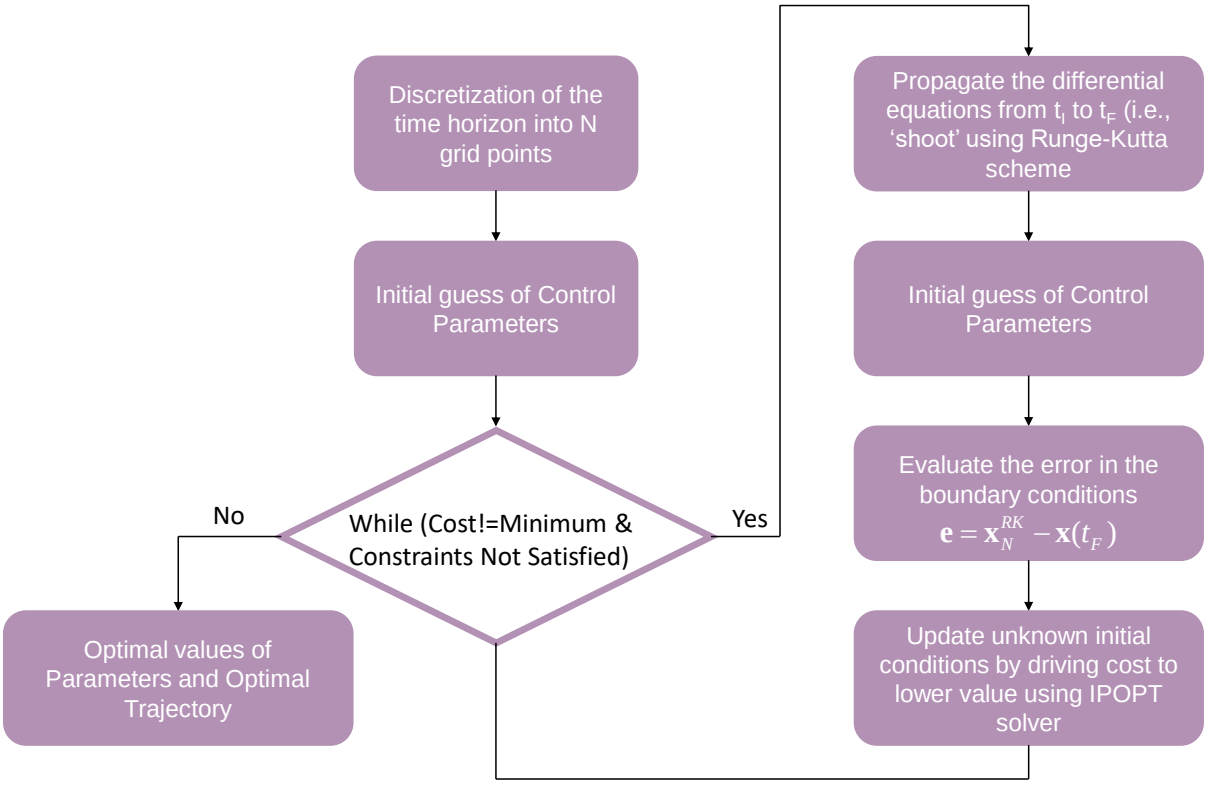


Figure 3-4. Single Shooting block diagram.

3.3.2 Multiple shooting

Two of the remarkable simultaneous methods, which will be explored, are multiple shooting and direct collocation. Multiple shooting works by breaking up a trajectory into several segments and using single shooting to solve for each segment. Figure 3-5 illustrates single shooting compared to the multiple shooting. In the former method, the defect constraint is enforced at the final point of the trajectory, while in the latter, these constraints are enforced at the final point of each segment. The integration of the ODE system is implemented through the Runge–Kutta 4th order (RK4) scheme. Both the controls $\mathbf{U} \in \mathbb{R}^{n_u \times N}$ and the states $\mathbf{X} \in \mathbb{R}^{n_s \times N+1}$ are decision variables. A defect constraint is added to each of the knot points, which are the points between two adjacent segments (Figure 3-5 (Right)), so that the trajectory is continuous. The values of the states derived from the RK4 integration $\mathbf{x}_n^{RK}$ (3-4) should match the state decision variables $\mathbf{x}_n$ at each grid point n. That is ensured through the equality constraints $\mathbf{g} = \mathbf{0}$ (3-5).

$$\mathbf{x}_{n+1}^{RK} = \mathbf{x}_n + \frac{h}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right), \qquad n = 0, \ldots, N \tag{3-4}$$

$$\mathbf{g} = \mathbf{x}_n - \mathbf{x}_n^{RK}, \qquad n = 1, \ldots, N+1 \tag{3-5}$$

This formulation increases the convergence and the stability since the inputs in one shot do not directly influence the trajectory of the next shot. The nonlinear effects of the continuity conditions are distributed over the whole horizon, while in single shooting the nonlinearity of the system is accumulated at the end condition. In conclusion, multiple shooting results in a higher dimensional problem, but it is sparse and less nonlinear than the NLP produced by single shooting. A block diagram that illustrates the steps of the multiple shooting method is given in Figure 3-6.

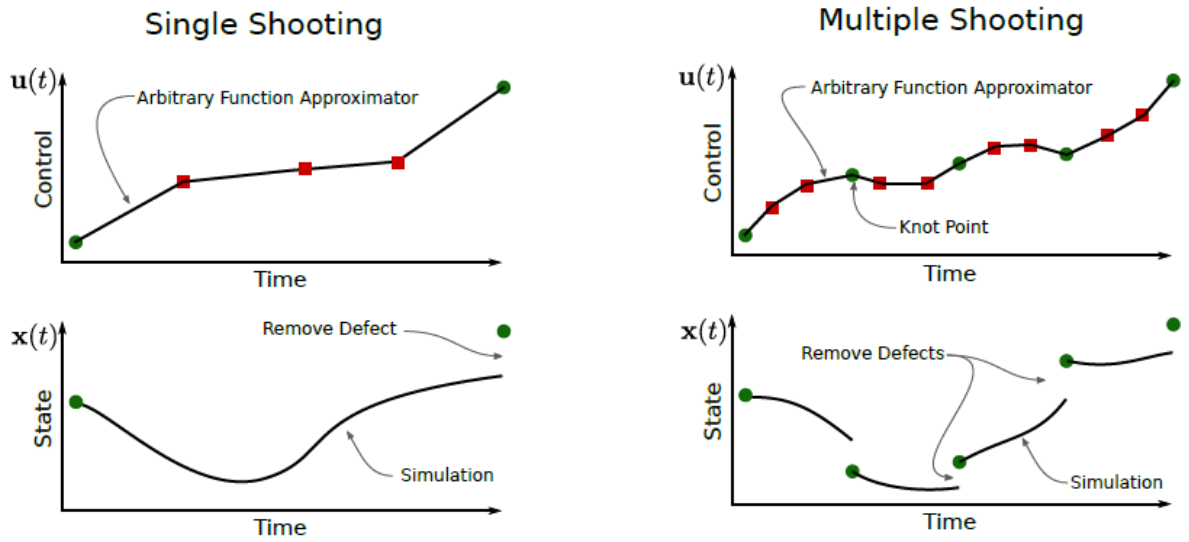


Figure 3-5. Left: Single shooting - Right: Multiple shooting [37].

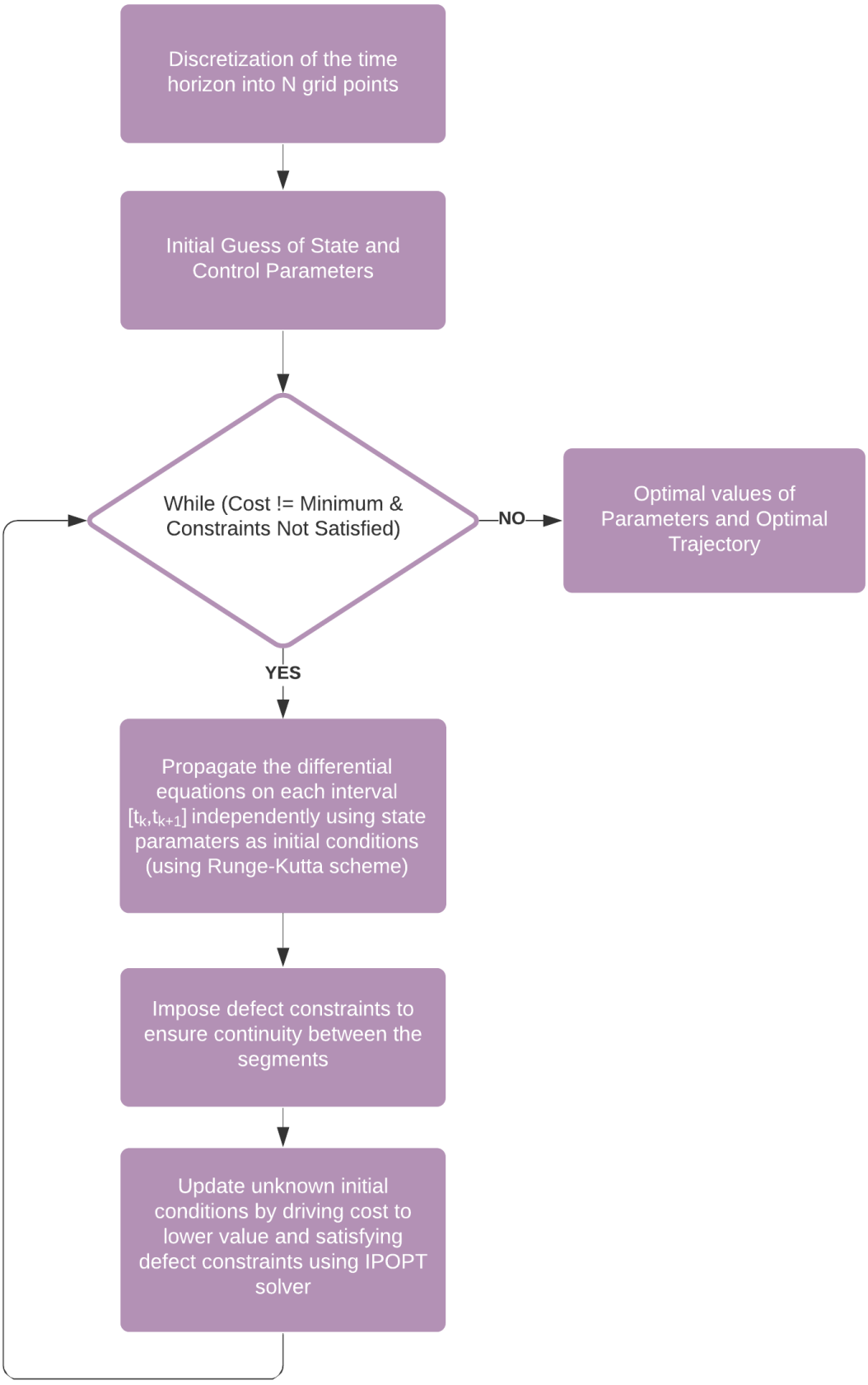


Figure 3-6. Multiple shooting block diagram.

3.3.3 Direct collocation

The basic idea behind these methods is the approximation of the state and control trajectories by polynomial splines. Starting with several predetermined points, called the *collocation points*, the interpolating polynomial is differentiated with respect to time. The differentiated polynomial equals the system dynamics $\dot{\mathbf{x}}(t) = \mathrm{f}\left(t, \mathbf{x}(t), \mathbf{u}(t)\right)$ at the collocation points. In direct collocation, the system dynamics are not enforced by forward integration, but through equality constraints. The solver varies the state x(t) and input u(t) trajectory simultaneously while enforcing the system dynamics relationship between them at specified times. These methods simulate the dynamics of the system implicitly (typically using implicit Runge-Kutta schemes) because the values of the state at each collocation point are obtained at the same time (as opposed to solving for the state sequentially). The decision variables include both the discretized state and control trajectories. An initial guess for these trajectories is required to be set up. It is worth mentioning that due to the decoupling between state and input, changes in the input only affect the state at that time instance, while future states remain unaffected.

Trapezoidal Direct collocation

Trapezoidal collocation is a commonly used low-order direct collocation method. The dynamics, path objective, and control are all represented using linear splines, and the dynamics are satisfied using trapezoidal quadrature. The continuous integral of the objective function is approximated as a summation that depends on the value of the integrand $w\left(t_k\right) = w_k$ at the collocation points t_k along the trajectory. Applying the trapezoid rule for integration between each collocation point results in (3-6), where $h_k = t_{\mathrm{k+1}} - t_{\mathrm{k}}$.

$$\int_{t_0}^{t_F} \mathrm{w}\left(t, x(t), u(t)\right) dt = \sum_{k=0}^{N-1} \frac{h_k}{2}\left(w_k + w_{k+1}\right) \tag{3-6}$$

The system dynamics $\mathrm{f}\left(t, x(t), u(t)\right)$ is represented by the collocation constraints. For trapezoidal collocation, these are constructed by writing the dynamics in integral form and then approximating that integral using trapezoidal quadrature. Assuming that x_k is the decision variable in the NLP and $f_k = \mathrm{f}\left(t_k, x_k, u_k\right)$ is the result of computing the system dynamics at each collocation point, then the following equations arise:

$$\dot{\mathbf{x}} = \mathbf{f} \tag{3-7}$$

$$\int_{t_k}^{t_{k+1}} \dot{\mathbf{x}}\, dt = \int_{t_k}^{t_{k+1}} \mathbf{f}\, dt \tag{3-8}$$

$$\mathbf{x}_{k+1} - \mathbf{x}_k \approx \frac{1}{2} h_k \left(\mathbf{f}_{k+1} - \mathbf{f}_k\right) \qquad k \in 0, \ldots, N-1 \tag{3-9}$$

In addition to the collocation constraints (3-9), which enforce the system dynamics, the problem might also have limits on the state and control, path constraints, and boundary constraints. The first one is expressed by setting a lower and upper limit on the state and control values at specific collocation points. The path constraints, which must be satisfied at specified collocation points along the trajectory, ensure that the output trajectories are feasible. Finally, the boundary constraints are expressions that are enforced at the initial and the final point of the trajectory. It is noteworthy that trajectory optimization problems with path constraints tend to be much harder to solve than those without.

After defining the objective function, the dynamics constraints, the boundary constraints and the path constraints, the problem can be solved using an off-the-self solver. The vectors that contain the optimal values of the states and controls at the collocation points, must be exploited to construct continuous trajectories. In trapezoidal collocation, the control trajectory and the system dynamics are approximated by piecewise linear functions. These, which are also called linear splines are constructed by two sequential collocation points. For trapezoidal collocation, all the collocation points are considered knot points, as they join any two polynomial segments.

The expression for $\mathbf{u}$ on the interval $t \in \left[t_k, t_{k+1}\right]$ requires the value of the control at the initial and the final knot point. Setting the parameter $\tau = t - t_k$, the control trajectory is a linear spline given by (3-10).

$$\mathbf{u}(t) = \mathbf{u}_k + \frac{\tau}{h_k}\left(\mathbf{u}_{k+1} - \mathbf{u}_k\right) \tag{3-10}$$

Similarly, the system dynamics on the interval $t \in \left[t_k, t_{k+1}\right]$ is approximated as shown in (3-11) :

$$\mathbf{f}(t) = \dot{\mathbf{x}}(t) \approx \mathbf{f}_k + \frac{\tau}{h_k}\left(\mathbf{f}_{k+1} - \mathbf{f}_k\right) \tag{3-11}$$

Integrating (3-11) results in the expression for the state, which is a quadratic polynomial (3-12) .

$$\mathbf{x}(t) = \int \dot{\mathbf{x}}(t)\, d\tau \approx \mathbf{x}_k + \mathbf{f}_k \tau + \frac{\tau^2}{2h_k}\left(\mathbf{f}_{k+1} - \mathbf{f}_k\right) \tag{3-12}$$

Hermite Simpson

Hermite-Simpson Collocation is a common medium-order direct collocation method, in which the state is represented by a cubic-Hermite spline. The integrand of the objective function and system dynamics are approximated as piecewise quadratic functions (3-13) using Simpson Quadrature, also known as Simpson's rule for integration. More details on Simpson's rule are given in Appendix C.

$$\int_{t_0}^{t_F} w(t)\, d\tau \approx \sum_{k=0}^{N-1} \frac{h_k}{6}\left(w_k + 4w_{k+\frac{1}{2}} + w_{k+1} \right) \tag{3-13}$$

Similarly, the continuous dynamics is transcribed to a set of collocation equations by approximating the continuous integral in (3-8) with Simpson quadrature (3-14).

$$\mathbf{x}_{k+1} - \mathbf{x}_k = \frac{1}{6} h_k \left(\mathbf{f}_k + 4\mathbf{f}_{k+\frac{1}{2}} + \mathbf{f}_{k+1} \right) \tag{3-14}$$

The dynamics at the midpoint of the segment $\mathbf{f}_{k+1/2}$ are a function of the state $\mathbf{x}_{k+1/2}$, which is calculated by constructing an interpolant for the state trajectory and then evaluating it at the midpoint of the interval (3-15):

$$\mathbf{x}_{k+\frac{1}{2}} = \frac{1}{2}\left(\mathbf{x}_k + \mathbf{x}_{k+1}\right) + \frac{h_k}{8}\left(\mathbf{f}_k - \mathbf{f}_{k+1}\right) \tag{3-15}$$

When the midpoint $\mathbf{x}_{k+1/2}$ is considered as an additional decision variable, then both (3-14) and (3-15) are constraint equations the formulation is reported as separated form. On the

other hand, if (3-15) is substituted in (3-14), a single collocation constraint arise and the formulation is reported as compressed form. Each form has pros and cons, but as a rule of thumb, the separated form is preferred when the number of segments is small.

The continuous trajectories that interpolate the solution of the controls between the collocation points are consisted of quadratic polynomial merged at the knot points. Each quadratic segment is fitted through three uniformly spaced points to approximate the integrand. A 2nd order polynomial that passes through the points (t_A,u_A), (t_B,u_B), (t_C,u_C) can be written in the form of (3-16).

$$\mathbf{u}(t)=\frac{(t-t_B)(t-t_C)}{(t_A-t_B)(t_A-t_C)}u_A+\frac{(t-t_A)(t-t_C)}{(t_B-t_A)(t_B-t_C)}u_B+\frac{(t-t_A)(t-t_B)}{(t_C-t_A)(t_C-t_B)}u_C \tag{3-16}$$

However, in this formulation the segments have equal duration h_k and point B is in the middle of the segment. If the indices A, B, C are substituted by the indices k, $k+1/2$, $k+1$, then $h_k=t_{k+1}-t_k$, $t_{k+1/2}=(t_k+t_{k+1})/2$. Defining the time as $\tau=t-t_k$, the simplified equation becomes:

$$\mathbf{u}(t)=\left(2\mathbf{u}_k-4\mathbf{u}_{k+1/2}+2\mathbf{u}_{k+1}\right)\left(\frac{\tau}{h}\right)^2+\left(-3\mathbf{u}_k+4\mathbf{u}_{k+1/2}-\mathbf{u}_{k+1}\right)\left(\frac{\tau}{h}\right)+\mathbf{u}_k \tag{3-17}$$

The system dynamics $\mathbf{f}(\cdot)=\dot{\mathbf{x}}$ is also represented by quadratic polynomials (3-18).

$$\mathbf{f}(t)=\dot{x}=\left(2\mathbf{f}_k-4\mathbf{f}_{k+1/2}+2\mathbf{f}_{k+1}\right)\left(\frac{\tau}{h}\right)^2+\left(-3\mathbf{f}_k+4\mathbf{f}_{k+1/2}-\mathbf{f}_{k+1}\right)\left(\frac{\tau}{h}\right)+\mathbf{f}_k \tag{3-18}$$

The states trajectories are derived by integrating (3-18) as

$$\mathbf{x}=\int\dot{\mathbf{x}}dt=\int\left(\left(2\mathbf{f}_k-4\mathbf{f}_{k+1/2}+2\mathbf{f}_{k+1}\right)\left(\frac{t}{h}\right)^2+\left(-3\mathbf{f}_k+4\mathbf{f}_{k+1/2}-\mathbf{f}_{k+1}\right)\left(\frac{t}{h}\right)+\mathbf{f}_k\right)dt \tag{3-19}$$

$$\begin{aligned}&\mathbf{x}=\frac{1}{3}h\left(2\mathbf{f}_k-4\mathbf{f}_{k+1/2}+2\mathbf{f}_{k+1}\right)\left(\frac{\tau}{h}\right)^3+\frac{1}{2}h\left(-3\mathbf{f}_k+4\mathbf{f}_{k+1/2}-\mathbf{f}_{k+1}\right)\left(\frac{\tau}{h}\right)^2+\mathbf{f}_k\tau+\mathbf{x}_k\Rightarrow\\&\mathbf{x}=\frac{1}{3h^2}\left(2\mathbf{f}_k-4\mathbf{f}_{k+1/2}+2\mathbf{f}_{k+1}\right)\tau^3+\frac{1}{2h}\left(-3\mathbf{f}_k+4\mathbf{f}_{k+1/2}-\mathbf{f}_{k+1}\right)\tau^2+\mathbf{f}_k\tau+\mathbf{x}_k\end{aligned} \tag{3-20}$$

It is noteworthy, that the state trajectory, which is represented by cubic Hermite splines, satisfies the continuity of the first derivative.

Orthogonal collocation

A subset of collocation methods that have seen extensive use in optimal control are orthogonal collocation methods [37]. Orthogonal collocation method differs from the aforementioned collocation methods in the determination of the collocation points positions along the trajectory. In view of this, in an orthogonal collocation method the collocation points are the roots of some orthogonal polynomial. Usually, these collocation points are obtained from the roots of either Chebyshev polynomials or Legendre polynomials. In such formulations, orthogonal polynomials are used to approximate both states and controls.

3.3.4 Comparison of shooting methods with collocation methods

The difference between multiple shooting and direct collocation lies on how each method enforces the constraint of the system's dynamics [37]. The dynamics constraints can be stated

in two different forms: derivative and integral. In the first one, the derivative of the state with respect to time must be equal to the dynamics function (3-21). On the other hand, the integral form requires that the state trajectory matches the integral of the dynamics with respect to time (3-22). Collocation methods comply with the derivative form, while shooting methods comply with the integral form. A graphic representation of these two methods is illustrated in Figure 3-7.

$$\dot{\mathbf{x}} = \mathrm{f}(\mathbf{x}, \mathbf{u}) \tag{3-21}$$

$$\mathbf{x} = \int \mathrm{f}(\mathbf{x}, \mathbf{u})\, dt \tag{3-22}$$

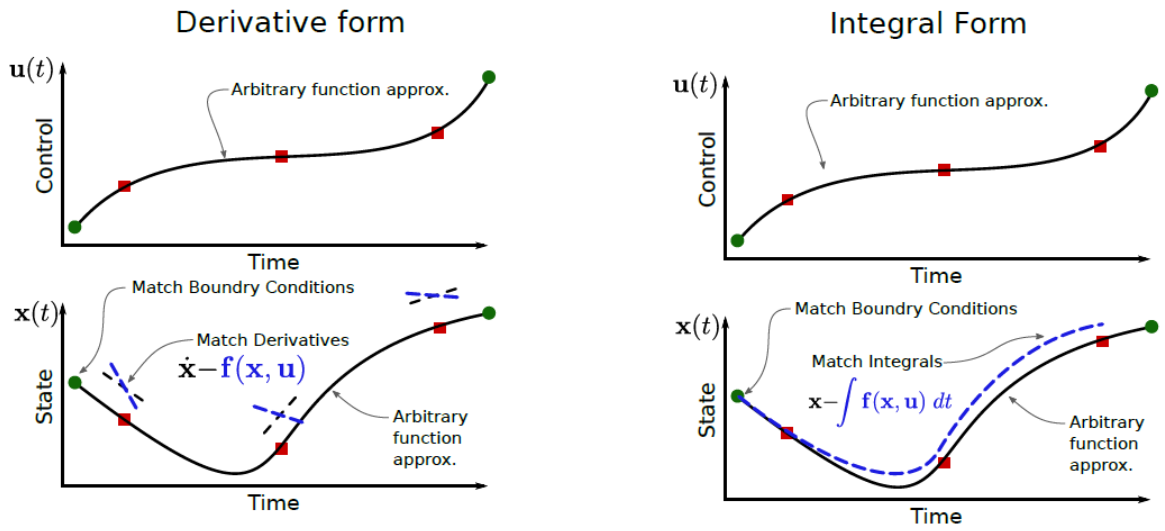


Figure 3-7. Integral vs. derivative form of the optimal control problem [37].

In addition, shooting methods could make use of adaptive, error-controlled ODE solvers. These can change their integration step-size depending on the current dynamics, thereby avoiding unnecessary calculation, and reducing computation time. Compared to collocation, the dimension of the NLP in multiple shooting has smaller dimension but it is less sparse. This loss of sparsity combined with the cost of the underlying ODE solution leads to theoretically higher costs per iteration.

3.4 Nonlinear Model Predictive Control

Nonlinear Model Predictive Control (NMPC) is an optimization-based feedback control technique [18]. The general idea is the generation of a closed-loop system whose feedback control is derived from the solution of an open-loop optimal control problem. The main advantage of NMPC is the fact that it allows the first value of optimal control vector to be executed, while keeping the rest as an initial guess for the next iteration. NMPC optimizes a finite time-horizon in a high frequency, and in each iteration, the most recent state observation is used as initial value $\mathbf{x}_0$. Consequently, the system can deal with unexpected disturbances and errors due to the inaccuracies of the dynamic model.

Another advantage of NMPC, is the capability of modelling nonlinear systems in the form $\dot{\mathbf{x}} = \mathrm{f}(\mathbf{x}, \mathbf{u})$. Additionally, the optimal control problem can be formulated with any desired objective function and the boundary and path constraints are treated as equality and inequality

constraints which can be handled easily. It appears that NMPC has potential in robotics applications. However, a deterrent reason to its use is the high on-line computational load that is often associated with NMPC, since at each sampling instant a nonlinear optimal control problem must be solved. The algorithm must predict and optimize repeatedly, while the system performs commands in real time. Therefore, most of the applications so far concern the process industries in chemical plants, where time scales are typically in the range of minutes [18]. Another reason is the need to solve a non-convex optimization problem on-line, no global minimum or feasible solution can be guaranteed.

3.5 Nonlinear programming solvers

Numerical methods for solving NLPs fall into two categories: gradient-based methods and heuristic methods [51]. In this work, emphasis is given to the first category which includes Sequential Quadratic Programming (SQP) and Interior Point methods. Examples of well-known software that use SQP methods include the dense NLP solver NPSOL [27] and the sparse NLP solvers SNOPT and SPRNLP [7]. Well-known sparse interior point NLP solvers include BARNLP [53] [5] [53], LOQO [5], KNITRO [14] [13] and IPOPT. Two of the most commonly used nonlinear programming solvers are SNOPT and IPOPT, which are based on mature methods in mathematical optimization. SNOPT is based on Sequential Quadratic Programming, while IPOPT uses the Interior Point Method. In [47], the performance of these two approaches is evaluated within a robotics application. In this survey, the trajectory optimization examples were set up according to the formulation of direct collocation method. The direct method can be configured in several ways depending on the accuracy and solving time requirements. In [47], two different integration methods (Trapezoidal, Hermite-Simpson) and three initialization methods (Zero, Linear and Incremental) were examined to enhance the validity of the results in different problem complexity. The performance criteria were set to be the accuracy, the solving time, and the quality of the solution. The accuracy of the solution is measured using the mean squared error between the planned and the actual control trajectories. In contrast to the number of iterations, running time is an absolute metric that provides information on how fast the motion planner compute trajectories online. The quality of the solutions is compared in terms of optimality using the resulting value of the objective function. The results of the study show that SNOPT requires more time to solve the same problem compared to IPOPT. The solving time deviation becomes noticeable in complex tasks. In terms of quality and accuracy, both solvers act similarly under different schemes. In summary, this comparison revealed that IPOPT is faster, so the NLP problems presented in Section 3.6 are solved by Interior Point algorithms. The first optimization library that is evaluated uses FMINCON's Interior Point algorithm, while the second one uses the IPOPT solver.

3.6 Evaluation of trajectory optimization methods

The direct methods can be implemented using several optimization libraries. However, the goal of this study is to choose a library which already has been used for the formulation of optimal control problems in legged systems. The comparison of methods and libraries along with the aforementioned trajectory optimization theory determines the direct method and library choices in Section 4.3. The performance metrics are evaluated by executing a simple optimization example i.e., the swing up stabilization of a two-link manipulator, whose dynamics

is similar to that of a quadruped's leg. It comprises two rotary actuators, one at the base and one at the elbow. The dynamics of this system is simple and can be derived using an Euler-Lagrange approach to result in (3-23)-(3-24).

$$u_1 = b_{11}\ddot{q}_1 + b_{12}\ddot{q}_2 + \mathrm{N}_1\left(q_1, q_2, \dot{q}_1, \dot{q}_2\right) \tag{3-23}$$

$$u_2 = b_{21}\ddot{q}_1 + b_{22}\ddot{q}_2 + \mathrm{N}_2\left(q_1, q_2, \dot{q}_1, \dot{q}_2\right) \tag{3-24}$$

where the terms $b_{11}, b_{12}, b_{21}, b_{22}, N_1, N_2$ are defined in (3-25)-(3-29):

$$b_{11} = I_1 + I_2 + m_1 l_{1c}^2 + m_2\left(l_1^2 + l_{2c}^2\right) + 2m_2 l_1 l_{2c}\cos\left(q_2\right) \tag{3-25}$$

$$b_{12} = b_{21} = I_2 + m_2 l_{2c}^2 + m_2 l_1 l_{2c}\cos\left(q_2\right) \tag{3-26}$$

$$b_{22} = \left(I_2 + m_2 l_{2c}^2\right)\ddot{q}_2 \tag{3-27}$$

$$\begin{aligned}\mathrm{N}_1 = {} & -2m_2 l_1 l_{2c}\sin\left(q_2\right)\dot{q}_1\dot{q}_2 - m_2 l_1 l_{2c}\sin\left(q_2\right)\dot{q}_2^2 \\ & + g\left(\left(m_1 l_{1c} + m_2 l_1\right)\sin\left(q_1\right) + m_2 l_{2c}\sin\left(q_1 + q_2\right)\right)\end{aligned} \tag{3-28}$$

$$\mathrm{N}_2 = m_2 l_1 l_{2c}\sin\left(q_2\right)\dot{q}_1^2 + g\left(m_2 l_{2c}\sin\left(q_1 + q_2\right)\right) \tag{3-29}$$

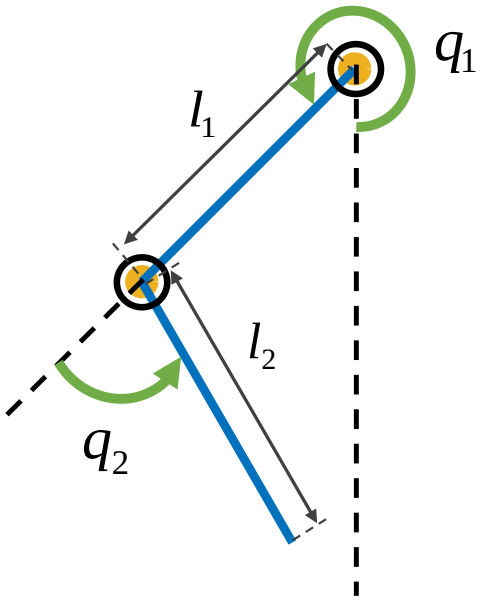


Figure 3-8. Two-link manipulator The two links are depicted in blue, the rotary joints and the joint angles are indicated by the yellow circles and green arrows respectively.

The subscript 1 indicates the link from the base to the elbow and the subscript 2 the link from the elbow to the end effector. Regarding the parameters, I is the inertia about link's center of mass, m is the mass, l is the length and l_{ic} is the distance between the center of mass of the i-th link and the i-th joint. The angles are given by q_1 and q_2 respectively and the control torques by u_1, u_2. All standard trajectory optimization methods require that the dynamics of the system are in first-order form. This is accomplished by including both the generalized coordinates (q_1 and q_2) and their derivatives in the state vector.

$$\mathbf{x} = \begin{bmatrix} q_1 & q_2 & \dot{q}_1 & \dot{q}_2 \end{bmatrix}^T \tag{3-30}$$

The dynamics of the system must be expressed in the form (3-31). The system of equations (3-23)-(3-24) can be written in compact matrix form. Introducing matrices in (3-32), the system of differential equations can be represented as shown in (3-33). Due to the inversibility of matrix $\mathbf{M}$ (3-34), the dynamics can be written in the desired form explicitly (3-35). The example was performed in two specialized libraries, i.e., OptimTraj and CasADi[2]. Table 3-1 summarizes the essential information in the formulation of each problem and how these four problems differ from each other.

$$\dot{\mathbf{x}} = \mathrm{f}\left(q_1, q_2, \dot{q}_1, \dot{q}_2\right) \tag{3-31}$$

$$\mathbf{M} = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \qquad \ddot{\mathbf{q}} = \begin{bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \end{bmatrix} \qquad \mathbf{N} = \begin{bmatrix} N_1 \\ N_2 \end{bmatrix} \qquad \mathbf{u} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \tag{3-32}$$

$$\mathbf{M}\ddot{\mathbf{q}} = \mathbf{N} - \mathbf{u} \tag{3-33}$$

$$\mathbf{M}^{-1} = \frac{1}{b_{11}b_{22} - b_{12}b_{21}} \begin{bmatrix} b_{22} & -b_{12} \\ -b_{21} & b_{11} \end{bmatrix} \tag{3-34}$$

$$\ddot{\mathbf{q}} = \mathbf{M}^{-1}\left(\mathbf{N} - \mathbf{u}\right) \tag{3-35}$$

Table 3-1. Differences in the formulation of the implemented trajectory optimization examples for the two-link manipulator.

Problem Setup	1	2	3	4
Library	OptimTraj	OptimTraj	CasADi	CasADi
Method	Trapezoidal Herm.Sim. Mult.Shoot.	Trapezoidal Herm.Sim. Mult.Shoot.	Single shooting	Multiple shooting
NMPC	No	No	Yes	Yes/No
Opt. Variables	$\mathbf{u}, \mathbf{x}$	$\mathbf{u}, \mathbf{x}$	$\mathbf{u}$	$\mathbf{u}, \mathbf{x}$
Objective Function	(3-37)	(3-49)	(3-49)	(3-49)
$\mathbf{U}_0$	$\mathbf{0}_{N+1\times 2}$	$\mathbf{0}_{N+1\times 2}$	$\mathbf{0}_{N+1\times 2}$	$\mathbf{0}_{N+1\times 2}$
$\mathbf{X}_0$	Linear interp.	Linear interp.	-	Linear interp.

[2] The MATLAB codes can be found here: https://bitbucket.org/csl_legged/two_link_manipulator_trajectory_optimization/src/master/

OptimTraj is a general-purpose trajectory optimization library that is written in MATLAB [61]. It handles continuous-time single-phase trajectory optimization problems. It is ideal for comparison of the optimization methods, as it includes the most commonly used algorithms which are:

- Trapezoidal Direct Collocation
- Hermite-Simpson Direct Collocation
- Runge-Kutta 4th-order Multiple Shooting
- Chebyshev-Lobatto Orthogonal Collocation

CasADi is an open-source, general purpose software tool for nonlinear optimization and can be used for dynamic optimization in a flexible, interactive and numerically efficient way [4]. It has been written to be able to deal with the optimization of large-scale optimal control problems (OCPs). It has been designed to provide the necessary building blocks for solving OCPs. Key features of CasADi are algorithmic differentiation, interfaces to NLP and ODE solvers, and functions tailored for implementing direct methods. It is implemented in self-contained C++ code and contains full interface capabilities to MATLAB. Thanks to convenient interfaces and efficient virtual machines, the implementation in a high-level language is as fast as an equivalent C implementation. The time spent in CasADi can furthermore be cut by a factor of five by generating C-code for the NLP functions [4].

3.6.1 Problem Setup 1 – OptimTraj

In this problem setup trapezoidal direct collocation, Hermite Simpson direct collocation and multiple shooting methods are compared in the OptimTraj library. The decision variables that are required in this formulation are the matrices that contain the states given in (3-30) and the controls over the optimization horizon (3-36). If the trajectory is discretized in N segments, then the number of grid points will be $N+1$.

$$\mathbf{x}=\begin{bmatrix}\mathbf{q}_1\\ \mathbf{q}_2\\ \dot{\mathbf{q}}_1\\ \dot{\mathbf{q}}_2\end{bmatrix}\in\mathbb{R}^{4\times N+1}\qquad \mathbf{u}=\begin{bmatrix}\mathbf{u}_1\\ \mathbf{u}_2\end{bmatrix}\in\mathbb{R}^{2\times N+1} \tag{3-36}$$

Objective Function

The objective function was chosen as the integral of the actuator-effort (control) squared (3-37). Choosing the trapezoidal collocation method, the continuous objective is transformed in discretized form as shown in (3-38), where T is the total simulation time.

$$J=\int_0^T \mathbf{u}^2(\tau)\,d\tau \tag{3-37}$$

$$\min J\approx\min\sum_{k=0}^{N-1}\frac{h_k}{2}\left(u_k^2+u_{k+1}^2\right) \tag{3-38}$$

This objective function tends to produce smooth trajectories, which are desirable for two reasons. The first is that the polynomial spline that is used by the most transcription methods will approximate very well the solution. The second is that a smooth solution can be found quickly and accurately.

Constraints

The dynamics constraints for the trapezoidal direct collocation, Hermite Simpson direct collocation and multiple shooting are described in (3-9), (3-14), (3-5) respectively, where $\mathbf{f}(\bullet)$ is the right-hand side of the dynamics defined in (3-31).

The boundary constraints restrict the full state of the two-link manipulator at both the initial and the final points on the trajectory. The manipulator starts from its equilibrium position where $q_1, q_2 = 0$ and the goal is to reach the upper vertical position (3-39).

$$\begin{aligned} q_1(t_0) &= 0 & q_1(t_F) &= \pi \\ q_2(t_0) &= 0 & q_2(t_F) &= 0 \\ \dot{q}_1(t_0) &= 0 & \dot{q}_1(t_F) &= 0 \\ \dot{q}_2(t_0) &= 0 & \dot{q}_2(t_F) &= 0 \end{aligned} \tag{3-39}$$

The angles q_1, q_2 and the control torques are bounded (3-40)-(3-42):

$$-\frac{3\pi}{2} \le q_1 \le \frac{3\pi}{2} \tag{3-40}$$

$$-\pi \le q_2 \le \pi \tag{3-41}$$

$$u_{\min} \le u_1, u_2 \le u_{\max} \tag{3-42}$$

Initialization

The trajectory optimization problem needs an initial guess for each decision variable. All segments are of uniform duration, so the vector that represents time is given by (3-43), where $N+1$ is the number of grid points. The state $\mathbf{q_1}$ is interpolated linearly between the initial and the final position (3-44), while the initial guess for the rest of the states and the controls is a zero matrix (3-45), (3-46).

$$t(k) = \frac{k-1}{N} t_F \qquad k = 1, \ldots N+1 \tag{3-43}$$

$$q_1(k) = \frac{k-1}{N} \pi \qquad k = 1, \ldots N+1 \tag{3-44}$$

$$\mathbf{q_2} = \dot{\mathbf{q}}_\mathbf{1} = \dot{\mathbf{q}}_\mathbf{2} = \mathbf{0}_{1 \times N+1} \tag{3-45}$$

$$\mathbf{u_1} = \mathbf{u_2} = \mathbf{0}_{1 \times N+1} \tag{3-46}$$

In this library, there is no need of determining an initial value at each grid point. Having defined the first and the final point, a linear interpolation can be automatically generated. The interpolation can also be piecewise linear. For instance, if four time instances between $[0, t_F]$ (including the boundaries of the interval) are specified (3-47), three piecewise linear trajectories connecting these four key frames will be generated (3-48). Then, the initial guess of the decision variables for each grid point results by substituting the time instances, represented by the grid points, to the piecewise linear function. In other words, the trajectory that was defined by four points is re-discretized into the number of grid points to obtain an initial guess for each grid point. The number of grid points can be specified by the user or selected by the library automatically.

$$\mathbf{t} = \begin{bmatrix} 0 & 0.25 & 0.75 & 1 \end{bmatrix} (t_F - t_0) \tag{3-47}$$

$$\mathbf{q_1} = \begin{bmatrix} 0 & \frac{\pi}{4} & \frac{3\pi}{4} & \pi \end{bmatrix} \quad \dot{\mathbf{q}}_1 = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix} \quad \mathbf{u_1} = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}$$
$$\mathbf{q_2} = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix} \quad \dot{\mathbf{q}}_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix} \quad \mathbf{u_2} = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix} \tag{3-48}$$

Results

This example was solved in MATLAB, using FMINCON's interior-point algorithm as the nonlinear programming solver. Setting up the optimization problem involves defining the objective function (3-38), the dynamics constraints (3-9), (3-14), (3-5), the boundary constraints (3-39), the range of acceptable values for each optimization variable (3-40)-(3-42) and the initial guess (3-43)-(3-48). The library allows to manually define the number of grid points, but if this option is not set by the user, the number of the required grid points is automatically determined by the library. Table 3-2 lists the solution information and the number of grid points that were chosen automatically for each method. The lower bound of the final time was set to $t_{FL} = 1s$, while the upper bound was set to $t_{FU} = 6s$, so the final time of the problem varies in each method. The results for the states and control torques are depicted in Figure 3-9.

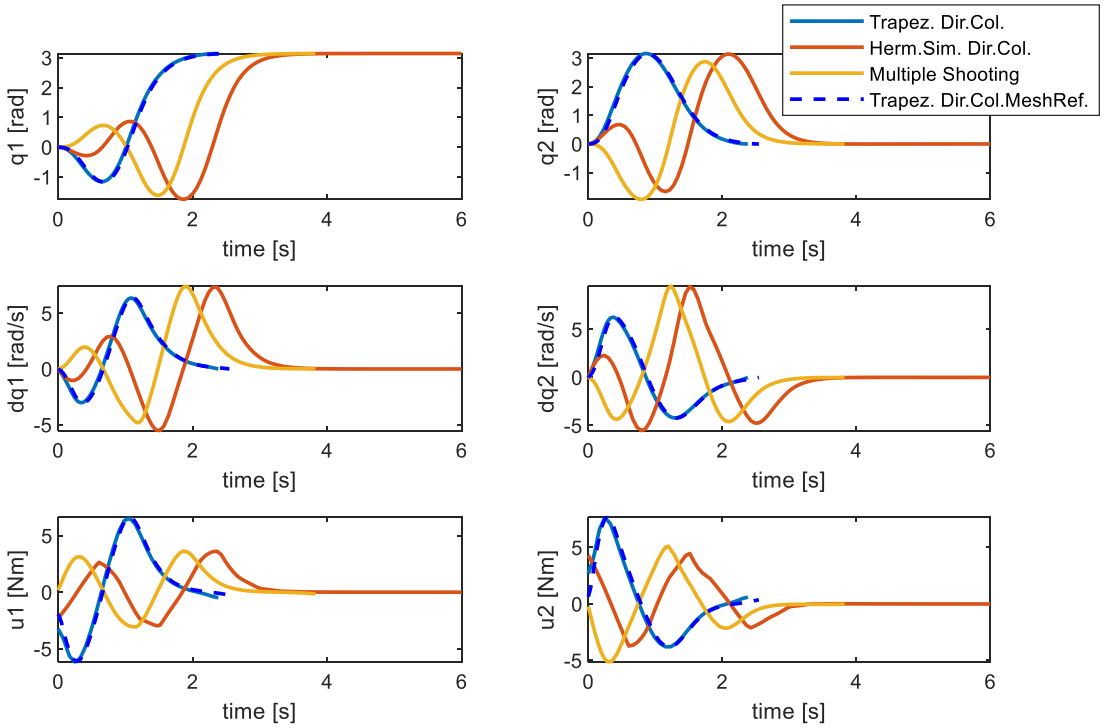


Figure 3-9. Trajectories of a two-link manipulator that reaches the upper position using three different direct methods. The dashed lines represent the case where the OCP was solved three sequential times using trapezoidal collocation. Each time the mesh was refined with more points.

Trapezoidal direct collocation method is the fastest but lacks in accuracy. One way to improve the accuracy is by refining the mesh. The problem was solved again using trapezoidal direct collocation on a sequence of 3 collocation grids, with 20, 40 and 60 grid points respectively. By solving the OCP multiple times, starting with a coarse grid, and using the previous solution as an initial guess for the next OCP, it is sometimes possible to achieve faster convergence than trying to solve a finely resolved OCP with a poor initial guess from

the start. This iterative strategy is implemented to obtain the most accurate solution with the least amount of computational effort. The solution on the initial (20 grid points) mesh needed 75 iterations, 2.42 s and the maximum value of error was 0.211. The solution on the (40 grid points) mesh needed 31 iterations, 1.9 s and the maximum value of error was 0.0305. The solution on the final (60 grid points) mesh needed 50 iterations, 6.94 s and the maximum value of error was 0.0095.

Hermite Simpson direct collocation and multiple shooting are both considered as high-accuracy methods but along with the accuracy, the solving time also increases. Between the two, Hermite Simpson direct collocation takes 0.6 times less time to solve the NLP than multiple shooting method. Figure 3-9 shows that the torque requirements in the Hermite-Simpson direct collocation and the multiple shooting methods are similar and almost half of torque requirements in the Trapezoidal direct collocation method. Table 3-2 reveals that the objective function value is minimum when the Hermite-Simpson direct collocation is used, which means that the target is reached with the least actuation effort.

Table 3-2. Solution information of the three different direct methods.

Info	Trapezoidal Direct Collocation	Hermite Simpson Direct Collocation	Multiple Shooting
NLP Time [s]	2.88	10.45	25.24
Iterations	79	189	204
Func. Count	14752	47985	52010
Grid points	30	40	40
Obj value	58.59	28.45	32.93

3.6.2 Problem Setup 2 – OptimTraj

In this case, an alternate objective function is defined and the system's response is examined when the final time is specified and when it is free. The boundary and path constraints are kept the same. The problem is solved using the OptimTraj library.

Objective Function

The objective function (3-49) is in quadratic form and the problem is considered as a quadratic programming (QP) problem. The first part of the equation penalizes the errors between the vector of each state and the reference state (goal) $\mathbf{x}_{\mathbf{F}_k}$ via the weighting matrix $\mathbf{Q} \geq \mathbf{0}$. This term minimizes the time needed to reach the goal. The second term which contains the weighting matrix $\mathbf{R} \geq \mathbf{0}$ penalizes large input torques, so that the desired task is accomplished in an energy efficient way. Compared to the previous objective function, this one is also affected by the deviation of optimal states from a reference value. The elements of the reference state vector represent the desired states at each grid point k, so if the desired states at all grid points are equal to the final states, it means that the faster the system reaches the final position, the smaller the objective function will be. In case of desired intermediate states along the trajectory, these elements can be chosen accordingly. So, in this example, $\mathbf{x}_{\mathbf{F}_k}$ represents the final states $\forall k \in \{1, 2, 3, \ldots N\}$.

$$\min J = \min \sum_{k=1}^{N} \left(\mathbf{x_k} - \mathbf{x_{F_k}}\right)^T \mathbf{Q}\left(\mathbf{x_k} - \mathbf{x_{F_k}}\right) + \mathbf{u_k}^T \mathbf{R} \mathbf{u_k} \tag{3-49}$$

$$\mathbf{Q} = \begin{bmatrix} 0.1 & 0 & 0 & 0 \\ 0 & 0.01 & 0 & 0 \\ 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 0.1 \end{bmatrix}, \quad \mathbf{R} = \begin{bmatrix} 0.08 & 0 \\ 0 & 0.08 \end{bmatrix} \tag{3-50}$$

Results

The example was solved in MATLAB, using FMINCON's interior-point algorithm as the nonlinear programming solver. Setting up the optimization problem involves defining the objective function (3-49), the dynamics constraints (3-9), (3-14), (3-5), the boundary constraints (3-39), the range of acceptable values for each optimization variable (3-40)-(3-42) and the initial guess (3-43)-(3-48). Figure 3-10 refers to the case where the lower bound of the final time was set to $t_{FL} = 1s$, while the upper bound was set to $t_{FU} = 6s$. Figure 3-11 refers to the case where both final time bounds were set equal to 6s. The simulation stops when the pendulum reaches its goal. If the lower and upper bounds of time are set equal to a specific time, the pendulum will take just that much time to reach the goal. Table 3-3 lists the solution information of the two cases that were examined. The objective function value is minimum when the Hermite-Simpson direct collocation method is used.

The maximum number of iterations is specified by the accuracy option. The default option, which is 'medium' accuracy, sets the maximum number of iterations equal to 400. This limit is reached when the final time is free and the problem is handled by Hermite Simpson direct collocation or multiple shooting method. If the accuracy is set to 'high', the maximum number of iterations becomes 800, but the allowable tolerances become smaller, so the maximum number of iterations is reached again. When the final time is free, the trajectories obtained from three different methods are almost identical. When the problem has additional restrictions (fixed final time), these trajectories are very different from each other.

Table 3-3. Solution information of the three different direct methods when the final time is free and when it is fixed.

Info	Trapezoidal Direct Collocation		Hermite Simpson Direct Collocation		Multiple Shooting	
	Free	Fixed	Free	Fixed	Free	Fixed
NLP Time [s]	10.25	8.58	31.27	11.46	73.95	30.48
Iterations	196	192	400	191	400	196
Func. Count	36569	35616	101085	47985	100970	49138
Grid points	30	30	40	40	40	40
Obj value	10.43	11.35	10.33	11.05	10.34	11.82

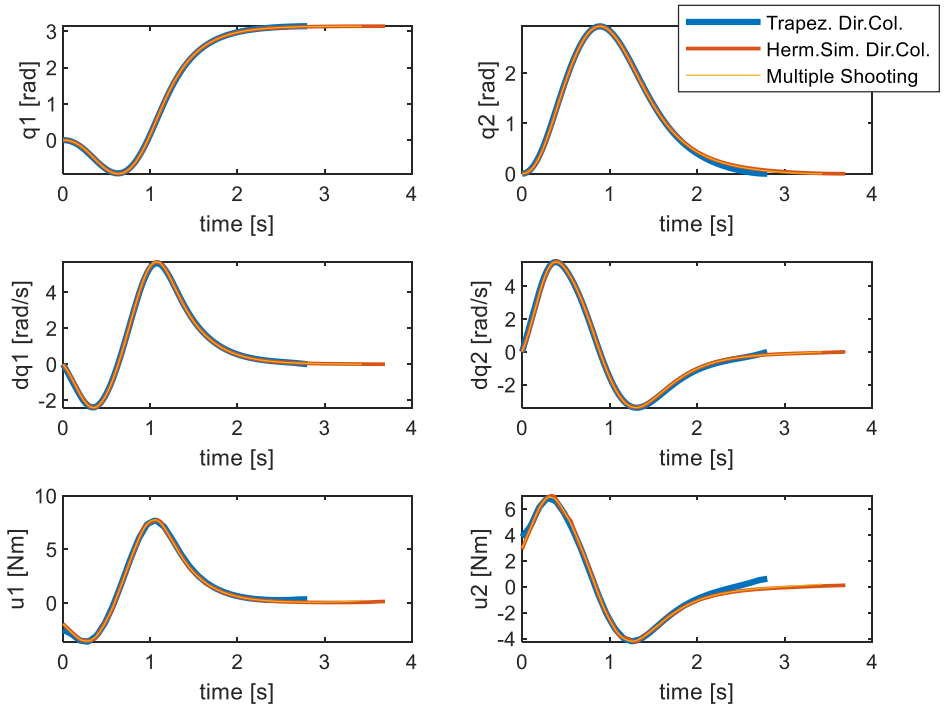


Figure 3-10. Trajectories of a two-link manipulator that reaches the upper position using three different direct methods when the final time is free.

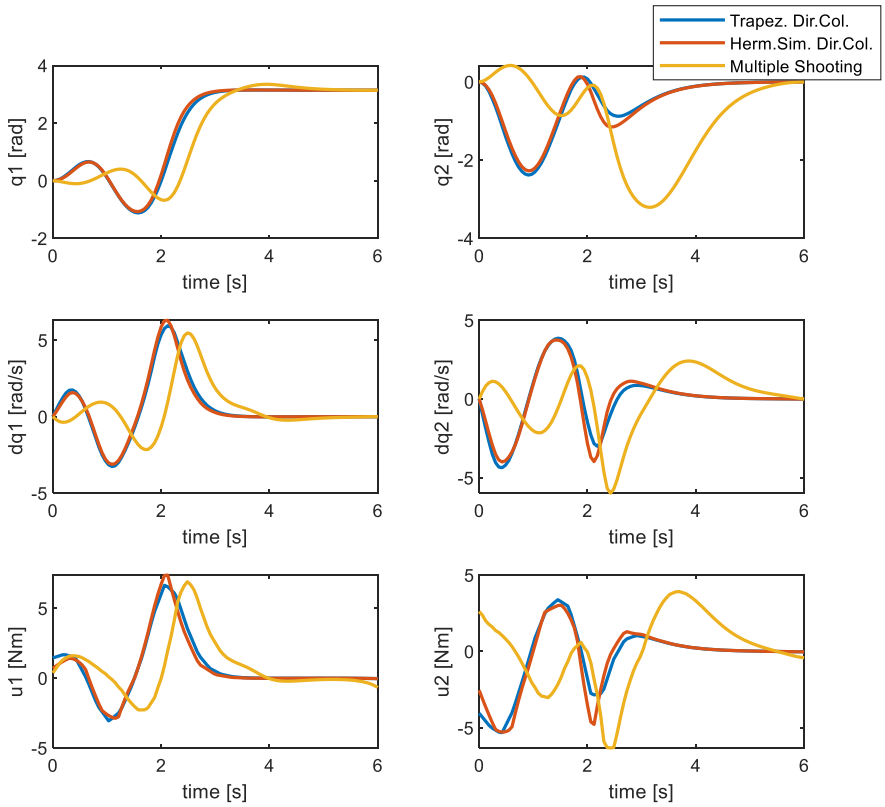


Figure 3-11. Trajectories of a two-link manipulator that reaches the upper position using three different direct methods when the final time is fixed.

3.6.3 Problem Setup 3 – CasADi – Single Shooting and NMPC

CasADi is evaluated in this and the next problem setup, where the trajectory optimization problem is handled by shooting methods. Also, these two examples highlight the potential of NMPC when its feedback is derived from the solution of the trajectory optimization problem. Single shooting is the simplest method, but it is not recommended for unstable systems. To handle the two-link manipulator, single shooting is used in combination with NMPC. The prediction horizon, N_{NMPC}, is the number of future control intervals the NMPC controller must evaluate by prediction when optimizing its optimization variables at control interval k. The prediction horizon is set to $N_{NMPC} = 20$ and the sampling time is $T = 0.05\,s$. The optimization variables are the controls over the optimization horizon (3-51). The states over the optimization problem depend on the control inputs and the initial value $\mathbf{x_0}$ as shown in (3-1).

$$\mathbf{u} = \begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \end{bmatrix} \in \mathbb{R}^{2\times N} \tag{3-51}$$

In this library, a vector of parameters $\mathbf{p}$ is introduced, which includes the initial and the final states (goal) (3-52). The length of this vector is two times the number of states n_{st}.

$$\mathbf{p} = \begin{bmatrix} \mathbf{x_0} \\ \mathbf{x_{ref}} \end{bmatrix} \in \mathbb{R}^{2n_{st}\times 1} \tag{3-52}$$

Objective Function

The objective function defined in (3-49) is written in the form (3-53), where the selected elements of the vector $\mathbf{p}$ are the final states.

$$J = \sum_{k=1}^{N} \left\{ \left(\mathbf{x_k} - \mathbf{x_{ref}}\right)^T \mathbf{Q} \left(\mathbf{x_k} - \mathbf{x_{ref}}\right) + \mathbf{u_k}^{\mathbf{T}} \mathbf{R} \mathbf{u_k} \right\} \tag{3-53}$$

NMPC loop

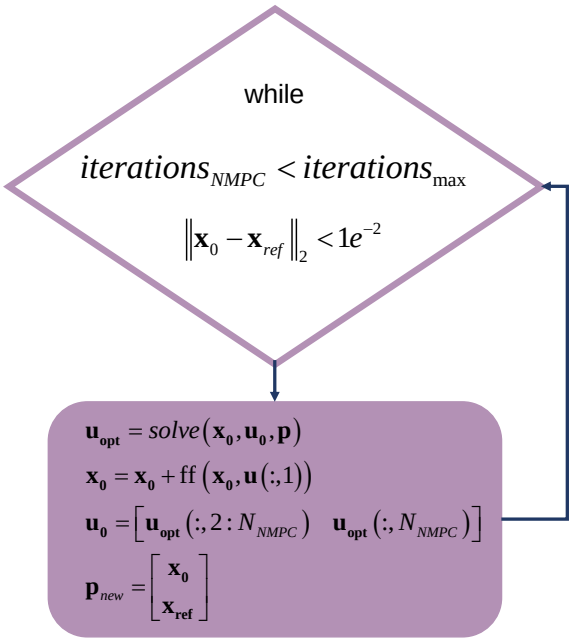


Figure 3-12. NMPC flow chart.

NMPC is initialized every timestep, so the first four elements of the vector $\mathbf{p}$ are the current states of the two-link manipulator. The solution of the optimization problem is a vector of the optimized controls $\mathbf{u}_{\mathbf{opt}}$. Only the first control input is used to compute the new $\mathbf{x}_{\mathbf{0}}$. The new $\mathbf{u}_{\mathbf{0}}$ vector consists of the optimized controls from the simulation, but the first element is trimmed because it has been used and the last element is repeated twice as a guess for the last timestep. Figure 3-12 represents a block diagram of the single shooting method combined with NMPC.

Results

The example was solved in MATLAB, using IPOPT as the nonlinear programming solver. Setting up the optimization problem involves defining the objective function (3-53), the dynamics constraints (3-2), the boundary constraints (3-39) and the range of acceptable values for each optimization variable (3-40)-(3-42). The final time was set $t_F = 6s$. As an initial guess for the optimization variables, all the control inputs were set to zero. The results for the states and control torques are depicted in Figure 3-13. The solution took 2.94 s in total and the average time of each MPC iteration was 0.0526 s.

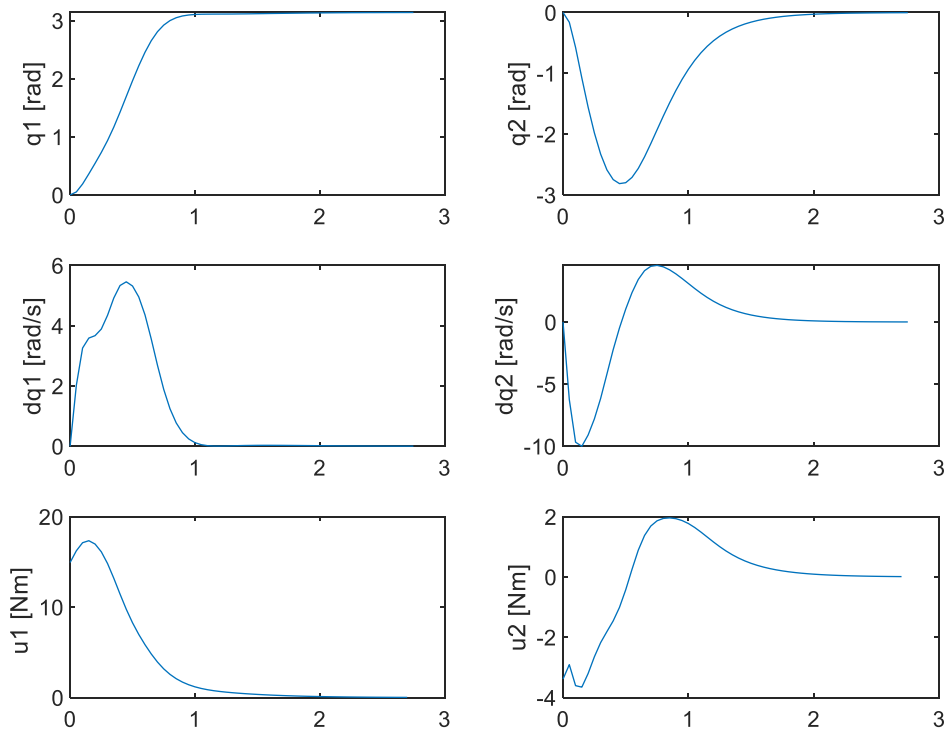


Figure 3-13. CasADi -Solution of the NLP problem using Single Shooting and NMPC.

3.6.4 Problem Setup 4 – CasADi – Multiple Shooting and NMPC

In this problem setup, the NLP problem is solved using the direct multiple shooting method and a fixed step explicit Runge-Kutta integration method. The optimization variables are given in (3-36). The time interval T is divided into N equidistant segments of length h according to (3-54):

$$h = \frac{T}{N} \tag{3-54}$$

where the timestep was set to $h = 0.06\,s$ and the total number of segments was set to $N = 100$. The objective function is the one defined in (3-53) and the defect constraints $\mathbf{g} = \mathbf{0}$ take the form of (3-5). The initialization is the same as in the problem setup 3.

Results

Figure 3-14 shows the open loop solution of the trajectory optimization problem, while Figure 3-15 represents the closed loop solution in which multiple shooting was used in combination with nonlinear model predictive control (NMPC). The weighting matrices $\mathbf{Q}$, $\mathbf{R}$ that were defined in (3-50) must be retuned when the problem formulation includes the NMPC. It was found that the objective function affects significantly the system, since the system fails to reach the final target when using the weights given in (3-50). The weighting matrices given in (3-55) lead to the response shown in Figure 3-15. The IPOPT solver was used, and the maximum number of iterations was set to 2000. The closed-loop solution took 5.07 s in total and the average time of each MPC iteration was 0.0818 s.

$$\mathbf{Q} = \begin{bmatrix} 0.6 & 0 & 0 & 0 \\ 0 & 0.06 & 0 & 0 \\ 0 & 0 & 0.001 & 0 \\ 0 & 0 & 0 & 0.01 \end{bmatrix}, \quad \mathbf{R} = \begin{bmatrix} 0.06 & 0 \\ 0 & 0.006 \end{bmatrix} \tag{3-55}$$

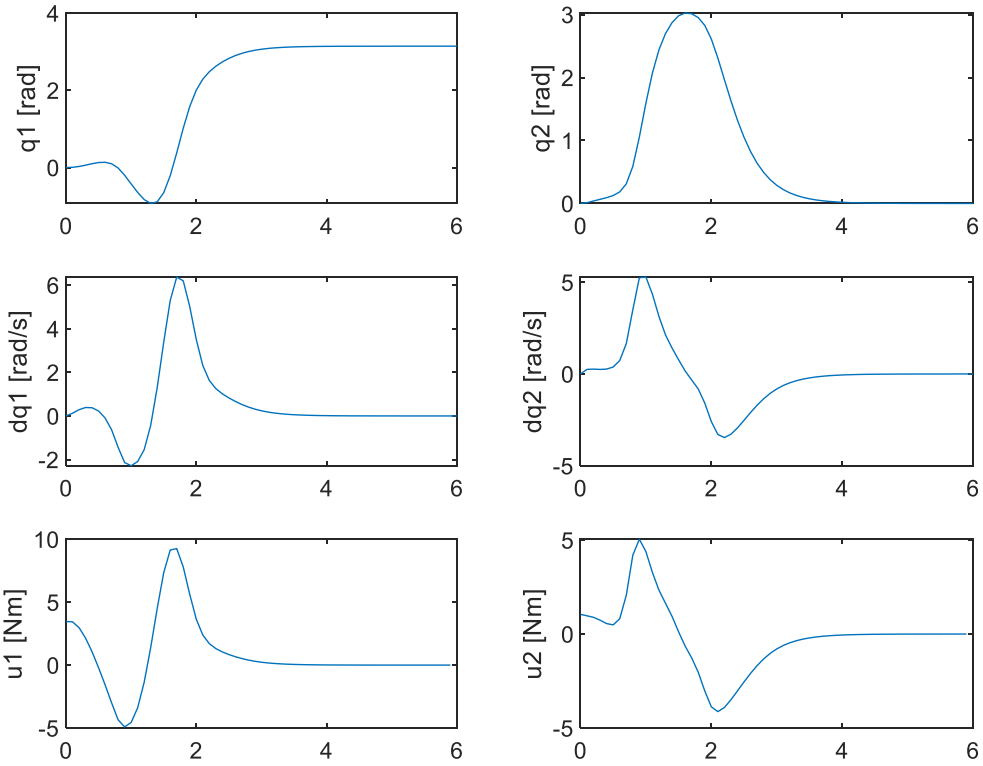


Figure 3-14. CasADi - Open-loop solution of the NLP problem using Multiple Shooting.

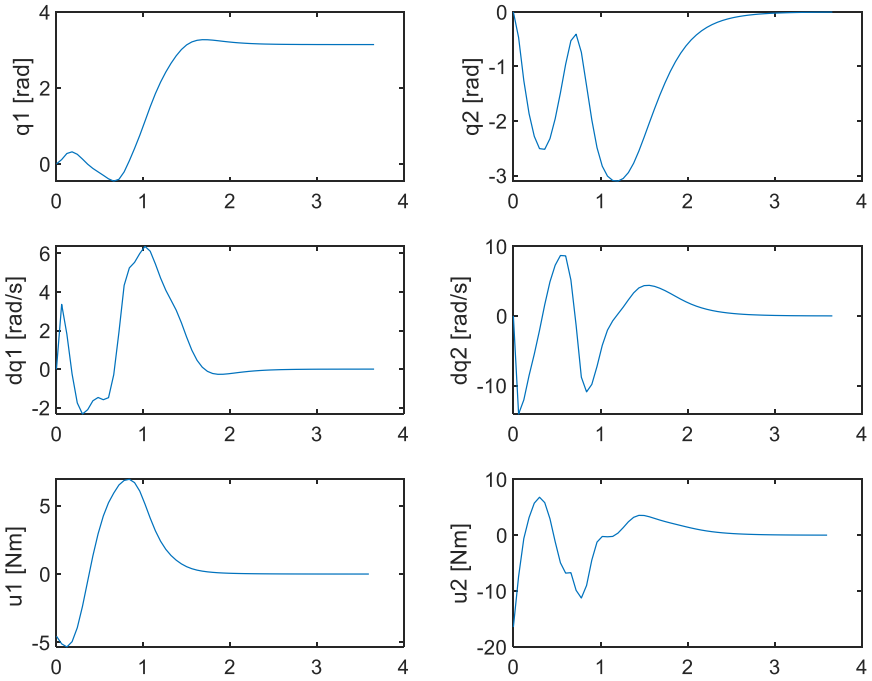


Figure 3-15. CasADi - Closed-loop solution of the NLP using Multiple shooting and NMPC.

3.6.5 Conclusion

The evaluation of the direct methods was performed in the OptimTraj library, which is easy to use and does not require expert knowledge of the numerical methods, since it includes the functions of the most important direct methods. The results proved that trapezoidal direct collocation is the fastest direct method, but when the grid points are not close enough, the linear interpolation does not provide sufficient accuracy. Regarding the high accuracy methods, Hermite Simpson direct collocation was much faster than multiple shooting.

The OCPs that were setup in CasADi evaluate the shooting methods and their solving time when they are combined with a NMPC which recomputes the problem in each time step. The results revealed that when the TO problem was handled by the single shooting method, the average time of each MPC iteration was 52 ms, and when the multiple shooting was used, the average time of each MPC iteration was 82 ms. When using NMPC, it was observed that the time horizon and the definition of the weighting matrices in (3-49), affect in large extent the response of the system. Having tried various weighting values, it was found that the system was not always able to reach the final target.

The solving times in the two optimization libraries can be compared by formulating the same problem. The stabilization of the two-link manipulator using multiple shooting and the objective function (3-49) was setup both in OptimTraj and CasADi and the graphs of the system's response are depicted in Figure 3-16. The upper vertical position is reached in 4 s in both cases. OptimTraj uses the FMINCON's interior point algorithm to solve the NLP, while CasADi uses IPOPT. Despite that in both cases the problem is solved by interior point algorithms, the required solving time differs significantly. In Casadi, the NLP was solved in 1.72 s, while in OptimTraj, it was solved in 73.95 s. According to this example, the choice of the most relevant library is obvious, provided that CasADi returns a result much faster. The

value of the objective function is nearly the same, so solving time is the determinant of choosing CasADi.

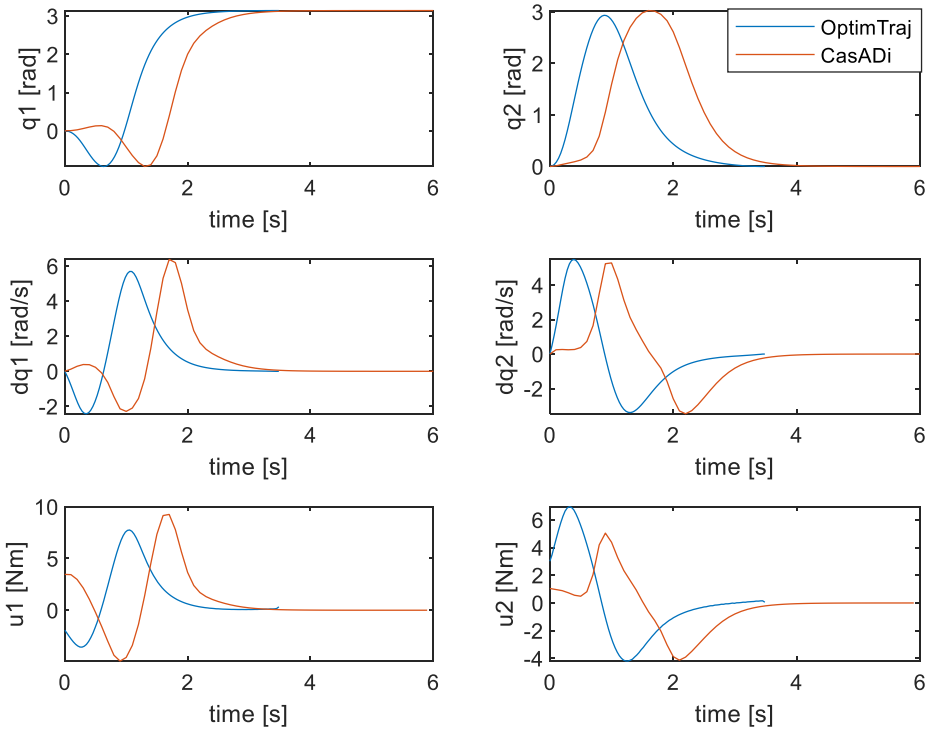


Figure 3-16. Trajectories of a two-link manipulator that reaches the upper position using multiple shooting in OptimTraj and CasADi.

4 Argos Motion Planning & Control

4.1 Introduction

This chapter focuses on effective motion planning algorithms related to Argos locomotion. It is divided in three main sections. Section 4.2 demonstrates a simple planning and control framework that consists of gait scheduling, definition of foot trajectories in Cartesian space and implementation of a PD controller that tracks the desired motion plans. In Section 4.3, an alternative motion planning approach is investigated, in which the robot tracks velocity and turn rate commands. Similarly, the sequence and timing of leg motions are determined by a gait pattern, but the desired footholds are indirectly derived from the high-level commands. The desired body pose is controlled by adjusting the virtual foot forces through the formulation of an optimization problem. A PD controller is also designed to track the desired foot trajectories. In Section 4.4, a multi-phase trajectory optimization problem is formulated, which not only produces valid motions plans for the main body and the feet, but also finds the required feet forces to execute these motion plans. The problem setup includes the equations that describe the physical constraints and an objective function that minimizes a selected criterion. The trajectory optimization problem is solved using the CasADi library. Simulation results are presented at the end of each motion planning approach.

4.2 Toe level trajectory tracking with active compliance control

The goal of this framework is the locomotion of the quadruped in flat terrain by specifying valid foot trajectories in the Cartesian space. It is based on a prior motion plan algorithm that has been developed by the Legged Robots Team [41], but the main difference is that in this approach, the foot trajectory is defined by polynomials. The quadruped's base is unactuated; thus, its motion is determined indirectly by the feet's motion and their interaction with the environment. The gait determines when each foot contacts the ground. The common gaits, like walk (& amble), trot, pace (rack), canter, and gallop can be executed, provided that the start and the end of the swinging phase are prespecified. The trotting gait has been assessed in the context of this work. The periodic motion that is illustrated in Figure 4-1 indicates that half of the period T is spent in stance phase (grey color) and the other half in swing phase. In trotting gait, the diagonal legs move simultaneously and when the one pair touches down, the other one lifts up exactly at the same time instance. This section includes the planning of the swing foot trajectory and the controller that tracks the desired plans is analyzed next. The control framework is verified in the Simscape simulation environment[3]. The structure of the Simscape model is described in Appendix D. The results of the quadruped's locomotion are presented at the end of this section. The simulation environment and the solver settings are described in Section 2.5

[3] The MATLAB codes can be found here:
https://bitbucket.org/csl_legged/argos_active_compliance_control/src/master/

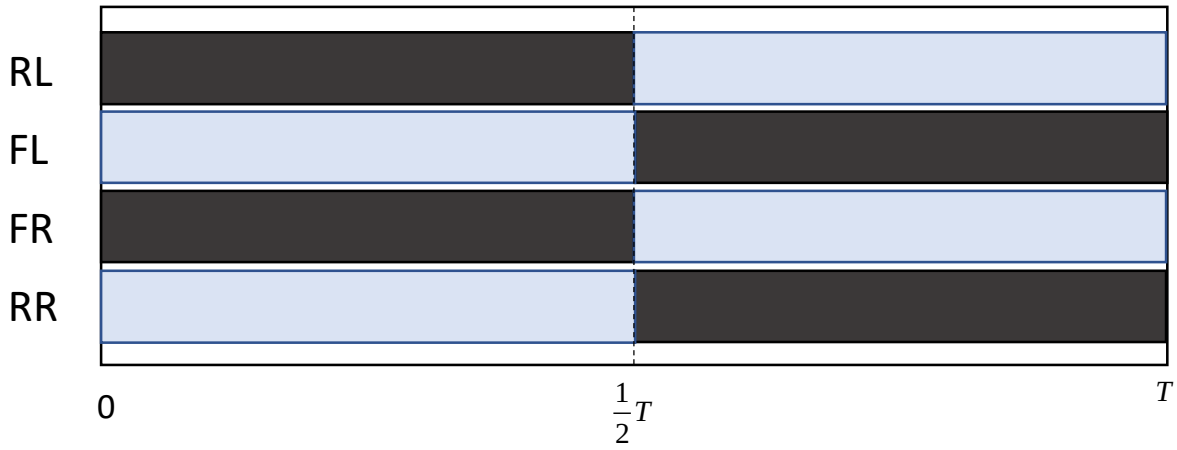


Figure 4-1. Trotting gait pattern.

4.2.1 Trajectory Planning

The desired foot trajectory is defined relative to the hip frame, and it determines the motion in the xy plane (Figure 4-2), while the desired angular position of the abduction/adduction joint is set to 0. The inverse kinematics problem is 2D and the output are the joint angles of the hip and knee joint. In Figure 4-2, the swing phase appears in light blue color. The stance phase is represented as a constant line in the xy plane because while the foot stays still on the ground, the hip frame is translated forward due to the body's movement. Then, the x, y coordinates of the trajectory must be expressed as a function of time. The polynomial that represents the swing phase can be defined by providing the desired position at the starting, the middle and the final point. However, four extra conditions are added for the transition points to ensure the continuity of the first and second derivative. Having already set seven conditions (4-4), the swing phase will be represented by a 6th order polynomial and the stance trajectory is derived easily by linear interpolation between the initial and the final point of the swing trajectory (4-1)-(4-2). While the foot is executing the flat part of the trajectory, it is assumed that it does not slip due to the ground friction. Thus, the foot stays still, and a forward force is generated that enables the main body to move.

$$x_f^d(t) = \begin{cases} x_{sw}^d = a_1 t^6 + b_1 t^5 + c_1 t^4 + d_1 t^3 + e_1 t^2 + f_1 t + g_1 & \text{(swing)} \\ x_{st}^d = a_3 t + b_3 & \text{(stance)} \end{cases} \quad (4\text{-}1)$$

$$y_f^d(t) = \begin{cases} y_{sw}^d = a_2 t^6 + b_2 t^5 + c_2 t^4 + d_2 t^3 + e_2 t^2 + f_2 t + g_2 & \text{(swing)} \\ y_{st}^d = 0 & \text{(stance)} \end{cases} \quad (4\text{-}2)$$

The polynomial coefficients in (4-1)-(4-2) can be found by specifying the initial, the intermediate and the final point of the swing phase as $(x_0,0),(x_1,y_1),(x_2,0)$. Then, the stance phase polynomial coefficients arise from (4-3), where T_{st} is the total duration of the stance phase. The variable T_{sw} in (4-4) denotes the total duration of the swing phase. In trotting gait T_{sw} is equal to T_{st}.

$$\left.\begin{matrix} x_{st}^d(0) = x_2 \\ x_{st}^d(T_{st}) = x_0 \end{matrix}\right\} \Rightarrow \left.\begin{matrix} b_3 = x_2 \\ a_3 T_{st} + b_3 = x_0 \end{matrix}\right\} \Rightarrow \begin{matrix} b_3 = x_2 \\ a_3 = \dfrac{(x_0 - x_2)}{T_{st}} \end{matrix} \quad (4\text{-}3)$$

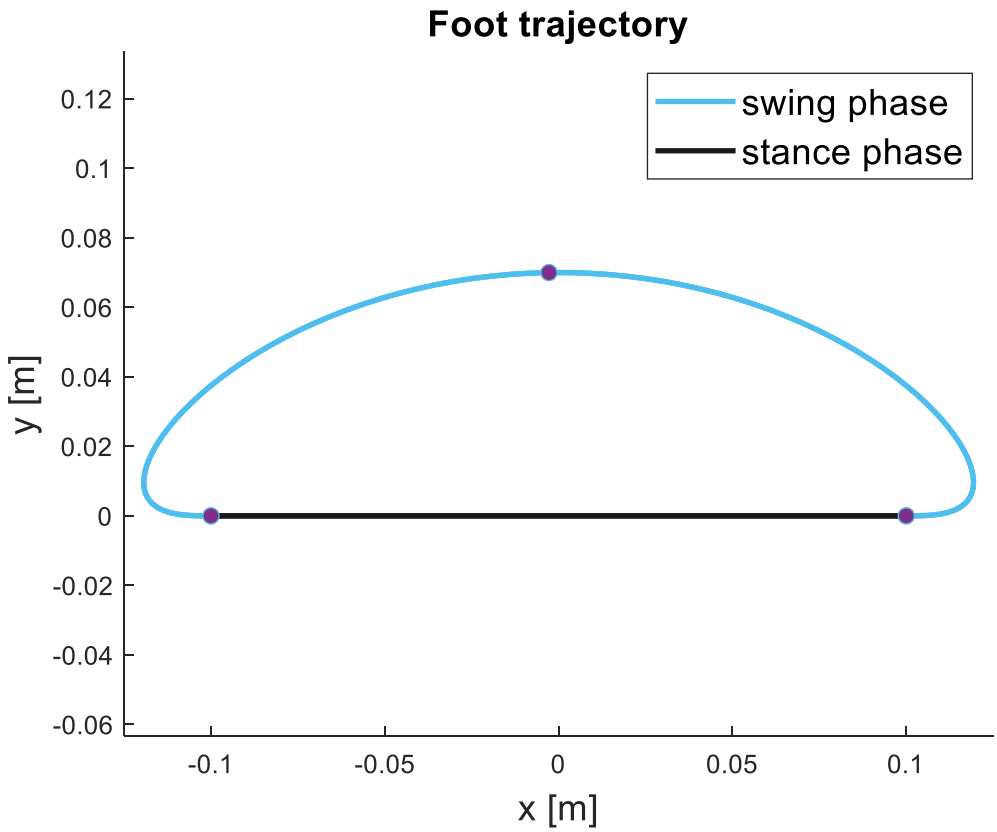


Figure 4-2. The swing trajectory is denoted by the light blue color while the stance trajectory is denoted by dark grey color.

$$
\begin{aligned}
x_{sw}^{d}(0) &= x_0 & y_{sw}^{d}(0) &= 0 \\
x_{sw}^{d}\left(\frac{T_{sw}}{2}\right) &= x_1 & y_{sw}^{d}\left(\frac{T_{sw}}{2}\right) &= y_1 \\
x_{sw}^{d}\left(T_{sw}\right) &= x_2 & y_{sw}^{d}\left(T_{sw}\right) &= 0 \\
\dot{x}_{sw}^{d}(0) &= \alpha_3 & \dot{y}_{sw}^{d}(0) &= 0 \\
\dot{x}_{sw}^{d}(T_{sw}) &= \alpha_3 & \dot{y}_{sw}^{d}(T_{sw}) &= 0 \\
\ddot{x}_{sw}^{d}(0) &= 0 & \ddot{y}_{sw}^{d}(0) &= 0 \\
\ddot{x}_{sw}^{d}(T_{sw}) &= 0 & \ddot{y}_{sw}^{d}(T_{sw}) &= 0
\end{aligned}
\tag{4-4}
$$

The derivatives of (4-1)-(4-2) give the desired velocity of the foot in Cartesian space (4-5)-(4-6). These are mapped to joint velocities through the inverse geometric Jacobian (4-7). The geometric Jacobian in 3D space is given by (2-12). Eliminating the third row and third column results in the simplified form of the Jacobian in 2D space (xy plane), which is given by (4-8). Apparently, the desired abduction/adduction velocity is set to zero.

$$\dot{x}_f^d = \begin{cases} x_{sw}^d = 6a_1t^5 + 5b_1t^4 + 4c_1t^3 + 3d_1t^2 + 2e_1t + f_1 \\ x_{st}^d = a_3 \end{cases} \tag{4-5}$$

$$\dot{y}_f^d = \begin{cases} y_{sw}^d = 6a_2t^5 + 5b_2t^4 + 4c_2t^3 + 3d_2t^2 + 2e_2t + f_2t \\ y_{st}^d = 0 \end{cases} \tag{4-6}$$

$$\dot{\mathbf{q}}^d = \mathbf{J}^{-1}\mathbf{v} \Rightarrow \begin{bmatrix} \dot{q}_{hip}^d \\ \dot{q}_{knee}^d \end{bmatrix} = \mathbf{J}^{-1} \begin{bmatrix} \dot{x}_f^d \\ \dot{y}_f^d \end{bmatrix} \tag{4-7}$$

$$\mathbf{J} = \begin{bmatrix} \frac{\partial^0 X_E}{\partial q_{hip}} & \frac{\partial^0 X_E}{\partial q_{knee}} \\ \frac{\partial^0 Y_E}{\partial q_{hip}} & \frac{\partial^0 Y_E}{\partial q_{knee}} \end{bmatrix} \tag{4-8}$$

4.2.2 Control

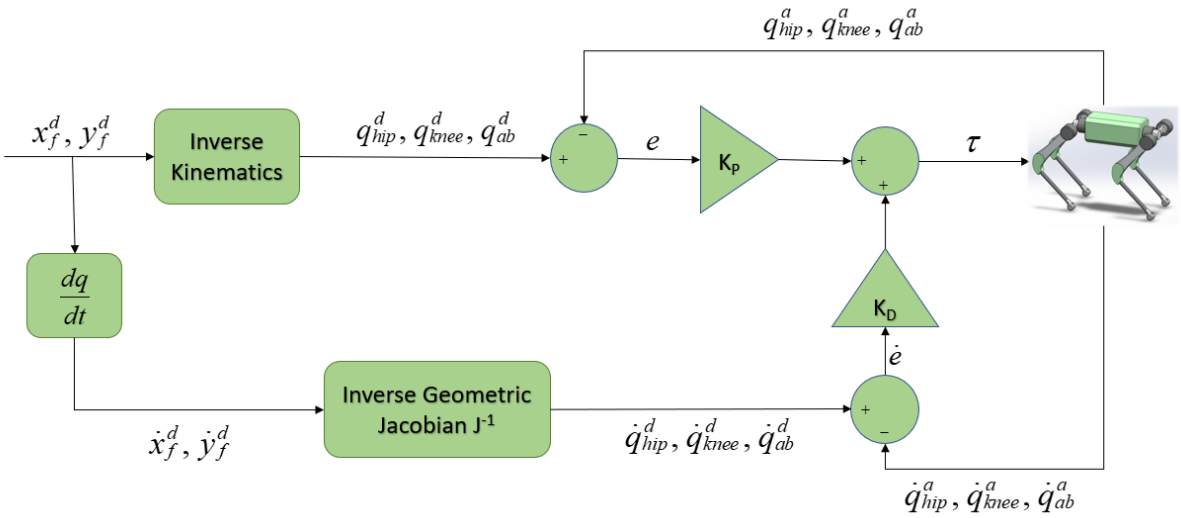


Figure 4-3. Block diagram of the control framework based on a joint space PD controller.

The desired trajectories and their derivatives in Cartesian space are transformed into joint angles and joint velocities respectively. Figure 4-3 represents the robot cartesian controller, which is based on a PD joint controller. Each leg and even each joint is independent of the others. To track the desired commands, a PD controller is employed, which takes as input the error between the desired and the actual joint angles and the error between the desired and the actual joint velocities. The output of the controller are the joint torques to the hip, knee and abduction/adduction actuators (4-9)-(4-11).

$$\tau_{hip} = K_P\left(q_{hip}^d - q_{hip}^a\right) + K_D\left(\dot{q}_{hip}^d - \dot{q}_{hip}^a\right) \tag{4-9}$$

$$\tau_{knee} = K_P\left(q_{knee}^d - q_{knee}^a\right) + K_D\left(\dot{q}_{knee}^d - \dot{q}_{knee}^a\right) \tag{4-10}$$

$$\tau_{ab} = K_P\left(q_{ab}^d - q_{ab}^a\right) + K_D\left(\dot{q}_{ab}^d - \dot{q}_{ab}^a\right) \tag{4-11}$$

The gains that appear in (4-9)-(4-11) are listed in Table 4-1 and were found experimentally by trial and error. These gains ensure good tracking of the trajectory and smooth movements.

Table 4-1. PD controller gains – Trotting gait.

Gains	Hip	Knee	Abduction
K_P	1000	2000	2000
K_D	300	300	100

4.2.3 Results

In this simulation, the period was set to $0.75s$, each new foothold was set $0.2m$ away from the current one and the maximum height of the swing trajectory was set to $0.07m$. The ground stiffness (k) and damping (b) were set equal to $1e^4\ N/m$ and $1e^3\ Ns/m$ respectively. A saturation block has been added to limit the actuators torques. to the range $\pm 120\ Nm$. The first three graphs refer to the body's motion (Figure 4-4-Figure 4-6). This formulation does not include desired commands for the body's pose. However, ideally, the orientation should not deviate significantly from the nominal position. The orientation error about the x axis (roll) does not exceed 2 degrees, which is acceptable (Figure 4-5). The orientation about the y and z axes (yaw and pitch) can have larger deviations without causing a problem in the quadruped's balance (Figure 4-5). Finally, the body's mean velocity at steady state is around $0.4\,m/s$ (Figure 4-6). The periodic oscillations are due to the collisions with the ground. Figure 4-7 depicts the tracking of the hip and knee joint angles. During the collision with the ground, the actual knee angular position deviates from the desired trajectory. Simulation results showed that if the gains are increased, better tracking is achieved. However, the torque requirements increase, thus there is a trade-off between the tracking accuracy and the magnitude of the torques. The large error in (4-10) is necessary to generate torques large enough to support the body weight and move the body forward due to the PD used. Figure 4-8 displays the joint torques for the front left and the rear right leg and it is revealed that the latter takes greater values, which is expected as the rear legs support a greater part of the total weight.

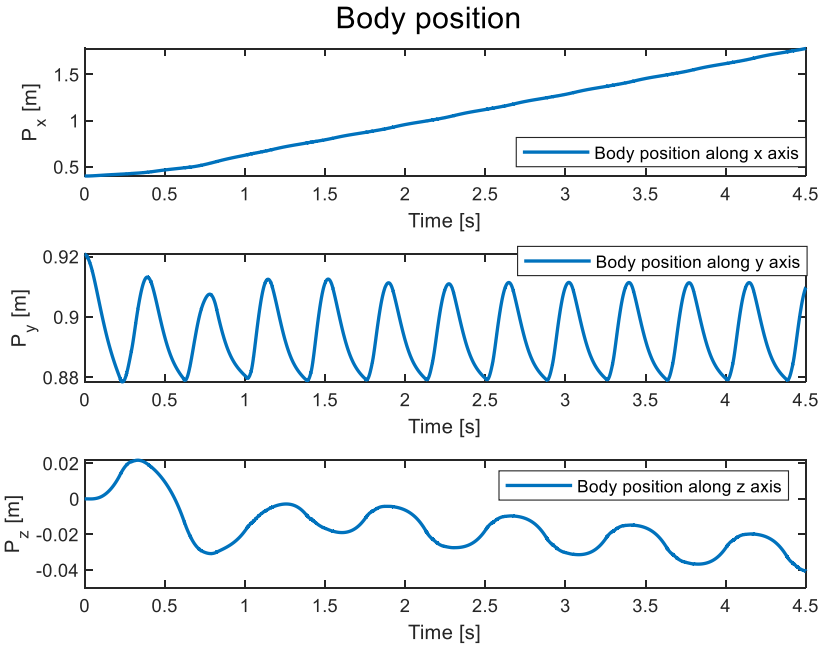


Figure 4-4. CoM's position.

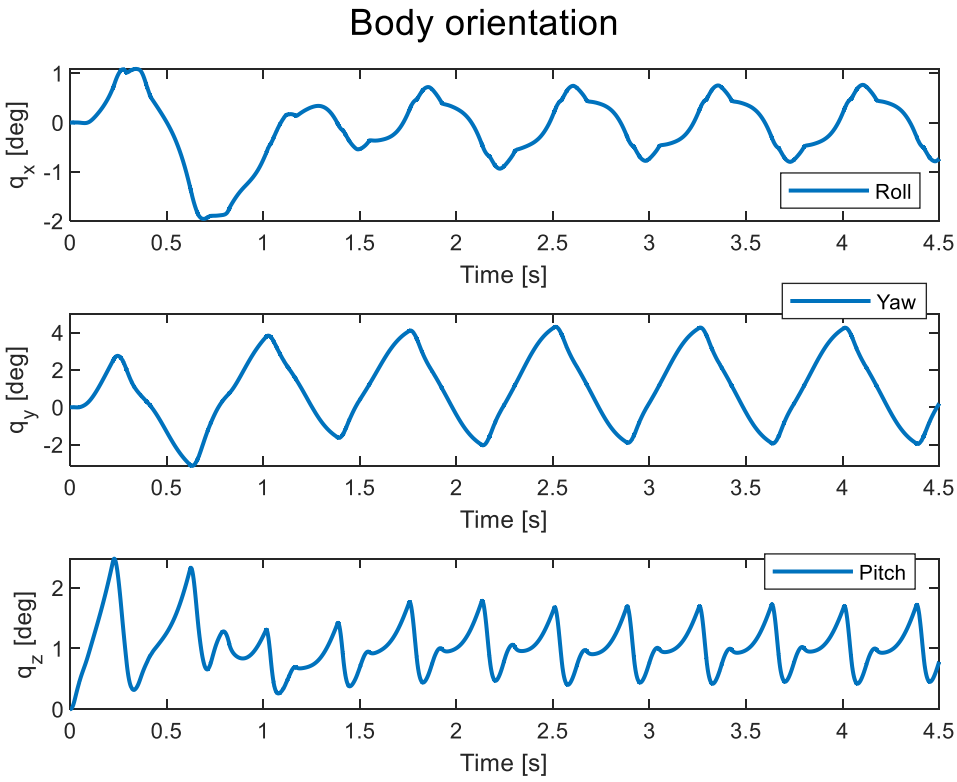


Figure 4-5. CoM's orientation described by roll, pitch, yaw Euler angles.

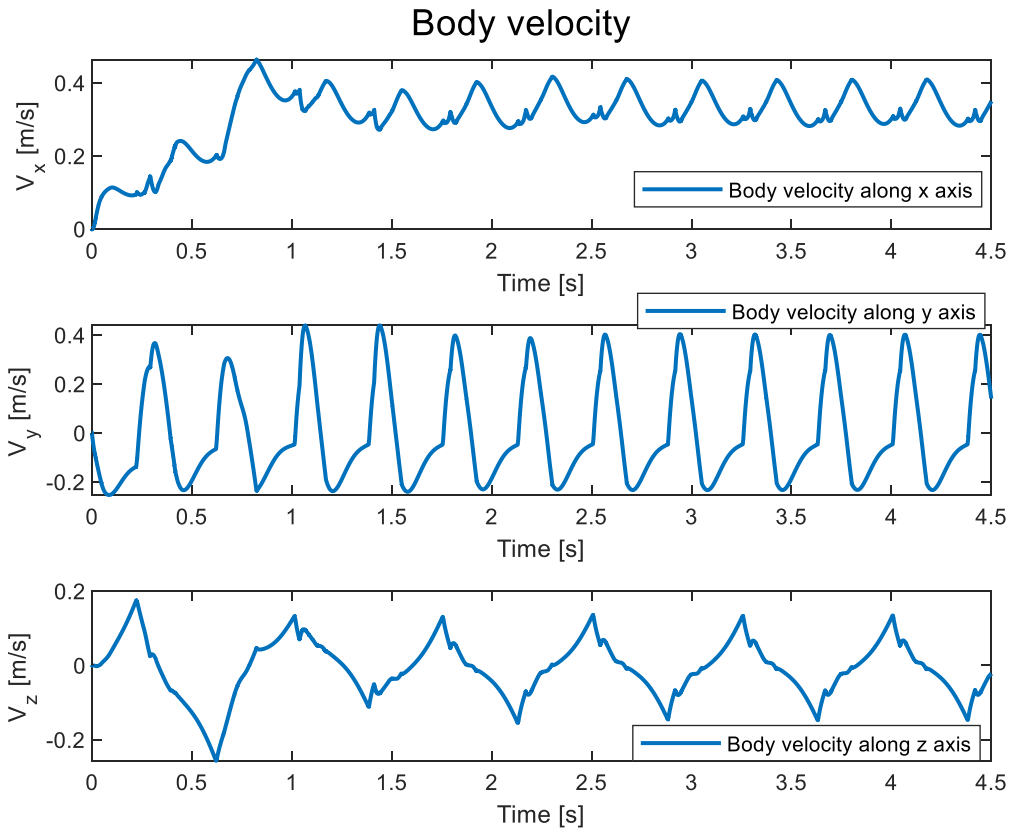


Figure 4-6. CoM's velocity.

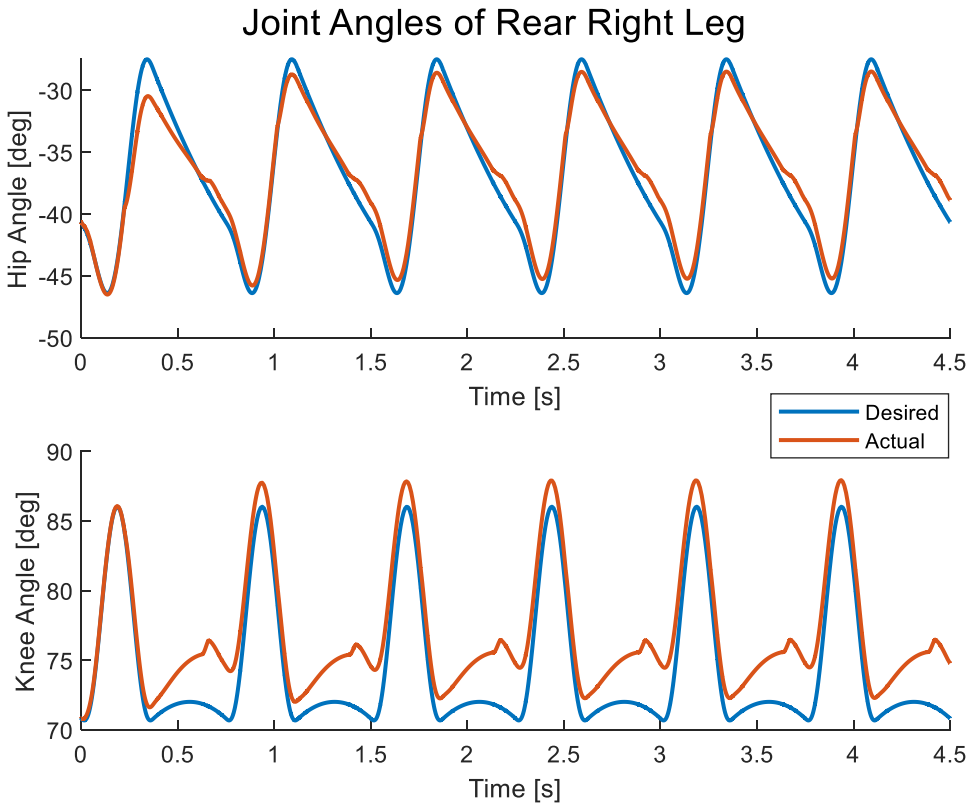


Figure 4-7. Desired vs actual joint angular position of the rear right leg. The upper graph refers to the hip joint and the bottom one refers to the knee joint.

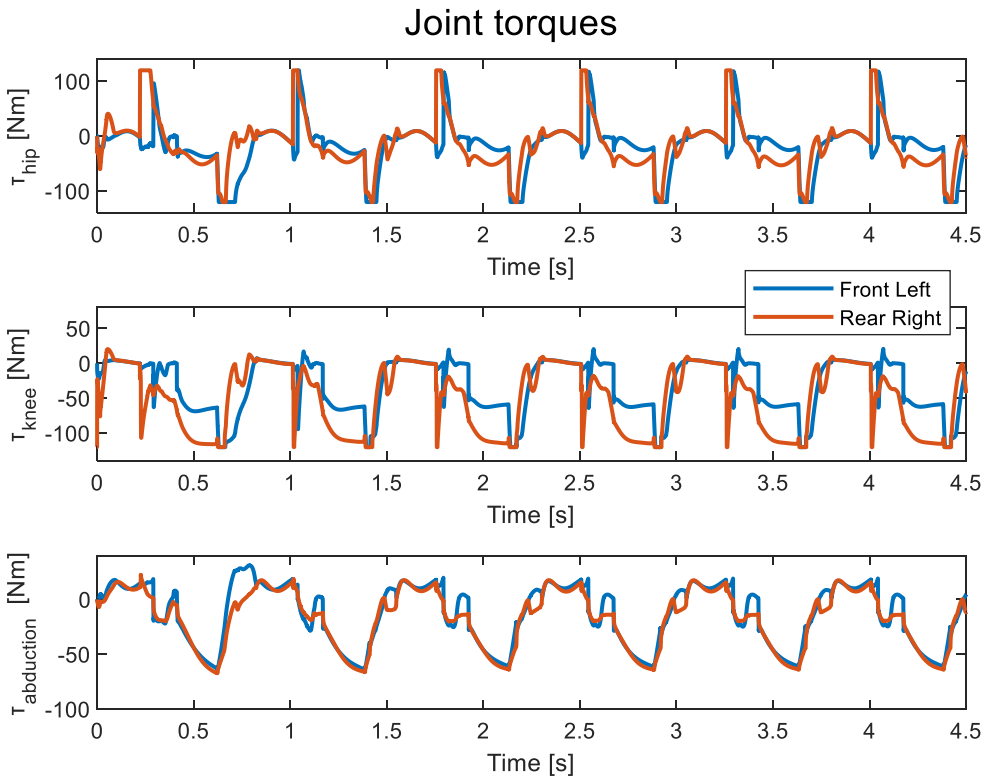


Figure 4-8. Joint torques of the front left and rear right leg.

4.2.4 Conclusion

A really simple but effective way to achieve the quadruped's locomotion is the one described in the previous sections. It incorporates three main parts; a gait pattern, which determines the swing and the stance timings for each foot, a motion planner through which the foot trajectories are defined and a controller that tracks the desired motions. The continuous transition from the swing to the stance trajectory results in smooth contact with the ground and no sudden change in torque requirements. The simulation revealed that the quadruped moves at a constant speed in steady state and the deviations in roll, yaw and pitch angles are negligible, so it keeps its balance. However, it cannot handle a perturbation by making adjustments at the footsteps. Thus, this formulation is recommended for walking on known terrain and especially on flat terrain.

4.3 Motion Planning & Control Based on Optimal Force Distribution

The original idea of this work comes from [23], which presents a control framework implemented on the quadruped StarlETH. The quadruped is able to perform various gaits and also to be robust to external disturbances. In comparison to the approach that was investigated in the previous section, in this formulation, the robot is moving according to high level commands, which determine the body's velocity and its orientation about the y axis (yaw). An optimization problem is formulated for the force distribution to the stance legs, so that the robot maintains its desired pose. The above framework was tested on Argos to find out how the optimization algorithm improves its locomotion. It consists of a motion planner and a motion controller. The outputs are the desired joint torques $\boldsymbol{\tau}^d$ for the stance legs or the desired joint angles $\mathbf{q}_j^d$, which are mapped to the desired torques using inverse dynamics. The optimization problem is formulated using CasADi; an open-source tool for nonlinear optimization and algorithmic differentiation [3]. An overview of the framework is shown in Figure 4-9.

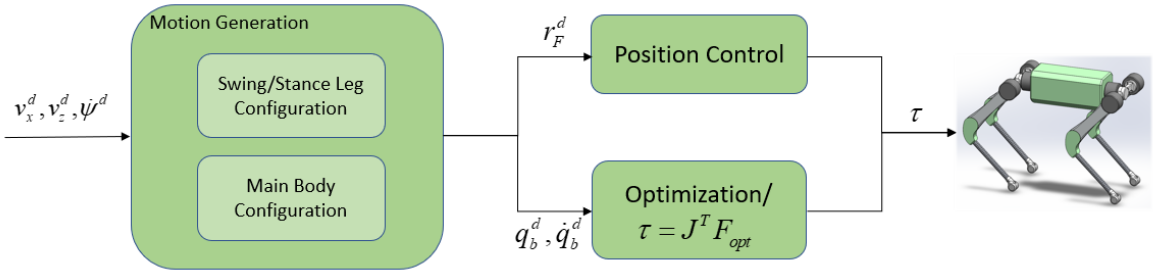


Figure 4-9. Block diagram of the control framework. Motion planning algorithms exploit high-level velocity and turn rate commands to output the desired body pose and foot position. The control algorithms take as input these desired commands and output the required torques at the actuated joints.

This section is organized as follows. The desired foothold selection at each gait cycle and the trajectory that the foot follows to reach the desired position are presented in Sections 4.3.2-4.3.3. The goal is to track this trajectory, which is addressed in Section 4.3.7. The determination of a desired body pose and how this is achieved is explored in Sections 4.3.5 and 4.3.8 through the formulation of an optimization problem. The desired velocity in x, z direction v_x^d, v_z^d and the yaw rate $\dot{\psi}^d$ are treated as high-level input parameters and can be

modified easily. The parameters related to the ground that were chosen for the simulation are discussed in Section 2.5.2 and the results are shown in Section 4.3.10.

Compared to [23], there have been some modifications. Regarding the swing leg trajectories, a semi-ellipse is defined, instead of splines, which is simpler and can be easily generated. As for the trajectory that is generated when the foot loses contact with the ground, in [23] the foot is translated 1cm lower than its current position. In this work, the foot moves vertically downwards until the force sensor gives positive values. Additionally, the control framework of [23] comprises an extra low-level controller that takes as input the desired joint angles or the desired torque and regulates the motor current I^d by considering the dynamics of the actuators. Finally, Argos has different mass properties, size and leg configuration and this framework is verified in a different simulation environment, so the parameters that have been tuned experimentally in [23] are adapted accordingly.

Regarding the desired body pose proposed in [23], even though the forward velocity is constant, the desired body position is not a linear function of time. It is defined relative to the footholds. It is claimed that this formulation produces smooth trajectories while ensuring the robot's stability and the compatibility with the desired velocity commands. Quantities such as position or velocity are described in the inertial frame I or relative to the hip frame H of each leg (Figure 4-10). Both frames' axes are aligned with the same orientation. The x axis indicates the heading direction, and the y axis is vertical to the ground.

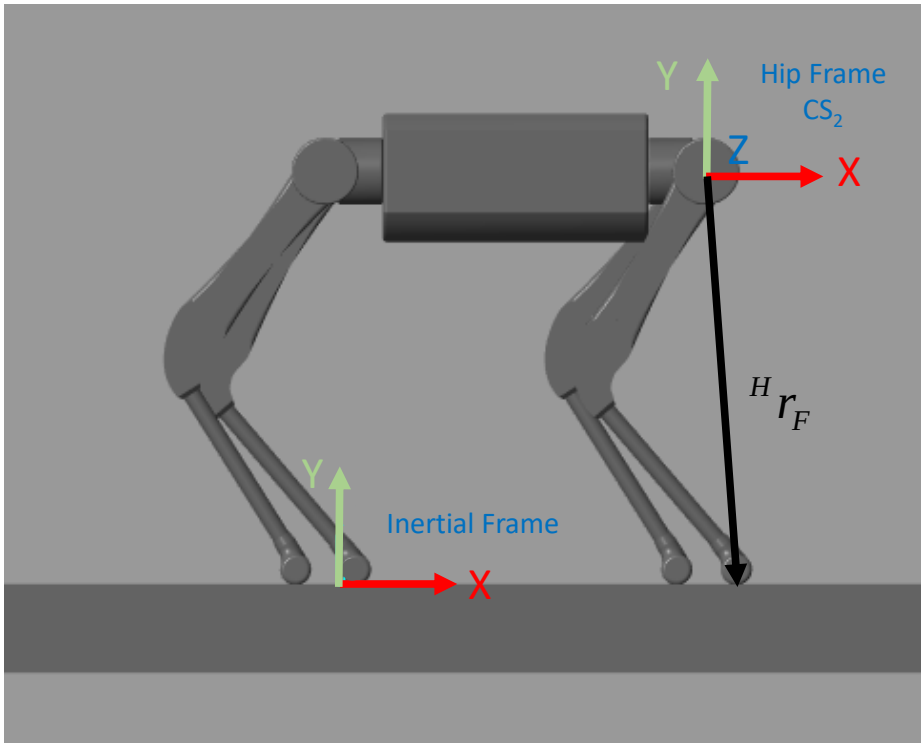


Figure 4-10. Inertial Frame and Front Right leg's Hip Frame. The foothold is defined relative to the Hip Frame.

4.3.1 Gait Scheduling

The quadruped moves forward according to a Gait pattern. The time normalized stride-phase $\varphi \in [0 \quad 1]$ is the equivalent of the gait cycle T and not only informs the controller regarding a leg's status (swing or stance phase), but also regarding the time left until the next phase. In a similar way, the variables φ_{sw} and φ_{st} indicate the time normalized progress made in swing

and stance mode (4-12), where T_{sw} and T_{st} are the time durations of swing and stance phase respectively. As a result, the controller anticipates how the support polygon will change (Figure 4-12).

$$\begin{aligned} \varphi &= \frac{t}{T} \qquad t \in [0 \quad T] \\ \varphi_{sw} &= \frac{t}{T_{sw}} \quad t \in [0 \quad T_{sw}] \\ \varphi_{st} &= \frac{t}{T_{st}} \quad t \in [0 \quad T_{st}] \end{aligned} \tag{4-12}$$

The vertices of the support polygon are described by the position of legs in contact with the ground. In case three legs are in stance at the same time, then it is a triangle, and if two legs are in stance, it is transformed to a support line. In general, static stability is ensured if the projection of the CoM lies over the support polygon and the polygon's area is greater than zero [56]. This condition requires at least three legs on the ground. The static stability criterion implies that a set of forces over the region of contact exactly counteracts the forces of gravity.

The gait that was proposed in this work is like the trotting gait, but with some modifications. In Figure 4-11, the dark areas indicate that a leg is in stance phase, while the light blue ones refer to the swing phase. After the swing phase of two diagonal legs, a phase where all legs are in stance is following and then the other two diagonal legs start their swing phase. In this way, if a swinging foot is late in contacting the ground, its diagonal leg will not have to support the whole weight of the robot on its own. Choosing properly the time intervals of each phase, at least two feet will be in contact with the ground simultaneously. The legs that are in swing phase shall reach the next foothold. On the contrary, the legs that are in stance phase shall ensure that the quadruped's body maintains the desired pose. Figure 4-13 shows that depending on the gait pattern and the contact with the ground, a different control method is chosen.

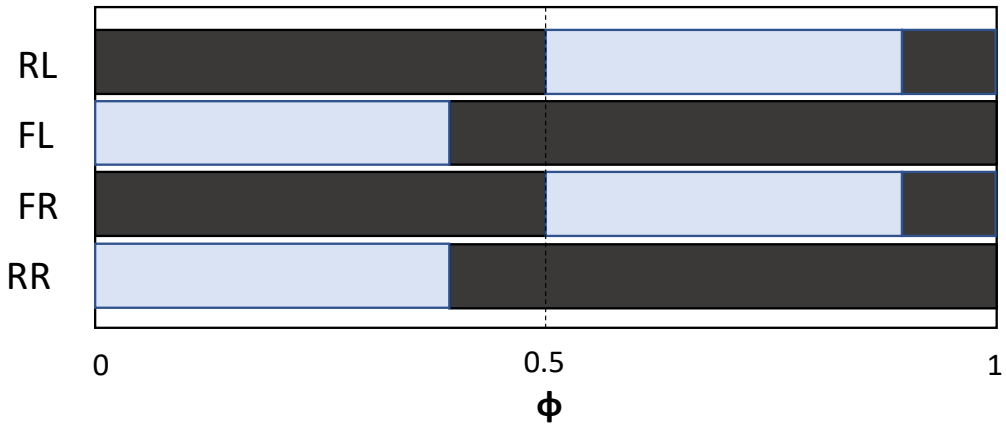


Figure 4-11. Gait pattern: the grey bar defines the stance phase of the rear left (RL), front left (FL), front right (FR), and rear right (RR) leg, respectively.

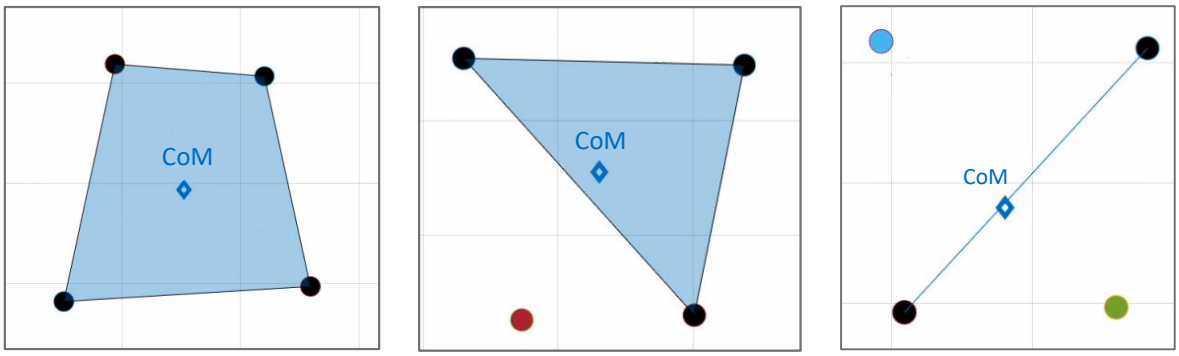


Figure 4-12. Support polygon/Support line that is generated from the feet that are in contact with the ground (black filled circles).

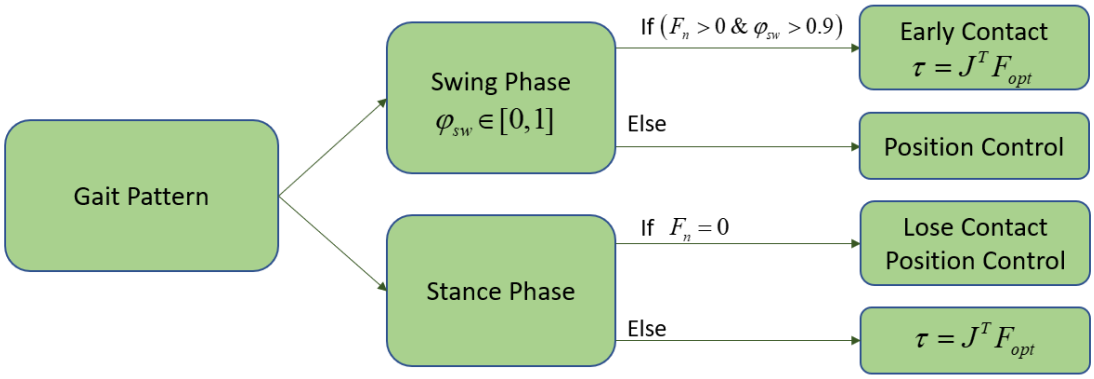


Figure 4-13. Controller selection for swing and stance phases.

4.3.2 Footstep planner

At every control cycle, a new foothold position in xz plane is calculated for each swing leg (4-13). The position of the foothold relative to the hip frame (CS_2) of each leg is computed by (4-14). Regarding the notation, the left superscript denotes in which coordinate system the vector is defined (H: Hip, see Figure 4-10), the right subscript refers to the position of the vector (e.g., *F* refers to foot), *ff* denotes a feed-forward term and *fb* a feedback term.

$$ {}^{H}\mathbf{r}_{F} = \begin{bmatrix} {}^{H}\mathbf{r}_{Fx} \\ {}^{H}\mathbf{r}_{Fz} \end{bmatrix} \tag{4-13} $$

$$ {}^{H}\mathbf{r}_{F} = {}^{H}\mathbf{r}_{F}^{ff} + {}^{H}\mathbf{r}_{F}^{fb} \tag{4-14} $$

$$ {}^{H}\mathbf{r}_{F}^{ff} = \frac{1}{2}\mathbf{v}^{d}\Delta t_{st} \tag{4-15} $$

$$ {}^{H}\mathbf{r}_{F}^{fb} = \eta(\mathbf{v} - \mathbf{v}^{d})\sqrt{\frac{\mathbf{y}_{hip}}{\mathbf{g}}} \tag{4-16} $$

The feedforward term takes into account the high-level command which is the desired velocity given by (4-17) and the stance duration Δt_{st}.

$$ \mathbf{v}^{d} = v_{x}^{d}\begin{bmatrix} 1 \\ 0 \end{bmatrix} + v_{z}^{d}\begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{4-17} $$

The feedback term is derived from the foothold prediction given by an inverted pendulum dynamic model. However, instead of incorporating the desired velocity, it includes the error between the desired and the actual velocity. The feedback term is treated as a corrective term that is used only when actual body velocity deviates from the desired command. The weight of this term is regulated through the scaling parameter η. It was set to 1.2, as it is in the original work, because it was found that other values in the same region did not improve the performance.

4.3.3 Foot trajectory planner

Having already defined the next step of the foot, the following paragraph presents the trajectory that the foot follows to reach the desired foothold. An elliptical trajectory is defined in the xy plane (Figure 4-14), which is described by (4-18)-(4-21).

$$ {}^{0}x^{d} = {}^{0}x_{c} + \alpha_{ellipse}\cos(\varphi) \tag{4-18} $$

$$ {}^{0}y^{d} = {}^{0}y_{c} - b_{ellipse}\sin(\varphi) \tag{4-19} $$

$$ \varphi = \omega_{traj} t + \Delta\varphi, \quad t \in [0, T] \tag{4-20} $$

$$ \omega_{traj} = \frac{2\pi}{T} \tag{4-21} $$

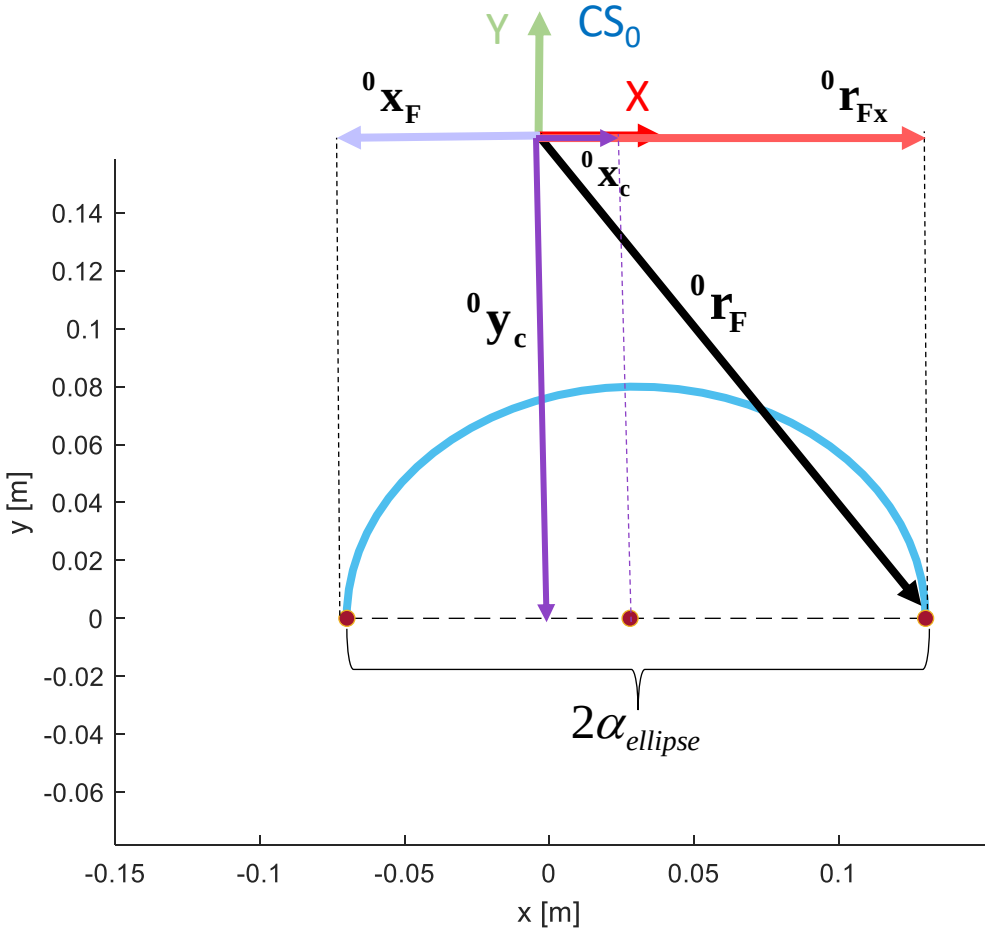


Figure 4-14. Elliptical trajectory. The y axis of this figure represents the y axis of the inertial frame.

The center of the ellipse is denoted by (x_c, y_c) and more specifically ${}^0y_c$ is the vertical vector from the CS_0 of each leg to the ground, $\alpha_{ellipse}$ is the semi-major axis and $b_{ellipse}$ is the semi-minor axis. $\Delta\varphi$ is set to π rads, $\alpha_{ellipse}$ and x_c are defined at the beginning of the aerial phase as:

$$\boldsymbol{\alpha}_{\mathbf{ellipse}} = \frac{{}^0\mathbf{r}_{\mathbf{Fx}} - {}^0\mathbf{x}_{\mathbf{F}}}{2} \tag{4-22}$$

$${}^0\mathbf{x}_{\mathbf{c}} = {}^0\mathbf{x}_{\mathbf{F}} + \boldsymbol{\alpha}_{\mathbf{ellipse}} \tag{4-23}$$

where ${}_0x_F$ is the foot position expressed at the CS_0 (see Figure 2-4).

Each leg has three degrees of freedom. Having already defined the next foothold in xz plane and the trajectories along x and y axes, a trajectory along z axis should be also defined. The desired position along the z axis and relative to the CS_0 is given in (4-25), which encompasses two terms. The first term is the vector from the CS_0 to the CS_2 and the second term is the foothold defined in (4-14) which is expressed relative to the hip frame (CS_2). The z component of the vector that relates the CS0 to the CS_2 is given in (4-24):

$${}^0\mathbf{z}_2 = l_3 \cos(th_3) \tag{4-24}$$

$${}^0\mathbf{z}^{\mathbf{d}} = \begin{Bmatrix} {}^0\mathbf{z}_2 + {}^H\mathbf{r}_{\mathbf{Fz}} & Right\ Leg \\ -{}^0\mathbf{z}_2 + {}^H\mathbf{r}_{\mathbf{Fz}} & Left\ Leg \end{Bmatrix} \tag{4-25}$$

Choosing a trapezoidal velocity profile (blue lines in Figure 4-15) is a way to determine the motion from an initial point to a final one. However, it is desired to keep the acceleration minimum. Given the total time t_f and the initial acceleration, the maximum velocity arises from (4-26), where t_b is the time spent until it reaches the maximum velocity.

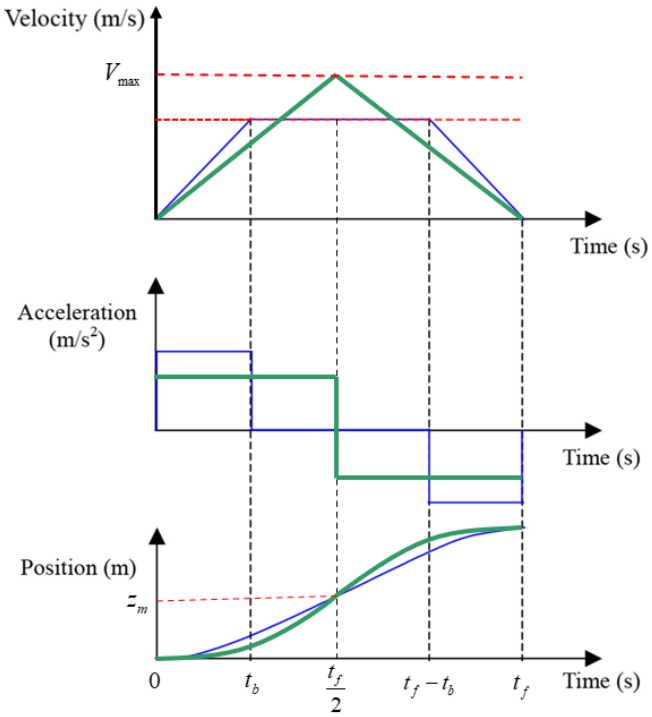


Figure 4-15. Acceleration and position profiles given a trapezoidal (blue) or triangular (green) velocity profile.

$$V_{\max} = \ddot{z}\, t_b = \frac{z_m - z_b}{\frac{t_f}{2} - t_b} \tag{4-26}$$

$$z_m = \frac{z_0 + z_f}{2} \tag{4-27}$$

$$z_b = \frac{1}{2}\ddot{z}\, t_b^2 \tag{4-28}$$

$$t_b = \frac{t_f}{2} - \frac{\sqrt{\ddot{z}^2 t_f^2 - 4\ddot{z}(z_f - z_0)}}{2\ddot{z}} \tag{4-29}$$

Equation (4-29) gives a valid solution for the time t_b if the inequality in (4-30) is satisfied.

$$\ddot{z} \geq \frac{4(z_f - z_0)}{t_f^2} = \ddot{z}_{\min} \tag{4-30}$$

Therefore, when t_b is equal to half the final time, the maximum acceleration is the minimum possible ($\ddot{z}_{\min}$), and the velocity profile is triangular (green lines in Figure 4-15). Equations (4-31)-(4-33) give the acceleration, the velocity and the position as a function of time.

$$\ddot{z} = \frac{4\left({}^0z^d - {}^0z^i\right)}{{T_{sw}}^2} \tag{4-31}$$

$$\dot{z} = \begin{Bmatrix} \ddot{z}t & t \leq t_b \\ \ddot{z}(T_{sw} - t) & t > t_b \end{Bmatrix} \tag{4-32}$$

$$z = \begin{Bmatrix} {}^0z^i + \frac{1}{2}\ddot{z}t^2 & t \leq t_b \\ {}^0z^d - \frac{1}{2}\ddot{z}\,(T_{sw} - t)^2 & t > t_b \end{Bmatrix} \tag{4-33}$$

where ${}^0z^i$ is the initial position of the foot relative to the CS_0, T_{sw} is the time duration of the swing phase of each leg and t_b is half of the T_{sw}.

4.3.4 Foot trajectory planning when contact is lost at stance phase

When the swing phase ends, the foot is supposed to have reached the desired foothold and to have contacted the ground. However, it is possible to delay arriving at the final position, which means that the leg will have passed to the stance phase according to the Gait pattern, but the foot will still be in the air. Therefore, in case the force sensor does not detect positive force values, it is considered that the foot is in the air, and it should regain contact with the ground instantly. To achieve this, the foot starting from its current position follows a line that is perpendicular to the ground at a constant speed until the force sensors detect ground reaction forces.

4.3.5 Pose finder

The pose of the main body shall be controlled in a way to prevent the quadruped from falling

and at the same time to make it move smoothly with the desired forward velocity. The robot mostly leans on the legs that has just started their stance phase, and it starts to move away from them when their swing phase is about to start. To achieve this, the desired position of the body ${}^{\mathbf{I}}\mathbf{r}_{\mathbf{B}}^{\mathbf{d}}$ in the xz plane is computed relative to the positions of the legs ${}^{\mathbf{I}}\mathbf{r}_{\mathbf{F_i}}$ multiplied by weights $w_i(\varphi)$ (4-34), [23]. The left superscript denotes in which coordinate system the vector is defined (I : Inertial Frame, see Figure 4-10).

$$ {}^{\mathbf{I}}\mathbf{r}_{\mathbf{B}}^{\mathbf{d}} = \frac{\sum_{i=1}^{i=4} w_i(\varphi)\,{}^{\mathbf{I}}\mathbf{r}_{\mathbf{F_i}}}{\sum_{i=1}^{i=4} w_i(\varphi)} \tag{4-34} $$

The leg weights w_i were found experimentally in [23], so their values vary according to the gait that was chosen. These are expressed as a function of the stride phase φ. For the trotting gait, w_i is given in (4-35) and its value is maximum at the start of the stance phase in which the foot has just landed and can support the body weight. When the foot is about to swing, this value declines linearly. At the beginning of the swing phase, w_i is constant, but when the foot prepares to touch down, the weight increments linearly until it reaches its maximum limit again.

$$ w_i(\varphi) = \begin{Bmatrix} 1 & 0 \le \varphi_{st} \le 0.7 \\ \dfrac{0.895 - 0.85\varphi_{st}}{0.3} & 0.7 < \varphi_{st} \le 1 \\ 0.15 & 0 \le \varphi_{sw} \le 0.7 \\ \dfrac{-0.55 + 0.85\varphi_{sw}}{0.3} & 0.7 < \varphi_{sw} \le 1 \end{Bmatrix} \tag{4-35} $$

The desired height of the body relative to the ground is constant and equal to the torso's height at the nominal position. The generalized desired position and velocity of the main body are given as:

$$ \mathbf{q}_{\mathbf{b}}^{\mathbf{d}} = [\,{}^{\mathbf{I}}\mathbf{r}_{\mathbf{B_x}}^{\mathbf{d}} \quad h_{CoM} \quad {}^{\mathbf{I}}\mathbf{r}_{\mathbf{B_z}}^{\mathbf{d}} \quad 0 \quad 0 \quad 0]^T \tag{4-36} $$

$$ \dot{\mathbf{q}}_{\mathbf{b}}^{\mathbf{d}} = [\,{}_{\mathbf{I}}\mathbf{v}_{\mathbf{x}}^{\mathbf{d}} \quad 0 \quad {}_{\mathbf{I}}\mathbf{v}_{\mathbf{z}}^{\mathbf{d}} \quad 0 \quad \dot{\boldsymbol{\psi}}^{\mathbf{d}} \quad 0]^T \tag{4-37} $$

4.3.6 Control selection

To achieve the desired motions that were described above, a hybrid control approach is implemented. When a leg is in the air, the foot's desired motion is tracked by using position control. On the other side, the forces and torques that are generated from the legs that are in stance mode can affect the main body's pose in a controlled manner to track the desired body positions. An optimization problem, which finds virtual forces that should act on each of the stance legs, is formulated and then these forces are mapped to joint torques.

4.3.7 Swing Phase Position Control

At swing phase, the desired joint angles $\mathbf{q}_j^d$ are obtained from the desired foot positions through inverse kinematics (see Section 2.2.2). Then, implementing a PD controller results in finding the torques needed to track the desired motion of the leg (4-38). The PD gains are listed in Table 4-2 and were found by trial and error.

$$\mathbf{u} = \mathbf{K_P}(\mathbf{q_j^d} - \mathbf{q_j}) + \mathbf{K_D}(\dot{\mathbf{q}}_\mathbf{j}^\mathbf{d} - \dot{\mathbf{q}}_\mathbf{j}) \tag{4-38}$$

Table 4-2. PD Controller Gains.

Gains	Hip	Knee	Abduction/Adduction
K_P	100	80	1000
K_D	10	8	90

4.3.8 Control at stance phase

The joint torques that need to be applied through the stance legs are investigated in this section, which is divided in three parts. Firstly, the forces and torques that should ideally act on the main body to force it to the desired pose, are calculated. These are optimally distributed to the stance legs. Finally, these virtual foot forces are mapped to joint torques. The desired forces and torques on the main body are given by (4-39):

$$\begin{bmatrix} {}_B\mathbf{F}_B^d \\ {}_B\mathbf{T}_B^d \end{bmatrix} = \mathbf{K}_p(\mathbf{q}_b^d - \mathbf{q}_b) + \mathbf{K}_d(\dot{\mathbf{q}}_b^d - \dot{\mathbf{q}}_b) + \mathbf{K}_{ff} \begin{pmatrix} {}_I v_x^d \\ mg \\ {}_I v_z^d \\ 0 \\ \dot{\psi}^d \\ 0 \end{pmatrix} \tag{4-39}$$

where $\mathbf{K}_p$, $\mathbf{K}_d$ and $\mathbf{K}_{ff}$ are the proportional, derivative and feed-forward gains respectively and m is the total mass of the robot. This equation is obtained by [23] and describes the required forces and torques on the main body, so that the actual pose matches the desired. The feed-forward gains improve tracking the desired velocities and compensate for gravity.

The desirable contact and friction forces $\mathbf{x}$ that should be applied at each stance foot arises from the solution of a convex optimization problem with linear constraints that is described below (4-40).

$$\begin{aligned} &minimize \quad (\mathbf{Ax}-\mathbf{b})^T\mathbf{S}(\mathbf{Ax}-\mathbf{b}) + \mathbf{x}^T\mathbf{Wx} \\ &subject\,to \quad F_{\min}^n \le F_{c,i}^n \le F_{\max}^n \\ &\qquad\qquad -\mu F_{c,i}^n \le F_{c,i}^t \le \mu F_{c,i}^n \end{aligned} \tag{4-40}$$

The vector $\mathbf{x}$ is given in (4-41), where the superscripts t_x and t_z denote the tangential forces along the x and z axis respectively and n denotes the normal forces. The subscripts $1,\ldots,4$ refer to each of the legs and the constraints in (4-40) bound the normal forces between a minimum and a maximum limit. The second equation that should be satisfied is the friction cone constraint which ensures that the foot does not slip. The magnitude of the tangential forces must be smaller than the normal force multiplied by the friction coefficient $\mu = 0.8$.

$$\mathbf{x} = \begin{bmatrix} F_{c,1}^{t_x} & F_{c,1}^n & F_{c,1}^{t_z} & \cdots & F_{c,4}^{t_x} & F_{c,4}^n & F_{c,4}^{t_z} \end{bmatrix}^{\mathrm{T}} \in \mathbb{R}^{12} \tag{4-41}$$

The first term of the objective function in (4-40) is used to minimize the deviation from the desired forces and torques that should be applied at the main body in the least-squares sense.

The matrix $\mathbf{A}$ (4-42) maps the foot forces on the main body (Figure 4-16), while the vector $\mathbf{b}$ (4-43) has already been calculated in (4-39).

$$\mathbf{A} = \begin{bmatrix} \mathbf{I} & \mathbf{I} & \cdots & \mathbf{I} \\ \mathbf{r}_0 \times & \mathbf{r}_1 \times & \cdots & \mathbf{r}_m \times \end{bmatrix} \in \mathbb{R}^{12\times 12} \tag{4-42}$$

$$\mathbf{b} = \begin{bmatrix} {}_B\mathbf{F}_B^d \\ {}_B\mathbf{T}_B^d \end{bmatrix} \in \mathbb{R}^{12\times 1} \tag{4-43}$$

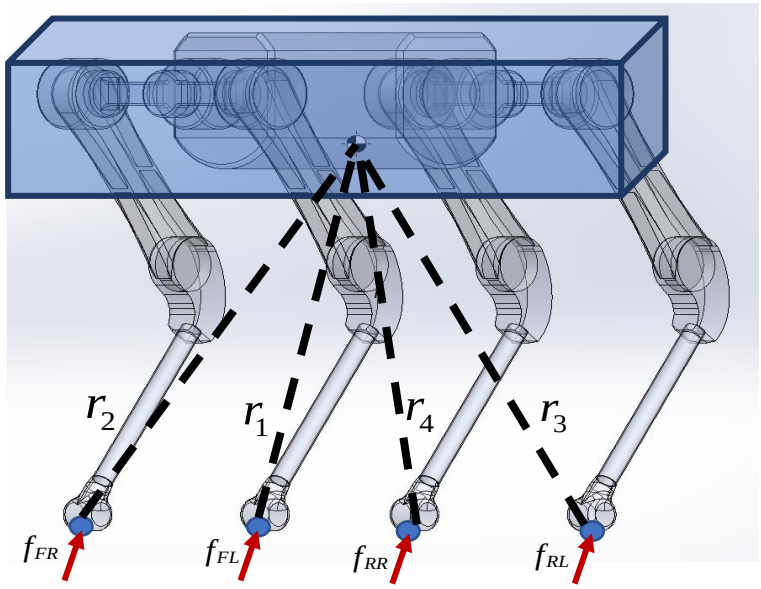


Figure 4-16. Illustration of the foot forces mapping at the CoM.

The weight matrix $\mathbf{S}$ adjusts the degree to which each of the actual variables matches the desired one. On the other hand, the weighting matrix $\mathbf{W}$, which is applied at the optimization variables, penalizes large output values at the leg forces. The optimization variables are bounded, not only to ensure that the virtual ground forces are always positive, but also that they are not excessively large. Assuming that the maximum force, that a leg can stand, is three times the total weight of the robot, this value is defined as the upper bound and the lower bound is chosen equal to $\mathrm{F}_{\min}^{\mathrm{n}} = 2\ \mathrm{N}$, because according to [23], this limitation makes the foot slip less times.

The general formulation of (2-34)-(2-39), through which the contact forces are mapped to joint torques, can be written in the form (4-44). The vector $\mathbf{u}_i \in \mathbb{R}^{3\times 1}$ encompasses the hip, knee and ab/ad torque for the leg i, $\mathbf{G}(\mathbf{q}) \in \mathbb{R}^{3\times 1}$ are the gravitational terms, $\mathbf{J}^T \in \mathbb{R}^{3\times 3}$ is the transpose geometrical Jacobian (see Appendix B) and $\mathbf{F}_{c,i} \in \mathbb{R}^{3\times 1}$ are the optimized contact forces. The foot is in stance, thus the inertial forces are zero.

$$\mathbf{u_i} = \mathbf{G}(\mathbf{q}) - \mathbf{J}^{\mathrm{T}}\mathbf{F_{c,i}} \tag{4-44}$$

The vector $\mathbf{G}(\mathbf{q}) \in \mathbb{R}^{3\times 1}$ arises from (4-45) where U is the leg's potential energy given by (4-46) and q_i are the joint angles.

$$\mathbf{G} = \begin{bmatrix} \dfrac{\partial U}{\partial q_1} & \dfrac{\partial U}{\partial q_2} & \dfrac{\partial U}{\partial q_3} \end{bmatrix}^{\mathrm{T}} \tag{4-45}$$

$$U = m_1 g\, y_1 + m_2 g\, y_2 + m_3 g\, y_3 \tag{4-46}$$

The term y_j, $j=1,2,3$ in (4-47) expresses the position of the center of mass of link j along the y axis relative to the CS_0.

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} l_3 \sin(q_3) - l_{1c} \cos(q_1)\cos(q_3) \\ l_3 \sin(q_3) - (l_1 \cos(q_1) + l_{2c} \cos(q_1 + q_2))\cos(q_3) \\ l_{3c} \sin(q_3) \end{bmatrix} \tag{4-47}$$

After calculating the partial derivatives of (4-45), the vector $\mathbf{G(q)}$ is written in the form (4-48), where l_{1c} is the distance from the leg upper segment's CoM to the hip joint, l_{2c} is the distance from the leg lower segment's CoM to the knee joint and l_{3c} is the distance from the leg roll segment's CoM to the ab/ad joint.

$$\mathbf{G(q)} = \begin{bmatrix} \left(m_1 g l_{1c} \sin(q_1) + m_2 g \left(l_1 \sin(q_1) + l_{2c} \sin(q_1+q_2)\right)\right)\cos(q_3) \\ m_2 g l_{2c} \sin(q_1+q_2)\cos(q_3) \\ \left(m_1 g l_{1c} \cos(q_1) + m_2 g \left(l_1 \cos(q_1) + l_{2c} \cos(q_1+q_2)\right)\right)\sin(q_3) + g\left((m_1+m_2) l_3 + m_3 l_{3c}\right)\cos(q_3) \end{bmatrix} \tag{4-48}$$

4.3.9 Optimization Solver

The optimization problem (4-40) is formulated using the external library CasADi (see Appendix E). The choice of a solver is of great importance to achieve a good solution in minimum time. An initial guess is not necessary, since the optimization problem is convex. The nonlinear programming solvers, presented in Section 3.5 can handle a nonlinear objective function and nonlinear constraints. In this problem, the objective function is in quadratic form and the constraints are linear, hence it is considered as a quadratic programming problem, which is one of the simplest forms of non-linear programming. There is no need to use a NLP solver in this class of problems. Supported solvers for the quadratic problem are qpOASES [20], OSQP (Operator Splitting Quadratic Program) [57], OOQP [24] and CPLEX [16]. The first one, which is an active-set QP solver, is widely used and the simulations at the current problem shoed that it outputs acceptable solutions in one or two iterations. OSQP was also tested but failed to converge in some simulation runs. Thus, qpOASES was chosen, as it is fast and reliable.

4.3.10 Results

The control framework was verified in MATLAB Simscape[4]. The simulation environment and the solver settings are described in Section 2.5. The structure of the Simscape model is described in Appendix E. The quadruped is walking forward using the trotting gait and its speed in x direction ranges between $0.2\,m/s - 0.9\,m/s$. It is not recommended to set the speed above the upper limit, because the distance between the current foothold to the next one (given by (4-15)) increases significantly. Thus, the leg would have to move faster or extend excessively, which may lead to singularities. However, if the frequency is increased, the quadruped's speed can also be increased without affecting the stride length, but then the torques increase. Simulations have been performed for the entire velocity range, but then the graphs that are presented below stand for the case in which $v_x^d = 0.4\,m/s$. The illustration of the quadruped's motion objectives at a certain speed gives all the essential data to assess its performance, provided that different forward velocity commands mostly affect the stride length

[4] The MATLAB codes can be found here:
https://bitbucket.org/csl_legged/argos_optimal_force_distribution/src/master/

and the rest quantities do not differ significantly. The other high-level parameters were set as $v_z^d = 0\, m/s$ and $\dot{\psi}^d = 0\, rad/s$. The total stride duration is set equal to $T = 0.8\, s$ and the swing phase duration of each leg is set as $T_{sw} = 0.3\, s$.

Simulations have revealed that high gait frequencies lead to robust locomotion, thus a commonly used value for the stride duration was chosen. Another parameter that was tuned for the swing leg motion is the semi-minor axis of the ellipse, for which a typical value was chosen, i.e., $b_{ellipse} = 0.08\, m$. The initial conditions for the joint angles are shown in Table 4-3 and the controller's gains that were found experimentally are presented in Table 4-2. Table 4-4 presents the parameters that were tuned in order to achieve good performance. Especially, the feedforward gain related to v_x^d needs a high value when the velocity is low and a smaller value as the speed increases. The gains have been tuned for the case of $v_x^d = 0.35\, m/s$, and the expression that adapts the gain value for the different values of v_x^d is given by (4-49). This section includes the settings and results regarding the optimization and after that, graphs for the body pose, toes' trajectories, forces and joint torques are presented.

$$k_{ffx} = 90\left(1 - \frac{v_x^d - 0.35}{v_x^d}\right) \tag{4-49}$$

The foot-ground interaction model parameters are described in Section 2.5.2.

Table 4-3. Initial Joint angles.

Hip	$th1_0$	-36 º
Knee	$th2_0$	72 º
Abduction/Adduction	$th3_0$	0

Table 4-4. Gains and weights for the optimization problem.

Parameter	Symbol	Value
virt. force proportional gain	$\boldsymbol{k}_p$	$diag\left(\begin{bmatrix}0 & 600 & 350 & 400 & 400 & 400\end{bmatrix}\right)$
virt. force derivative gain	$\boldsymbol{k}_d$	$diag\left(\begin{bmatrix}150 & 120 & 100 & 40 & 20 & 40\end{bmatrix}\right)$
virt. force feed-forward gain	$\boldsymbol{k}_{ff}$	$diag\left(\begin{bmatrix}k_{ffx} & 1 & 0 & 0 & 0 & 0\end{bmatrix}\right)$
weights for matching the des. virt. forces	$\boldsymbol{S}$	$\begin{bmatrix}1 & 1 & 1 & 20 & 5 & 20\end{bmatrix}$
weights for reducing joint torques	$\boldsymbol{W}$	$\begin{bmatrix}0.00001 & \dots & 0.00001\end{bmatrix} \in \mathbb{R}^{1\times 12}$

Optimization

The solving time of the optimization problem ranges between 1-4ms and the average time is 2ms.The simulation was executed on a 2-core Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz computer.

Body Pose

Figure 4-17 displays the position of the center of mass of the robot body. The desired velocity is constant, so a linear displacement with time is expected in x direction. The height of the

robot in steady state is greater than that given as initial condition. This has the advantage that when the legs are extended, the knee torques will be relatively small. However, if the legs are stretched excessively, this will lead to a singularity. The orientation of the quadruped has a slight deviation from the desired values which ensures that the robot will not tumble over (Figure 4-18).

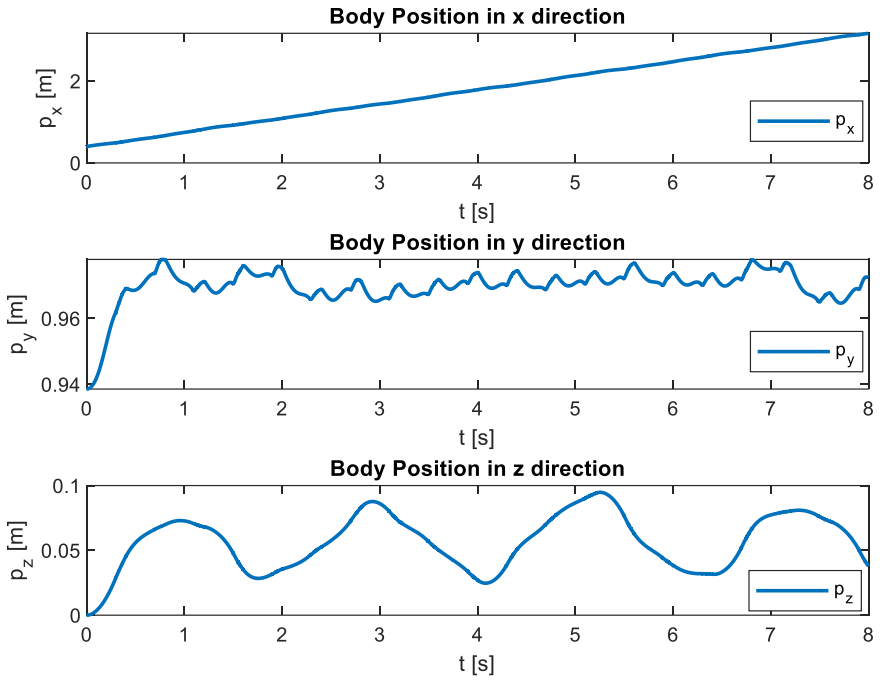


Figure 4-17. CoM's position

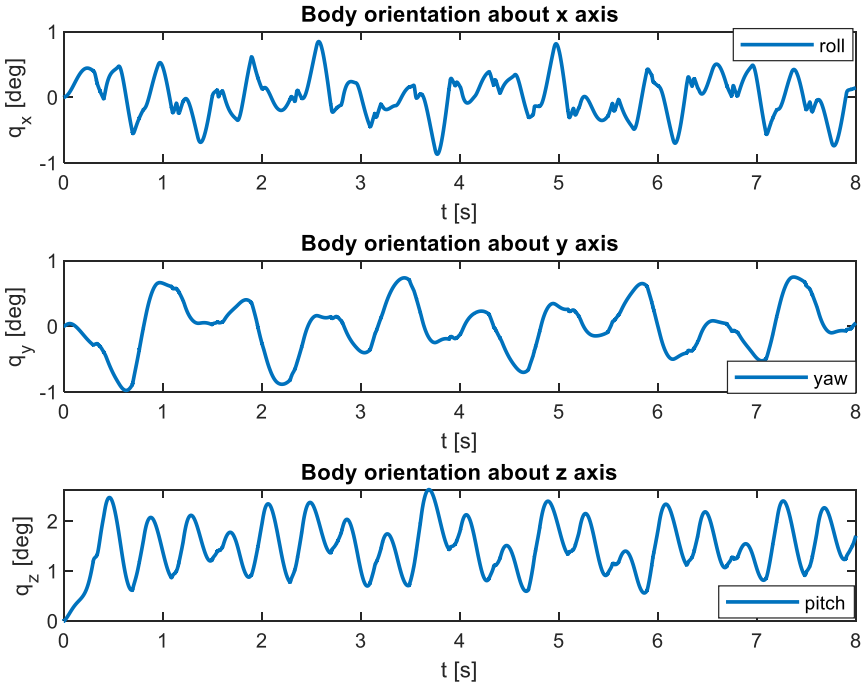


Figure 4-18. CoM's orientation.

Figure 4-20 depicts the robot's speed in four different cases to demonstrate the controller's robustness. The quadruped cannot move continuously at the desired forward velocity because after a swinging phase of two diagonal legs, a short phase with all the feet on ground has been introduced. This happens to ensure that the robot is supported by at least two legs even if a swinging leg delay making contact with the ground. However, Table 4-5 attests that the mean velocity is almost equal to the desired. Figure 4-19 portrays a snapshot from an animation that was created in MATLAB to illustrate the body's position in xz-plane (top view) and the support polygon created by the legs that are in stance phase.

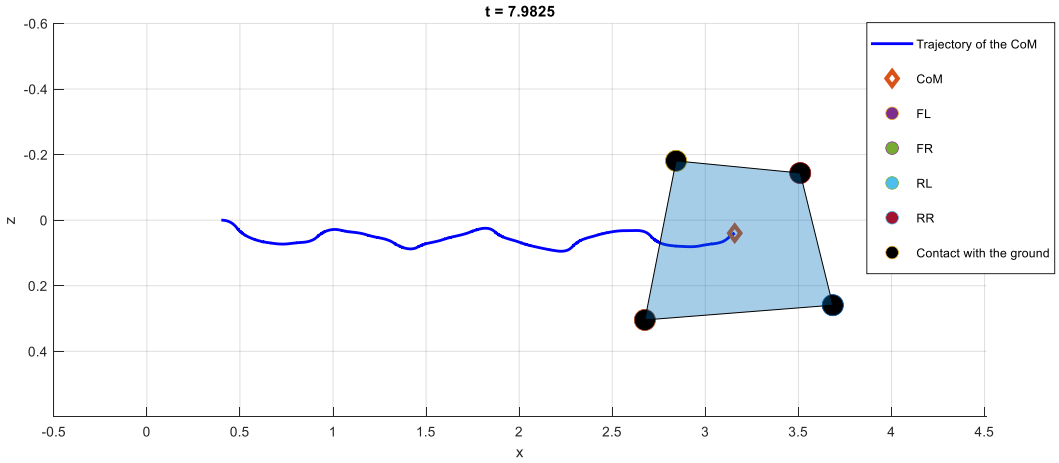


Figure 4-19. CoM's trajectory in xz plane illustrated with solid blue line. The black circles are the footholds in stance phase and the polygon that connects them is the support polygon. In this snapshot, all the legs are in stance.

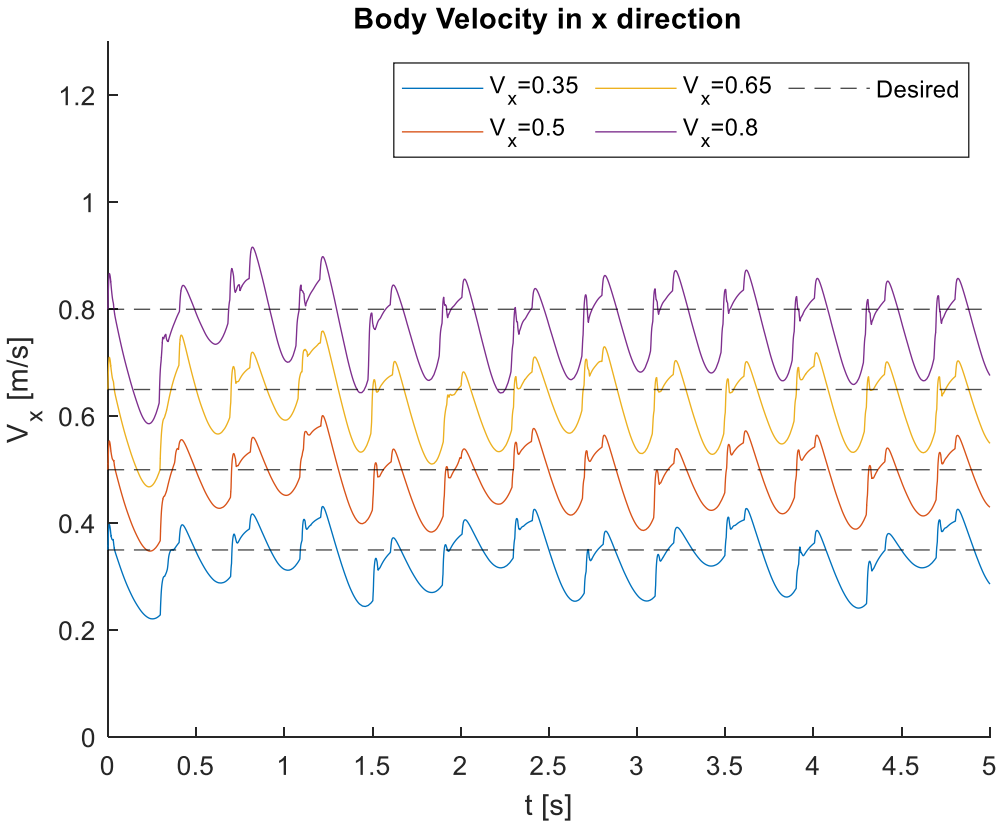


Figure 4-20. CoM's forward velocity. The black dashed lines are the desired velocity commands.

Table 4-5. The average forward velocity that the robot achieves given the desired command.

v_x^d	0.2	0.35	0.5	0.65	0.8	0.9
$\bar{v}_x$	0.1945	0.348	0.4986	0.6499	0.7873	0.8734

Toe Trajectory

Figure 4-21 illustrates in blue color the desired toe trajectory of the front left leg during the swing phase and the actual trajectory is portrayed in orange color. The purple vertical line is the desired trajectory of the foot when it is in stance phase but loses contact with the ground. The green line depicts the trajectory that it actually follows to regain contact. The deviation between the actual trajectories and the desired ones does not have a negative effect on the robot's locomotion, since the foot's motion is still smooth, and the quadruped achieves to step on the desired footholds. The foot trajectories of the other three legs are similar to that presented in Figure 4-21, so they can be omitted.

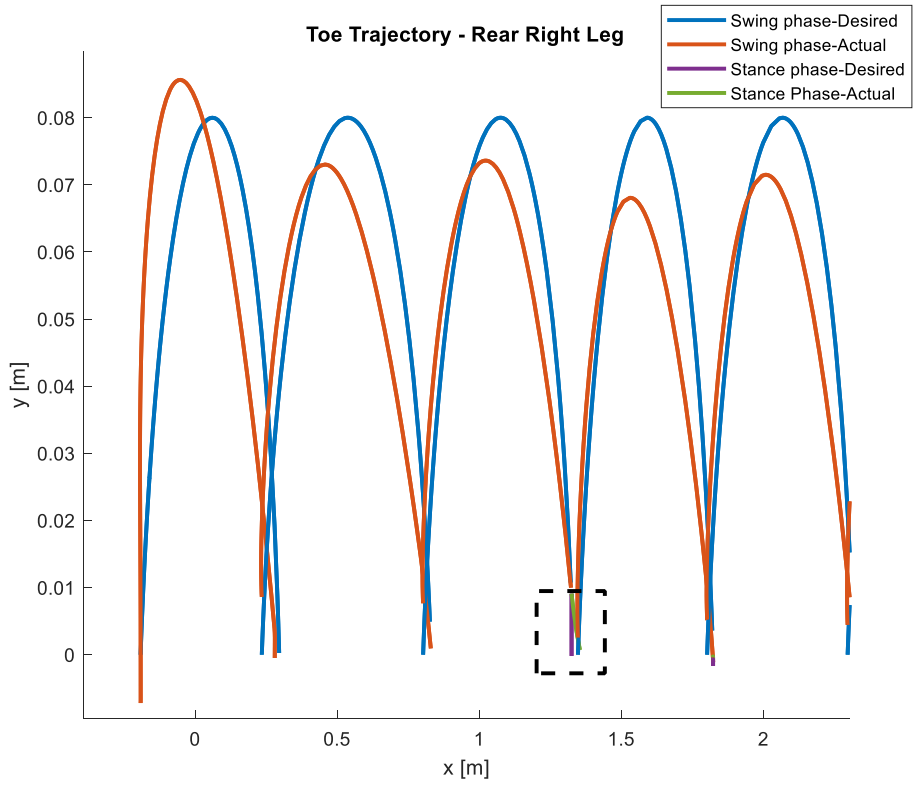


Figure 4-21. Toe's trajectory – Rear Right Leg. The detail represents the trajectory that the foot follows to regain contact with the ground.

Joint Angles and Velocities

The following figures (Figure 4-22 - Figure 4-27) represent the joint angles and joint velocities of the front left leg. The foot follows an elliptical trajectory in the xy-plane. On the other hand, a second order polynomial resulting from the triangular velocity profile is the desired command in the z direction. The blue lines represent the desired motion given during swing phase. The controller tracks the desired joint angles and angular velocities satisfyingly, but further gain tuning may lead to even better tracking.

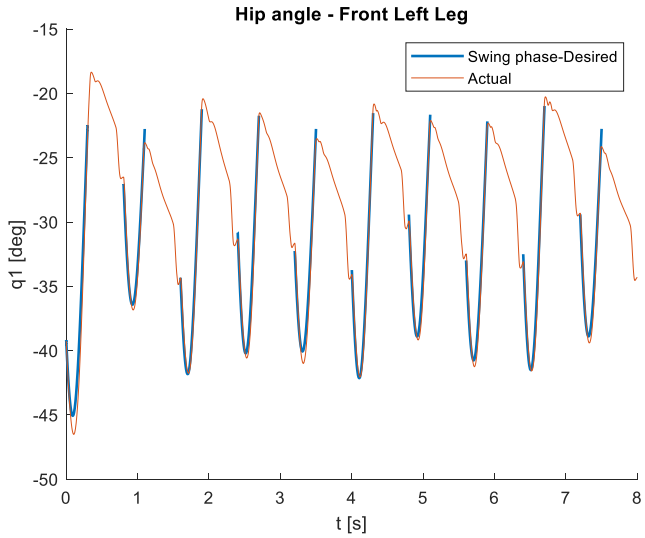


Figure 4-22. Desired and actual hip joint angle of the front left leg.

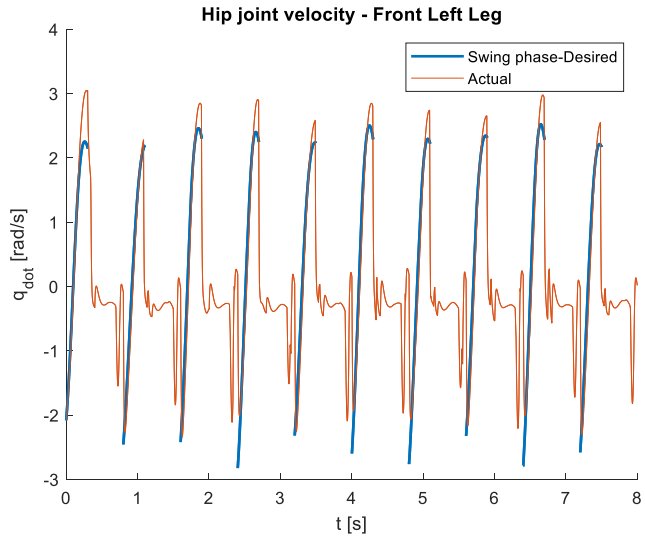


Figure 4-23. Desired and actual hip joint angular velocity of the front left leg.

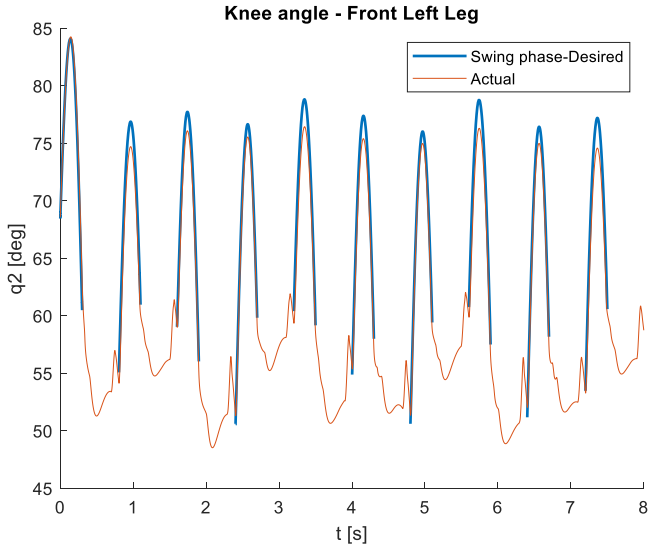


Figure 4-24. Desired and actual knee joint angle of the front left leg.

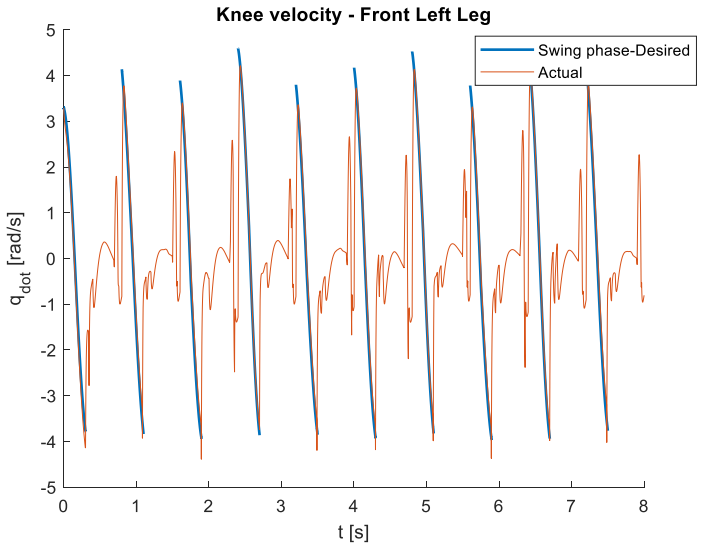


Figure 4-25. Desired and actual knee joint angular velocity of the front left leg.

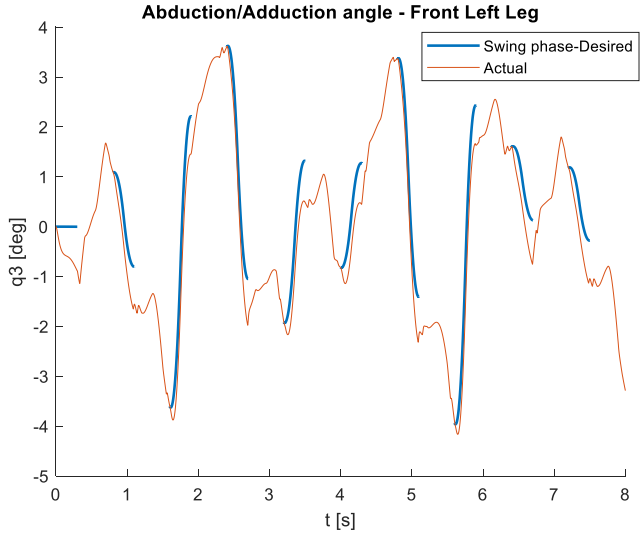


Figure 4-26. Desired and actual abduction joint angle of the front left leg.

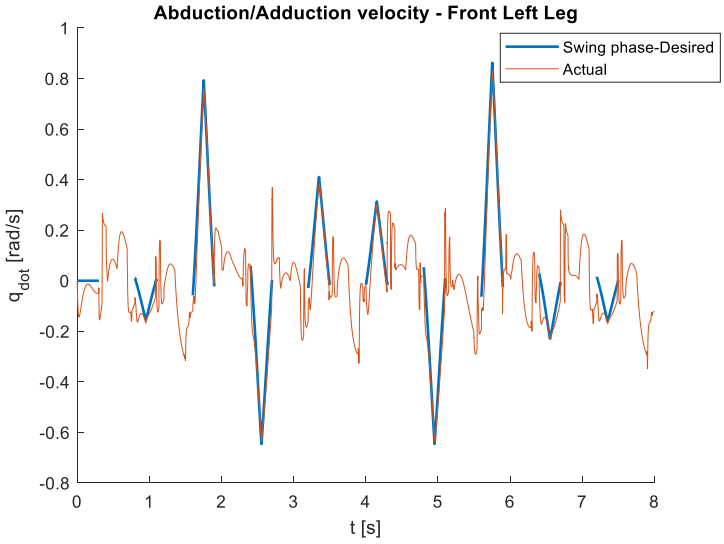


Figure 4-27. Desired and actual abduction joint angular velocity of the front left leg.

Virtual Optimized Forces – Actual contact forces

Figure 4-28 shows the front left leg's virtual forces that result from the solution of the optimization problem and the actual ones that are computed by the force sensors according to the foot-ground contact model. The spikes happen when the foot contacts the ground. If the ground stiffness is reduced, the surface will become softer, and the force change will not be so abrupt. Therefore, the maximum force values depend on the terrain and the toe material. Then, Figure 4-29 and Figure 4-30 show the resulting force and torques on the main body. The weighting gains that are listed in Table 4-4 affect the tracking of the desired forces and torques and the higher values assigned at the X-axis and Z-axis torque mean that these quantities need better tracking. The forces and the torques that are applied on the center of mass are calculated through mapping of the leg forces to the CoM. As a result, abrupt changes at the leg forces when the foot contacts the ground cause the spikes at the Figure 4-29 and Figure 4-30.

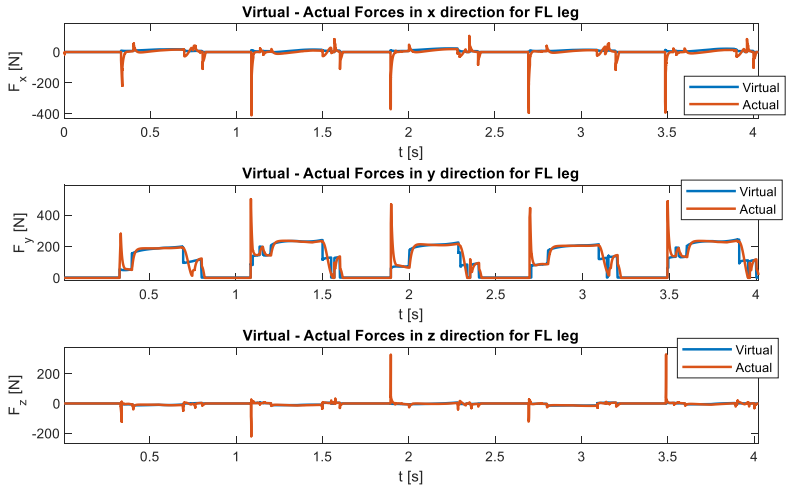


Figure 4-28. Virtual-Actual Forces for Front Left Leg.

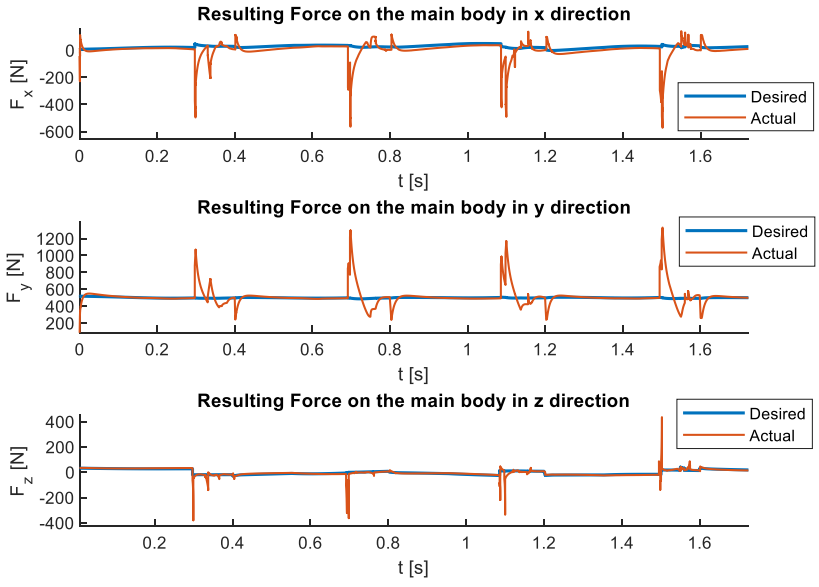


Figure 4-29. Resulting force on the main body.

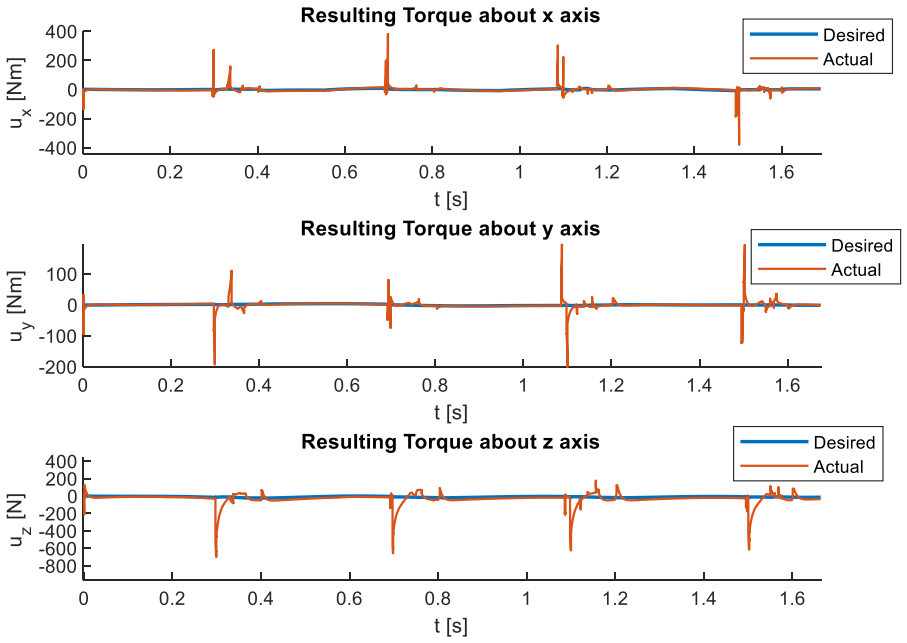


Figure 4-30. Resulting torque on the main body.

Joint Torques

The figures below show the torque required at the hip, knee, and abduction/adduction joints respectively. The torques at the rear legs are larger than those of the front legs. Also, the highest torque values appear at the knee joint because of the long lever arm.

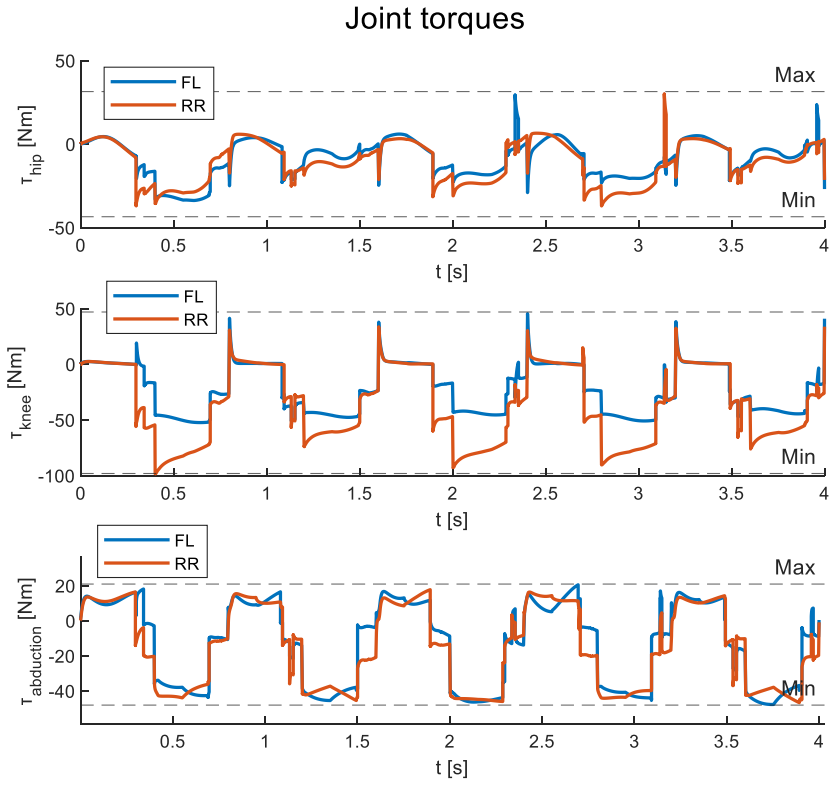


Figure 4-31. Hip, Knee and Abduction torques of the diagonal legs Front Left and Rear Right.

4.3.11 Conclusion

The framework that was investigated can be used for a wide range of speeds. Both the formulation of the planning and the control has a lot of tunable parameters. Consequently, the parameter choice becomes a complex task. However, the conclusion that was drawn in [23], according to which a large range of parameters results in successful motions, is validated in this work. A change in one of the parameters cannot be easily related to a specific effect on the motion objectives. Thus, the tuning of the parameters has been based on the values used in [23], but the adjustments were made intuitively. Finally, the generation of other gaits will require different tuning of parameters.

The mean forward velocity is almost identical to the desired, thus the high-level velocity commands are tracked very well. The orientation of the robot does not deviate significantly from the nominal position, so it remains stable. The slight lateral drift that is presented in [23], also appears in this simulation, but it does not affect the whole motion. The spikes that are shown in the force graphs are caused by the impact with the ground and can be eliminated if the shocks are absorbed by adding high damping feet elements, as it is done in [23]. Also, StarlETH is able to either damp out a certain amount of energy or to store it as potential energy in the actuator springs. The passive mechanical compliance in the actuators, as well as the high damping feet elements protect the system from the impact at landing [30]. The required joint torques are twice of those needed in [23], but this is expected, provided that the weight of StarlETH is less than half the weight of Argos. In summary, the optimal force distribution method improved quadruped's locomotion by controlling indirectly the pose of the robot's body, but further experiments, where external disturbances are acting on the body can be performed to verify the controller's robustness. It is also worth identifying in a future work if the recovery process from an external disturbance leads to excessively large joint torques. Another advantage of this framework is its modularity, meaning that each of the small subsystems can be replaced by other algorithms, while the rest are kept as they are.

4.4 Application of Trajectory Optimization in Argos

4.4.1 Introduction

In this section, trajectory optimization is applied in the quadruped Argos. In general, a high-level task is translated into a desired motion plan for the legged robot, while satisfying the physical constraints. Trajectory optimization generates optimal plans, which drive the system to the final goal, given the initial state of the system and an estimate of the total time needed. The total time horizon is discretized in a number of grid points, each of which shall satisfy the according physical constraints. This work is based on [63], in which trajectory optimization is used for the generation of complex motions in several legged systems.

4.4.2 Approaches in legged systems

A walking robot is a dynamical system which goes through multiple phases of continuous dynamics, separated by discrete transitions [37]. The underlying optimization algorithm is typically a smooth, gradient-based method. Common nonlinear programming solvers such as SNOPT [25], [26], IPOPT [6], [60], [61], [8], FMINCON [44] rely on gradient based methods. The discrete transitions of such hybrid systems results in non-smooth dynamics. There are a few techniques to handle the discontinuous dynamics such as phase-based optimization,

through-contact optimization, techniques with slack variables and constraints and smoothing. The choice of the appropriate method depends on the problem, but in the context of this work, it is worth investigating the first two methods [37].

If the discontinuity is simple, such as an absolute value sign in the objective function, then there are many clever tricks with slack variables and constraints. If there is a simple non-linearity, then every instance of the discontinuous function can be replaced with a smooth approximation. In [7], a smooth contact model which provides good gradients of the dynamics is used. This tends to be simple and fast, but less accurate than the slack variable method. There are two types of smoothing: exponential and polynomial. If polynomial smoothing is used, it has to be ensured that it is smooth to second order at least.

Phase-based optimization

If there is a small number of well-understood discontinuities, such as foot collisions with the ground during walking, that occur in a known sequence, then the best option is to pose a multi-phase trajectory optimization problem. This method has been applied for the quadruped robot ANYmal [4]. Multi-phase methods implement a basic multiple shooting or collocation algorithm in each phase, but additional constraints ensure that the phases are linked together. First, the sequence of phases is defined and then standard constraints at the continuous phase and extra constraints at the transition are set up. They are faster and more accurate than through-contact optimization, but they require explicit knowledge of the sequence of transitions. A software specialized for this type of problem is GPOPS-II [5].

Through-contact optimization

Through-contact (contact invariant) optimization is best for systems with more complicated contact models and/or discontinuities, where the sequence of discontinuities is not known. The principle is to include some special constraints and extra decision variables to force all of the discontinuities into constraints, leaving the resulting NLP smooth. The system dynamics includes the contact impulses which are arbitrary. Compared to the multi-phase method, where the constraints at each grid point are known a priori, in this formulation, contact constraints are directly added to the required grid points. These impulses are then treated as a control variable, which is subject to constraints. If the foot is in the air, these constraints ensure that there is a gap between the foot and the ground. Otherwise, the foot forces are non-zero. These constraints form what is known as a linear complementarity problem (LCP). This method can deal with arbitrary sequences of contacts, but it is slower and less accurate.

4.4.3 Optimization problem structure

The problem is formulated according to the phase-based approach. An overview of the optimization problem is depicted in Figure 4-32. The inputs are the initial and final state of the system, the number of steps of each leg $n_{s,i}$, the total duration T, the durations of the swing and the stance phases $\Delta T_{sw}, \Delta T_{st}$ and subsequently the timings at which the feet make contact with the ground. A major difference from [63] is that the optimization problem is simplified by predefining the type of gait; it is not generated by the algorithm. The parameter selection concerns the trotting gait, thus if a different gait is chosen, the problem should be set up accordingly. The outputs are the base position $r(t) \in \mathbb{R}^3$ and orientation $\theta(t) \in \mathbb{R}^3$ (parameterized by Euler angles), the feet positions p_i and the feet forces f_i. The pose of the robot (base position and orientation) is described by cubic polynomials merged at the boundary points, so that a continuous spline is created. On the other hand, the foot position

is parameterized using one quartic polynomial at swing phase and a constant value at stance phase. For the foot force, three quadratic polynomials describe the stance phase, while at swing phase, the force is set to zero. Both the states (base pose) and the controls (feet positions and forces) are optimization variables; thus, a simultaneous method is pertinent for solving such problems. The optimization problem is formulated according to the Hermite-Simpson direct collocation because according to the comparison made in Section 3.6, it provides satisfying accuracy and it is much faster compared to multiple shooting. Regarding the optimization software, CasADi was chosen as it turned out to be far more efficient than OptimTraj.

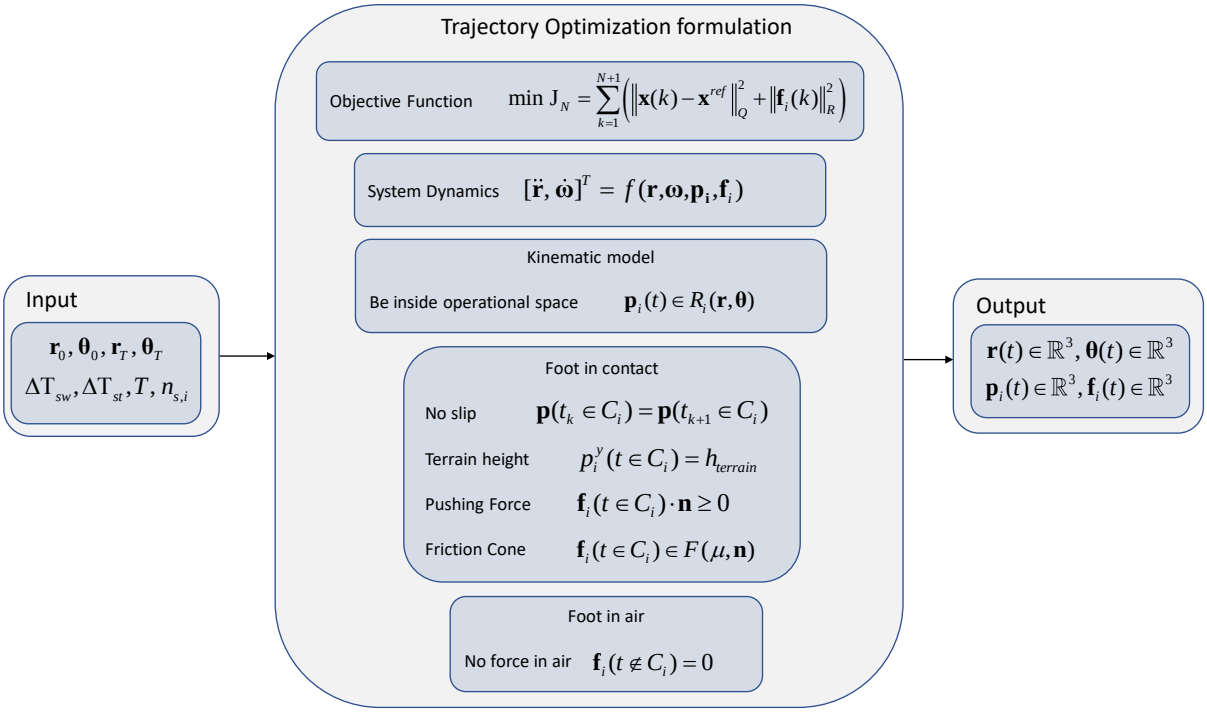


Figure 4-32. Trajectory optimization formulation.

To speed up the computations the optimization problem that was formulated in [63] does not include an objective function, but in this work it does. Problems without an objective function are known as *Constraint Satisfaction Problems*. In contrast, problems that include an optimization criterion are known as *Constraint Optimization Problems*. Not all optimization solvers require the specification of an objective function. Even if they do, it is possible to specify some dummy objective. However, it is important to note that introducing a dummy variable is a bad practice, unless all optimization variables are bounded.

4.4.4 Objective function

The goal of this implementation is the generation of feasible movements to reach a desired position in Cartesian space. The optimal trajectory is determined by the criteria that are set via the objective function $\mathbf{J}_N$.

$$\text{minimize } \mathrm{J}_N = \sum_{k=1}^{N+1} \left(\left(\mathbf{x}(k) - \mathbf{x}^{ref} \right)^T \mathbf{Q} \left(\mathbf{x}(k) - \mathbf{x}^{ref} \right) + \left(\mathbf{f}_\mathbf{i}(k) \right)^T \mathbf{R} \left(\mathbf{f}_\mathbf{i}(k) \right) \right) \tag{4-50}$$

The objective function given in (4-50) consists of two terms. The first penalizes states $\mathbf{x}$ that deviate significantly from the reference states, $\mathbf{x}^{ref}$ while the second term penalizes large

values for the feet forces $\mathbf{f_i}$. The weighting matrix $\mathbf{Q}$ trades off the degree to which each optimized state will match the respective reference state over the rest of them. The reference value for the forward velocity is the desired forward velocity, thus the reference body position along x axis is a linear function from the initial to the final position. The CoM's reference height is equal to the nominal height in a leg configuration where the joint angles (Figure 2-1) are set as $th_1 = -36^\circ$ and $th_2 = 72^\circ$. The reference values for the rest of the states are set to 0. Regarding the weighting matrix $\mathbf{R}$, the higher its elements are, the more high output forces are discouraged. Equation (4-51) gives an initial estimation of the weights.

$$\begin{aligned} Q_{ii} &= \frac{1}{\left(x_{\max} - x^{ref}\right)_i^2} \\ R_{ii} &= \frac{1}{f_{i,\max}^2} \\ Q_{ij} &= 0 \qquad i \neq j \\ R_{ij} &= 0 \end{aligned} \tag{4-51}$$

Then, these values are adjusted until the optimal performance is achieved. The feet forces are significantly larger than the error of the state's variables. To compensate this difference, the values of the weighting matrix $\mathbf{R}$ should be at least three orders of magnitude smaller. The index k specifies the discretization points.

4.4.5 Dynamic model

In the trajectory optimization formulation, the dynamic model that is defined, is treated as a constraint that should be satisfied at every grid point. The dynamic models that were presented in Section 2.3 are approximations of the physical model, but the choice depends on the application. The simplest one, which is the Linear Inverted Pendulum Model (LIPM) is rejected because it cannot be used to generate complex motions and it is inappropriate for uneven terrain. More accurate dynamic models are the rigid body dynamics and the centroidal dynamics models which takes as input the joint torques and the contact forces respectively. Their states are the six-dimensional base pose and the joint angles. Although these models are good representations of the actual physics, the physical constraints cannot be easily set in the joint space. On the other hand, the constraints can be described in Cartesian space efficiently.

For the aforementioned reasons, the single rigid body dynamics model is chosen, which can be written in the form of (4-52)-(4-53). The variables are described in detail in Section 2.3.3. Regarding the coordinate systems (see Figure 2-3), there is one defined at the CoM of the robot which is the base c.s. B and the other one is the inertial c.s. I in which the optimization variables are expressed (Figure 2-3). The direction of the forward movement is aligned with x-axis and y-axis is defined to be vertical to the ground.

$$\ddot{\mathbf{r}}(t) = \frac{1}{m}\left(\sum_{i=1}^{i=4} \mathbf{f_i}(t) - m\mathbf{g}\right) \tag{4-52}$$

$$\dot{\boldsymbol{\omega}}(t) = \frac{1}{\mathbf{I}}\left(\sum_{i=1}^{i=4} \mathbf{f}_i(t) \times \left(\mathbf{r}(t) - \mathbf{p_i}(t)\right) - \boldsymbol{\omega}(t) \times \mathbf{I}\boldsymbol{\omega}(t)\right) \tag{4-53}$$

4.4.6 Dynamics constraints using Hermite Simpson direct collocation

The base position, orientation and their derivatives are continuous variables that are transcribed into decision variables for the NLP by splitting these continuous trajectories into segments. Having already defined the total time T that the robot needs to accomplish a specified task, then this time interval is discretized every h seconds. Consequently, the total number of segments will be:

$$N = \frac{T}{h} \tag{4-54}$$

The implementation of Hermite-Simpson collocation method requires the following constraints. Firstly, a vector $\mathbf{x}$ that contains the base position $\mathbf{r}$, the velocity $\dot{\mathbf{r}}$, the orientation $\boldsymbol{\theta}$ and the angular velocity $\boldsymbol{\omega}$ is defined as:

$$\mathbf{x} = \begin{bmatrix} \mathbf{r} \\ \dot{\mathbf{r}} \\ \boldsymbol{\theta} \\ \boldsymbol{\omega} \end{bmatrix} \tag{4-55}$$

For the implementation of Hermite-Simpson direct collocation method, an extra collocation point in the middle of each segment should be defined. Then the system dynamics given in (4-52)-(4-53) and (2-29) have to be satisfied every $h/2$ seconds. The collocation constraints (4-58)-(4-59) are constructed to approximate the system dynamics. The change in state between any two knot points k and $k+1$ should be equal to the integral of the system dynamics $\mathrm{F}(\bullet)$ between those two knot points [35]:

$$\dot{\mathbf{x}} = \mathrm{F}(\mathbf{x}, \mathbf{p}, \mathbf{f}) \tag{4-56}$$

$$\int_{t_k}^{t_{k+1}} \dot{\mathbf{x}}\, dt = \int_{t_k}^{t_{k+1}} \mathbf{F}\, dt \tag{4-57}$$

Transcription occurs when the continuous integral in (4-57) is approximated with Simpson quadrature (analyzed in Appendix C) as shown in (4-58):

$$\mathbf{x}_{k+1} - \mathbf{x}_k = \frac{1}{6} h \left(\mathbf{F}_{\mathrm{k}} + 4\mathbf{F}_{\mathrm{k}+\frac{1}{2}} + \mathbf{F}_{\mathrm{k}+1} \right) \tag{4-58}$$

where $\mathbf{F}_{\mathrm{k}+\frac{1}{2}}$ is the dynamics at the midpoint $\mathbf{x}_{\mathrm{k}+\frac{1}{2}}$ calculated as follows:

$$\mathbf{x}_{k+\frac{1}{2}} = \frac{1}{2}\left(\mathbf{x}_k + \mathbf{x}_{k+1}\right) + \frac{h}{8}\left(\mathbf{F}_k - \mathbf{F}_{k+1}\right) \tag{4-59}$$

Given that the state trajectories are discretized every h seconds, then the algorithm will provide the solution of the states and their derivatives at the initial and the final point of this time interval. Consequently, there are four conditions available, and the state trajectory can be approximated by a cubic polynomial. The final point of one segment h is the initial point of the next segment. Thus, the cubic polynomials that are merged together, create a continuous spline and the whole trajectory is smooth.

4.4.7 Feet motion and forces parameterization

Quadrupeds can move in several ways and every gait has a distinctive pattern, with one or more feet leaving the ground at a time. The most common gaits are static walk, trotting, galloping and pace. The description of these gaits leads to a great number of phases, each of which represent a different combination of feet that makes contact with the ground. To simplify the problem, each foot is treated separately from the others, which results in two phases; contact and swing. These phases alternate, meaning that after a flight phase, a contact with the ground will occur and so on. One step includes a swing and a stance phase. When the physical problem is transcribed into an NLP, the decision variables have to be discretized. The position of each foot is parameterized using one quartic polynomial of duration $2h$ at swing phase. At stance phase, the foot stays still, so the position is described by a constant function. On the other hand, foot force is parameterized using three quadratic polynomials of equal duration at stance phase and at swing phase, the force is set to zero. Apparently, the last point of one polynomial is the starting point of the next one. The total number of discretization points is $2N+1$. The quadratic polynomials that represent the force trajectories are constructed from the initial and the final point of each segment h and the middle point that has been introduced in the problem. The quartic polynomials require 5 points, especially the two boundary points of two adjacent segments and the 3 intermediate points.

4.4.8 Foot operational space

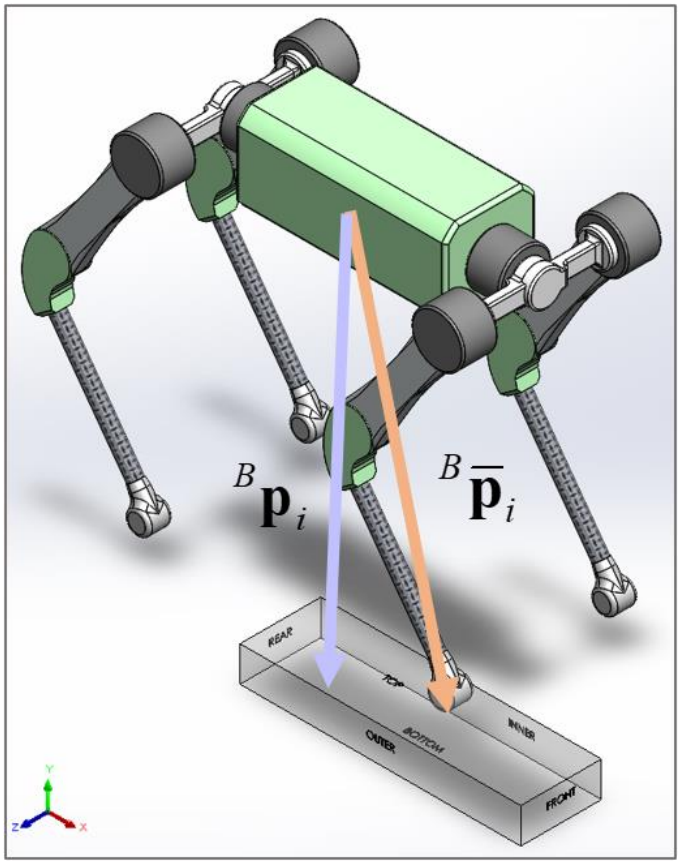


Figure 4-33. Foot's operational space.

In general, joint angles are restricted to be away from singularities to avoid undesirable effects. Also, robot's configurations have to be restricted in a space to avoid self-collisions. Since, the optimization problem has been defined in the Cartesian space, it is convenient that the

constraints are defined in Cartesian space too. So, instead of restricting the joint angles, the position of the foot relative to the base has to be kept within the bounds of a hexahedron (Figure 4-33). The hexahedron is centered at the nominal position of the foot and the length of the cube determines how much the foot can deviate from its nominal position. The equation that describes this constraint is:

$$\left| {}^{B}\mathbf{R}_{I}\left[\mathbf{p}_{i}(t)-\mathbf{r}(t)\right]-{}^{B}\bar{\mathbf{p}}_{i}\right|<\mathbf{b} \tag{4-60}$$

$${}^{B}\mathbf{p}_{i}(t)={}^{B}\mathbf{R}_{I}\left[\mathbf{p}_{i}(t)-\mathbf{r}(t)\right] \tag{4-61}$$

where $\mathbf{p}_i(t)$ is the current foot position relative to the world frame, ${}^{B}\bar{\mathbf{p}}_i$ is the nominal position of the foot relative to the base frame and $\mathbf{b}$ is the vector that represents the allowable offsets from the nominal position in each direction.

4.4.9 Position and force constraints

The nodes of the swing polynomials are treated differently from those of stance polynomials, as they are subject to different constraints. As the swing polynomials always alternate the stance polynomials, it is known a priori in which polynomial each node belongs. Firstly, it has to be ensured that the feet do not slip during the stance phase (denoted by C_i) by setting the foot position constant:

$$\mathbf{p}_{i,st}=const.\Rightarrow\mathbf{p}_{i}(t_{k}\in C_{i})=\mathbf{p}_{i}(t_{k+1}\in C_{i}) \tag{4-62}$$

Additionally, the position of the foot in y direction should be equal to the height of the terrain $h_{i,terrain}$ at the point of contact:

$$p_{i,st}^{y}=h_{i,terrain} \tag{4-63}$$

On the other hand, the contact forces determine the base motion, but they are subject to constraints. Regarding their direction, a unilateral constraint is added to ensure that these can only push into the ground and not pull on it. So, the nodes that belong to the stance phase polynomial have to be constrained as:

$$\mathbf{f}_{n,st}=\mathbf{f}^{T}\mathbf{n}(p_{i,st}^{xz})\geq 0 \tag{4-64}$$

where $\mathbf{n}(p_{i,st}^{xz})$ is the normal vector to the ground at the position of the foot which is in stance. For flat terrain $\mathbf{n}(p_{i,st}^{xz})=\begin{bmatrix}0 & 1 & 0\end{bmatrix}^{T}$. The upper bound of the normal foot force, when the leg is in stance phase, is set equal to three times the body weight. Another essential constraint is that the tangential forces along x f_{t_x} and z axis f_{t_z} have to be inside the friction cone, to ensure that the foot does not slip:

$$-\mu\mathbf{f}_{n,st}\leq\mathbf{f}_{\{t_x,t_z\}}\leq\mu\mathbf{f}_{n,st} \tag{4-65}$$

Apparently, the greater the normal force, the larger the acceptable tangential forces can be without slipping. The cone is also defined by the friction coefficient μ, which is set equal to 0.8. The above-mentioned constraints refer to the stance phase. At swing phase, the only restriction is that the feet forces should be zero:

$$\mathbf{f}_{i,sw}=\mathbf{0} \tag{4-66}$$

where $i=1,\ldots,4$ indicates each of the four legs.

4.4.10 Hermite-Simpson Collocation: Interpolation

The solution of the trajectory optimization problem outputs the optimal values of the decision variables at each of the grid points. The goal is the construction of continuous trajectories which is achieved by polynomial interpolation passing through these optimal points. The unknown values that lie in between the known data points are determined through the interpolation process. As already mentioned, the feet forces are represented by quadratic polynomials. A 2nd order polynomial that passes through the points (t_A, u_A), (t_B, u_B), (t_C, u_C) can be written in the form of (4-67).

$$u(t) = \frac{(t-t_B)(t-t_C)}{(t_A-t_B)(t_A-t_C)}u_A + \frac{(t-t_A)(t-t_C)}{(t_B-t_A)(t_B-t_C)}u_B + \frac{(t-t_A)(t-t_B)}{(t_C-t_A)(t_C-t_B)}u_C \tag{4-67}$$

However, in this formulation the segments have equal duration h and the point B is at the middle of the segment. If the indices A, B, C are substituted by the indices k, k+1, k+2, then h is given by (4-68) and the intermediate point by (4-69). Defining the time as $\tau = t - t_k$, the simplified equation becomes:

$$h = t_{k+2} - t_k \tag{4-68}$$

$$t_{k+1} = \frac{t_k + t_{k+2}}{2} \tag{4-69}$$

$$u(t) = \left(2u_k - 4u_{k+1} + 2u_{k+2}\right)\left(\frac{\tau}{h}\right)^2 + \left(-3u_k + 4u_{k+1} - u_{k+2}\right)\left(\frac{\tau}{h}\right) + u_k \tag{4-70}$$

The system dynamics $\mathrm{F}(\cdot) = \dot{x}$ is also represented by quadratic polynomials (4-71).

$$\mathrm{F}(t) = \dot{x} = \left(2\,\mathrm{F}_k - 4\,\mathrm{F}_{k+1} + 2\,\mathrm{F}_{k+2}\right)\left(\frac{\tau}{h}\right)^2 + \left(-3\,\mathrm{F}_k + 4\,\mathrm{F}_{k+1} - \mathrm{F}_{k+2}\right)\left(\frac{\tau}{h}\right) + \mathrm{F}_k \tag{4-71}$$

The states trajectories are derived by integrating (4-72) as

$$x = \int \dot{x}dt = \int \left(\left(2\,\mathrm{F}_k - 4\,\mathrm{F}_{k+1} + 2\,\mathrm{F}_{k+2}\right)\left(\frac{t}{h}\right)^2 + \left(-3\,\mathrm{F}_k + 4\,\mathrm{F}_{k+1} - \mathrm{F}_{k+2}\right)\left(\frac{t}{h}\right) + \mathrm{F}_k \right) dt \Rightarrow$$

$$x = \frac{1}{3}h\left(2\,\mathrm{F}_k - 4\,\mathrm{F}_{k+1} + 2\,\mathrm{F}_{k+2}\right)\left(\frac{\tau}{h}\right)^3 + \frac{1}{2}h\left(-3\,\mathrm{F}_k + 4\,\mathrm{F}_{k+1} - \mathrm{F}_{k+2}\right)\left(\frac{\tau}{h}\right)^2 + \mathrm{F}_k\,\tau + x_k \Rightarrow \tag{4-72}$$

$$x = \frac{1}{3h^2}\left(2\,\mathrm{F}_k - 4\,\mathrm{F}_{k+1} + 2\,\mathrm{F}_{k+2}\right)\tau^3 + \frac{1}{2h}\left(-3\,\mathrm{F}_k + 4\,\mathrm{F}_{k+1} - \mathrm{F}_{k+2}\right)\tau^2 + \mathrm{F}_k\,\tau + x_k$$

4.4.11 Quartic interpolation

The foot position in swing phase is represented by a quartic polynomial which is given by (4-73). Given five data pairs $\{t_i, p_i\}$, the coefficients of the polynomial can be found by solving the linear system (4-74). The 5×5 interpolation matrix is known as Vandermonde matrix.

$$p(t) = \alpha_0 + \alpha_1 t + \alpha_2 t^2 + \alpha_3 t^3 + \alpha_4 t^4 \tag{4-73}$$

$$\begin{pmatrix} 1 & t_0 & t_0^2 & t_0^3 & t_0^4 \\ 1 & t_1 & t_1^2 & t_1^3 & t_1^4 \\ 1 & t_2 & t_2^2 & t_2^3 & t_2^4 \\ 1 & t_3 & t_3^2 & t_3^3 & t_3^4 \\ 1 & t_4 & t_4^2 & t_4^3 & t_4^4 \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{pmatrix} = \begin{pmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \\ p_4 \end{pmatrix} \tag{4-74}$$

An alternative way to create an interpolant is by expressing the polynomial p(t) as a sum of four quartic polynomials $l_i(t)$ which satisfy the "delta conditions" (4-77), given by:

$$p(t) = \sum_{i=0}^{4} p_i l_i(t)\,, \tag{4-75}$$

$$l_i(t) = \frac{\sum_{j=1,\, j\neq i}^{j=4} \left(t - t_j\right)}{\sum_{j=1,\, j\neq i}^{j=4} \left(t_i - t_j\right)} \tag{4-76}$$

$$l_i(t) = \begin{cases} 1 & if\ t = t_i \\ 0 & if\ t \neq t_i \end{cases} \tag{4-77}$$

This approach is called the Lagrange form of the interpolating polynomial. The quartic polynomial refers to two adjacent segments which last $h_{sw} = 2h$, so the optimal points which are evenly spaced will be $\{0,\, p_0\}$, $\{h_{sw}/4,\, p_1\}$, $\{h_{sw}/2,\, p_2\}$, $\{3h_{sw}/4,\, p_3\}$, $\{h_{sw},\, p_4\}$. After substituting the data points at (4-75)-(4-76), the resulting polynomial coefficients are:

$$\alpha_0 = p_0 \tag{4-78}$$

$$\alpha_1 = \left(-\frac{25}{3}p_0 + 16p_1 - 12p_2 + \frac{16}{3}p_3 - p_4\right)\frac{1}{h_{sw}} \tag{4-79}$$

$$\alpha_2 = \left(\frac{70}{3}p_0 - \frac{208}{3}p_1 + 76p_2 - \frac{112}{3}p_3 + \frac{22}{3}p_4\right)\frac{1}{h_{sw}^2} \tag{4-80}$$

$$\alpha_3 = \left(-\frac{80}{3}p_0 + 96p_1 - 128p_2 + \frac{224}{3}p_3 - 16p_4\right)\frac{1}{h_{sw}^3} \tag{4-81}$$

$$\alpha_4 = \left(\frac{32}{3}p_0 - \frac{128}{3}p_1 + 64p_2 - \frac{128}{3}p_3 + \frac{32}{3}p_4\right)\frac{1}{h_{sw}^4} \tag{4-82}$$

The aforementioned method simplifies the construction of a 4th order polynomial, which is essential as this formula creates smooth trajectories while exploiting all the available optimal points.

4.4.12 Results

The trajectory optimization problem that was analyzed in the previous sections incorporates differential-algebraic equations (DAE). The numerical solution of this problem requires its transcription to an NLP and then it can be solved by off-the-self NLP solvers. Section 3.6 highlights the reasons to formulate the problem in CasADi, while Section 3.5 demonstrates

that IPOPT is the most efficient NLP solver compared to the others that were discussed. Therefore, CasADi is imported to MATLAB, the variables are defined as CasADi symbols and the NLP is solved by IPOPT. The code[5] is included in Appendix F.

Apparently, the elapsed time depends both on the time horizon and the discretization of the trajectories. A 2-core Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz computer needs 2-3 s to solve the problems that are presented below. Especially, the total CPU time in IPOPT (w/o function evaluations) is the $1/5$ of the total time and the rest is the CPU time in NLP function evaluations.

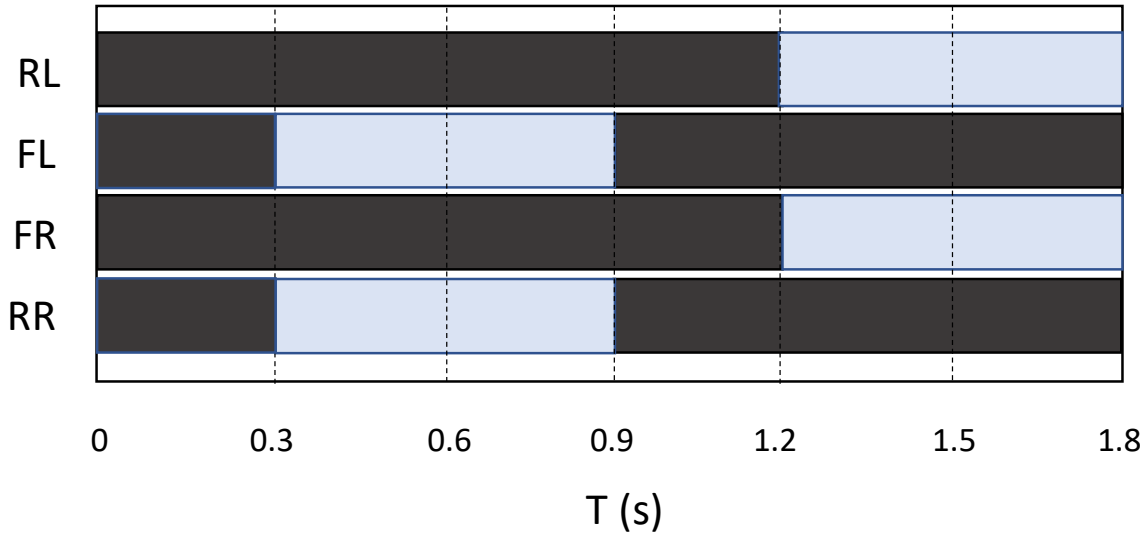


Figure 4-34. Gait pattern. The grey color indicates the stance phase while the light color indicates the swing phase.

The results that are presented in the following sections include both the motion objectives of the quadruped and the torques required to achieve these motions. The robustness of the algorithm is tested in 3 simulation experiments with different final position and consequently different forward velocity. The target of the system is the final position, but the mean forward velocity is introduced indirectly as a reference state in the objective function and as an initial guess. The time horizon is kept constant in these three cases. The number of steps is set to 4, which is equivalent to 4 full periods. The trajectory is discretized every $h = 0.3s$. The swing phase includes 2 adjacent segments (4-83) and the stance phase 4 segments, see (4-84).

$$\Delta T_{sw} = 2h \tag{4-83}$$

$$\Delta T_{st} = 4h \tag{4-84}$$

In this phase-based trajectory optimization formulation, the sequence of the phases for each leg is defined by the gait pattern that is illustrated in Figure 4-34.

The weighting matrix $\mathbf{Q} \in \mathbb{R}^{12\times12}$ (4-85) is a diagonal matrix that penalizes the deviation from the reference states. The values at the diagonal are equal to one except for the second element which is three orders of magnitude smaller.

[5] The MATLAB code can be found here:
https://bitbucket.org/csl_legged/argos_trajectory_optimization/src/master/

$$\mathbf{Q} = \begin{bmatrix} 1 & 0 & \cdots & \cdots & \cdots & 0 \\ 0 & 1e^{-3} & 0 & & & \vdots \\ \vdots & 0 & 1 & 0 & & \vdots \\ \vdots & & 0 & \ddots & 0 & \vdots \\ \vdots & & & 0 & \ddots & 0 \\ 0 & \cdots & \cdots & \cdots & 0 & 1 \end{bmatrix} \tag{4-85}$$

This gain penalizes to a lesser extent the deviation of the CoM's height to its nominal value, which makes the movement seem more physical. If the value of the quadruped's height exceeds the upper acceptable limit, this means that the feet are extended excessively which may lead to singularities. On the other hand, if the robot stops below the lowest acceptable position, then the torques at the knee will become really high. To avoid the aforementioned issues, strict constraints are set, which ensures that the body position along y axis is bounded by a lower and an upper limit. The weighting matrix $\mathbf{R} \in \mathbb{R}^{4\times4}$ (4-86) penalizes large values at the components of the feet forces vectors which are normal to the ground. As mentioned in Section 4.4.4, the gains are chosen significantly smaller than those of the matrix $\mathbf{Q}$.

$$\mathbf{R} = \begin{bmatrix} 1e^{-3} & 0 & 0 & 0 \\ 0 & 1e^{-3} & 0 & 0 \\ 0 & 0 & 1e^{-3} & 0 \\ 0 & 0 & 0 & 1e^{-3} \end{bmatrix} \tag{4-86}$$

As mentioned in Section 4.4.8, the foot is restricted to move inside a hexahedron. The operational space of the foot is constrained by the offsets given in Table 4-6.

Table 4-6. Offsets from the nominal foot position

X axis	Front	0.4 m
	Rear	0.4 m
Y axis	Top	0.05 m
	Bottom	0.05 m
Z axis	Inner	0.03 m
	Outer	0.2 m

Body poses and velocity

The final position that the robot should reach is expressed as x_F . Three simulation experiments with different target position x_F are presented in Figure 4-35. The quadruped starts with the desired mean forward velocity and at the end of the simulation time, it stops, i.e., its velocity becomes 0. The quadruped reaches the goal while keeping almost constant velocity. This is due to the fact that the smaller the deviation of the actual speed from the reference speed, the smaller the value of the objective function will be. The CoM's position along the y axis of the inertial frame is periodical and its frequency depends on the gait pattern which is the same for the three cases. The amplitude of the oscillation is acceptable and does not affect the whole movement. The CoM's position along the z axis is kept constant and equal to 0 as it is desired (Figure 4-35). To keep up with the reference velocity, the stride length of the third case is too long, so in order not to violate the foot operational constraints, the robot's main body may tilt to the left or right side. Figure 4-36 displays that the order of magnitude of

the angular deviation along the x, y and z axis is $1e^{-6}\,rad$ (roll), $1e^{-7}\,rad$ (yaw) and $1e^{-3}\,rad$ (pitch) respectively. Therefore, it is ensured that the quadruped will not tumble over.

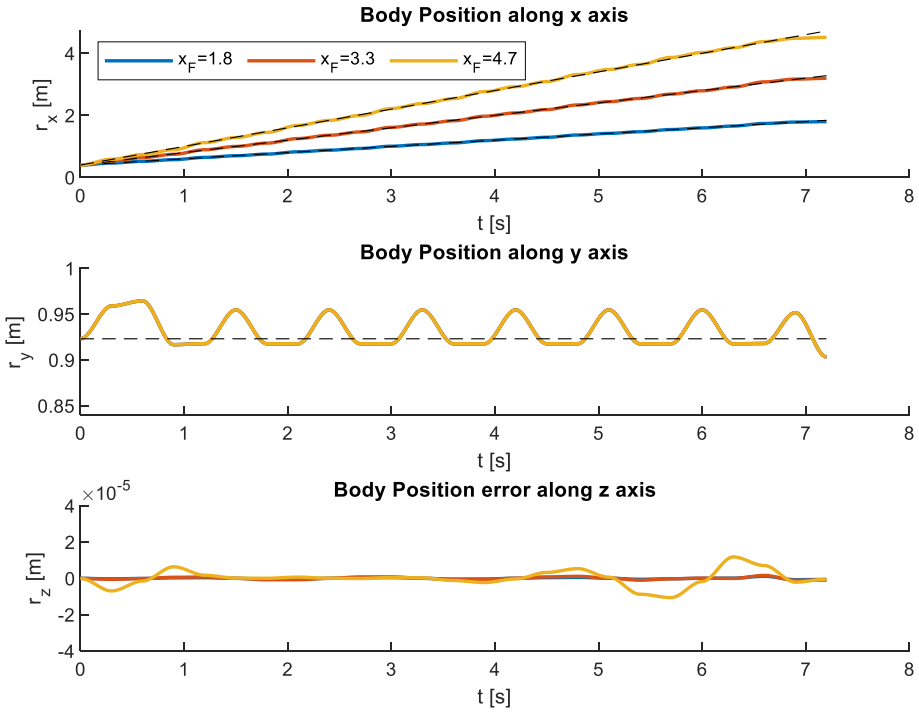


Figure 4-35. Desired and actual body position in Cartesian space for three different target positions x_F.

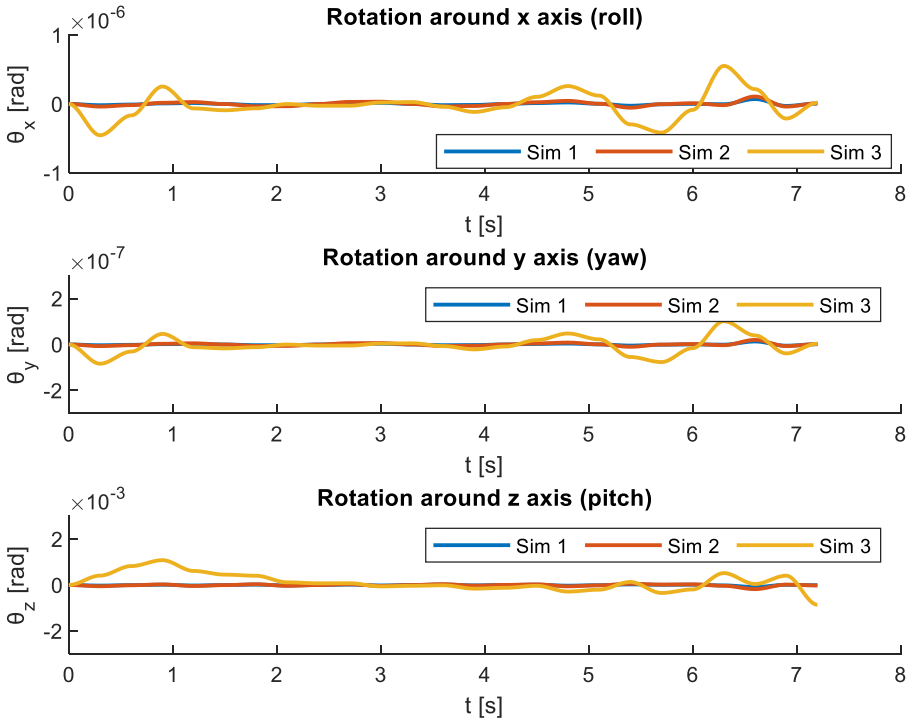


Figure 4-36. Body orientation in Cartesian space.

The velocity along the x and y axes (Figure 4-37) is changing periodically. Among the aforementioned constraints, the decision variables could also be bounded to avoid infeasible movements, so the acceptable values of v_y range between $\pm 0.3\, m/s$ in order to avoid the abrupt changes in CoM's height. As it was described earlier, between the swing phases of the two sets of diagonal legs, a stance phase occurs when all the feet make contact with the ground. The forward velocity v_x falls below the reference value due to this intermediate stance phase. However, the mean forward velocity is slightly lower than the reference velocity (Table 4-7), but this happens because the quadruped stops at the end and the final value is 0. Figure 4-37 also depicts that the velocity along z is almost 0 (order of magnitude $1e^{-4}\, m/s$) which means it tracks the desired command.

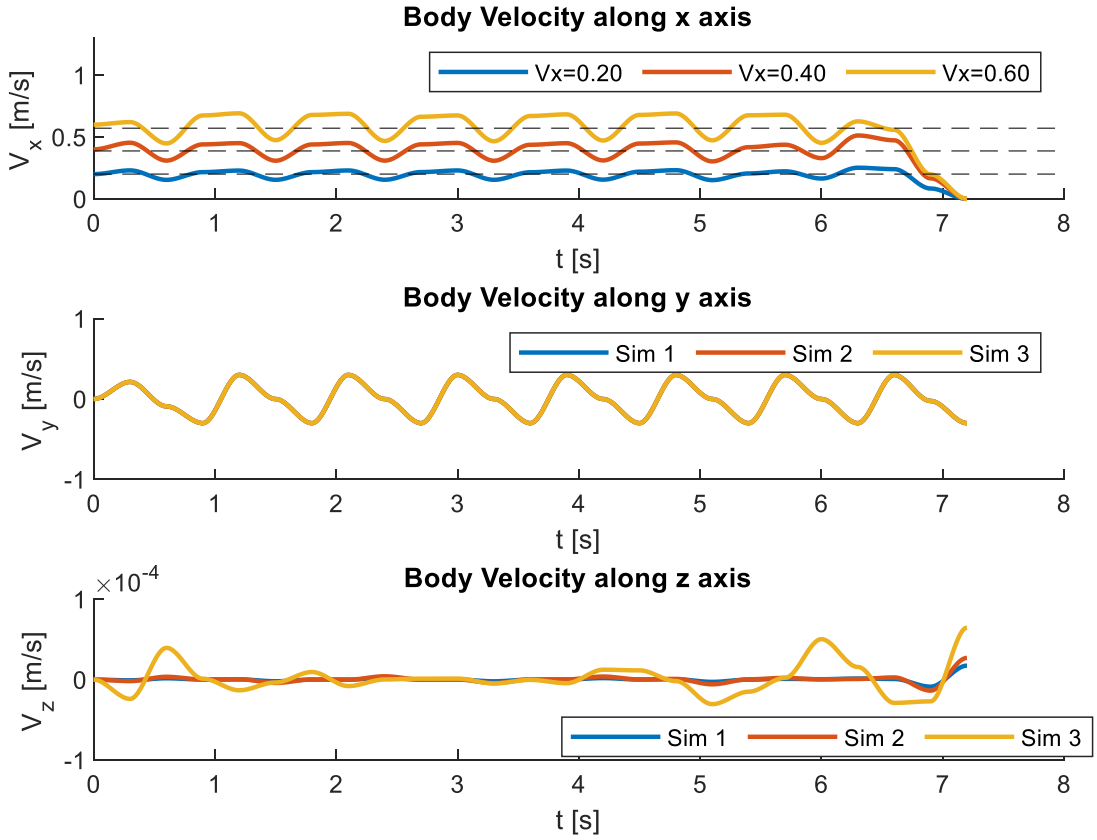


Figure 4-37. Body velocity in Cartesian space.

Table 4-7. The average forward velocity that the robot achieves given the desired command.

v_x^d	0.2	0.4	0.6
$\bar{v}_x$	0.19	0.39	0.57

Foot trajectory

The trajectories that are illustrated in Figure 4-38 refer to the rear left leg, but similar response is noticed for the rest of the legs. The foot position along the x axis is represented by a constant line in stance phase, while the foot position along the y axis is 0. The quartic polynomials that

are generated in swing phase seem to be really smooth and inside the acceptable range of values. As the distance that the quadruped should travel increases, while keeping during the simulation time and the frequency (or the number of steps) fixed, the foot should make bigger steps to reach the desired final position. The foot trajectory along y axis remains the same for all the different cases that were examined.

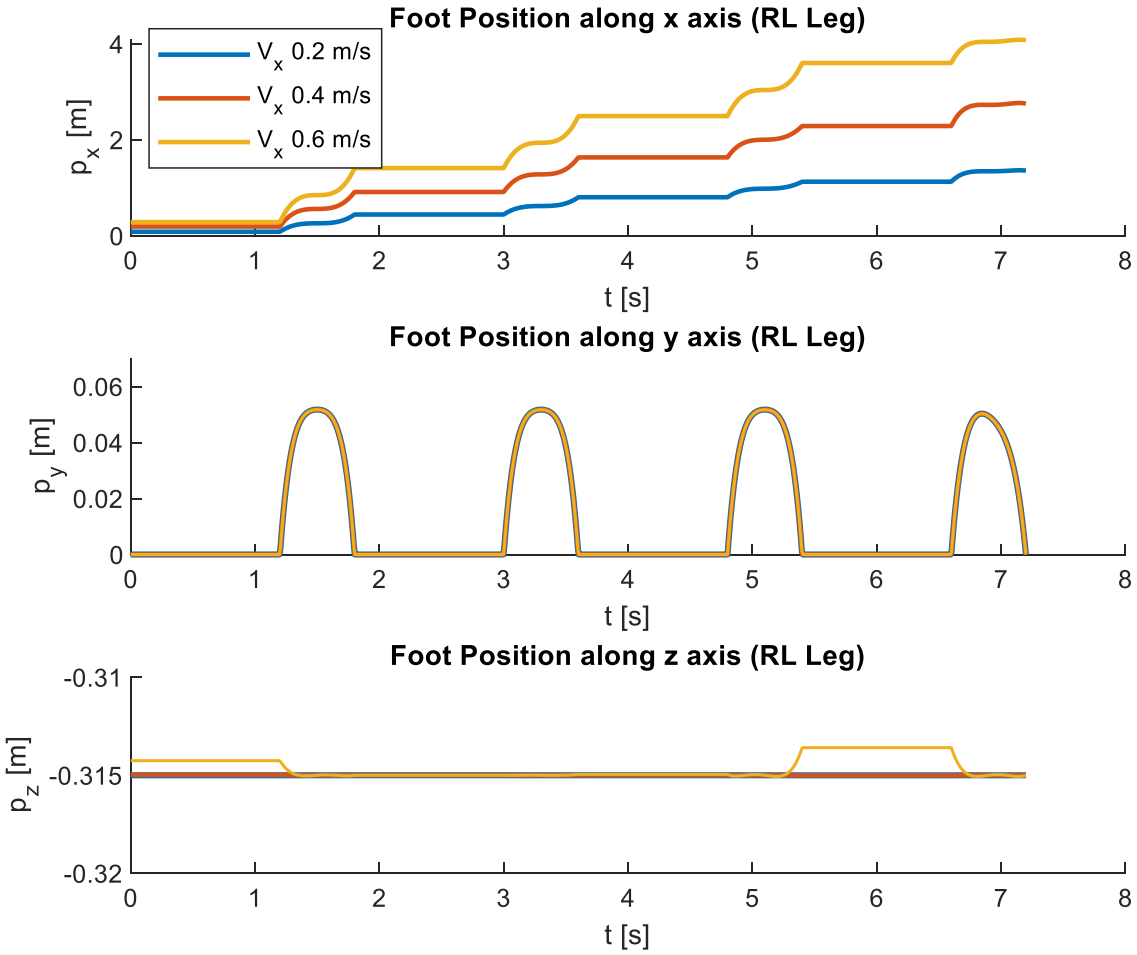


Figure 4-38. Foot Trajectory (Rear Left leg).

Foot forces and joint torques

Figure 4-39 depicts the feet for the three different simulations. The forces during the swing phase are represented by four 2nd order polynomials. The normal forces are identical in all simulation experiments. The forces along the x and z axes should satisfy the friction cone constraints as:

$$\left| f_x, f_z \right| \leq \mu f_y \tag{4-87}$$

When the foot is in the air, the forces are zero. The feet forces are mapped to the joint torques (4-88) via the geometrical Jacobian given in (2-12). Figure 4-40 illustrates the torques during stance phase which are the most critical.

$$\boldsymbol{\tau} = -\mathbf{J}^{\mathrm{T}}\mathbf{F} \tag{4-88}$$

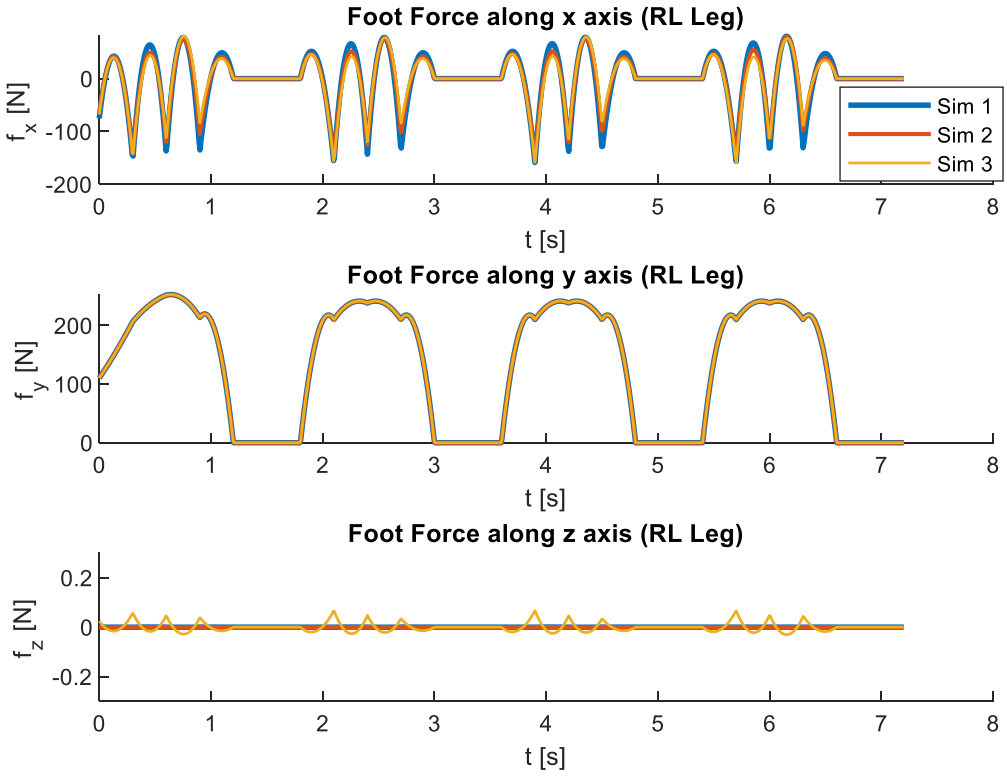


Figure 4-39. Foot Forces (Rear Left leg).

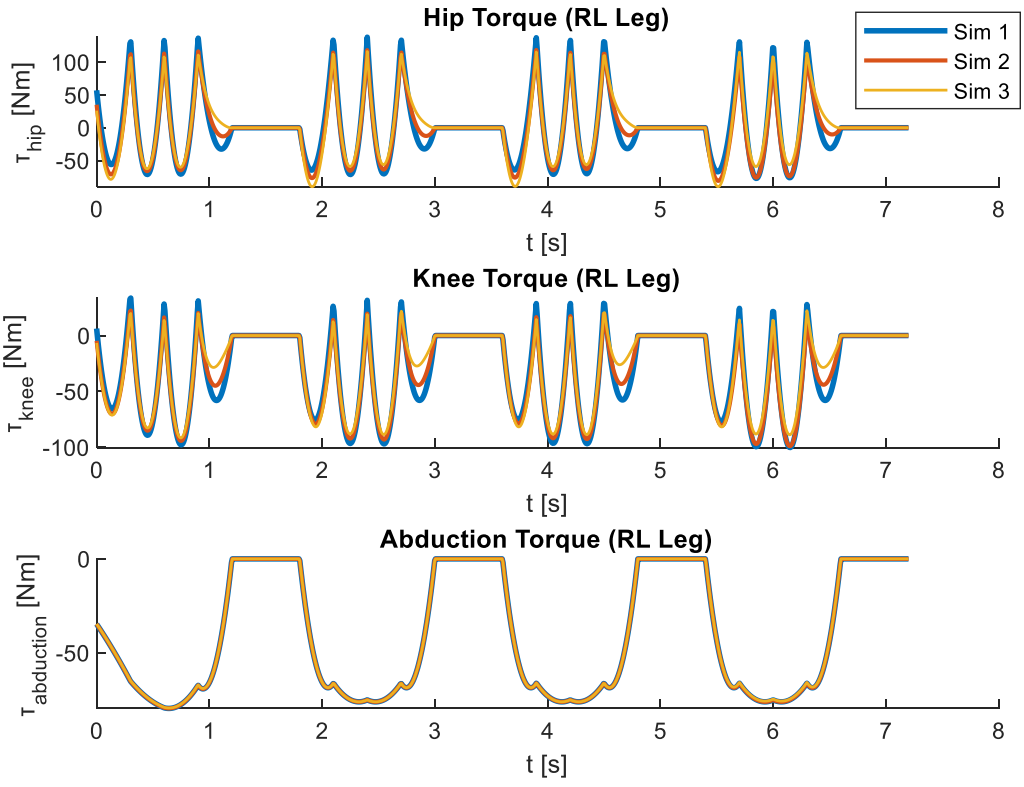


Figure 4-40. Joint torques.

4.4.13 Conclusion

The implementation of trajectory optimization in a multi-phase problem was successful and the process of finding valid trajectories for the CoM and the feet was automated. This formulation is applicable when the terrain map and the timings that each foot interacts with the environment are provided. This happens because the constraints at swing phase differ from those at stance phase. The outputs of the optimization problem are comparable to those of the previous models, so these motions are feasible. However, the optimal planning is based on a single rigid body dynamics model, so deviations may appear in the real experiment.

The model can be further improved if a continuity constraint is added at the points between the three different polynomial splines that represent the forces at stance phase. This can be achieved by introducing the derivative of the force as decision variable. In [63], the trajectories are continuous in their entire range, but it is not stated clearly how this is achieved. As a final point, this optimal planning method should be combined with a whole-body controller or a MPC to evaluate the output trajectories and verify the overall response of the system.

5 Conclusions and Future Work

5.1 Conclusion

The first approach proves that a simple control framework leads to successful quadrupedal locomotion. The tunable parameters are few, so it is easy to observe the effect of changing one parameter on the response of the system. However, feasible motions are limited to simple foot trajectories that are predetermined and executed according to a gait pattern.

The contribution of the second approach compared to the first is that the quadruped motion is determined by high-level velocity and yaw rate commands, which makes target specification easy. The optimal force distribution and the footstep planning ensure that in the event of a disturbance, the quadruped will place its feet at an appropriate position and the forces acting on the stance feet will restore the body to its desired posture. The optimization problem implicitly incorporates the torque limits by bounding the output foot forces under a specified limit, which leads to feasible output commands. The conclusion drawn is that footholds and body motion are important quantities in legged locomotion and tightly coupled. Another advantage of this framework is that it is highly modular. Each of the subsystems describing either the leg trajectories or their control can be replaced by other algorithms, thus allowing the exploration of new movements. Although tuning the various parameters to the optimum is a tedious process and even impractical, a wide range of parameters has been found to lead to successful movements. Overall, it is an efficient motion planning algorithm for legged robots that could be applied to unstructured terrain locomotion situations.

Regarding the trajectory optimization process, the simulations of the two-link manipulator revealed that TO is a really efficient way to generate optimal trajectories, which can be easily adjusted by modifying the objective function. Additionally, the combination of trajectory optimization with MPC results in a good trajectory tracking and the final goal was reached successfully. Repeatedly generating motion plans starting from the current state of the system was feasible due to the fast control loop speed.

The last approach demonstrated that the discrete dynamics of the legged locomotion problem can be handled successfully by the phase-based trajectory optimization technique. The process of finding valid trajectories for the CoM and the feet were automated and the range of achievable motions has been expanded. However, this technique requires a complete definition of the terrain map and the timings that each foot makes contact with the ground in order to enforce the appropriate constraints at each grid point. It is worth mentioning that the optimal planning is based on a single rigid body dynamics model, so deviations may appear at the real experiment. The outputs of the optimization problem are expected motions for the quadruped locomotion on flat terrain. The usefulness of this approach could be highlighted in more complex tasks, where the definition of all these tightly coupled trajectories is not evident.

According to the aforementioned evaluation, each approach has its advantages and disadvantages. In summary, the choice of the appropriate motion planning and control scheme depends on the needs of the application.

5.2 Future Work

Each of the three approaches that were investigated has different potential for improvement. Due to the simplicity of the first approach, it is ideal for the initial real experiments of the quadruped Argos. The response of the system during the trajectory tracking and during the interaction with the ground can be clearly observed. The simulations of the second motion planning approach can be extended to sloped and rough terrain. In this way, the contribution of the optimization in the control of the quadruped's body pose can be better assessed. Further experiments with external disturbances acting on the body can be performed to verify system robustness.

Trajectory optimization is an attractive approach, as a large variety of motions can be generated. Future work may include a model predictive control formulation that would allow the robot to re-plan and take corrective steps if the actual plan deviates from the desired one. Also, it would be robust to pushes or foot slips. The modelling errors that arise due to the approximation of the real dynamics by the SRBD model will not affect the performance because the quick replanning will correct any model inaccuracies. The only discouraging factor may be the control loop speed, but MPC is worth investigating further. In conclusion, improving the capabilities of the motion-planning algorithm and embedding it in an MPC formulation on a physical system, seems to have high potential.

6 References

[1] (2016). Matthewpeterkelly.com. http://www.matthewpeterkelly.com/tutorials/trajectoryOptimization/cartPoleCollocation.svg#frame1129

[2] *32 Photos That Prove Goats Are the World's Best Climbers*. (n.d.). Matador Network. Retrieved November 1, 2022, from https://matadornetwork.com/life/32-photos-that-prove-goats-are-the-worlds-best-climbers/?epik=dj0yJnU9VDd5ZU5rMFNzdUZVZEJCV1huQWlKbWFaZ2NyRWJnX0omcD0wJm49aDZRZk5JNXZtcGw2VlpucDBmTlhQQSZ0PUFBQUFBR05Md1U0

[3] Andersson, J. A. E., Gillis, J., Horn, G., Rawlings, J. B., & Diehl, M. (2018). CasADi: a software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, *11*(1), 1–36. https://doi.org/10.1007/s12532-018-0139-4

[4] Andersson, J., Akesson, J., & Diehl, M. (2012). Dynamic optimization with CasADi. *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*. https://doi.org/10.1109/cdc.2012.6426534

[5] Benson, H. Y., Shanno, D. F., & Vanderbei, R. J. (2004). Interior-point methods for nonconvex nonlinear programming: Jamming and numerical testing. *Mathematical Programming*, *99*(1), 35–48. https://doi.org/10.1007/s10107-003-0418-2

[6] Betts, J. T. (2010). *Practical methods for optimal control and estimation using nonlinear programming*. Society For Industrial And Applied Mathematics, Cop.

[7] Betts, J. T., & Frank, P. D. (1994). A sparse nonlinear optimization algorithm. *Journal of Optimization Theory and Applications*, *82*(3), 519–541. https://doi.org/10.1007/bf02192216

[8] Biegler, L. T., & Zavala, V. M. (2009). Large-scale nonlinear programming using IPOPT: An integrating framework for enterprise-wide dynamic optimization. *Computers & Chemical Engineering*, *33*(3), 575–582. https://doi.org/10.1016/j.compchemeng.2008.08.006

[9] Biegler, L. T., Ghattas, O., Matthias Heinkenschloss, & Bart. (2012). *Large-Scale PDE-Constrained Optimization*. Springer Science & Business Media.

[10] Bledt, G., Powell, M. J., Katz, B., Di Carlo, J., Wensing, P. M., & Kim, S. (2019). MIT Cheetah 3: Design and Control of a Robust, Dynamic Quadruped Robot. *Other Repository*. https://dspace.mit.edu/handle/1721.1/126619

[11] Bledt, G., Wensing, P. M., & Kim, S. (2017). Policy-regularized model predictive control to stabilize diverse quadrupedal gaits for the MIT cheetah. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. https://doi.org/10.1109/iros.2017.8206268

[12] Blickhan, R. (1989). The spring-mass model for running and hopping. *Journal of Biomechanics*, *22*(11-12), 1217–1227. https://doi.org/10.1016/0021-9290(89)90224-8

[13] Byrd, R. H., Hribar, M. E., & Nocedal, J. (1999). An Interior Point Algorithm for Large-Scale Nonlinear Programming. *SIAM Journal on Optimization*, *9*(4), 877–900. https://doi.org/10.1137/s1052623497325107

[14] Byrd, R., Nocedal, J., & Waltz, R. A. (2006). Knitro: An Integrated Package for Nonlinear Optimization. *Undefined*. https://www.semanticscholar.org/paper/Knitro%3A-An-Integrated-Package-for-Nonlinear-Byrd-Nocedal/d34319bafaaaf3dd5f315edff819a1cd9a38b6d6

[15] Coros, S., Karpathy, A., Jones, B., Reveret, L., & van de Panne, M. (2011). Locomotion skills for simulated quadrupeds. *ACM Transactions on Graphics*, *30*(4), 1–12. https://doi.org/10.1145/2010324.1964954

[16] Dallas, S., Machairas, K., Koutsoukis, K., & Papadopoulos, E. (2017, September 1). *A leg design method for high speed quadrupedal locomotion*. IEEE Xplore. https://doi.org/10.1109/IROS.2017.8206365

[17] Di Carlo, J., Wensing, P. M., Katz, B., Bledt, G., & Kim, S. (2018, October 1). *Dynamic Locomotion in the MIT Cheetah 3 Through Convex Model-Predictive Control*. IEEE Xplore. https://doi.org/10.1109/IROS.2018.8594448

[18] Diehl, M., Bock, H., Diedam, H., & Wieber, P.-B. (2005). Fast Direct Multiple Shooting Algorithms for Optimal Robot Control. In *Fast Motions in Biomechanics and Robotics*. https://hal.inria.fr/inria-00390435/PDF/Diehl.pdf

[19] Farshidian, F., Neunert, M., Winkler, A. W., Rey, G., & Buchli, J. (2017). An efficient optimal planning and control framework for quadrupedal locomotion. *2017 IEEE International Conference on Robotics and Automation (ICRA)*. https://doi.org/10.1109/icra.2017.7989016

[20] Ferreau, Joachim. (2007). textttqpOASES -- An Open-Source Implementation of the Online Active Set Strategy for Fast Model Predictive Control.

[21] Gehring, C., Bellicoso, C. D., Fankhauser, P., Coros, S., & Hutter, M. (2017). *Quadrupedal locomotion using trajectory optimization and hierarchical whole body control*. Www.research-Collection.ethz.ch; IEEE. https://doi.org/10.1109/ICRA.2017.7989557

[22] Gehring, C., Bellicoso, D., Bloesch, M., Sommer, H., Fankhauser, P., Hutter, M., & Siegwart, R. (n.d.). *Kindr Library -Kinematics and Dynamics for Robotics*. Retrieved October 31, 2022, from https://docs.leggedrobotics.com/kindr/cheatsheet_latest.pdf

[23] Gehring, C., Coros, S., Hutter, M., Bloesch, M., Hoepflinger, M. A., & Siegwart, R. (2013). Control of dynamic gaits for a quadrupedal robot. *2013 IEEE International Conference on Robotics and Automation*. https://doi.org/10.1109/icra.2013.6631035

[24] Gertz, E. M., & Wright, S. J. (2003). Object-oriented software for quadratic programming. *ACM Transactions on Mathematical Software*, *29*(1), 58–81. https://doi.org/10.1145/641876.641880

[25] Gill, P. E., Murray, W., & Saunders, M. A. (2005). SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization. *SIAM Review*, *47*(1), 99–131. https://doi.org/10.1137/s0036144504446096

[26] Gill, P., Murray, W., & Saunders, M. (2008). *User's Guide for SNOPT Version 7: Software for Large-Scale Nonlinear Programming **. https://web.stanford.edu/group/SOL/guides/sndoc7.pdf

[27] Gill, P., Murray, W., Saunders, M., & Wright, M. (n.d.). *USER'S GUIDE FOR NPSOL 5.0: A FORTRAN PACKAGE FOR NONLINEAR PROGRAMMING*. Retrieved November 1, 2022, from http://www.ccom.ucsd.edu/~peg/papers/npdoc.pdf

[28] Havoutis, I., Semini, C., Buchli, J., & Caldwell, D. (n.d.). *Progress in quadrupedal trotting with active compliance*. Retrieved October 31, 2022, from https://ihavoutis.github.io/publications/2012/2012_DynamicWalkingAbstract.pdf

[29] HilberTraum. (2017, November 26). *English: Slopes used by the classical Runge-Kutta method (rk4)*. Wikimedia Commons. https://commons.wikimedia.org/w/index.php?curid=64366870

[30] Hutter, M., Gehring, C., Bloesch, M., Hoepflinger, M., Remy, C., & Siegwart, R. (2012). Starleth: A compliant quadrupedal robot for fast, efficient, and versatile locomotion. *Undefined*. https://www.semanticscholar.org/paper/Starleth%3A-A-compliant-quadrupedal-robot-for-fast%2C-Hutter-Gehring/049424c48e5449a9ff135534faaf3cf356214b43

[31] *IBM ILOG CPLEX Optimization Studio CPLEX User's Manual*. (n.d.). https://www.ibm.com/docs/en/SSSA5P_12.8.0/ilog.odms.studio.help/pdf/usrcplex.pdf

[32] *International Cheetah Day: running fast toward extinction*. (2019, December 4). Tehran Times. https://www.tehrantimes.com/news/442666/International-Cheetah-Day-running-fast-toward-extinction

[33] Jia, Y., Luo, X., Han, B., Liang, G., Zhao, J., & Zhao, Y. (2018). Stability Criterion for Dynamic Gaits of Quadruped Robot. *Applied Sciences*, *8*(12), 2381. https://doi.org/10.3390/app8122381

[34] Katz, B., Carlo, J., & Kim, S. (2019). Mini Cheetah: A Platform for Pushing the Limits of Dynamic Quadruped Control. *Undefined*. https://www.semanticscholar.org/paper/Mini-Cheetah%3A-A-Platform-for-Pushing-the-Limits-of-Katz-Carlo/bb7e50d5d25ebf46f04c6d8cabdaac72a0e9d297

[35] Kelly, M. (2017). An Introduction to Trajectory Optimization: How to Do Your Own Direct Collocation. *SIAM Review*, *59*(4), 849–904. https://doi.org/10.1137/16m1062569

[36] Kelly, M. (2022, July 30). *OptimTraj - Trajectory Optimization for Matlab*. GitHub. https://github.com/MatthewPeterKelly/OptimTraj

[37] Kelly, M. P. (2017). Transcription Methods for Trajectory Optimization: a beginners tutorial. *ArXiv:1707.00284 [Math]*. https://arxiv.org/abs/1707.00284

[38] Klein, J.-F. (2018). *A simulation framework for a hybrid leg-wheeled robot using Gazebo and ROS*. [Master's Thesis]. Available at: https://www.dropbox.com/s/2b9c5cbx3wzcmqd/masters_thesis.pdf?dl=0.

[39] Kudruss, M., Naveau, M., Stasse, O., Mansard, N., Kirches, C., Soueres, P., & Mombaur, K. (2015). Optimal control for whole-body motion generation using center-of-mass dynamics for predefined multi-contact configurations. *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*. https://doi.org/10.1109/humanoids.2015.7363428

[40] Li, H., & Wensing, P. M. (2020). Hybrid Systems Differential Dynamic Programming for Whole-Body Motion Planning of Legged Robots. *IEEE Robotics and Automation Letters*, *5*(4), 5448–5455. https://doi.org/10.1109/lra.2020.3007475

[41] Machairas, K., & Papadopoulos, E. (2016). An active compliance controller for quadruped trotting. *2016 24th Mediterranean Conference on Control and Automation (MED)*. https://doi.org/10.1109/med.2016.7536064

[42] Mastalli, C., Focchi, M., Havoutis, I., Radulescu, A., Calinon, S., Buchli, J., Caldwell, D. G., & Semini, C. (2017). Trajectory and foothold optimization using low-dimensional models for rough terrain locomotion. *2017 IEEE International Conference on Robotics and Automation (ICRA)*. https://doi.org/10.1109/icra.2017.7989131

[43] Neunert, M., Stäuble, M., Giftthaler, M., Bellicoso, C. D., Carius, J., Gehring, C., Hutter, M., & Buchli, J. (2018). Whole-Body Nonlinear Model Predictive Control Through Contacts for Quadrupeds. *IEEE Robotics and Automation Letters*, *3*(3), 1458–1465. https://doi.org/10.1109/LRA.2018.2800124

[44] *Optimization Toolbox*. (n.d.). www.mathworks.com. https://www.mathworks.com/products/optimization.html

[45] Orin, D. E., Goswami, A., & Lee, S.-H. (2013). Centroidal dynamics of a humanoid robot. *Autonomous Robots*, *35*(2-3), 161–176. https://doi.org/10.1007/s10514-013-9341-4

[46] Orsolino, R., Focchi, M., Caron, S., Raiola, G., Barasuol, V., Caldwell, D. G., & Semini, C. (2020). Feasible Region: An Actuation-Aware Extension of the Support Region. *IEEE Transactions on Robotics*, *36*(4), 1239–1255. https://doi.org/10.1109/tro.2020.2983318

[47] Pardo, D., Moller, L., Neunert, M., Winkler, A. W., & Buchli, J. (2016). Evaluating Direct Transcription and Nonlinear Optimization Methods for Robot Motion Planning. *IEEE Robotics and Automation Letters*, *1*(2), 946–953. https://doi.org/10.1109/lra.2016.2527062

[48] Park, H.-W., Wensing, P., & Kim, S. (n.d.). *Online Planning for Autonomous Running Jumps Over Obstacles in High-Speed Quadrupeds*. Retrieved October 31, 2022, from http://www.roboticsproceedings.org/rss11/p47.pdf

[49] Pratt, J., Carff, J., Drakunov, S., & Goswami, A. (2006). Capture Point: A Step toward Humanoid Push Recovery. *Undefined*. https://www.semanticscholar.org/paper/Capture-Point%3A-A-Step-toward-Humanoid-Push-Recovery-Pratt-Carff/9d9aabdfd5b47862bdba0b9bb35711065b673418

[50] Raibert, M. H. (2000). *Legged robots that balance*. Mit P.

[51] Rao, A. (n.d.). *A SURVEY OF NUMERICAL METHODS FOR OPTIMAL CONTROL*. Retrieved October 31, 2022, from http://www.anilvrao.com/Publications/ConferencePublications/trajectorySurveyAAS.pdf

[52] *Release R2022a · mathworks/Simscape-Multibody-Contact-Forces-Library*. (n.d.). GitHub. Retrieved October 31, 2022, from https://github.com/mathworks/Simscape-Multibody-Contact-Forces-Library/releases/tag/22.1.5.1

[53] *Sarah (cheetah)*. (2022, May 22). Wikipedia. https://en.wikipedia.org/wiki/Sarah_(cheetah)#/media/File:Sarah_(cheetah).jpg

[54] Sen, Muhammed & Bakircioglu, Veli & Kalyoncu, Mete. (2017). Inverse Kinematic Analysis of a Quadruped Robot. *International Journal of Scientific & Technology Research*. 6. 285-289.

[55] Sideris, A., & Bobrow, J. E. (n.d.). An efficient sequential linear quadratic algorithm for solving nonlinear optimal control problems. *Proceedings of the 2005, American Control Conference, 2005.* https://doi.org/10.1109/acc.2005.1470308

[56] Srinivas, T., Madhusudhan, A. K. K., Manohar, L., Stephen Pushpagiri, N. M., Ramanathan, K. C., Janardhanan, M., & Nielsen, I. (2021). Valkyrie—Design and Development of Gaits for Quadruped Robot Using Particle Swarm Optimization. *Applied Sciences*, *11*(16), 7458. https://doi.org/10.3390/app11167458

[57] Stellato, B., Banjac, G., Goulart, P., Bemporad, A., & Boyd, S. (2018). OSQP: An Operator Splitting Solver for Quadratic Programs. *2018 UKACC 12th International Conference on Control (CONTROL)*. https://doi.org/10.1109/control.2018.8516834

[58] Sutyasadi, P., & Parnichkun, M. (2020). Push recovery control of quadruped robot using particle swarm optimization based structure specified mixed sensitivity H_2/H_∞ control. *Industrial Robot: The International Journal of Robotics Research and Application*, *47*(3), 423–434. https://doi.org/10.1108/ir-06-2019-0135

[59] Todorov, E., & Weiwei Li. (n.d.). A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems. *Proceedings of the 2005, American Control Conference, 2005.* https://doi.org/10.1109/acc.2005.1469949

[60] Wächter, A. An Interior Point Algorithm for Large-Scale Nonlinear Optimization with Applications in Process Engineering. PhD thesis, Department of Chemical Engineering, Carnegie-Mellon University, Pittsburgh, PA, 2002.

[61] Wächter, A., & Biegler, L. T. (2005). On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, *106*(1), 25–57. https://doi.org/10.1007/s10107-004-0559-y

[62] Winkler, A. W. (2018, May 14). *Optimization-based motion planning for legged robots*. Www.research-Collection.ethz.ch. https://www.research-collection.ethz.ch/handle/20.500.11850/272432

[63] Winkler, A. W., Bellicoso, C. D., Hutter, M., & Buchli, J. (2018). Gait and Trajectory Optimization for Legged Systems Through Phase-Based End-Effector Parameterization. *IEEE Robotics and Automation Letters*, *3*(3), 1560–1567. https://doi.org/10.1109/lra.2018.2798285

Appendix A

Mathematica Code – Forward Kinematics of the quadruped Argos

The coordinate systems of the main body, hip, knee and foot are expressed relative to the inertial frame.

```
Clear["Global`*"]
Rx = {{1,0,0,0},{0 ,Cos[qx],-Sin[qx],0},{0,Sin[qx],Cos[qx],0},{0,0,0,1}};
Ry ={{Cos[qy],0,Sin[qy],0},{0 ,1, 0,0},{-Sin[qy],0,Cos[qy],0},{0,0,0,1}};
Rz ={{Cos[qz],-Sin[qz],0,0},{Sin[qz],Cos[qz],0,0},{0,0,1,0},{0,0,0,1}};
TCoM ={{1,0,0,rx},{0,1,0,ry},{0,0,1,rz},{0,0,0,1}}.Rx.Ry.Rz  ;

T0front={{1,0,0,Lx},{0,1,0,0},{0,0,1,0},{0,0,0,1}};
T0rear={{1,0,0,-Lx},{0,1,0,0},{0,0,1,0},{0,0,0,1}};

TR01 = {{0,0,-1,0},{0,1,0,0},{1,0,0,0},{0,0,0,1}} ;  (*ROTy(-pi/2) *)
TL01 = {{0,0,1,0},{0,1,0,0},{-1,0,0,0},{0,0,0,1}};  (*ROTy(pi/2) *)

TR12 = {{Cos[Q3],-Sin[Q3],0,0},{Sin[Q3],Cos[Q3],0,0},{0,0,1,0},{0,0,0,1}}.
{{1,0,0,l3},{0,1,0,0},{0,0,1,0},{0,0,0,1}}.
{{0,0,1,0},{0,1,0,0},{-1,0,0,0},{0,0,0,1}};  (*ROTz(Q3).TRANSX(l3).ROTy(pi/2) *)
TL12= {{Cos[Q3],-Sin[Q3],0,0},{Sin[Q3],Cos[Q3],0,0},{0,0,1,0},{0,0,0,1}}.
{{1,0,0,l3},{0,1,0,0},{0,0,1,0},{0,0,0,1}}.
{{0,0,-1,0},{0,1,0,0},{1,0,0,0},{0,0,0,1}};  (*ROTz(Q3).TRANSX(l3).ROTy(-pi/2) *)

T23 = {{Cos[Q1],-Sin[Q1],0,0},{Sin[Q1],Cos[Q1],0,0},{0,0,1,0},{0,0,0,1}}.
{{1,0,0,0},{0,1,0,-l1},{0,0,1,0},{0,0,0,1}};(*ROTz(Q1).TRANSy(-l1)*)

T34 ={{Cos[Q2-theta],-Sin[Q2-theta],0,0},{Sin[Q2-theta],Cos[Q2-
theta],0,0},{0,0,1,0},{0,0,0,1}}.{{1,0,0,0},{0,1,0,-l2},{0,0,1,0},{0,0,0,1}} ;
(*ROTz(Q2-theta) .TRANSy(-l2)*)

TR04 = TR01.TR12.T23.T34;
TL04 = TL01.TL12.T23.T34;

TFL0 =TCoM.T0front;  (*Front Left *)
FullSimplify[MatrixForm[TFL0]] ;

XFL0 =FullSimplify[TFL0[[1,4]]];
YFL0 =FullSimplify[TFL0[[2,4]]];
ZFL0 =FullSimplify[TFL0[[3,4]]];

TFLhip = TFL0.TL01.TL12 ;
XFLhip =FullSimplify[TFLhip[[1,4]]];
YFLhip =FullSimplify[TFLhip[[2,4]]];
```

```
ZFLhip =FullSimplify[TFLhip[[3,4]]];

TFLknee = TFLhip .T23;
XFLknee =FullSimplify[TFLknee[[1,4]]];
YFLknee =FullSimplify[TFLknee[[2,4]]];
ZFLknee =FullSimplify[TFLknee[[3,4]]];

TFLe = TFLknee.T34;
XFLe=FullSimplify[TFLe [[1,4]]];
YFLe=FullSimplify[TFLe [[2,4]]];
ZFLe=FullSimplify[TFLe [[3,4]]];

TFR0 =TCoM.T0front; (*Front Right*)
FullSimplify[MatrixForm[TFR0]] ;

XFR0 =FullSimplify[TFR0[[1,4]]];
YFR0 =FullSimplify[TFR0[[2,4]]];
ZFR0 =FullSimplify[TFR0[[3,4]]];

TFRhip = TFR0.TR01.TR12 ;
XFRhip =FullSimplify[TFRhip[[1,4]]];
YFRhip =FullSimplify[TFRhip[[2,4]]];
ZFRhip =FullSimplify[TFRhip[[3,4]]];

TFRknee = TFRhip .T23;
XFRknee =FullSimplify[TFRknee[[1,4]]];
YFRknee =FullSimplify[TFRknee[[2,4]]];
ZFRknee =FullSimplify[TFRknee[[3,4]]];

TFRe = TFRknee.T34;
XFRe=FullSimplify[TFRe [[1,4]]];
YFRe=FullSimplify[TFRe [[2,4]]];
ZFRe=FullSimplify[TFRe [[3,4]]];

TRL0 =TCoM.T0rear; (*Rear Left*)
FullSimplify[MatrixForm[TRL0]] ;

XRL0 =FullSimplify[TRL0[[1,4]]];
YRL0 =FullSimplify[TRL0[[2,4]]];
ZRL0 =FullSimplify[TRL0[[3,4]]];

TRLhip = TRL0.TL01.TL12 ;
XRLhip =FullSimplify[TRLhip[[1,4]]];
YRLhip =FullSimplify[TRLhip[[2,4]]];
ZRLhip =FullSimplify[TRLhip[[3,4]]];
```

```
TRLknee = TRLhip .T23;
XRLknee =FullSimplify[TRLknee[[1,4]]];
YRLknee =FullSimplify[TRLknee[[2,4]]];
ZRLknee =FullSimplify[TRLknee[[3,4]]];

TRLe = TRLknee.T34;
XRLe=FullSimplify[TRLe [[1,4]]];
YRLe=FullSimplify[TRLe [[2,4]]];
ZRLe=FullSimplify[TRLe [[3,4]]];

TRR0 =TCoM.T0rear; (*Rear Right*)
FullSimplify[MatrixForm[TRR0]] ;

XRR0 =FullSimplify[TRR0[[1,4]]];
YRR0 =FullSimplify[TRR0[[2,4]]];
ZRR0 =FullSimplify[TRR0[[3,4]]];

TRRhip = TRR0.TR01.TR12 ;
XRRhip =FullSimplify[TRRhip[[1,4]]];
YRRhip =FullSimplify[TRRhip[[2,4]]];
ZRRhip =FullSimplify[TRRhip[[3,4]]];

TRRknee = TRRhip .T23;
XRRknee =FullSimplify[TRRknee[[1,4]]];
YRRknee =FullSimplify[TRRknee[[2,4]]];
ZRRknee =FullSimplify[TRRknee[[3,4]]];

TRRe = TRRknee.T34;
XRRe=FullSimplify[TRRe [[1,4]]];
YRRe=FullSimplify[TRRe [[2,4]]];
ZRRe=FullSimplify[TRRe [[3,4]]];
```

Appendix B

Mathematica Code – Jacobian Calculation of the three DoF leg

```
(*Right Side*)
(*Foot Position*)
xeR =l1*Sin[th1[t]]+l2*Sin[th1[t]+th2[t]-theta] ;
yeR = l3*Sin[th3[t]]-(l1*Cos[th1[t]]+l2*Cos[th1[t]+th2[t]-theta])*Cos[th3[t]];
zeR = (l1*Cos[th1[t]]+l2*Cos[th1[t]+th2[t]-theta])*Sin[th3[t]]+l3*Cos[th3[t]];
(*Foot Velocity*)
xeR' = FullSimplify[D[xeR,t]] ;
yeR' = FullSimplify[D[yeR,t]] ;
zeR' = FullSimplify[D[zeR,t]] ;
(*Foot Acceleration*)
xeR'' = FullSimplify[D[xeR',t]] ;
yeR'' = FullSimplify[D[yeR',t]] ;
zeR'' = FullSimplify[D[zeR',t]] ;

(*Jacobian*)
j11R= D[xeR',th1'[t]];
j12R= D[xeR',th2'[t]];
j13R= D[xeR',th3'[t]];
j21R= D[yeR',th1'[t]];
j22R= D[yeR',th2'[t]];
j23R= D[yeR',th3'[t]];
j31R= D[zeR',th1'[t]];
j32R= D[zeR',th2'[t]];
j33R= D[zeR',th3'[t]];

(*Jacobian Derivative*)
jd11R= D[j11R,t];
jd12R= D[j12R,t];
jd13R= D[j13R,t];
jd21R= D[j21R,t];
jd22R= D[j22R,t];
jd23R= D[j23R,t];
jd31R= D[j31R,t];
jd32R= D[j32R,t];
jd33R= D[j33R,t];

(*Left Side*)
(*Foot Position*)
xeL =l1*Sin[th1[t]]+l2*Sin[th1[t]+th2[t]-theta] ;
yeL = l3*Sin[th3[t]]-(l1*Cos[th1[t]]+l2*Cos[th1[t]+th2[t]-theta])*Cos[th3[t]];
```

```
zeL = -(l1*Cos[th1[t]]+l2*Cos[th1[t]+th2[t]-theta])*Sin[th3[t]]-l3*Cos[th3[t]];
(*Foot Velocity*)
xeL' = FullSimplify[D[xeL,t]] ;
yeL' = FullSimplify[D[yeL,t]] ;
zeL' = FullSimplify[D[zeL,t]] ;
(*Foot Acceleration*)
xeL'' = FullSimplify[D[xeL',t]] ;
yeL'' = FullSimplify[D[yeL',t]] ;
zeL'' = FullSimplify[D[zeL',t]] ;

(*Jacobian*)
j11L= D[xeL',th1'[t]];
j12L= D[xeL',th2'[t]];
j13L= D[xeL',th3'[t]];
j21L= D[yeL',th1'[t]];
j22L= D[yeL',th2'[t]];
j23L= D[yeL',th3'[t]];
j31L= D[zeL',th1'[t]];
j32L= D[zeL',th2'[t]];
j33L= D[zeL',th3'[t]];

(*Jacobian Derivative*)
jd11L= D[j11L,t];
jd12L= D[j12L,t];
jd13L= D[j13L,t];
jd21L= D[j21L,t];
jd22L= D[j22L,t];
jd23L= D[j23L,t];
jd31L= D[j31L,t];
jd32L= D[j32L,t];
jd33L= D[j33L,t];
```

Appendix C

Simpson's rules

Simpson's rules are several approximations for definite integrals of a function F by evaluating it at the boundaries and the middle point of the segment. This approximation is exact if F is a polynomial up to 3rd degree. Simpson's 1/3 is the most common rule and requires 3 points at each segment. If the discretized segments are equal then the weights alternate between 4/3 and 2/3. Simpson 3/8 rule which requires one more function evaluation gives lower error bounds, but it does not increment the order of the error. Equation (C-89) represents a quadratic curve that is defined at the interval [0, h] [35].

$$v(t) = A + Bt + Ct^2 \tag{C-89}$$

The integral of (C-89) is given at the following equations (assuming $x(0) = 0$):

$$x(h) - \cancel{x(0)} = \int_0^h v(t)dt \tag{C-90}$$

$$x(h) = \int_0^h \left(A + Bt + Ct^2\right)dt \tag{C-91}$$

$$x(h) = Ah + \frac{1}{2}Bh^2 + \frac{1}{3}Ch^3 \tag{C-92}$$

To determine the coefficients A, B, C, three points that belong to the curve $v(t)$ are needed. Choosing the two boundary points and the midpoint of the polynomial $v(t)$ as shown in (C-93) results in coefficient values that depend on the lower the midpoint and the upper point of the polynomial $v(t)$ as shown in (C-94)-(C-96).

$$v(0) = v_L \quad v\left(\frac{h}{2}\right) = v_M \quad v(h) = v_U \tag{C-93}$$

$$A = v_L \tag{C-94}$$

$$Bh = -3v_L + 4v_M - v_U \tag{C-95}$$

$$Ch^2 = 2v_L - 4v_M + 2v_U \tag{C-96}$$

The sum of the weights at (C-95) and (C-96) is equal to zero. Substituting (C-94)-(C-96) at (C-89) gives the Simpson's rule for quadrature:

$$x(h) = \frac{h}{6}\left(v_L + 4v_M + v_U\right) \tag{C-97}$$

Appendix D

Simscape model description

A complete representation of the Simscape model presented in Section 4 is shown in Figure D-1. It consists of the main body which is linked with the four legs and the feet interact with the floor. The body fixed frame which is located at the CoM of the main body is defined relative to an inertial frame.

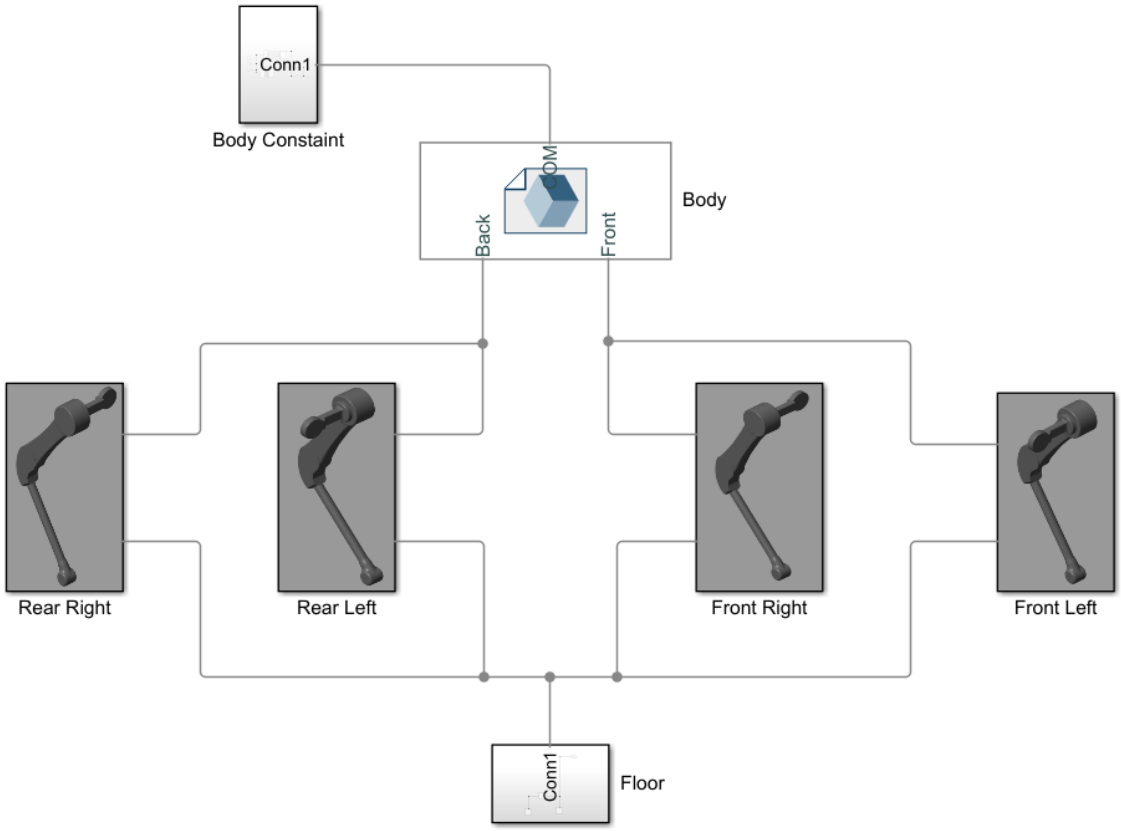


Figure D-1. Simscape Model.

Figure D-2 displays the fundamental building blocks to begin the quadruped's modelling. A bushing force provides a full, six degree-of-freedom (DOF) connector between the main body and the fixed world frame. The solver block defines the solver settings used in simulation. The mechanism configuration defines uniform gravity along y axis. A Transform Sensor passively senses this 3-D time-varying transformation, and its derivatives, between the two frames.

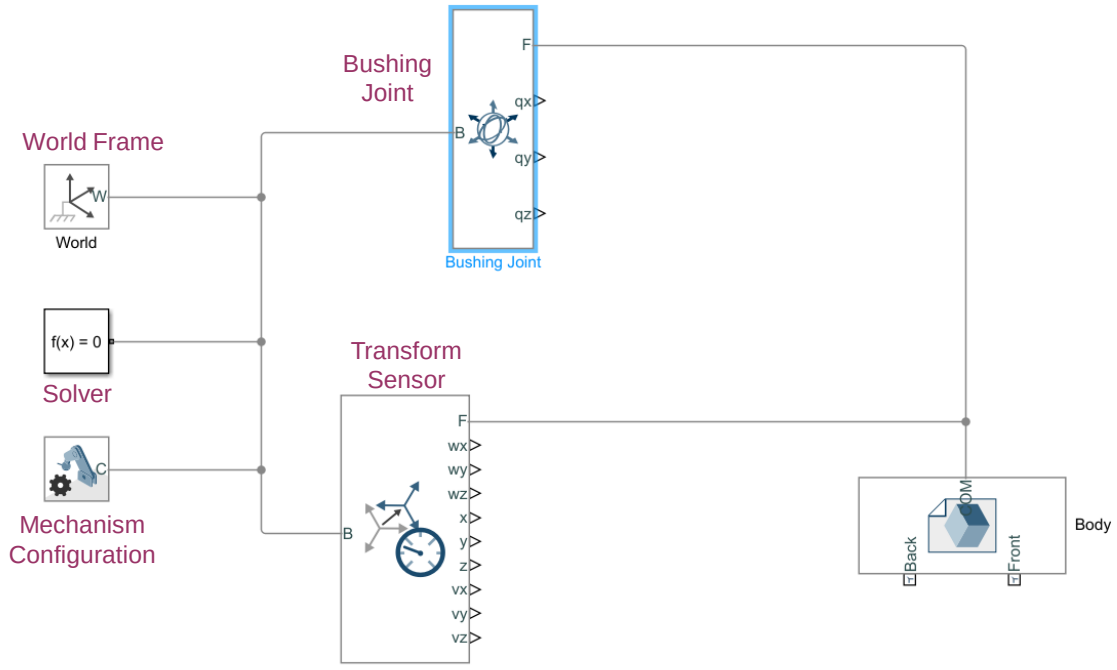


Figure D-2. The main body is connected to the fixed world frame through a bushing joint.

The rigid transform (Figure D-3) defines a fixed 3-D rigid transformation (translation and rotation) between the coordinate systems. Each actuated joint (abduction/hip/knee) is described by a revolute joint block, which takes torque as input and the sensors measurements are the joint angles and the joint velocities. Each link of the leg is imported as STEP file, and it is connected between the relevant coordinate systems.

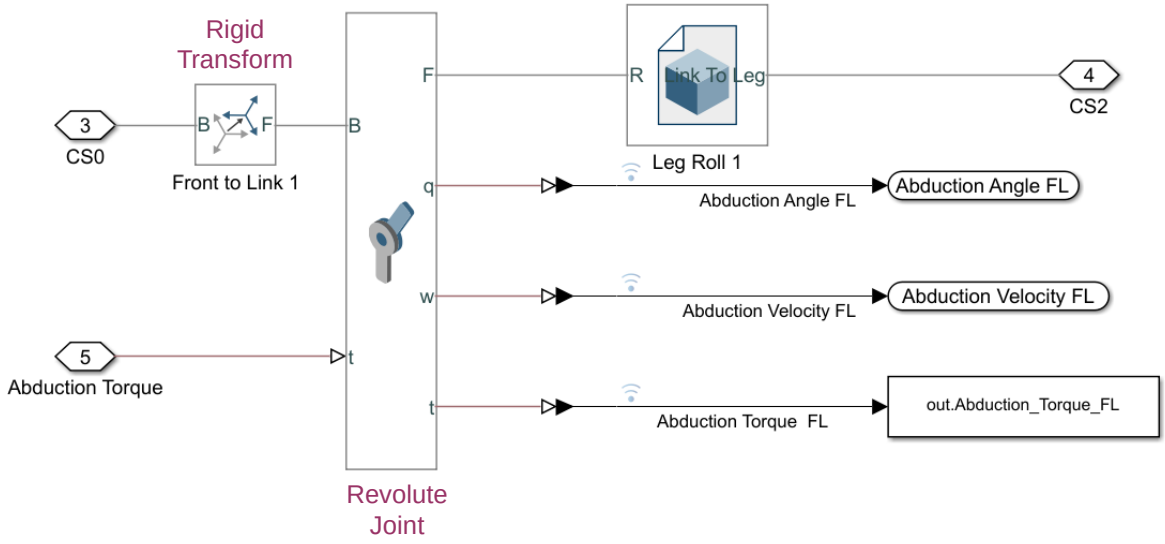


Figure D-3. Actuated joints modelling.

Each of the leg subsystems consists of a planner and the controller (Figure D-4). The blocks of the controller are depicted in Figure D-5 and the Matlab function of the planner is given right after that figure.

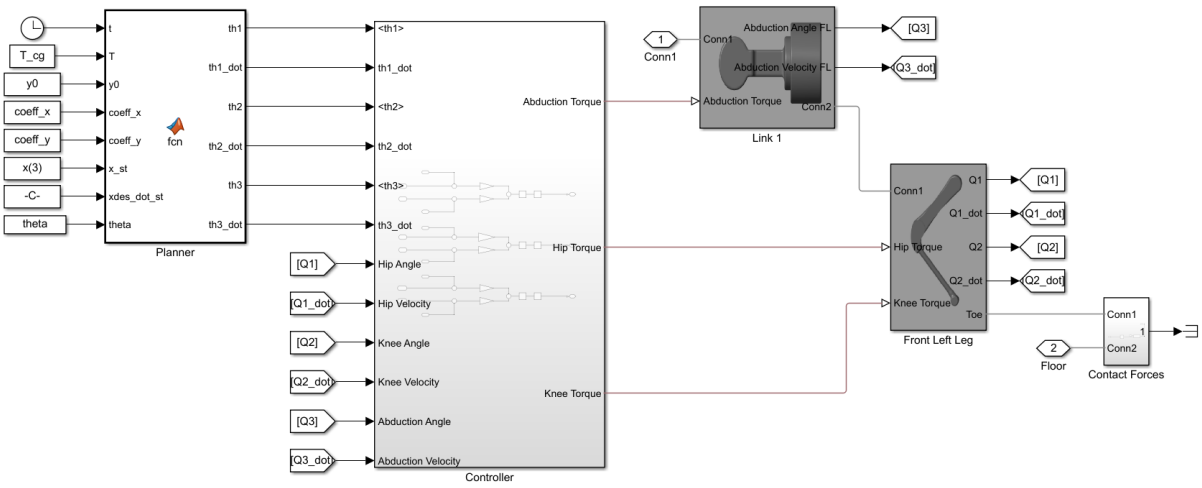


Figure D-4. The planner outputs the desired joint angles and the controller outputs the joint torques.

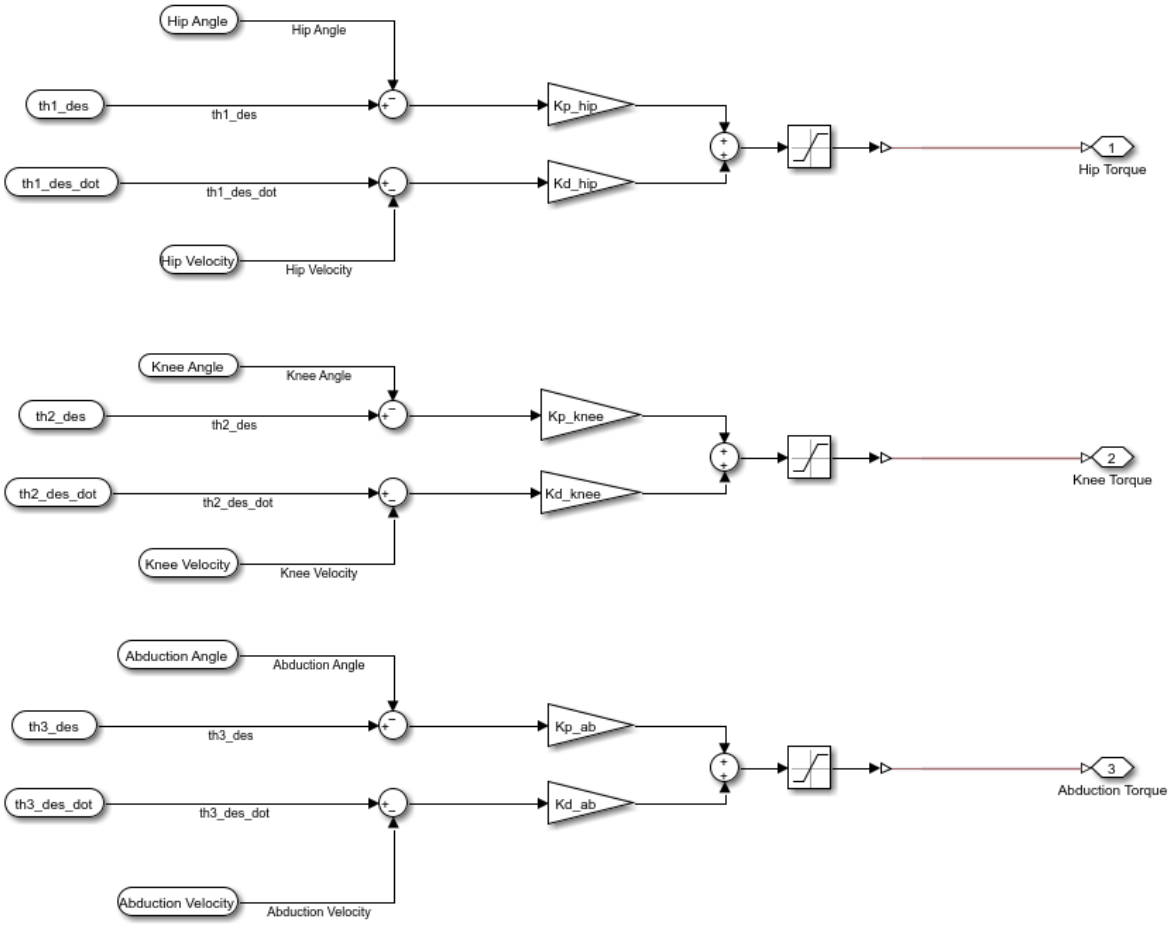


Figure D-5. Implementation of a PD controller.

Matlab Code – Front Left Leg Planner

```
function [th1,th1_dot,th2,th2_dot,th3,th3_dot] =
fcn(t,T,y0,coeff_x,coeff_y,x_st,xdes_dot_st,theta)

% Trajectory Planning
% Parameters
l1 = 0.45034 ;          % Length of Upper Segment
l2 = 0.61763 ;          % Length of Lower Segment

% Trajectory in cartesian space
t = mod(t,T) ;

if t<=T/2               % Swing phase trajectory
    xdes= coeff_x.'* [t^6; t^5; t^4; t^3; t^2;t; 1];
    ydes= coeff_y.'* [t^6; t^5; t^4; t^3; t^2;t; 1];
    xdes_dot= coeff_x(1:end-1).'* [6*t^5; 5*t^4; 4*t^3; 3*t^2; 2* t; 1];
    ydes_dot= coeff_y(1:end-1).'* [6*t^5; 5*t^4; 4*t^3; 3*t^2; 2* t; 1];
else
    t=t-T/2;            % Stance phase trajectory
    xdes = x_st+xdes_dot_st*t;
    ydes=0;
    xdes_dot= xdes_dot_st;
    ydes_dot=0;
end

ydes = ydes-y0;

% Inverse Kinematics
c_th2 = (ydes^2 + xdes^2 - l1^2 - l2^2)/(2*l1*l2);
th2= acos(c_th2)+theta ;
phi = atan2(xdes, -ydes);
psi = atan2(l2*sin(th2-theta),l1+l2*cos(th2-theta));
th1 = phi-psi;

% 2D Jacobian
J = [l1*cos(th1)+l2*cos(th1+th2-theta) l2*cos(th1+th2-theta);
l1*sin(th1)+l2*sin(th1+th2-theta) l2*sin(th1+th2-theta)] ;
th_dot = J\([xdes_dot; ydes_dot]) ;
th1_dot = th_dot(1) ;
th2_dot = th_dot(2) ;

% Abduction joint angle/angular velocity
th3=0;
th3_dot=0;
end
```

The interaction of the foot with the ground is described by a sphere to plane contact model (Figure D-6). This block is included at the Simscape Contact Forces Library and the parameters that are tuned are the contact stiffness (k), the contact damping (b) and the friction coefficients which are discussed in Section 2.5.2. Other parameters to be specified are the sphere radius, which is $0.035m$ and the depth of the contact surface to the reference plane, which is half the height of the 3D ground model. The ground is represented by a $40m \times 0.2m \times 5m$ block. These are the dimensions along x, y and z axis, so the height which is defined along y axis is equal to $0.2m$.

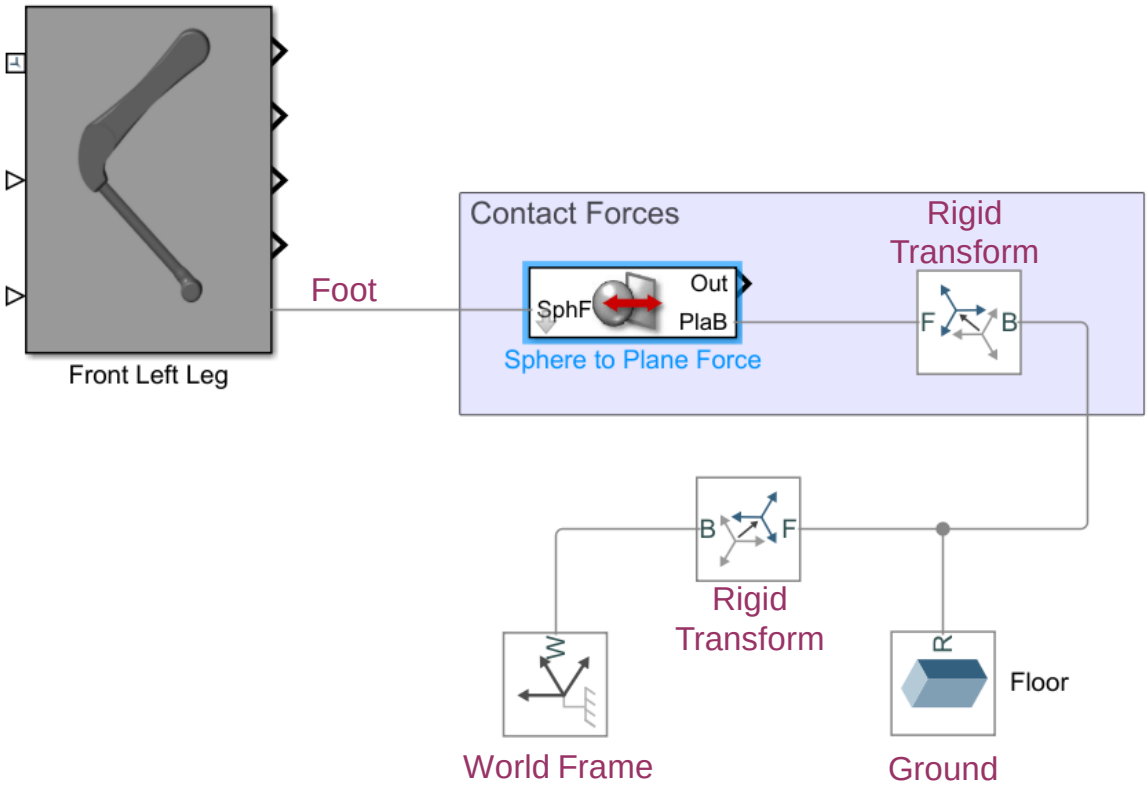


Figure D-6. Contact Forces model for the interaction between the foot and the floor.

Appendix E

Simscape model description

A complete representation of the Simscape model presented in 4.3 is shown in Figure E-1. It consists of the Gait Pattern, which determines the actions of each leg at any moment, the main body which is linked with the four legs and the feet-floor contact model.

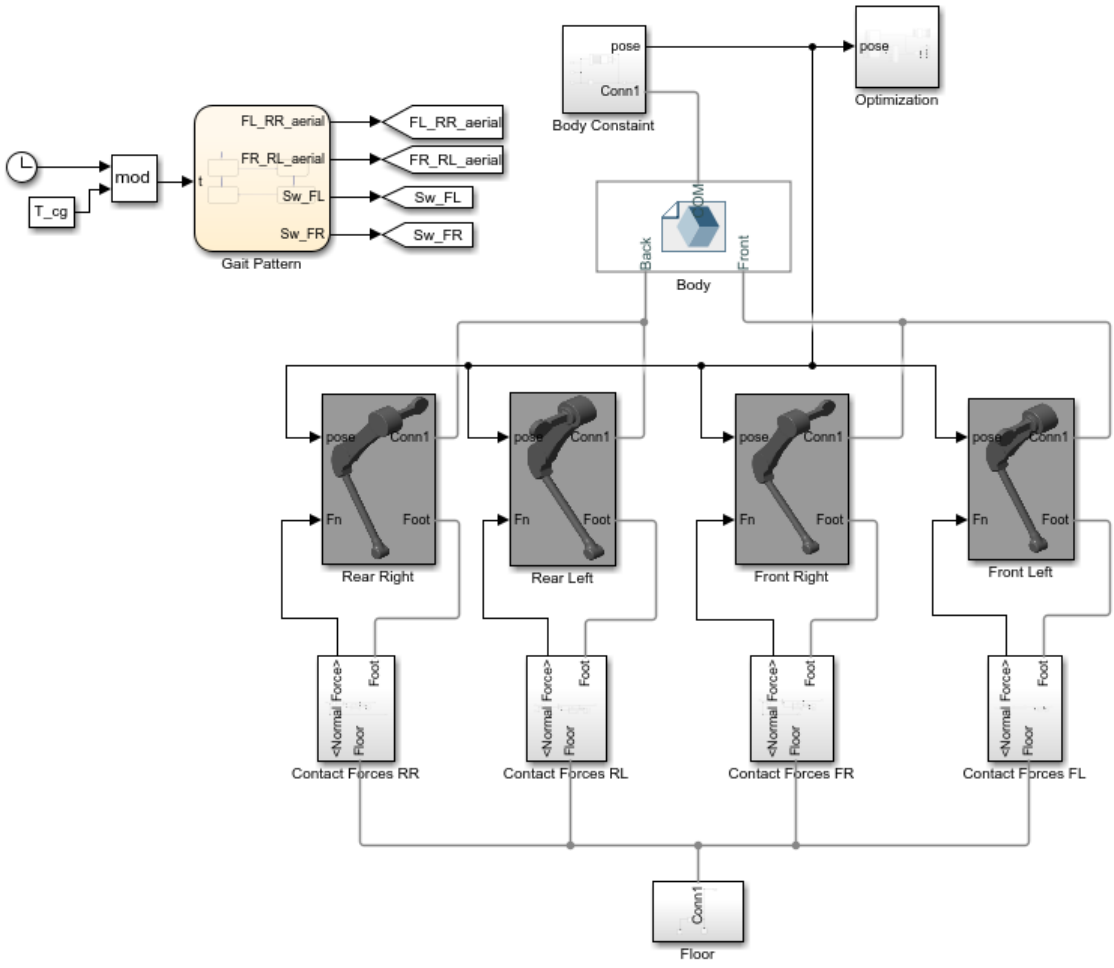


Figure E-1. Simscape Model.

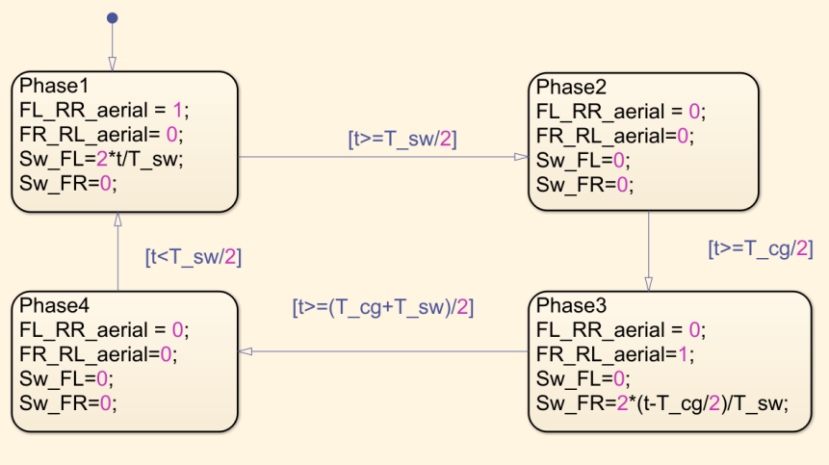


Figure E-2. Gait pattern.

Figure E-2 illustrates the trotting gait pattern and outputs the state of each leg (swing-stance) the current time instance. An overview of a leg subsystem, which includes the links, the connections between them and the control algorithms, is portrayed in Figure E-3.

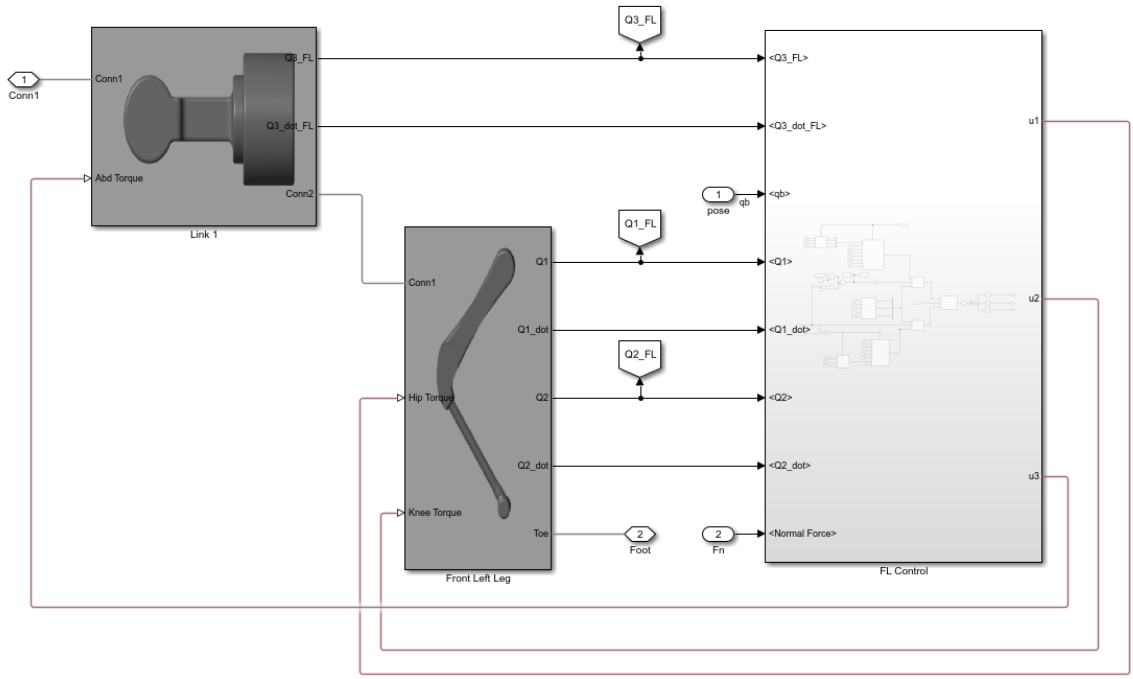


Figure E-3. Leg subsystem.

The control subsystem shown in Figure E-4, Figure E-5, and Figure E-3 incorporates switches that determine the control selection according to the gait pattern and the detected contact forces.

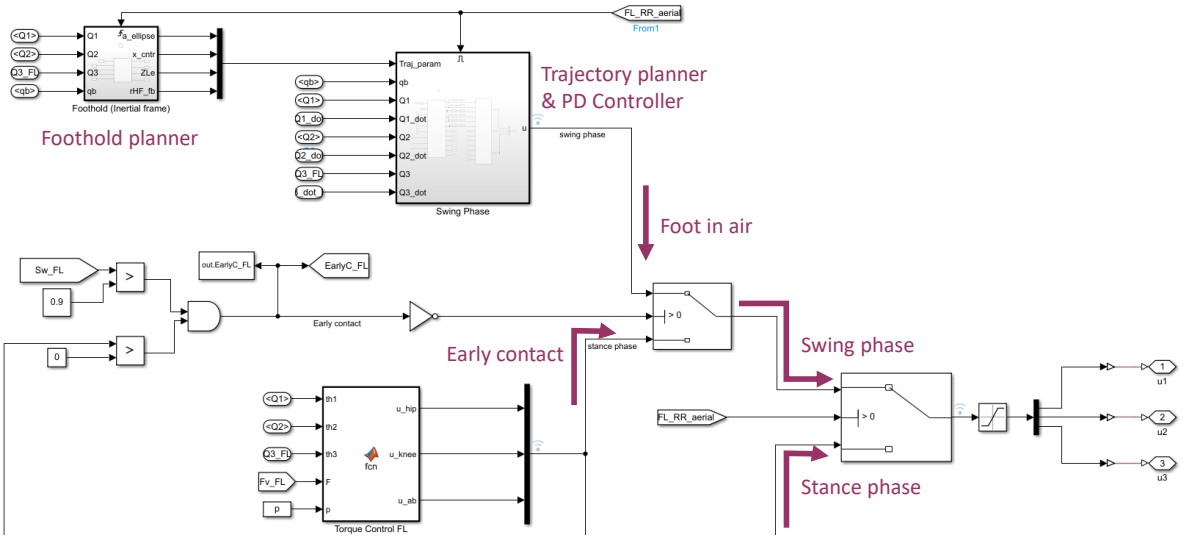


Figure E-4. Controller selection at swing phase.

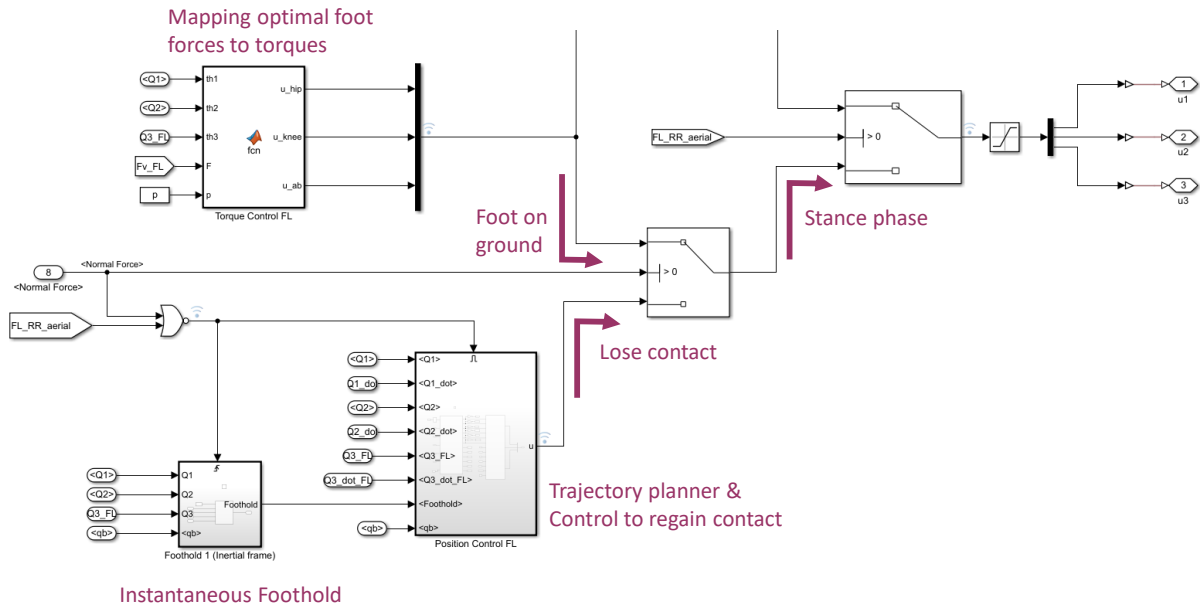


Figure E-5. Controller selection at stance phase.

The following section documents the code for the optimal force distribution which is written in CasADi, but there is an interface with MATLAB/Simscape.

Matlab-CasADi Code – Optimal Force Distribution

```
classdef CasADi_block < matlab.System & matlab.system.mixin.Propagates
    % Quadratic Optimization problem for Force distribution
    % This template includes the minimum set of functions required
    % to define a System object with discrete state.

    % Public, tunable properties
    properties

    end

    properties(DiscreteState)

    end

    % Pre-computed constants
    properties(Access = private)
       % Pre-computed constants.
        casadi_solver;
        lbx;
        ubx;
        ubg;
    end

    methods(Access = protected)
        function num = getNumInputsImpl(~)
            num = 2;
        end
        function num = getNumOutputsImpl(~)
            num = 1;
        end
        function dt1 = getOutputDataTypeImpl(~)
             dt1 = 'double';
        end
        function dt1 = getInputDataTypeImpl(~)
             dt1 = 'double';
        end
        function sz_1 = getOutputSizeImpl(~)
            sz_1 = [12,1];
        end
        function [sz_1,sz_2] = getInputSizeImpl(~)
             sz_1 = [6,12];
            sz_2 = [6,1];
        end
        function cp1 = isInputComplexImpl(~)
             cp1 = false;
        end
        function cp1 = isOutputComplexImpl(~)
             cp1 = false;
        end
        function fz1 = isInputFixedSizeImpl(~)
             fz1 = true;
        end
        function fz1 = isOutputFixedSizeImpl(~)
             fz1 = true;
        end
        function setupImpl(obj,~,~)
            % Perform one-time calculations, such as computing constants
```

```
            import casadi.*

            x = SX.sym( 'x', 12) ;

            S = [1 1 1 20 5 20];
            W = 0.00001*ones(1,12);
            Asys = SX.sym('Asys',6,12);
            Fdes = SX.sym('Fdes',6,1);

            % Objective Function
            J= S*(Asys*x-Fdes).^2+W*x.^2 ;

            % Friction cone constraints
            A = zeros(16,12);
            c = [1 -0.8 0; -1 -0.8 0; 0 -0.8 1; 0 -0.8 -1];
            A(1:4,1:3)=c; A(5:8,4:6)=c; A(9:12,7:9)=c; A(13:16,10:12)=c;
            con=A*x ;

            % NLP
            qp = struct( 'x' ,x , 'f' ,J, 'g', con, 'p', [Asys, Fdes]);

%             opts = struct;
%             opts.osqp.eps_abs = 1e-02;
%             opts.osqp.eps_prim_inf = 1e-02;
%             opts.osqp.eps_dual_inf =1e-02;
%             opts.osqp.alpha = 1.9;

            % Solver
%             solver = qpsol('solver', 'osqp', qp, opts);
          solver = qpsol('solver', 'qpoases', qp);

            % Bounds
            lbx(1:12) = -inf;
            lbx(2:3:12) = 2;
            ubx(1:12) = 1500;
            ubg(:) = 0;

            % Initialization
            obj.casadi_solver = solver;
            obj.lbx = lbx;
            obj.ubx = ubx;
            obj.ubg = ubg;

        end

        function Fopt = stepImpl(obj,Asys,Fdes)
            % Implement algorithm

            lbx = obj.lbx;
            ubx = obj.ubx;
            ubg = obj.ubg;

            % Initialization
%             x0 = zeros(12,1);
%             x0(2:3:6) = Fdes(2)/4;
%             x0(8:3:12) = Fdes(2)/4;

            solver = obj.casadi_solver;
            sol = solver('lbx', lbx, 'ubx', ubx,'ubg', ubg, 'p', [Asys, Fdes]);
```

```
%             sol = solver('x0', x0, 'lbx', lbx, 'ubx', ubx,'ubg', ubg, 'p',
[Asys, Fdes]);
            Fopt = full(sol.x(:));
        end

        function resetImpl(~)
            % Initialize / reset discrete-state properties
        end
    end
end
```

Appendix F

Matlab-CasADi Code – Trajectory Optimization

```
clear;
close all
clc;
% Specify the path of CasAdi library
addpath("C:\Users\...\casadi-windows-matlabR2016a-v3.5.5")
import casadi.*

%% Robot Parameters & initial conditions
L = 0.8 ;          % Whole Body length
y0 = 0.9231;       % Robot's height at nominal position
vx0 = 0.3;         % Initial velocity in x direction
% Feet's nominal position relative to the CoM
p_nom = [0.42; -0.9231; -0.23; 0.42; -0.9231; 0.23; -0.44; -0.9231; -0.23; -0.44;
-0.9231; 0.23];

m = 52.96;         % Robot's total mass
% Moments of inertia[kg*m^2]:
% Taken at the center of mass and aligned with the output coordinate system
I = [3.7223 0.6859 0; 0.6859 8.1901 0; 0 0 8.8015];

% Environment
g = 9.81;                  % Gravity acceleration
friction_coef = 0.8;       % Friction coefficient

%% Optimization setup
% Discretization
h = 0.3;                   % Timestep
ns = 4;                    % Number of steps
sw_segm=2;                 % 2 segments for swing phase
st_segm=4;                 % 4 segments for stance phase
N = ns*(sw_segm+st_segm); % Number of segments
T_sim = N*h;               % Total simulation time

% System Dynamics
r = MX.sym('r',3);
rdot = MX.sym('rdot',3);
th = MX.sym('th',3);
w = MX.sym('w',3);
pos = MX.sym('pos',12);
fi = MX.sym('fi',12);

states = [r; rdot; th; w];
n_states = length(states);

rdd = 1/m*[sum(fi(1:3:end));sum(fi(2:3:end));sum(fi(3:3:end))]-g*[0;1;0];
thdot = [cos(th(3))*sec(th(2)) sin(th(3))*sec(th(2)) 0; -sin(th(3)) cos(th(3)) 0;
cos(th(3))*tan(th(2)) sin(th(3))*tan(th(2)) 1]*w;
ext_torque = MX.zeros(3,1);
for i=1:4
    ext_torque = ext_torque+([0 -fi(i*3) fi(i*3-1); fi(i*3) 0 -fi(i*3-2); -fi(i*3-
1) fi(i*3-2) 0]*[r(1)-pos(i*3-2); r(2)-pos(i*3-1); r(3)-pos(i*3)]);
end
```

```
wd = I\(ext_torque-[0 -w(3) w(2); w(3) 0 -w(1); -w(2) w(1) 0]*I*w);

rhs = [rdot;rdd;thdot;wd]; % right hand system

% Nonlinear mapping function f(x,u)
f = Function('f',{states,pos,fi},{rhs});

Pos = MX.sym('Pos',12,(2*N+1));   % Decision variables (controls)
Fi = MX.sym('Fi',12,(2*N+1));     % Decision variables (controls)

% Parameters (which include the initial state and the reference state)
P = MX.sym('P',n_states + n_states);

% A vector that represents the states over the optimization problem
X = MX.sym('X',n_states,(2*N+1));

obj = 0; % Objective function
g = [];  % Constraints vector

C0 = [0; 1; 1; 0]; % Starting phase for each leg (FL,FR,RL,RR)
% 1 indicates contact (stance phase), 0 indicates swing phase.
% In this case, FL and RR legs start in swing phase

xdes = zeros(1,2*N+1);
for i=1:2*N+1
    xdes(i) = L/2+T_sim*vx0*(i-1)/(2*N) ;
end

st = X(:,1);                    % Initial state
g = [g;st-P(1:n_states)];       % Initial condition constraints

Q = 1*eye(n_states,n_states);  % Weighing matrices (states)
Q(2,2) = 1e-4;
Q(4,4) = 0;
R = 1e-3*eye(4,4);


for k=1:2:2*N-1
       st = X(:,k);
    st_middle = X(:,k+1);
    st_next = X(:,k+2);
    fy = Fi(2:3:end,k);
    fy_middle = Fi(2:3:end,k+1);
    fy_next = Fi(2:3:end,k+2);

    obj_k = (st-[xdes(k); P(n_states+2:2*n_states)])'*Q*(st-[xdes(k);
P(n_states+2:2*n_states)])+fy'*R*fy;
    obj_mid = (st_middle-[xdes(k+1); P(n_states+2:2*n_states)])'*Q*(st_middle-
[xdes(k+1); P(n_states+2:2*n_states)])+fy_middle'*R*fy_middle;
    obj_next = (st_next-[xdes(k+2); P(n_states+2:2*n_states)])'*Q*(st_next-
[xdes(k+2); P(n_states+2:2*n_states)])+fy_next'*R*fy_next;

%     obj_k = (st-[xdes(k); P(n_states+2:2*n_states)])'*Q*(st-[xdes(k);
P(n_states+2:2*n_states)]);
%     obj_mid = (st_middle-[xdes(k+1); P(n_states+2:2*n_states)])'*Q*(st_middle-
[xdes(k+1); P(n_states+2:2*n_states)]);
%     obj_next = (st_next-[xdes(k+2); P(n_states+2:2*n_states)])'*Q*(st_next-
[xdes(k+2); P(n_states+2:2*n_states)]);
```

```
    obj = obj+h/6*(obj_k+4*obj_mid+obj_next); % calculate obj

    st_middle_col = 1/2*(st+st_next)+h/8*(f(st,Pos(:,k),Fi(:,k))-
f(st_next,Pos(:,k+2),Fi(:,k+2)));
    st_next_col=
st+h/6*(f(st,Pos(:,k),Fi(:,k))+4*f(st_middle,Pos(:,k+1),Fi(:,k+1))+f(st_next,Pos(:
,k+2),Fi(:,k+2)));
    g = [g;st_middle-st_middle_col; st_next-st_next_col]; % compute constraints

end

for k=1:2*N+1
%     Rotqx = [1 0 0; 0 cos(X(7,k)) -sin(X(7,k)); 0 sin(X(7,k)) cos(X(7,k))];
%     Rotqy = [cos(X(8,k)) 0 sin(X(8,k)); 0 1 0; -sin(X(8,k)) 0 cos(X(8,k))];
%     Rotqz = [cos(X(9,k)) -sin(X(9,k)) 0; sin(X(9,k)) cos(X(9,k)) 0; 0 0 1];
%     Rotq = (Rotqx*Rotqy*Rotqz)'; %Rotation matrix from the world frame to the
base frame
%     g = [g; kron(eye(4),Rotq)*(Pos(:,k)-repmat(X(1:3,k),4,1))-p_nom] ;  % Foot
position should be inside a cube centered at nominal foot position
    g = [g; Pos(:,k)-repmat(X(1:3,k),4,1)-p_nom] ;  % Foot position should be
inside a cube centered at nominal foot position
end

% Foot Position constraints at stance phase
st_points = zeros(12,2*N+1);
for j=1:4
    if C0(j)==0
        for ind_st=[1:2,st_segm/2+2*sw_segm+1:st_segm/2+2*sw_segm+2*(st_segm-1)]
         g = [g;reshape(Pos(3*(j-1)+1:3*j,ind_st:2*(sw_segm+st_segm):2*N)-
Pos(3*(j-1)+1:3*j,ind_st+1:2*(sw_segm+st_segm):2*N+1),3*ns,1)];
         st_points(3*(j-1)+2,ind_st:2*(sw_segm+st_segm):2*N+1)=1;
        end
    else
        for ind_st=1:2*st_segm
         g = [g;reshape(Pos(3*(j-1)+1:3*j,ind_st:2*(sw_segm+st_segm):2*N)-
Pos(3*(j-1)+1:3*j,ind_st+1:2*(sw_segm+st_segm):2*N+1),3*ns,1)];
;
         st_points(3*(j-1)+2,ind_st:2*(sw_segm+st_segm):2*N+1)=1;
        end
    end
end

sw_points = zeros(12,2*N+1);
for j=1:4
    if C0(j)==1
        for ind_sw=1:2*sw_segm+1
        sw_points(3*(j-1)+1:3*j,2*st_segm+ind_sw:2*(sw_segm+st_segm):2*N+1)=1;
        end
    else
        for ind_sw=1:2*sw_segm+1
        sw_points(3*(j-1)+1:3*j,st_segm/2+ind_sw:2*(sw_segm+st_segm):2*N+1)=1;
        end
    end
end

% Forces should be inside the friction cone
for k=1:2*N+1
    for j=1:4
```

```
        g = [g;reshape(Fi(3*(j-1)+1:2:3*(j-1)+3,k)-friction_coef*Fi(3*(j-
1)+2,k)*DM.ones(2,1),2,1);reshape(-Fi(3*(j-1)+1:2:3*(j-1)+3,k)-
friction_coef*Fi(3*(j-1)+2,k)*DM.ones(2,1),2,1)];
    end
end

OPT_variables =
[reshape(X,12*(2*N+1),1);reshape(Pos,12*(2*N+1),1);reshape(Fi,12*(2*N+1),1)];

nlp_prob = struct('f', obj, 'x', OPT_variables, 'g', g, 'p', P);

opts = struct;
opts.ipopt.max_iter = 2000;
opts.ipopt.print_level =5;
opts.print_time = 0;
opts.ipopt.acceptable_tol =1e-6;
opts.ipopt.acceptable_obj_change_tol = 1e-6;

solver = nlpsol('solver', 'ipopt', nlp_prob, opts);

args = struct;

% Equality & Inequality constraints
p_offest_x = 0.4;
p_offest_y = 0.05;
p_in_offest_z = 0.03;
p_out_offest_z = 0.2;

args.lbg(1:length(g)) = 0;  % -1e-20  % Equality constraints
args.ubg(1:length(g)) = 0;  % 1e-20   % Equality constraints
args.lbg(n_states*(2*N+1)+1:3:(n_states+12)*(2*N+1)) = -p_offest_x;
args.ubg(n_states*(2*N+1)+1:3:(n_states+12)*(2*N+1)) = p_offest_x;
% The leg should not extend too much to avoid singularities
args.lbg(n_states*(2*N+1)+2:3:(n_states+12)*(2*N+1)) = -p_offest_y;
% The leg should not retract too much to avoid high knee torques
args.ubg(n_states*(2*N+1)+2:3:(n_states+12)*(2*N+1)) = p_offest_y;
% Constraint along z axis for FR&RR legs
args.lbg(n_states*(2*N+1)+6:6:(n_states+12)*(2*N+1)) = -p_in_offest_z;
args.ubg(n_states*(2*N+1)+6:6:(n_states+12)*(2*N+1)) = p_out_offest_z;
% Constraint along z axis for FL&RL legs
args.lbg(n_states*(2*N+1)+3:6:(n_states+12)*(2*N+1)) = -p_out_offest_z;
args.ubg(n_states*(2*N+1)+3:6:(n_states+12)*(2*N+1)) = p_in_offest_z;
% Fricition cone inequality constraints
args.lbg(length(g)-4*4*(2*N+1)+1:end) = -inf;  % 1e-20


% Descision variables constraints
args.lbx(1:length(OPT_variables),1)= -inf;
args.ubx(1:length(OPT_variables),1)= inf;
% CoM position along y at the final point shall be equal to the nominal height
% args.lbx(n_states*(2*N+1)-10,1)= y0;
args.ubx(n_states*(2*N+1)-8,1)= 0;                          % The forward velocity
is set to 0 at the end
args.lbx(5:12:n_states*(2*N+1),1)= -0.3;                    % Velocity along y
axis (Lower bound)
args.ubx(5:12:n_states*(2*N+1),1)= 0.3;                     % Velocity along y
axis (Upper bound)
args.lbx(n_states*(2*N+1)+2:3:(n_states+12)*(2*N+1),1)= 0;  % Feet position shall
be positive in y direction
```

```
args.ubx(n_states*(2*N+1)+find(st_points),1)= 0;            % Feet position shall
be zero in y direction in stance phase
args.lbx(end-12*(2*N+1)+2:3:end,1)= 0;                      % Feet force along y
axis shall be positive (push the ground)
args.lbx(end-10,1)= 200;                                    % Feet force along y
axis at final point for FL
args.lbx(end-1,1)= 200;                                     % Feet force along y
axis at final point for RR
args.lbx(end-12*(2*N+1)+find(sw_points),1)= 0;              % Feet force shall be
zero in stance phase
args.ubx(end-12*(2*N+1)+find(sw_points),1)= 0;              % Feet force shall be
zero in stance phase

%% THE SIMULATION LOOP SHOULD START FROM HERE
%-----------------------------------------

x0 = [L/2; y0; 0; vx0; 0; 0; 0; 0; 0; 0; 0; 0];         % State initial conditions
xs = [L/2+T_sim*vx0; y0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0]; % Reference posture

X0 = repmat(x0,1,2*N+1)';   % Initialization of the states decision variables

X0(:,1) = xdes' ;


p0 = repmat(x0(1:3),4,1)+p_nom;
ps = repmat(xs(1:3),4,1)+p_nom;

P0 = repmat(p0,1,2*N+1)';   % Initialization of the feet's positions

for i=1:2*N+1
    for j=1:3:10
    P0(i,j) = p0(j,1)+(ps(j,1)-p0(j,1))*(i-1)/(2*N) ;
    end
end


for j=1:4
    if C0(j)==0
        Py = [0; 0; 0; 0.1; 0.2; 0.1; 0; 0; 0; 0; 0; 0];
    else
        Py = [0; 0; 0; 0; 0; 0; 0; 0; 0; 0.1; 0.2; 0.1];
    end
    P0(:,3*(j-1)+2) = [repmat(Py,ns,1);0] ;
end


F0 = zeros(2*N+1,12);

for j=1:4
    if C0(j)==0
        Fy = [200; 200; 0; 0; 0; 0; 0; 200; 300; 300; 300; 200];
    else
        Fy = [0; 200; 300; 300; 300; 300; 300; 200; 0; 0; 0; 0;];
    end
    F0(:,3*(j-1)+2) = [repmat(Fy,ns,1);Fy(1)] ;
end

% Initial value of the optimization variables
args.x0  = [reshape(X0',n_states*(2*N+1),1);reshape(P0',12*(2*N+1),1);...
```

```
    reshape(F0',12*(2*N+1),1)];
args.p   = [x0;xs]; % Set the values of the parameters vector

tic
sol = solver('x0', args.x0, 'lbx', args.lbx, 'ubx', args.ubx,...
    'lbg', args.lbg, 'ubg', args.ubg,'p',args.p);
toc


CoM_pos(1:n_states,:)= reshape(full(sol.x(1:n_states*(2*N+1)))',n_states,2*N+1); %
Get CoM's trajectory from the solution
foot_pos =
reshape(full(sol.x(n_states*(2*N+1)+1:(n_states+12)*(2*N+1)))',12,2*N+1); % Get
feet's positions from the solution
foot_force =
reshape(full(sol.x((n_states+12)*(2*N+1)+1:(n_states+24)*(2*N+1)))',12,2*N+1); %
Get feet's positions from the solution

%% Interpolate the solution on a uniform grid for plotting and animation:
tGrid = zeros(1,2*N+1);
for i=1:2*N+1
    tGrid(i) = (i-1)*T_sim/(2*N) ;
end
xGrid = CoM_pos;
uGrid = [foot_pos; foot_force];
fGrid=zeros(n_states,2*N+1);

for i=1:2*N+1
    pos=uGrid(1:12,i);
    fi=uGrid(13:24,i);
    r=xGrid(1:3,i);
    rdot=xGrid(4:6,i);
    th=xGrid(7:9,i);
    w=xGrid(10:12,i);

    g = 9.81; % gravity [m/s^2]
    rdd = 1/m*[sum(fi(1:3:end));sum(fi(2:3:end));sum(fi(3:3:end))]-g*[0;1;0];
    thdot = [cos(th(3))*sec(th(2)) sin(th(3))*sec(th(2)) 0; -sin(th(3)) cos(th(3))
0; cos(th(3))*tan(th(2)) sin(th(3))*tan(th(2)) 1]*w;
    ext_torque = zeros(3,1);
    for i=1:4
        ext_torque = ext_torque+([0 -fi(i*3) fi(i*3-1); fi(i*3) 0 -fi(i*3-2); -
fi(i*3-1) fi(i*3-2) 0]*[r(1)-pos(i*3-2); r(2)-pos(i*3-1); r(3)-pos(i*3)]);
    end
    wd = I\(ext_torque-[0 -w(3) w(2); w(3) 0 -w(1); -w(2) w(1) 0]*I*w);

    fGrid(:,i) = [rdot;rdd;thdot;wd]; % state derivative
end

soln.interp.state = @(t)( pwPoly3(tGrid,xGrid,fGrid,t) );
soln.interp.control = @(t)(pwPoly2(tGrid,uGrid(13:end,:),t));
soln.interp.footpos = @(t)(pwPoly5(tGrid,uGrid(1:12,:),t,C0));

tc1 = linspace(tGrid(1),tGrid(end),1000);

% State Interpolation
xc1 = soln.interp.state(tc1);       % Main body's pose
fc1 = soln.interp.control(tc1);     % Foot forces
pc1 = soln.interp.footpos(tc1);     % Foot position
```

```
% uc1 = [pc1;fc1];

% Mean forward velocity
vx_mean = mean(xc1(4,:));

%% Functions

function x = pwPoly3(tGrid,xGrid,fGrid,t)
nGrid = length(tGrid);
if mod(nGrid-1,2)~=0 || nGrid < 3
    error('The number of grid-points must be odd and at least 3');
end

% Figure out sizes
n = floor((length(tGrid)-1)/2);
m = size(xGrid,1);
k = length(t);
x = zeros(m, k);

% Figure out which segment each value of t should be on
edges = [-inf, tGrid(1:2:end), inf];
[~,~,bin] = histcounts(t,edges);

% Loop over each quadratic segment
for i=1:n
    idx = bin==(i+1);
    if sum(idx) > 0
        kLow = 2*(i-1) + 1;
        kUpp = kLow + 2;
        h = tGrid(kUpp)-tGrid(kLow);
        xLow = xGrid(:,kLow);
        fLow = fGrid(:,kLow);
        xUpp = xGrid(:,kUpp);
        fUpp = fGrid(:,kUpp);
        alpha = t(idx) - tGrid(kLow);
        x(:,idx) = cubicInterp(h,xLow, fLow, xUpp, fUpp,alpha);
    end
end

% Replace any out-of-bounds queries with NaN
outOfBounds = bin==1 | bin==(n+2);
x(:,outOfBounds) = nan;

% Check for any points that are exactly on the upper grid point:
if sum(t==tGrid(end))>0
    x(:,t==tGrid(end)) = xGrid(:,end);
end

end

function x = cubicInterp(h,xLow, fLow, xUpp, fUpp,del)
%
% This function computes the interpolant over a single interval
%
% INPUTS:
%   h = time step (tUpp-tLow)
%   xLow = function value at tLow
%   fLow = derivative at tLow
%   xUpp = function value at tUpp
```

```
%   fUpp = derivative at tUpp
%   del = query points on domain [0, h]
%
% OUTPUTS:
%   x = [m, p] = function at query times
%

%%% Fix matrix dimensions for vectorized calculations
nx = length(xLow);
nt = length(del);
xLow = xLow*ones(1,nt);
fLow = fLow*ones(1,nt);
xUpp = xUpp*ones(1,nt);
fUpp = fUpp*ones(1,nt);
del = ones(nx,1)*del;

a = (2.*(xLow-xUpp)+h.*(fLow + fUpp))./(h.^3);
b = -(3.*(xLow-xUpp)+h.*(2.*fLow + fUpp))./(h.^2);
c = fLow;
d = xLow;

x = d + del.*(c + del.*(b + del.*a));

end

function x = pwPoly2(tGrid,xGrid,t)
% x = pwPoly2(tGrid,xGrid,t)
%
% This function does piece-wise quadratic interpolation of a set of data,
% given the function value at the edges and midpoint of the interval of
% interest.
%
% INPUTS:
%   tGrid = [1, 2*n+1] = time grid, knot idx = 1:2:end
%   xGrid = [m, 2*n+1] = function at each grid point in tGrid
%   t = [1, k] = vector of query times (must be contained within tGrid)
%
% OUTPUTS:
%   x = [m, k] = function value at each query time
%
% NOTES:
%   If t is out of bounds, then all corresponding values for x are replaced
%   with NaN
%

nGrid = length(tGrid);
if mod(nGrid-1,2)~=0 || nGrid < 3
    error('The number of grid-points must be odd and at least 3');
end

% Figure out sizes
n = floor((length(tGrid)-1)/2);
m = size(xGrid,1);
k = length(t);
x = zeros(m, k);

% Figure out which segment each value of t should be on
edges = [-inf, tGrid(1:2:end), inf];
[~,~,bin] = histcounts(t,edges);
```

```
% Loop over each quadratic segment
for i=1:n
    idx = bin==(i+1);
    if sum(idx) > 0
        gridIdx = 2*(i-1) + [1,2,3];
        x(:,idx) = quadInterp(tGrid(gridIdx),xGrid(:,gridIdx),t(idx));
    end
end

% Replace any out-of-bounds queries with NaN
outOfBounds = bin==1 | bin==(n+2);
x(:,outOfBounds) = nan;

% Check for any points that are exactly on the upper grid point:
if sum(t==tGrid(end))>0
    x(:,t==tGrid(end)) = xGrid(:,end);
end

end


function x = quadInterp(tGrid,xGrid,t)
%
% This function computes the interpolant over a single interval
%
% INPUTS:
%   tGrid = [1, 3] = time grid
%   xGrid = [m, 3] = function grid
%   t = [1, p] = query times, spanned by tGrid
%
% OUTPUTS:
%   x = [m, p] = function at query times
%

% Rescale the query points to be on the domain [-1,1]
t = (t-tGrid(1))/(tGrid(3)-tGrid(1)) ;

% Compute the coefficients:
a = 2*(xGrid(:,3) + xGrid(:,1)) - 4*xGrid(:,2);
b = -3*xGrid(:,1)+4*xGrid(:,2)-xGrid(:,3);
c = xGrid(:,1);

% Evaluate the polynomial for each dimension of the function:
p = length(t);
m = size(xGrid,1);
x = zeros(m,p);

tt = t.^2;
for i=1:m
    x(i,:) = a(i)*tt + b(i)*t + c(i);
end

end

function x = pwPoly5(tGrid,xGrid,t,C0)
% x = pwPoly5(tGrid,xGrid,t,C0)
%
% This function does piece-wise quartic interpolation of a set of data,
```

```
% given the function value at the edges and 3 evenly spaced of the interval of
% interest.
%
% INPUTS:
%   tGrid = [1, 2*n+1] = time grid, knot idx = 1:2:end
%   xGrid = [m, 2*n+1] = function at each grid point in tGrid
%   t = [1, k] = vector of query times (must be contained within tGrid)
%
% OUTPUTS:
%   x = [m, k] = function value at each query time
%
% NOTES:
%   If t is out of bounds, then all corresponding values for x are replaced
%   with NaN
%
nGrid = length(tGrid);
if mod(nGrid-1,2)~=0 || nGrid < 5
    error('The number of grid-points must be odd and at least 5');
end

% Figure out sizes
n = floor((length(tGrid)-1)/2);
m = size(xGrid,1);
k = length(t);
x = zeros(m, k);

% Figure out which segment each value of t should be on
edges = [-inf, tGrid(1:2:end), inf];
[~,~,bin] = histcounts(t,edges); % Bin sorts the query times in the appropriate
segment

% Loop over each segment
for j=1:4
    for i=1:n
        idx = bin==(i+1); % Logical variable which indicated which query times
belong to the current segment
        if sum(idx) > 0
            if C0(j)==0
                if  (mod(bin(find(idx,1))-3,6)==0 || mod(bin(find(idx,1))-3,6)==1)
                    kLow = fix(bin(find(idx,1))/6)*12+3;
                    kUpp = kLow + 4;
                    h = tGrid(kUpp)-tGrid(kLow);
                    x1 = xGrid(3*(j-1)+1:3*j,kLow);
                    x2 = xGrid(3*(j-1)+1:3*j,kLow+1);
                    x3 = xGrid(3*(j-1)+1:3*j,kLow+2);
                    x4 = xGrid(3*(j-1)+1:3*j,kLow+3);
                    x5 = xGrid(3*(j-1)+1:3*j,kUpp);
                    alpha = t(idx) - tGrid(kLow);
                    x(3*(j-1)+1:3*j,find(idx)) =
quarticInterp(h,x1,x2,x3,x4,x5,alpha);
                else
                    kstance = 2*i-1;
                    x(3*(j-1)+1:3*j,find(idx)) = repmat(xGrid(3*(j-
1)+1:3*j,kstance),1,sum(idx));
                end
            else
                if  mod(bin(find(idx,1)),6)==0 || mod(bin(find(idx,1)),6)==1
                    kLow = fix(bin(find(idx,1))/6)*12-3;
                    kUpp = kLow + 4;
```

```
                    h = tGrid(kUpp)-tGrid(kLow);
                    x1 = xGrid(3*(j-1)+1:3*j,kLow);
                    x2 = xGrid(3*(j-1)+1:3*j,kLow+1);
                    x3 = xGrid(3*(j-1)+1:3*j,kLow+2);
                    x4 = xGrid(3*(j-1)+1:3*j,kLow+3);
                    x5 = xGrid(3*(j-1)+1:3*j,kUpp);
                    alpha = t(idx) - tGrid(kLow);
                    x(3*(j-1)+1:3*j,find(idx)) =
quarticInterp(h,x1,x2,x3,x4,x5,alpha);
                else
                    kstance = 2*i-1;
                    x(3*(j-1)+1:3*j,find(idx)) = repmat(xGrid(3*(j-
1)+1:3*j,kstance),1,sum(idx));
                end
            end
        end
    end
end

% Replace any out-of-bounds queries with NaN
outOfBounds = bin==1 | bin==(n+2);
x(:,outOfBounds) = nan;

% Check for any points that are exactly on the upper grid point:
if sum(t==tGrid(end))>0
    x(:,t==tGrid(end)) = xGrid(:,end);
end

end

function x = quarticInterp(h,x1,x2,x3,x4,x5,del)
%
% This function computes the interpolant over a single interval
%
% INPUTS:
%   h = time step (tUpp-tLow)
%   del = query points on domain [0, h]
%
% OUTPUTS:
%   x = [m, p] = function at query times
%

%%% Fix matrix dimensions for vectorized calculations
nx = length(x1);
nt = length(del);
x1 = x1*ones(1,nt);
x2 = x2*ones(1,nt);
x3 = x3*ones(1,nt);
x4 = x4*ones(1,nt);
x5 = x5*ones(1,nt);
del = ones(nx,1)*del;

a = (32/3*x1-128/3*x2+64*x3-128/3*x4+32/3*x5)./(h.^4);
b = (-80/3*x1+96*x2-128*x3+224/3*x4-16*x5)./(h.^3);
c = (70/3*x1-208/3*x2+76*x3-112/3*x4+22/3*x5)./(h.^2);
d = (-25/3*x1+16*x2-12*x3+16/3*x4-x5)./h;
e = x1;
x = e+del.*(d + del.*(c + del.*(b + del.*a)));
end
```