Modelling and in-flight torso attitude stabilization of a jumping quadruped.

National Technical University
School of Mechanical Engineering

Department of Mechanical Design & Automatic Control

Author: Michail Papadakis
Supervisor: Prof. Ioannis Poulakakis
Co-supervisor: Prof. Kostas Alexis

Athens, July 2024

Abstract

This thesis contributes to the modelling, simulation and attitude control of the jumping legged robot *Olympus*, designed for Martian lava tube exploration. The robot is currently being developed at the Autonomous Robot Lab (A.R.L.) at the Norwegian University of Science and Technology (N.T.N.U.).

The first step involved the detailed kinematic and dynamic modelling of the quadruped. Special attention was given to handling the robot's workspace constraints, as limb movements during reorientation manoeuvres are close to its torso.

Next, a hierarchical model based attitude controller was developed for the robot. The first module, the Body Planner, optimizes the torso trajectory to track a reference orientation based on virtual torques applied on a single rigid body that has comparable inertia to the quadruped. The second control module, the Leg Planner, attempts to track these virtual torques for only one leg while respecting its workspace and input constraints. To achieve high solution speeds and online calculation of the optimal trajectories, a switching strategy is utilized to update various controller parameters through a Finite State Machine (FSM)-based resetting approach. Finally, an allocation strategy projects the optimized joint trajectory of the one leg to the others. This projection ensures that collisions between legs cannot occur, cancels parasitic torques and lowers the overall computational cost of the controller as only one optimization problem must be solved to track the virtual torques.

The performance of the controller is evaluated in a high fidelity simulation. The robot is able to stabilize 90° single axis orientation manoeuvres in 6 seconds for roll, 2.5 seconds for pitch, and 5.5 seconds for yaw in free floating conditions. Additionally, a parametric study investigated the effect of increasing the paw and torso mass. The proposed controller is deployed on the actual robot. In the first experiment the robot achieves 90° single axis turns in 4s for roll and 7s for yaw. In the second test, the robot manages to track changing orientation references.

The outcome of this thesis is a high fidelity, configurable simulation framework for further development of *Olympus* and an attitude controller that manages to stabilize the desired orientation while respecting the robot's operational constraints.

Περίληψη

Αυτή η διπλωματική εργασία συμβάλλει στη μοντελοποίηση, τα εργαλεία προσομοίωσης και τον έλεγχο προσανατολισμού του αλτικού τετράποδου ρομπότ *Olympus*, το οποίο έχει σχεδιαστεί για εξερεύνηση υπόγειων σπηλιών με λάβα στον Άρη. Το ρομπότ αναπτύσσεται επί του παρόντος στο Autonomous Robot Lab (A.R.L.) του Norwegian University of Science and Technology (N.T.N.U.).

Το πρώτο βήμα περιλάμβανε τη λεπτομερή κινηματική και δυναμική μοντελοποίηση του τετράποδου ρομπότ. Ιδιαίτερη προσοχή δόθηκε στη διαχείριση των περιορισμών του χώρου εργασίας του ρομπότ, καθώς οι κινήσεις των άκρων κατά τη διάρκεια των κινήσεων επαναπροσανατολισμού είναι κοντά στον κορμό του.

Στη συνέχεια, αναπτύχθηκε ένας ιεραρχικός βάσει μοντέλου ελεγκτής προσανατολισμού για το ρομπότ. Το πρώτο υποσύστημα, ο Ελεγκτής Σώματος, βελτιστοποιεί την τροχιά του κορμού για να επιτύχει έναν επιθυμητό προσανατολισμό βάσει εικονικών ροπών που εφαρμόζονται σε ένα άκαμπτο σώμα με αδράνεια παρόμοια με του τετράποδου. Το δεύτερο υποσύστημα ελέγχου, ο Ελεγκτής Ποδιού, προσπαθεί να παράξει αυτές τις εικονικές ροπές για μόνο ένα πόδι, σεβόμενο παράλληλα τους περιορισμούς του χώρου εργασίας και των κινητήρων του. Για να επιτευχθούν υψηλές ταχύτητες λύσης και ο υπολογισμός των βέλτιστων τροχιών σε πραγματικό χρόνο, χρησιμοποιείται μια στρατηγική εναλλαγής (switching strategy) για την ανανέωση διαφόρων παραμέτρων του ελεγκτή μέσω μιας στρατιγικής επαναφοράς βασισμένης σε Μηχανή Πεπερασμένων Καταστάσεων (Finite State Machine). Τέλος, η βελτιστοποιημένη τροχιά των αρθρώσεων του ενός ποδιού προβάλλεται στα υπόλοιπα με κατάλληλο τρόπο ώστε να παραχθεί μια συγχρονισμένη και ασφαλής κίνηση. Αυτή η προβολή εξασφαλίζει ότι δεν μπορούν να συμβούν συγκρούσεις μεταξύ των ποδιών, ακυρώνει τις παρασιτικές ροπές και μειώνει το συνολικό υπολογιστικό κόστος του ελεγκτή, καθώς μόνο ένα πρόβλημα βελτιστοποίησης πρέπει να λυθεί για να επιτευχθούν οι εικονικές ροπές.

Η απόδοση του ελεγκτή αξιολογείται σε προσομοίωση υψηλής πιστότητας. Το ρομπότ είναι σε θέση να σταθεροποιήσει περιστροφές 90° γύρω από έναν άξονα σε 6 δευτερόλεπτα για κλίση (roll), 2,5 δευτερόλεπτα για πρόνευση (pitch) και 5,5 δευτερόλεπτα για εκτροπή (yaw) σε συνθήκες ελεύθερης αιώρησης. Επιπλέον, μια παραμετρική μελέτη διερεύνησε την επίδραση της αύξησης της μάζας των ποδιών και του κορμού. Ο προτεινόμενος ελεγκτής εφαρμόστηκε στο πραγματικό ρομπότ σε δύο πειράματα. Στο πρώτο πείραμα, το ρομπότ επιτυγχάνει περιστροφή 90° σε 4 δευτερόλεπτα για κλίση και 7 δευτερόλεπτα για εκτροπή. Στη δεύτερη δοκιμή, το ρομπότ καταφέρνει να ακολουθήσει μεταβαλλόμενους επιθυμητούς προσανατολισμούς.

Το αποτέλεσμα της διπλωματικής εργασίας είναι ένα υψηλής πιστότητας, παραμετροποιήσιμο εργαλείο προσομοίωσης για περαιτέρω ανάπτυξη του *Olympus* και ένας ελεγκτής προσανατολισμού που καταφέρνει να σταθεροποιήσει τον επιθυμητό προσανατολισμό, εξασφαλίζοντας παράλληλα την ασφάλεια του ρομπότ.

Acknowledgements

I would like to express my gratitude and appreciation to professor K. Alexis for providing me with the opportunity to work on this project and visit the ARL lab during my thesis. The expertise, professionalism and organization of the lab was truly inspiring. A special thanks goes to J.A.Olsen for his guidance and support throughout this year and especially during the experiments and M.Harms and W.Rehberg for their counselling. I would also like to thanks professor I.Poulakakis for his insightful feedback and for agreeing to supervise me in co-operation with K.Alexis.

I would like to express my sincere gratitude to my parents, Dimitris and Marina, and my sister Stella for their love and encouragement throughout my whole life. Without them, none of what I have accomplished would have been possible. A special thanks is owed to my sister for tolerating me in my worst throughout my studies. Also, i would like to thank my girlfriend, Sofia, who was always supportive and her love and attitude was always refreshing and was giving me strength.

I would like to thank all my friends for their personal support. They have made my time outside of the university enjoyable and filled with great memories. I would also like to thank the people from the collective in Trondheim, for welcoming me and providing me with memorable experiences. Finally, a special thanks goes to Leuteris and Antonis Kantonias who always believed in me and also inspired me. The last years of my studies would be much harder and less impactful without them.

Contents

List of Figures

List of Tables

List of Algorithms

1. Introduction

1.1 Motivation

Martian lava tubes are of particular interest due to their scientific value and practical advantages. These lava tubes provide access to pristine bedrock and potential materials, and are ideal locations for conducting seismic investigations. They are also theorized to contain ice water. Additionally, the environment of lava tubes is stable, characterized by minimal temperature variations, shielding from cosmic radiation, and protection from regolith dust. These attributes make lava tubes an excellent choice for in-situ laboratories, enable larger missions, and reduce the need for protective payloads [1].

Exploring these tubes poses a significant challenge. Their topological complexity, characterized by uneven terrain, numerous obstacles, and collapsed sections, hinders wheeled exploration. Additionally, UAVs have limited payload capacity and/or limited flight time to explore these large caves. On the other hand, legged systems are an excellent choice, having the capability to traverse unstructured environments and the versatility to accomplish complex tasks. In particular, jumping legged systems can leverage the low gravity to achieve complex traversal capabilities, such as jumping over large obstacles, crossing gaps, or entering these caves from skylight entrances [2].

This thesis aims to model and develop a controller for in-air stabilization of such a jumping quadruped, Olympus, developed at the Autonomous Robot Lab (ARL) in National Technical University of Norway (NTNU).

1.2 Literature Review

Over the last decade, there have been huge leaps in the capabilities of quadrupedal robots. Commercially available quadrupedal robots, such as ANYmal and Spot, have demonstrated promising results conducting missions in harsh, unstructured, and difficult-to-traverse environments on Earth, such as underground complexes and mines [3, 4]. At the same time, these systems have become more agile and capable of performing more aggressive manoeuvres. Quadrupeds have successfully executed back-flips [5] and barrel rolls [6] by utilizing model predictive control (MPC) based architectures to track offline generated trajectories. Additionally, reinforcement learning techniques have further enhanced the locomotion robustness of these systems, enabling ANYmal to navigate deformable terrain, rubble, thick vegetation, and gushing water [7]. This progress has motivated the development of such systems for planetary exploration. Several prominent results are presented below:

- *Spacebok* was one of the first quadrupedal robots designed for space exploration. A four-bar parallel motion linkage with parallel springs, enabling it to jump up to 1.3 meters. Additionally, a reaction wheel is used to control its pitch orientation during the flight phase, with a PID controller [8]. In later iterations, *Spacebok* incorporated a deep reinforcement learning-based controller to stabilize its attitude solely by moving its limbs. It has the capability to re-orient itself safely for a safe landing after a jump and is able to stabilize its attitude in under six seconds [9].

- *Spacehopper* is a three-legged, CubeSat-sized, and lightweight robot specifically designed for controlled low-gravity locomotion. It employs a deep reinforcement learning controller to control its orientation. In simulations, it can reorient itself in one second in zero-gravity, while experimental results on Earth demonstrated a five-second settling time [10].

- *Olympus* is the quadrupedal robot developed at NTNU, which employs a five bar mechanism for its legs with springs for improved jumping. A two-leg prototype could reach jump heights of 1.5 meters in Earth's gravity and 3.63 meters in Mars' gravity [11]. The first iteration of *Olympus* employs a hierarchical controller to control its attitude, where an MPC generates appropriate body torques and a torque allocation algorithm determines the suitable movements for the legs. Simulation results demonstrate its ability to stabilize its attitude on average in three seconds [12].

1.3 Contributions

Despite the significant advancements in quadrupedal robotics for planetary exploration, there are still a lot of unexplored areas. The model based controller used in [12] does not account for operational constraints, such as self-collision avoidance. As reorientation movements take place near the torso, this compromises the safety of the controller. Also, it relies on a heuristic algorithm to generate the necessary body torques from the leg movements, which does not guarantee optimal performance. While reinforcement learning can achieve high-performance results, guaranteeing its reliability, which is crucial for deploying robots in space, remains challenging. Therefore,

there is merit in further developing model based controllers to execute dynamic and agile manoeuvres and enable quadrupedal robots to contribute to planetary exploration.

This thesis first contributes to the development of a simulation framework for *Olympus*, with detailed kinematic modelling and handling of self collisions. Also, a cascaded MPC approach is investigated. Although the proposed controller does not surpass its predecessor in terms of performance, it successfully stabilizes the robot's orientation while avoiding collisions. The proposed controller is also experimentally tested to verify its performance.

In detail, the contributions are listed below:

- Forward and inverse analytical kinematic analysis of Olympus's legs (C++ kinematics library).
- Detailed dynamic modelling of Olympus's legs.
- Optimized collision modelling to allow the full range of motion of the legs and at the same time handle contacts in real time in simulation.
- Mathematical description of the workspace collision constraints.
- Development of a simulation framework in Drake which interfaces with Robot Operating System (ROS) for testing new controllers (written in C++).
- Development of a hierarchical controller that stabilizes the attitude of the robot.
- Investigation of various aspects of the architecture of the proposed controller (e.g. orientation error metric, tuning procedure and insights, supporting modules) to aid in its further development.
- Evaluation of the proposed controller in simulation.
- Evaluation of the proposed controller in experiments.

The code from the thesis can be accessed in:

- `olympus_tools`: `https://github.com/ntnu-arl/olympus_tools`. It contains the state estimation, gui and attitude control (`mpc_controller`) packages as well as the kinematics library.
- `olympus_simulation`: `https://github.com/ntnu-arl/olympus_simulation`. It contains the simulation code.

1.4 Olympus Overview

The quadruped *Olympus* is shown in figure 1.

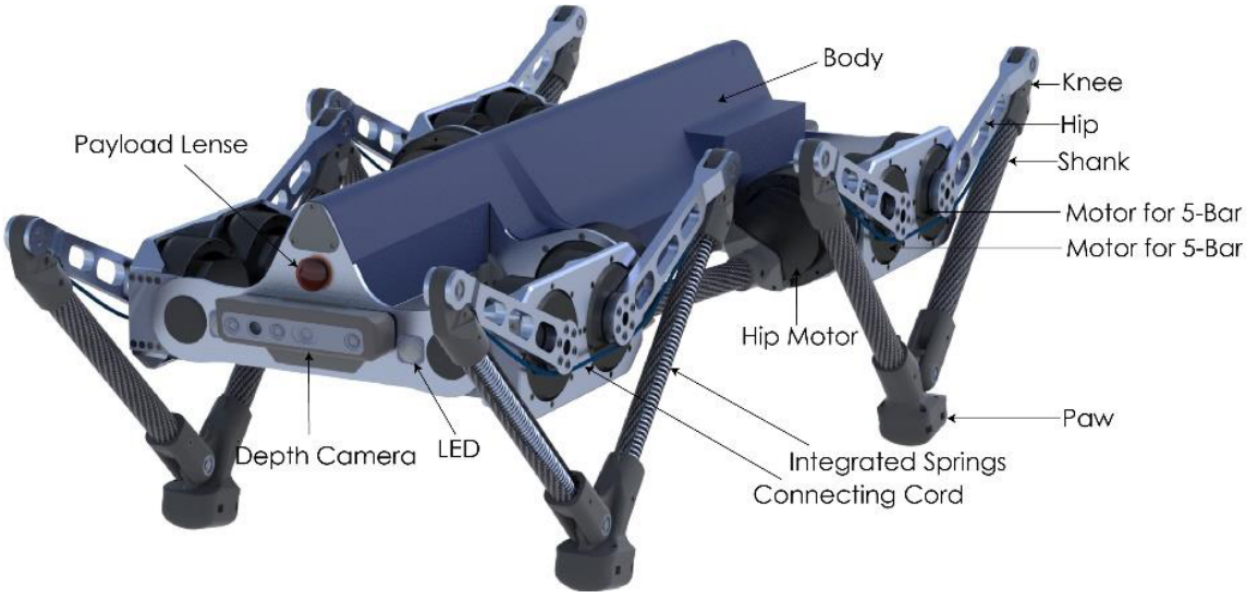


Figure 1: Overview of robot *Olympus*.

The robot's distinctive features include a five-bar leg design with integrated springs connected by high-strength cords. Various design parameters, including spring stiffness and body size, are optimized to enhance its reorientation and jumping capabilities. An overview of such design parameters[1] is presented in table 1. The actuators of the system are brush-less DC motors (BLDC) with integrated encoders and planetary gearboxes. Details about the selected motors are presented in table 2. The adduction/abduction joints utilize AK 80-9 actuators, complemented by AK70-10 actuators in the hip joints. For computation, it employs an Nvidia OrinX onboard computer, which communicates via four different CAN buses with each leg. The energy autonomy of the system is ensured by a 48V battery pack. Finally, it is equipped with exteroceptive sensors such as an IMU and a depth camera.

Table 1: Basic design parameters of Olympus.

Quantity	**Value**	**Leg Part**	**Mass**	**Length**
Total Height	570 mm	**Torso Weight**	5.68 kg	-
Total Width	400 mm	**Leg Weight**	2.017 kg	-
Total Length	679 mm	**Motor housing**	1.437kg	-
Total Weight	13.747 kg	**Hip Link**	81g	175 mm
Spring Resting Length	0.175 m	**Shank Link**	199g	300 mm
Spring Stiffness	800 N/m	**Paw**	20 g	-

Table 2: Characteristics of Olympus's motors.

	AK 80-9	**AK 70-10**
Internal gear ratio	9:1	10:1
Weight	485 g	521 g
Mechanical Time Constant	0.94 ms	0.74 ms
Torque (Rated/Maximum)	9 / 18 Nm	8.3 / 24.8 Nm
Rated Speed	390 rpm	310 rpm

1.5 Thesis Structure

The thesis is structured as follows:

- Chapter 2 offers a comprehensive theoretical foundation covering topics from rigid body dynamics, quaternions, Lagrangian dynamics, and optimal control. It also presents the primary tools employed in the thesis.
- Chapter 3 is concerned with the mathematical modelling of the system. The robot kinematics and dynamics are presented along with the handling of self collisions. Also, important modelling implementation details are discussed.
- Chapter 4 provides an in-depth presentation of the control design, including information on various modules and an overview of the entire architecture.
- Chapter 5 presents the main simulation results, describes the setup of the experiment that took place and presents the obtained data. In addition, the experimental results are compared with the simulations.
- Finally, chapter 6 concludes the thesis, discussing the strengths and weaknesses of the proposed methods and offering recommendations for future extensions of the current work.

[1]The springs are not connected between fixed points, but rather pull on each other through the cord. Thus the presented values are a linear spring approximation which is being used.

2. Background

This chapter provides a brief theoretical introduction to important aspects of the thesis. It begins with key concepts of rigid body rotational dynamics, followed by an introduction to quaternions and their properties. Next, an overview of the Euler-Lagrange formalism is presented, with a focus on constrained systems. Subsequently, fundamental ideas from optimal control and model predictive control theory are discussed. Finally, the software tools utilized in this thesis are presented.

Before delving into the specific topics, some notation conventions used throughout the thesis are introduced. The position of point B as expressed in frame $\{A\}$ is:

$$^{A}\mathbf{P}_{B}$$

Similarly, the orientation of frame $\{B\}$ with respect to frame $\{A\}$ is:

$$^{A}\mathbf{R}_{B}$$

The relative position of point B to point C as expressed in frame $\{A\}$ is:

$$^{A}\mathbf{P}_{B}^{C}$$

A skew symmetric matrix is symbolized with:

$$[\mathbf{v}^{\times}] = \begin{bmatrix} 0 & -v_z & v_y \\ v_z & 0 & -v_x \\ -v_y & v_x & 0 \end{bmatrix}$$

Finally, a $n \times m$ matrix filled with ones is symbolized with $\mathbf{1}_{n\times m}$.

2.1 Rigid Body Rotational Dynamics

First, the angular momentum $\mathbf{H}$ of a rigid body in an inertial frame is defined:

$$\mathbf{H} = \mathbf{I}\boldsymbol{\omega} \tag{1}$$

where $\mathbf{I}$ is the inertial tensor of the rigid body calculated at the inertial frame and $\boldsymbol{\omega}$ is the angular velocity vector expressed in the inertial frame. It must be noted, that $\mathbf{I}$ is not constant and depends on the orientation of the body.

In an inertial frame, Newton's second law of rotation states [13]:

$$\left.\frac{d\mathbf{H}}{dt}\right|_{i} = \boldsymbol{\tau}_{ext} \tag{2}$$

where $\boldsymbol{\tau}_{ext}$ are the external torques acting on the body. The subscript i is used because the time derivative is with respect to axes that do not rotate with the body. The time derivative of $\mathbf{H}$ is:

$$\left.\frac{d\mathbf{H}}{dt}\right|_{i} = \dot{\mathbf{I}}\boldsymbol{\omega} + \mathbf{I}\dot{\boldsymbol{\omega}} \tag{3}$$

The expression above can be further simplified. If $\mathbf{I}_B$ is the inertia tensor calculated at a body frame, $\mathbf{I}_I$ is the inertial tensor at an inertial frame and $\mathbf{R} = {}^{I}\mathbf{R}_{B}$ is the rotational matrix that represents the orientation of the body frame in the inertial frame, then the following is true [14] :

$$\mathbf{I}_I = \mathbf{R}\mathbf{I}_B\mathbf{R}^T \tag{4}$$

As $\mathbf{I}_B$ is calculated in a body frame, it is constant. So differentiating the above expression yields:

$$\dot{\mathbf{I}}_I = \dot{\mathbf{R}}\mathbf{I}_B\mathbf{R}^T + \mathbf{R}\mathbf{I}_B\dot{\mathbf{R}}^T \tag{5}$$

Using $\dot{\mathbf{R}} = [\boldsymbol{\omega}]^{\times}\mathbf{R}$ the above expression becomes:

$$\dot{\mathbf{I}}_I = [\boldsymbol{\omega}^{\times}]\mathbf{R}\mathbf{I}_B\mathbf{R}^T + \mathbf{R}\mathbf{I}_B([\boldsymbol{\omega}^{\times}]\mathbf{R})^T$$

$$\dot{\mathbf{I}}_I = [\boldsymbol{\omega}^{\times}]\mathbf{I}_I - \mathbf{I}_I[\boldsymbol{\omega}^{\times}]$$

So, the first term in equation 3 becomes:

$$\dot{\mathbf{I}}\boldsymbol{\omega} = \left([\boldsymbol{\omega}^{\times}]\mathbf{I} - \mathbf{I}[\boldsymbol{\omega}^{\times}]\right)\boldsymbol{\omega}$$

As $[\boldsymbol{\omega}^{\times}]\boldsymbol{\omega} = 0$, the above expression simplifies to the following.

$$\dot{\mathbf{I}}\boldsymbol{\omega} = [\boldsymbol{\omega}^{\times}]\mathbf{I}\boldsymbol{\omega} = \boldsymbol{\omega} \times \mathbf{I}\boldsymbol{\omega}$$

Thus (3) becomes:

$$\boldsymbol{\omega} \times \mathbf{I}\boldsymbol{\omega} + \mathbf{I}\dot{\boldsymbol{\omega}} = \boldsymbol{\tau}_{ext} \tag{6}$$

Expression (6) holds in an inertial frame as all the quantities $(\boldsymbol{\omega}, \boldsymbol{\tau}_{ext}, \mathbf{I})$ are specified in the inertial frame. To overcome this, at each timestep, an inertial frame is created that coincides with the body frame. Equation (6) holds in this inertial frame. Also, $\mathbf{I} = \mathbf{I}_B$ and $\boldsymbol{\tau}_{ext} = {}^{B}\boldsymbol{\tau}_{ext}$.

Angular velocities between two different fixed frames are related using the expression below:

$${}^{I}\boldsymbol{\omega} = \mathbf{R}\ {}^{B}\boldsymbol{\omega}$$

Differentiating the above expression gives:

$${}^{I}\dot{\boldsymbol{\omega}} = \mathbf{R}\ {}^{B}\dot{\boldsymbol{\omega}} + [{}^{I}\boldsymbol{\omega}^{\times}]\mathbf{R}\ {}^{B}\boldsymbol{\omega}$$

As $\mathbf{R}\ {}^{B}\dot{\boldsymbol{\omega}} = {}^{I}\dot{\boldsymbol{\omega}}$ and ${}^{I}\boldsymbol{\omega} \times {}^{I}\boldsymbol{\omega} = \mathbf{0}$ angular accelerations between two different fixed frames are related using the expression below:

$${}^{I}\dot{\boldsymbol{\omega}} = R\ {}^{B}\dot{\boldsymbol{\omega}}$$

As at each timestep, the inertial frame is chosen to be coincident with the body frame, $\mathbf{R}$ is the identity matrix and thus ${}^{I}\boldsymbol{\omega} = {}^{B}\boldsymbol{\omega}$ and ${}^{I}\dot{\boldsymbol{\omega}} = {}^{B}\dot{\boldsymbol{\omega}}$. So, (6) is transformed to the following:

$$\boldsymbol{\omega} \times \mathbf{I}_B\boldsymbol{\omega} + \mathbf{I}_B\dot{\boldsymbol{\omega}} = {}^{B}\boldsymbol{\tau}_{ext} \tag{7}$$

Equation (7) is the Euler's rotation equation for a rigid body. The inertial tensor $\mathbf{I}_B$ is constant. Also, the torques are expressed in the body frame, which better suits a rotating robotic system that has its actuators in fixed places in its body. As the inertial tensor is non singular, (7) can take an explicit form:

$$\dot{\boldsymbol{\omega}} = \mathbf{I}_B^{-1}\left({}^{B}\boldsymbol{\tau}_{ext} - \boldsymbol{\omega} \times \mathbf{I}_B\boldsymbol{\omega}\right) \tag{8}$$

2.2 Quaternion Basics

To describe the orientation of a rigid body, quaternions are a useful formalism. This representation does not suffer from singularities like Euler angles or angle-axis pair and has only four parameters, in contrast to rotation matrices that have 9 parameters [15]. However, quaternions are inherently more complex, and thus it is essential to understand them and their associated algebra.

2.2.1 Quaternion Basic Properties

A quaternion q is defined as:

$$q = w + xi + yj + zk \tag{9}$$

where $\{w, x, y, z\} \in \mathbb{R}$ and $\{i, j, k\}$ are imaginary numbers with the following properties:

$$\begin{gathered} ij = k = -ji \\ jk = i = -kj \\ ki = j = -ik \\ i^2 = j^2 = k^2 = ijk = -1 \end{gathered} \tag{10}$$

It is more useful to represent quaternion using 4-vectors, which allows matrix algebra to be used for quaternion operations. An even more compact form, is combining the imaginary part of the quaternion to a 3 element vector, $\mathbf{q}_v$.

$$q = \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix} = \begin{bmatrix} w \\ \mathbf{q}_v \end{bmatrix} \tag{11}$$

Next, some of the most important properties of the quaternions will be presented. The identity quaternion is:

$$q_I = [1,\ 0,\ 0,\ 0]^T \tag{12}$$

The sum of quaternions is defined as:

$$q_1 \pm q_2 = \begin{bmatrix} w_1 \pm w_2 \\ \mathbf{q}_1 \pm \mathbf{q}_2 \end{bmatrix} \tag{13}$$

The product of quaternions is denoted using the $\otimes$ operator, and is called the Hamilton product. It is a direct result of using the algebra defined in (10).

$$q_1 \otimes q_2 = \begin{bmatrix} w_1 w_2 - \mathbf{q}_{v,1}^T \mathbf{q}_{v,2} \\ w_1 \mathbf{q}_{v,2} + w_2 \mathbf{q}_{v,2} + \mathbf{q}_{v,1} \times \mathbf{q}_{v,2} \end{bmatrix} = \begin{bmatrix} w_1 w_2 - x_1 x_2 - y_1 y_2 - z_1 z_2 \\ w_1 x_2 + w_2 x_1 + y_1 z_2 - y_2 z_1 \\ w_1 y_2 + w_2 y_1 - x_1 z_2 + x_2 z_1 \\ w_1 z_2 + w_2 z_1 + x_1 y_2 - x_2 y_1 \end{bmatrix} \tag{14}$$

The conjugate of a quaternion is defined as:

$$q^* = \begin{bmatrix} w \\ -\mathbf{q}_v \end{bmatrix} = \begin{bmatrix} w \\ -x \\ -y \\ -z \end{bmatrix} \tag{15}$$

The following identity is true:

$$(q_1 \otimes q_2)^* = q_2^* \otimes q_1^* \tag{16}$$

The norm of a quaternion is defined as:

$$||q|| = \sqrt{q^T q^*} = \sqrt{w^2 + \ x^2 + \ y^2 + \ z^2} \tag{17}$$

A quaternion is called unitary if $\|q\| = 1$. The inverse of a quaternion is defined as:

$$q^{-1} = q^* / \left\|q\right\| \tag{18}$$

The following is true for the inverse:

$$q^{-1} \otimes q = q \otimes q^{-1} = q_I$$

For unitary quaternions $q^{-1} = q^*$.

2.2.2 Quaternion Geometric Meaning And Error Calculation

Apart from the quaternion properties, it is important to understand the quaternion geometric meaning. The usefulness of quaternions comes down to the fact that they encode orientations and rotations [16]. If $\mathbf{u}$ is the axis of rotation and φ is the right-hand rotation angle, a quaternion q can be written as:

$$q = \begin{bmatrix} w \\ \mathbf{q}_v \end{bmatrix} = \begin{bmatrix} cos(\varphi/2) \\ sin(\varphi/2)\mathbf{u} \end{bmatrix} \tag{19}$$

It can be proved that the product $q_2 \otimes q_1$ has the following meanings [16]:

- It represents a new rotation that first applies the rotation associated with q_1, followed by the rotation associated with q_2.
- It represents a new orientation of a frame, where the initial orientation encoded by q_1 is rotated by the rotation associated with q_2

This leads to the following remark: If the desired orientation is q_D and the initial orientation is q_0, then q_0 needs to be rotated by q_e to reach the desired orientation:

$$q_D = q_e \otimes q_0$$

Thus, the orientation error is given by the following expression:

$$q_e = q_D \otimes q_0^{-1} \tag{20}$$

In order to use the quaternion error in penalty functions, a corresponding metric[2] must be found. Also, as quaternions double cover the $SO(3)$, its use must always lead to the shortest path being taken [5]. A suitable metric is obtained by calculating the following quantities [17]:

$$\mathbf{u} = \frac{\mathbf{q}_{e,v}}{||\mathbf{q}_{e,v}||} \tag{21}$$

$$\theta \triangleq 2\varphi = acos\left(2 < q_d, q(t) >^2 \ - 1 \right) \tag{22}$$

Equation 22 returns an angle between $[0, \pi]$. The corresponding error metric is then:

$$\mathbf{f}_q = \theta \mathbf{u}_e \tag{23}$$

2.2.3 Quaternion Kinematics - Simulation of Rotating Rigid Bodies

A relation connecting the rate of change of quaternions to angular velocity is needed, to have a complete state space representation of rotational dynamics. This is presented in (24) [18].

$$\dot{\mathbf{q}} = \frac{1}{2} \begin{bmatrix} 0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & \omega_z & -\omega_y \\ \omega_y & -\omega_z & 0 & \omega_x \\ \omega_z & \omega_y & -\omega_x & 0 \end{bmatrix} \mathbf{q} \triangleq \frac{1}{2} \mathbf{\Omega q} \tag{24}$$

So, a rotating rigid body with the following state:

$$\mathbf{x} = \begin{bmatrix} \mathbf{q} \\ \boldsymbol{\omega} \end{bmatrix}$$

has the following state space dynamics:

$$\dot{\mathbf{x}} = \begin{bmatrix} \frac{1}{2} \mathbf{\Omega q} \\ \mathbf{I}_B^{-1} \left({}^B\boldsymbol{\tau}_{ext} - \boldsymbol{\omega} \times \mathbf{I}_B \boldsymbol{\omega} \right) \end{bmatrix} \tag{25}$$

2.3 Euler - Lagrange Dynamics

The dynamics of rigid body systems can be derived using the Euler-Lagrange method, which can be extended to include kinematic constraints such as the loop closure constraint of the legs.

2.3.1 Unconstrained Dynamics

The Euler-Lagrange unconstrained equations of motion have the following general form [14].

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) + \mathbf{D}\dot{\mathbf{q}} + \mathbf{F}_s sign(\dot{\mathbf{q}}) + \mathbf{J}^T(\mathbf{q})\mathbf{h}_e = \boldsymbol{\tau} \tag{26}$$

Explanation of the terms follows:

- $\mathbf{q}$ is the generalized position vector, and $\dot{\mathbf{q}}$ is the generalized velocities vector.
- inertial forces: $\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}}$
- centrifugal and Coriolis forces: $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}$
- gravity terms: $\mathbf{G}(\mathbf{q})$
- damping forces : $\mathbf{D}\ \dot{\mathbf{q}} = diag(D_1, ..., D_i, ..) \cdot \dot{\mathbf{q}}$

[2]The norm of the quaternion is not suitable, as for unit quaternions $\|q\| = 1$, no matter the value of q.

- the coulomb friction: $\mathbf{F_s}\ sign(\dot{\mathbf{q}}) = diag(F_{s,1}, ..., F_{s,i}, ..) \cdot sign(\dot{\mathbf{q}})$
- forces due to interactions with the environment: $\mathbf{J}^T(\mathbf{q})\mathbf{h}_e$, where $\mathbf{J}^T$ is the geometric Jacobian and $\mathbf{h}_e$ is the wrench vector from the end-effector to the environment
- motor inputs: $\boldsymbol{\tau}$

However, to adequately model Olympus during attitude stabilization manoeuvres during free fall, (26) can be simplified by making some reasonable assumptions.

Task specific assumptions

1. The legs in the flight face do not interact with the environment[3], and thus $\mathbf{J}^T(\mathbf{q})\mathbf{h}_e = \mathbf{0}$.
2. Also, during the flight phase, the robot is free falling. Since the frame of reference for the leg dynamics is attached to the robot, the **apparent gravity is effectively zero**, and thus $\mathbf{G} = \mathbf{0}$.
3. Coulomb friction is neglected. Thus $\mathbf{F_s} = \mathbf{0}$
4. The damping force coefficients are assumed: $\mathbf{D} = 2.5\text{e-}3 \cdot \mathbf{I}_5$

Thus (26) simplifies to the following:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{D}\dot{\mathbf{q}} = \boldsymbol{\tau} \tag{27}$$

2.3.2 Constrained Dynamics

Equation (27) describes the unconstrained dynamics. However, the leg is a closed kinematic chain, and thus there is a holonomic constraint, in the form of $\mathbf{h}(\mathbf{q}) = 0$ to ensure the closure of the mechanism.

Constraints are added in (27) using the Lagrange multipliers approach [19], and the constrained dynamics become:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{D}\dot{\mathbf{q}} = \boldsymbol{\tau} + \frac{\partial \mathbf{h}}{\partial \mathbf{q}}^T \boldsymbol{\lambda} \tag{28}$$

Defining:

$$\boldsymbol{\tau}_G = -\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} - \mathbf{D}\dot{\mathbf{q}} + \boldsymbol{\tau},\ \mathbf{H} = \frac{\partial \mathbf{h}}{\partial \mathbf{q}}$$

Equation (28) becomes:

$$\mathbf{M}\ddot{\mathbf{q}} = \boldsymbol{\tau}_G + \mathbf{H}^T\boldsymbol{\lambda} \tag{29}$$

Also, differentiating $\mathbf{h}$, results in:

$$\dot{\mathbf{h}} = \mathbf{H}\dot{\mathbf{q}} \tag{30}$$

$$\ddot{\mathbf{h}} = \mathbf{H}\ddot{\mathbf{q}} + \dot{\mathbf{H}}\dot{\mathbf{q}} \tag{31}$$

By combining (31) and (29), $\boldsymbol{\lambda}$ can be calculated:

$$\boldsymbol{\lambda} = -(\mathbf{H}\mathbf{M}^{-1}\mathbf{H}^T)^{-1}(\mathbf{H}\mathbf{M}^{-1}\boldsymbol{\tau}_G + \dot{\mathbf{H}}\mathbf{q}) \tag{32}$$

Having $\boldsymbol{\lambda}$, (29) can be used to calculate $\ddot{\mathbf{q}}$. The procedure above assumes that the expression $(\mathbf{H}\mathbf{M}^{-1}\mathbf{H}^T)$ is invertible. Due to the properties of the Euler-Lagrange equation of motions, $\mathbf{M}$ is non-singular [14], and thus $\mathbf{H}$ must be full rank to solve for both $\boldsymbol{\lambda}$ and $\ddot{\mathbf{q}}$ [20].

[3] If there is interaction, this will be considered an unknown disturbance in the model.

2.4 Optimal Control

A general optimal control problem (OCP) can be written as [21]:

$$\min_{\mathbf{x},\mathbf{u},t_0,t_f} F(\mathbf{x},\mathbf{u},t_0,t_f) \tag{33}$$

so that

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}(t),\mathbf{u}(t),t) \tag{34a}$$
$$\mathbf{F}_{NL}(\mathbf{x}(t),\mathbf{u}(t)) \leq \mathbf{0} \tag{34b}$$
$$\mathbf{F}_{Input}(\mathbf{u}(t)) \leq \mathbf{0} \tag{34c}$$
$$\mathbf{F}_{State}(\mathbf{x}(t)) \leq \mathbf{0} \tag{34d}$$
$$\mathbf{F}_{Boundary}(t_0,t_f,\mathbf{x}(t_0),\mathbf{x}(t_f)) \leq \mathbf{0} \tag{34e}$$
$$\tag{34f}$$

where where $\mathbf{x}(t)$ is the state trajectory, $\mathbf{u}(t)$ is the control input, F is the objective function,F_{NL} are general nonlinear constraints, $\mathbf{F}_{Input}$ are the input constraints, $\mathbf{F}_{State}$ are the state constraints, $\mathbf{F}_{Boundary}$ are the boundary constraints and (34a) is the dynamics constraint. The optimization variables $\mathbf{x}$, $\mathbf{u}$ are infinite-dimensional decision variables, as they are functions.

2.4.1 Numerical Methods for Solving OCPs

The formulation of the OCP in (33,34) is very general and therefore allows one to encode a wide variety of problems. Approaches to solve the OCP are outlined below:

1. **Dynamic Programming**: These methods find the optimal cost-to-go function $V(t_0,x_0)$ (also called the Value function). They do so by discretizing time and space and applying Dynamic Programming (DP). The major benefit of these methods is that they provide a closed-loop solution. However, it is well-known that the complexity of this strategy increases exponentially with the number of states and controls. Therefore, it is not directly applicable to most legged robots [21].

2. **Indirect Methods**: Indirect methods transform the original OCP into a Boundary Value Problem by using Pontryagin's Maximum Principle to formulate the so-called co-state equations (*"first optimize, then discretize"*). The major benefit of an indirect method, when compared to a direct method, is that an indirect method will generally be more accurate and will have a more reliable error estimate [22]. However, these methods are highly sensitive to initialization, require complex derivation of the necessary conditions, and lack flexibility [21, 23].

3. **Direct Methods**: In direct methods, the OCP is transcribed into a finite-dimensional Nonlinear Program (NLP), by discretizing the controls and states with respect to time (*"first discretize, then optimize"*). The NLP can be solved with well-established optimization techniques, e.g. Sequential Quadratic Programming (SQP) [21]. One of the most important advantages of direct compared to indirect methods is that they can easily treat (nonlinear) inequality constraints [24].

2.4.2 Direct Methods

In the context of Nonlinear Model Predictive Control (NMPC), direct methods are favoured [25]. To solve the OCP, one has to transcribe it in a NLP. In this form, the problem can be passed to a commercial solver, such as qpOASES [26] or HPIPM [27]. Transcription greatly affects: accuracy, numerical stability and computational complexity. A transcribed problem has the following form:

$$\min_{\mathbf{z}\in\mathbb{R}^n} J(\mathbf{z}) \tag{35a}$$

subject to

$$\mathbf{z}_{min} \leq \mathbf{z} \leq \mathbf{z}_{max} \tag{35b}$$
$$\mathbf{A}\mathbf{z} \leq \mathbf{0} \tag{35c}$$
$$\mathbf{c}(\mathbf{z}) \leq 0 \tag{35d}$$

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. The direct methods are the following:

1. **Single shooting**: Only the continuous input $\mathbf{u}(t)$ is parameterized through a set of discrete variables, e.g., polynomial coefficients w. The state $\mathbf{x}(t)$ results from forward simulation. The solver verifies that the input and the states satisfy the various constraints.

2. **Multiple shooting**: This method works by breaking up a trajectory into some number of segments and using single shooting to solve for each segment. In this method, *defect constraints* are introduced; constraints to match the end of one segment with the start of the next one. Also, there are more decision variables, as the initial state $\mathbf{x}_i$ for each segment is also a decision variable. Although more constraints and decision variables are added, the optimization problem becomes easier. That is because the relationship between the decision variables and the objective function and constraints becomes more linear [28]. Also, the problem is sparse, and efficient algorithms can be used to solve it [23].

   A schematic overview of single and multiple shooting is presented in figure 2.

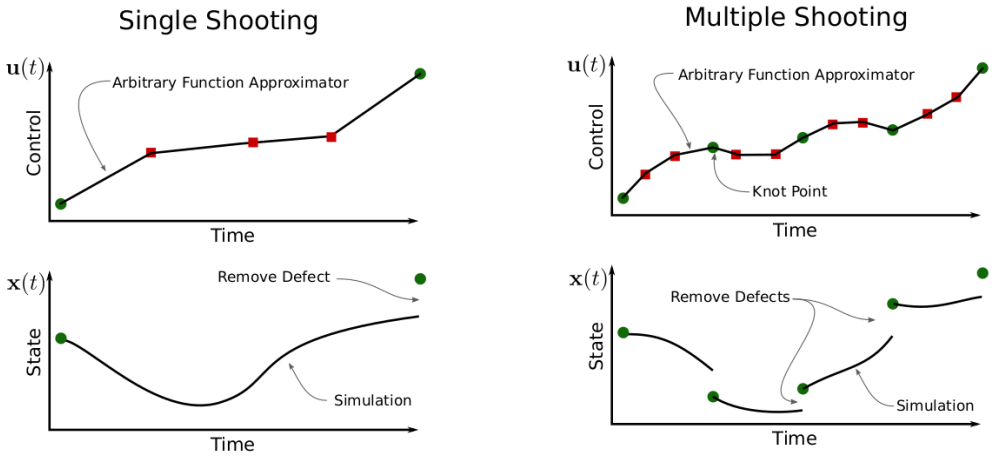


Figure 2: Single Shooting vs Multiple Shooting.

3. **Collocation**: Both the continuous input $\mathbf{u}(t)$ and state $\mathbf{x}(t)$ are parameterized. The dynamics are enforced as constraints at special points in the trajectory, called collocation points. The dynamics constraints can be enforced in either integral ($\mathbf{x} = \int \mathbf{f}(\mathbf{x}, \mathbf{u})dt$) or derivative form ($\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$).

2.4.3 Nonlinear Model Predictive Control

The principle of NMPC is to repeatedly solve finite-horizon optimal control problems described by (33) and (34), using direct numerical methods. The procedure is described in Algorithm 1 [29].

Algorithm 1 Model Predictive Control Loop.

1: Set $\mathbf{x}_0 = \hat{\mathbf{x}}(t)$ ▷ where $\hat{\mathbf{x}}(t)$ is the latest available estimate of the system state.
2: Solve a finite-horizon OCP.
3: Apply the first input ($\mathbf{u}_0^*$) to the system.
4: The state of the system evolves based on its dynamics.
5: After T_s, repeat the procedure. ▷ T_S is the sampling time of the controller.

The final time t_f of the OCP is called prediction horizon T_h and is usually fixed. There is also the concept of control horizon T_c, if the control input is allowed to change until $T_c < T_h$. As the OCP is solved using direct methods, it must be first discretized. The continuous OCP is sampled at $N + 1$ points, which are called stages. The objective function (33) is evaluated only based on the state $\mathbf{X} = [\mathbf{x}_0,\ \mathbf{x}_1,\ ...,\ \mathbf{x}_i,\ ...,\ \mathbf{x}_N]$ and control input $\mathbf{U} = [\mathbf{u}_0,\ \mathbf{u}_1,\ ...,\ \mathbf{u}_i,\ ...,\ \mathbf{u}_{N_C}]$ (with $N_C \leq N - 1$) at these points. To achieve higher resolution in the prediction, the *simulation steps* N_s between two consecutive stages can be increased. These concepts are illustrated in figure 3.

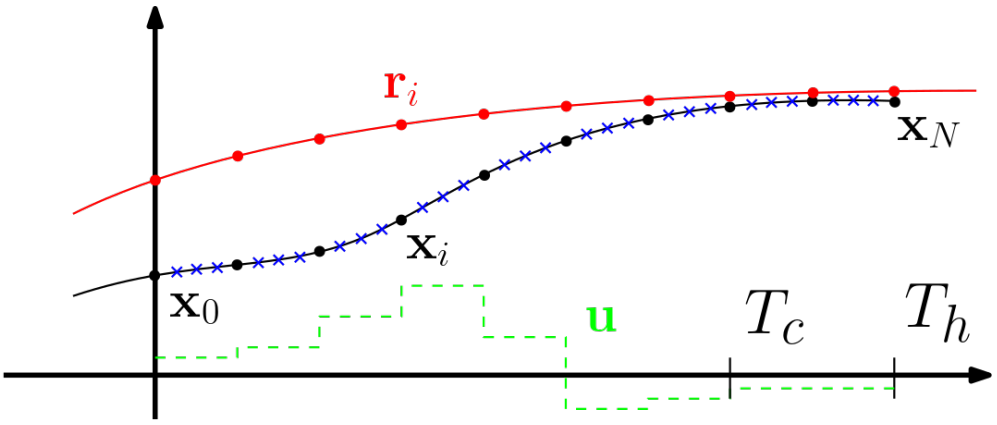


Figure 3: Principle of a Model Predictive Controller. The black line represents the predicted state trajectory, with $\mathbf{x}_i$ denoting the state values at each stage, and blue crosses indicating simulation steps. $\mathbf{r}_i$ denotes the reference state trajectory when MPC is used for trajectory tracking. $\mathbf{u}$ represents the control input at each stage within the control horizon T_c, while the prediction horizon is denoted by T_h.

2.5 Software tools

This subsection will present in short the software tools that were used in the thesis.

2.5.1 Robot Operating System - ROS

The Robot Operating System (ROS) is a software development kit (SDK) used to create robotic applications. The fundamental concepts of ROS are nodes, messages, topics and services. A ROS *node* is a standalone software module. It can communicate with other nodes via exchanging *messages*, which are strictly-typed data structures. A node sends a message by publishing it to a given *topic*, and any node requiring this message subscribes to the corresponding topic. Apart from this continuous and asynchronous data streaming between nodes, synchronous, request-response communication can be achieved through the use of *services* [30].

ROS is designed to support multi-computer networks, employing a peer-to-peer topology for node communication. This architecture facilitates efficient data exchange and distributed processing. Additionally, Secure Shell (SSH) protocol can be utilized to remotely connect to the robot's computer, allowing for logging information and remote operational control. ROS was used to create the software modules that were used by the actual robot.

2.5.2 Matlab - Simulink

Simscape is a physical modelling language using block diagrams. It is possible to create electronic and mechanical systems by assembling fundamental components into a schematic. It also provides a multibody simulation environment for 3D mechanical systems. The software includes functionality for importing CAD models, contact scenarios and visualization. The user friendly interface makes simulink suitable for rapid prototyping of new control architectures.

2.5.3 Drake

Drake is a C++ toolbox started by the Robot Locomotion Group at the MIT and is actively maintained by the Toyota Research Center. It features a collection of tools for analyzing the dynamics of robots and building control systems for them, with a heavy emphasis on optimization-based design/analysis [31]. Drake supports multibody simulations and has advanced contact modelling capabilities [32].

The modelling philosophy of drake is similar to simulink. Each system is treated as a block with input-output (I/O) ports and is connected with other systems. The main components of a drake simulation are presented below.

- **Diagrams**: It represents a combination of systems and it contains information about their connectivity. It is essentially the top-most abstraction layer. It is considered a new system and it is created through a **DiagramBuilder**.
- **LeafSystem**: A template class for generic static and dynamic subsystems that interact with the rest of the Diagram through their user-declared I/O ports. Used to implement e.g. controllers, sensors and actuator dynamics. Every user defined system inherits from LeafSystem.

- **Context**: It is a class containing the state, the input and system parameters of a system[4]. A context is designed to be used only with its corresponding system. Systems usually provide methods to modify their context, such as methods to set initial conditions and retrieve the current state.
- **Simulator**: An object advancing the state of a continuous/discrete/hybrid dynamical system (i.e. Diagram) forward in time. The Diagram itself only provides the Simulator with information such as state derivatives and is otherwise unchanged. Only the Diagram's Context gets modified by the Simulator.
- **MultibodyPlant**: It represents a physical system consisting of a collection of interconnected bodies. A MultiBodyPlant may contain multiple model instances. Each model instance corresponds to a set of bodies and their connections (joints). Usually a new model instance is added from a URDF file.
- **MeshcatVisualizer**: A wrapper for Meshcat[5], a 3D visualizer running in its own thread spawned by the MeshcatVisualizer. The object receives geometry changes from the **SceneGraph**, which handles all the geometry based operations, and publishes them to the underlying visualizer. This object provides the preferred method for visualizing simulations in Drake.

2.5.4 Acados Framework Details

Acados is a free and open source software package that contains solvers for fast embedded optimization intended for fast embedded applications. It interfaces with high-level languages such as Python and Matlab for quickly designing optimization-based control algorithms and can generate efficient C code for deployment.

Acados can handle the following[6] optimization problem:

$$\min_{\mathbf{x},\mathbf{u},\boldsymbol{\sigma}} \int_0^T \left[l(\mathbf{x},\mathbf{u},\mathbf{p}) + \mathbf{z}_s(\mathbf{s})\right] dt + m(\mathbf{x}_E,\mathbf{p}) + \mathbf{z}_{s,E}(\mathbf{s}_E) \tag{36}$$

so that

$$\mathbf{f}(\mathbf{x},\dot{\mathbf{x}},\mathbf{u},\mathbf{p}) = 0 \tag{37a}$$

$$\mathbf{g}_l \leq \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u} \leq \mathbf{g}_u \tag{37b}$$

$$\mathbf{h}_l \leq \mathbf{h}(\mathbf{x},\mathbf{u},\mathbf{p}) \leq \mathbf{h}_u \tag{37c}$$

where:

- $\mathbf{x}$ is the state vector,
- $\mathbf{u}$ is the control input vector,
- $\mathbf{p}$ are the model parameters,
- $\boldsymbol{\sigma} \geq \mathbf{0}$ is the slack variables vector,
- $l(\mathbf{x},\mathbf{u},\mathbf{p})$ represents the stage cost,
- $m(\mathbf{x}_E,\mathbf{p})$ represents the terminal cost,
- $\mathbf{z}_s(\mathbf{s})$ is the slack variables cost function, and $\mathbf{z}_{s,E}(\mathbf{s}_E)$ represents the terminal slack variables cost.
- Equation (37a) represents the dynamics constraint
- Equations (37b,37c) are generic constraint formulations supported by acados.

Below, some more details are given for the above formulation.

- **Cost**: The stage and terminal cost can be formulated using linear least squares (LS), non-linear least squares (NLS), or even a generic nonlinear function.

[4] A diagram is also a system, and thus it also has a corresponding diagram.

[5] `https://github.com/meshcat-dev/meshcat`

[6] This is not the most general description. Acados can handle DAE dynamics, which include algebraic variables. Also, input and state box constraints and terminal constraints can be specified directly. The description in this section is used to guide the discussion. A more detail description can be found here: `https://github.com/acados/acados/blob/master/docs/problem_formulation/problem_formulation_ocp_mex.pdf`

- **Equality constraints**: *Acados* support only inequality constraints, so all equality constraints are transformed to inequalities with identical lower and upper bounds.
- **Slacking constraints**: Slacked constraints are formulated as follows (where $\mathbf{f}$ is any possible constraint):

$$\mathbf{f}_l - \boldsymbol{\sigma}_l \leq \mathbf{f}(\mathbf{x}, \mathbf{u}) \leq \mathbf{f}_u + \boldsymbol{\sigma}_u \tag{38}$$

and the following term is added in the objective function:

$$\mathbf{z}_s(\mathbf{s}) = \begin{bmatrix} \boldsymbol{\sigma}_l & \boldsymbol{\sigma}_u & 1 \end{bmatrix} \begin{bmatrix} \mathbf{Z}_l & \mathbf{0} & \mathbf{z}_l \\ \mathbf{0} & \mathbf{Z}_u & \mathbf{z}_u \\ \mathbf{z}_l & \mathbf{z}_u & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\sigma}_l \\ \boldsymbol{\sigma}_u \\ 1 \end{bmatrix} \tag{39}$$

Usually $\mathbf{Z}_l = \mathbf{Z}_u = \mathbf{Z}$ and $\mathbf{z}_l = \mathbf{z}_u = \mathbf{z}$. Selecting $\mathbf{z} > 0$, enforces the constraints more strictly[7].

The OCP described by (36) and (37) is discretized using multiple shooting [33]. To solve the resulting NLP, acados interfaces several solvers; *HPIPM*, which uses an interior-point method [27], and *qpOASES* which is an active set QP solver [26].

[7]The expression (39) evaluates to $Z\sigma^2 + 2z\sigma$ for a single slack variable. The minimum is found at $\sigma^* = -z/Z$, $(Z > 0)$. For $z < 0$, the constraint is enlarged. For $z > 0$, as $\sigma > 0$, the slope at $\sigma = 0$ becomes more steep, which increases the rate at which the slack variable penalizes the objective function as the constraints are violated.

3. Modelling

This section provides a comprehensive overview of the system's physical modeling. Initially, the kinematic analysis of the robot takes place. Next, key points about the dynamic modeling and the verification of analytical models are presented. The workspace of the robot is then analyzed in order to extract various operational constraints for the system. Finally, the section discusses the corresponding models in simulation software.

3.1 Leg Kinematics

Olympus' leg is depicted in figure 4, in which the frames that will be used in the kinematic analysis are included. $\{BL\}$ is the base frame of the leg and it coincides with $\{MH\}$ for $q_{MH} = 0$. The axis of rotation is the z axis, apart from the motor housing joint, which rotates about the x axis, according to the right hand rule. It must be noted, that chain 1 **always** includes the leg paw. The generalized position of the leg is given by (these angles are defined in figures 4 and 5):

$$\mathbf{q} = [q_{MH},\ q_{11},\ q_{21},\ q_{12},\ q_{22}]^T \tag{40}$$

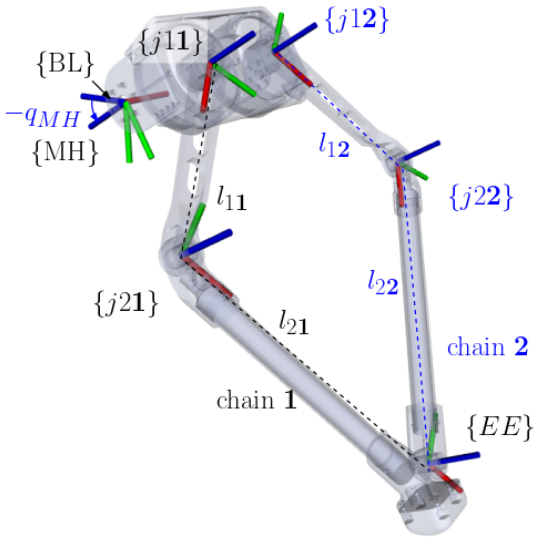


Figure 4: Kinematic Definitions and coordinate frames for Olympus' leg. (Red: x-axis, Green: y-axis, Blue: z-axis)

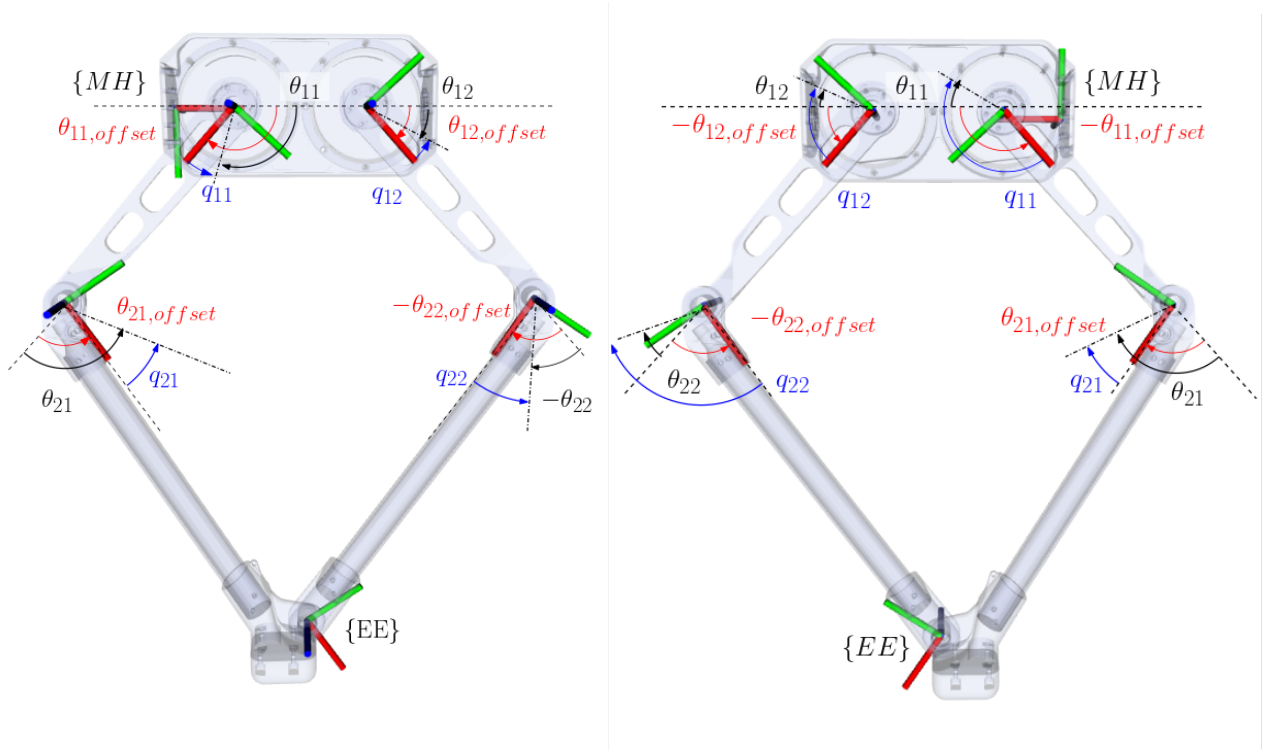
There are two different configurations[8].

- Configuration 1: front right (FR) and rear left (RL) legs
- Configuration 2: front left (FL) and rear right (RR) legs

In figure 5 these configurations are displayed, along with the joint zero positions. In table 3 the corresponding angle offsets[9] are presented. In table 4, the angle limits of the joints are displayed (red angles are correspond to configuration 2).

[8]The front right leg can be rotated and used as is (with its corresponding frames) for the rear left one. The same is true for the front left and rear right ones. But the front right leg cannot be rotated to be inserted in the front left one.

[9]Offsets (as they are extracted from the urdf) and θ_{ij} are measured from the previous frame, while q_{ij} are measured from the joint zero position.



(a) Configuration 1.

(b) Configuration 2.

Figure 5: Joint angles of the leg in each configuration.

Table 3: Joint angle offsets.

	Offsets	**Configuration 1**	**Configuration 2**
$q_{MH,offset}$	$-\pi/2$	$\theta_{MH} = \theta_{MH,offset} + q_{MH}$	$\theta_{MH} = \theta_{MH,offset} + q_{MH}$
$q_{11,offset}$	2.3095	$\theta_{11} = \theta_{11,offset} - q_{11}$	$\theta_{11} = -\theta_{11,offset} + q_{11}$
$q_{21,offset}$	1.3265	$\theta_{21} = \theta_{21,offset} + q_{21}$	$\theta_{21} = \theta_{21,offset} + q_{21}$
$q_{12,offset}$	0.83482	$\theta_{12} = \theta_{12,offset} - q_{12}$	$\theta_{12} = -\theta_{12,offset} + q_{12}$
$q_{22,offset}$	-1.3233	$\theta_{22} = \theta_{22,offset} + q_{22}$	$\theta_{22} = \theta_{22,offset} + q_{22}$

Table 4: Joint angle limits.

	Lower Limit	**Upper Limit**
q_{MH}	$-\pi$ / $-\pi/2$	$\pi/2$ / π
q_{11}	$-\pi$	1.94
q_{21}	-2.4	1.58
q_{12}	-1.94	π
q_{22}	-1.51	2.4

3.1.1 Forward Kinematics

The aim of the forward kinematics is to find the end effector position in the $\{BL\}$ frame of the leg, ${}^{BL}P_{EE}$. Also, as the robot only reads the actuated joint values (through the servo feedback), the direct kinematics are needed to get the **full generalized position** of the leg.

The kinematic analysis of each leg is done with respect to the leg frame. The kinematics are presented for configuration 1, and indices $i = 1, 2$ signify the kinematic chains. The parameters that change for the second configuration are marked with red . The procedure is described below; firstly ${}^{MH}P_{EE}$ is calculated based on the closed kinematics of an RR manipulator, and then the position is transformed in the $\{BL\}$ frame.

Forward Kinematics Procedure

1. Initially, θ_{1i} is calculated based on table 3.

$$\theta_{1i} \leftarrow -(q_{1i} - \theta_{off,1i}) \tag{41}$$

2. Then the position of joints $j21,\ j22$ is calculated in the $\{MH\}$ frame.

$$\mathbf{P}_{ci} \triangleq {}^{MH}\mathbf{P}_{j2i} = {}^{MH}\mathbf{P}_{j1i} + l_{1i} \begin{bmatrix} cos(\theta_{1i}) \\ sin(\theta_{1i}) \end{bmatrix} \tag{42}$$

3. Then the end effector position ${}^{MH}\mathbf{P}_{EE}$ is calculated as the intersection of $C_1(\mathbf{P}_{c1}, l_{21})$ and $C_2(\mathbf{P}_{c2}, l_{22})$. By defining the following:

$$\mathbf{v}_c = \mathbf{P}_{c2} - \mathbf{P}_{c1},\ d = ||\mathbf{v}_c||_2,\ \mathbf{v}_n = \frac{1}{d}[-v_{c,Y},\ v_{c,X}]^T$$

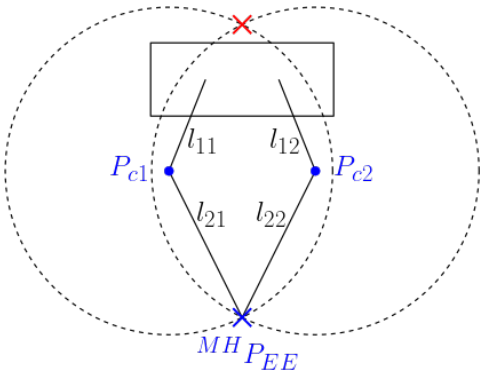
and calculating the quantities presented in figure 6b :

$$a = \frac{d^2 + l_{21}^2 - l_{22}^2}{2d},\ h = \sqrt{l_{21}^2 - a^2}$$

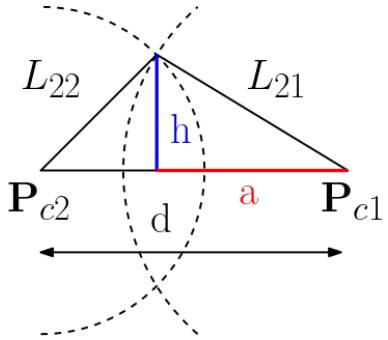
The intersection point is:

$$\mathbf{P}_{EE,2D} \triangleq {}^{MH}\mathbf{P}_{EE} = \mathbf{P}_{c1} + a\mathbf{v}_c + h\mathbf{v}_n; \tag{43}$$

There is always only one valid intersection point (blue point in in figure 6a). Also, because $l_{11}^2 + l_{12}^2 < l_{22}^2 + l_{21}^2$, it is not possible for the leg to reach the other intersection point[10].



(a) Selected intersection point for the calculation of ${}^{MH}\mathbf{P}_{EE}$.



(b) Circle intersection helper quantities.

Figure 6: Circle intersection as part of the Direct Kinematics.

4. Finally, the intersection point is transformed from $\{MH\}$ to $\{BL\}$:

$${}^{BL}P_{EE} \leftarrow \mathbf{R}_x(q_{MH})\ {}^{MH}P_{EE} \tag{44}$$

The forward kinematics procedure is presented in algorithm 2.

The blue quantities are saved to be used in the state estimation algorithm, presented in algorithm 3. Having found the intersection point, it is easy to calculate the joint angles of the whole leg.

Joint angle estimation

1. Calculate the vectors along *link1i* and *link2i*:

$$\mathbf{v}_{1i} = P_{ci} - {}^{MH}\mathbf{P}_{j1i} \tag{45}$$

$$\mathbf{v}_{2i} = {}^{MH}\mathbf{P}_{EE} - P_{ci} \tag{46}$$

[10]Without resulting in a self-colliding configuration.

Algorithm 2 Forward Kinematics Algorithm.

function DK($\mathbf{q}_m$) ▷ $\mathbf{q}_m = [q_{MH},\ q_{11},\ q_{12}]^T$
  Call END_EFFECTOR_2D($\mathbf{q}_m$)
  return END_EFFECTOR_BL(q_{MH})
end function
function END_EFFECTOR_2D($\mathbf{q}_m$)
  Calculate θ_{1i}
  Calculate $\mathbf{P}_{ci}$
  Calculate $\mathbf{P}_{EE,2D}$ as the intersection of $C_1(\mathbf{P}_{c1}, l_{21})$ and $C_2(\mathbf{P}_{c2}, l_{22})$
end function
function END_EFFECTOR_BL(q_{MH})
  Transform intersection point from $\{MH\}$ to $\{BL\}$
  return ${}^{BL}P_{EE}$
end function

2. Calculate θ_{2i}:

$$\theta_{2i} = \ acos\left(\frac{< \mathbf{v}_{1i}, \mathbf{v}_{2i} >}{||\mathbf{v}_{1i}||_2\ ||\mathbf{v}_{2i}||_2}\right) \tag{47}$$

   The *acos* function return values in $[0, \pi]$. Thus, the possible angles for θ_{2i} are 2. The angle that results in a configuration of a convex quadrilateral is initially selected. That means that at first $\theta_{21} \in [0, \pi]$ and $\theta_{22} \in [-\pi, 0]$. This assumption is checked in the following step.

3. To find the sign of θ_{2i}, the angle of the position of the end effector is compared with θ_{1i}. The angles of the end effector are found using the following expressions:

$$^{MH}\mathbf{P}^{j1i}_{EE,i} = {}^{MH}\mathbf{P}_{EE} - \ {}^{MH}\mathbf{P}_{j1i}$$

$$\theta_{EE,i} = atan2(^{MHr}\mathbf{P}^{j1i}_{Y,EE,i}, {}^{MHr}\mathbf{P}^{j1i}_{X,EE,i})$$

   Then, these angles are expressed in $[-\pi/2, 3\pi/2]$ for configuration 1 and $[-3\pi/2, \pi/2]$ for configuration 2 before doing the comparison.

4. Finally, q_{2i} is calculated[11] using table 3.

$$q_{2i} \leftarrow \theta_{2i} - q_{off,2i} \tag{48}$$

The state estimation algorithm is presented in Algorithm 3.

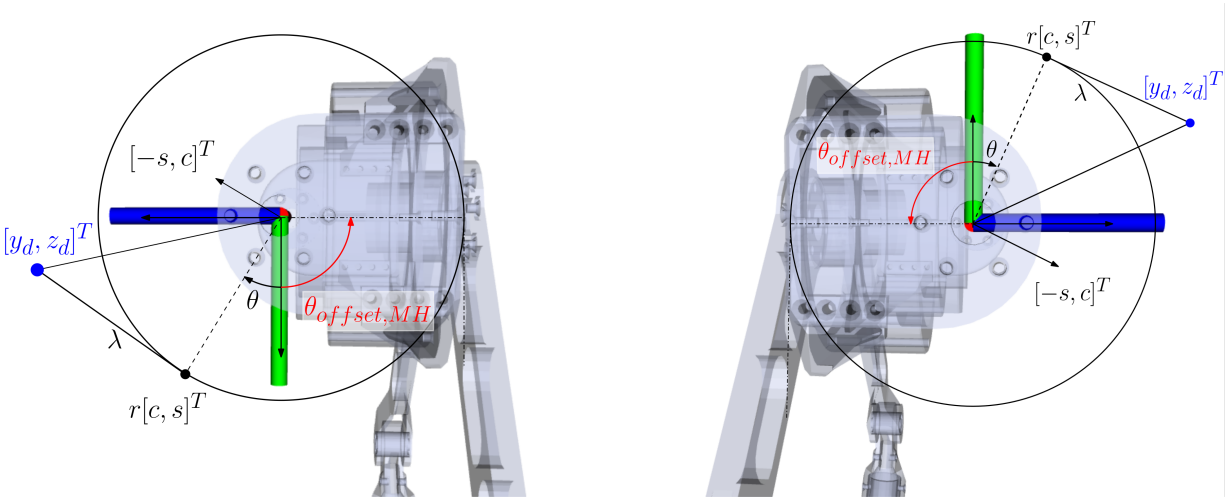
Algorithm 3 State Estimation Algorithm.

function STATE_ESTIMATION($\mathbf{q}_m$)
  Call END_EFFECTOR_2D($\mathbf{q}_m$)
  Calculate θ_{2i}
  Verify sign of θ_{2i}
  $q_{2i} \leftarrow \theta_{2i} - q_{off,2i}$
  return $[q_{MH},\ q_{11},\ q_{21},\ q_{12},\ q_{22}]^T$
end function

3.1.2 Inverse Kinematics

Given a desired end effector position ${}^{BL}\mathbf{P}_d$, the corresponding joint angles ($\mathbf{q} = [q_{MH},\ q_{11},\ q_{21},\ q_{12},\ q_{22}]^T$) must be found. The desired position is expressed in the $\{BL\}$ frame.

[11] Previously, the θ_{2i} is calculated exactly as the one shown in 5. This is in contrast to the IK calculation that are presented next.



(a) Finding q_{MH} for configuration 1.

(b) Finding q_{MH} for configuration 2.

Figure 7: Finding q_{MH}.

Finding q_{MH}

Using $r \triangleq {}^{MH}P_{z,j1i}$ and figure 7, the desired end effector position in $\{BL\}$ is:

$$\mathbf{P}_d = \begin{bmatrix} y_d \\ z_d \end{bmatrix} = r \begin{bmatrix} cos(\theta) \\ sin(\theta) \end{bmatrix} + \lambda \begin{bmatrix} -sin(\theta) \\ cos(\theta) \end{bmatrix}$$

By squaring both sides and adding them, we get:

$$\lambda = \pm\sqrt{y_d^2 + z_d^2 - r^2} \tag{49}$$

As the clock-wise normal vector was used (as seen in figure 7), the sign of λ in (49) can be specified for each configuration.

- For configuration 1: $\lambda > 0$
- For configuration 2: $\lambda < 0$

The quantity inside the square root must be non-negative, and thus the first check for the feasibility of the problem is:

$$\boxed{y_d^2 + z_d^2 - r^2 > 0} \tag{50}$$

Knowing λ, the following system of equations is created:

$$\begin{bmatrix} r & -\lambda \\ \lambda & r \end{bmatrix} \begin{Bmatrix} cos(\theta) \\ sin(\theta) \end{Bmatrix} = \begin{Bmatrix} y_d \\ z_d \end{Bmatrix}$$

$$c \triangleq cos(\theta) = \frac{r y_d + \lambda z_d}{r^2 + \lambda^2}$$

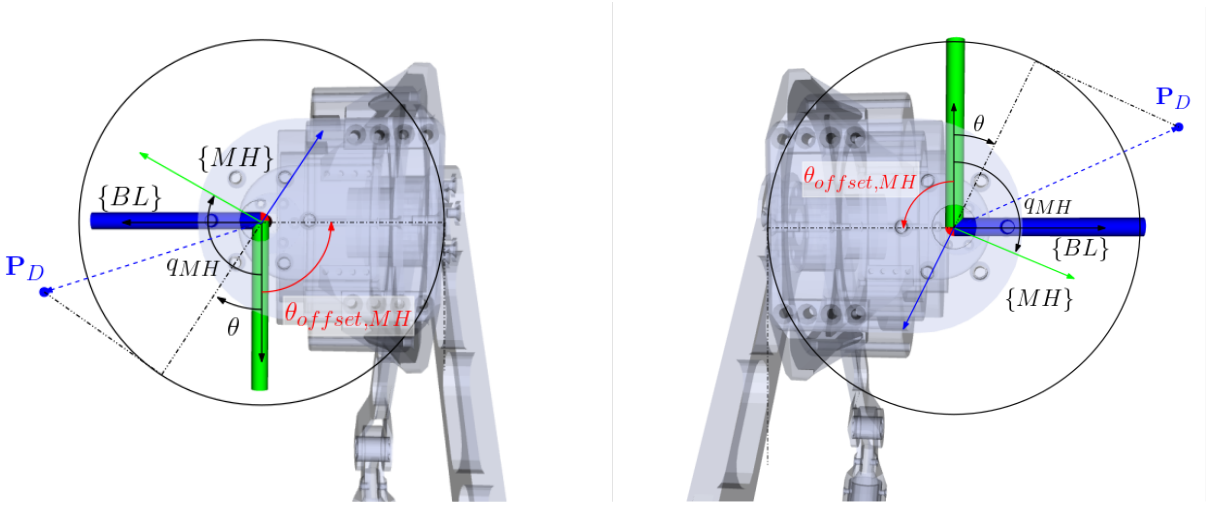
$$s \triangleq sin(\theta) = \frac{r z_d - \lambda y_d}{r^2 + \lambda^2}$$

$$\theta = atan2(s, c) \tag{51}$$

Thus, q_{MH} can be calculated[12] using table 3:

$$\boxed{q_{MH} = \theta + \pi/2} \tag{52}$$

[12]The determinant can be skipped when calculating θ, as it doesn't affect the result.



(a) Expressing ${}^{MH}\mathbf{P}_D$ for configuration 1.

(b) Expressing ${}^{MH}\mathbf{P}_D$ for configuration 2.

Figure 8: Expressing ${}^{MH}\mathbf{P}_D$.

Finding $q_{11},\ q_{21},\ q_{12},\ q_{22}$

$\mathbf{P}_d$ must be expressed in the $\{MH\}$ frame.

$$ {}^{BL}\mathbf{R}_{MH} = \mathbf{R}_x(q_{MH}) $$

and thus

$$ {}^{MH}\mathbf{P} = \mathbf{R}_x(-q_{MH})\ {}^{BL}\mathbf{P} \tag{53} $$

Next the distance of the end effector from each hip joint is calculated.

$$ \mathbf{P}_i^* = {}^{MH}\mathbf{P}^* = {}^{MH}\mathbf{P} - {}^{MH}\mathbf{P}_{j1i} \tag{54} $$

The inverse kinematics problem is now a RR manipulator inverse kinematics problem, with a known solution.The following is defined[13]:

$$ \theta_{2i}^* = \begin{cases} -\theta_{2i}, & \text{for configuration 1} \\ \theta_{2i}, & \text{for configuration 2} \end{cases} \tag{55} $$

The RR inverse kinematics problem is the following:

$$ \mathbf{P}_i^* = l_{1i} \begin{bmatrix} c_1 \\ s_1 \end{bmatrix} + l_{2i} \begin{bmatrix} c_{1+2^*} \\ s_{1+2^*} \end{bmatrix} \tag{56} $$

Solving for θ_{2i}:

$$ \boxed{\theta_{2i}^* = \pm acos\left(\frac{P_{i,X}^{*2} + P_{i,Y}^{*2} - l_{1i}^2 - l_{2i}^2}{2 l_{1i} l_{2i}} \right)} \tag{57} $$

The *acos* function is defined in $[-1, 1]$, and thus the second feasibility test is the following:

$$ \boxed{-1 \leq \frac{P_{i,X}^{*2} + P_{i,Y}^{*2} - l_{1i}^2 - l_{2i}^2}{2 l_{1i} l_{2i}} \leq 1} \tag{58} $$

There are in general two solutions. Knowing θ_{2i}^*, system 56 becomes:

[13] In the classic kinematic analysis of RR, all angles are measured from the previous link in the *Fixed Frame* (which here is the $\{MH\}$ frame, as $\mathbf{P}_i^*$ is the relative distance from hip joints.). See this: `http://www.diag.uniroma1.it/~deluca/rob1_en/10_InverseKinematics.pdf#page=18`. It must be noted that θ_{1i} angles match the above specification by definition (the parent frame of $\{j1i\}$ is $\{MH\}$, as seen in figure 5), **but θ_{2i} in configuration 1 have opposite positive direction**

$$\begin{bmatrix} l_{1i} + l_{2i}c_{2^*} & -l_{2i}s_{2^*} \\ l_{2i}s_{2^*} & l_{1i} + l_{2i}c_{2^*} \end{bmatrix} \begin{Bmatrix} c_1 \\ s_1 \end{Bmatrix} = \begin{bmatrix} k_1 & -k_2 \\ k_2 & k_1 \end{bmatrix} \begin{Bmatrix} c_1 \\ s_1 \end{Bmatrix} = \mathbf{P}^*$$

The determinant is $det = k_1^2 + k_2^2 > 0$, so the system always has a solution, which is:

$$c_1 = \frac{k_1 P_X^* + k_2 P_Y^*}{k_1^2 + k_2^2}$$

$$s_1 = \frac{k_1 P_Y^* - k_2 P_X^*}{k_1^2 + k_2^2}$$

$$\boxed{\theta_{1i} = atan2(s_1, c_1)} \tag{59}$$

If the IK problem is mathematically feasible (meaning conditions (50) and (58) are true) there are two solutions sets:

$$\begin{Bmatrix} (\theta_{1i},\ \theta_{2i}^*) \\ (\theta_{1i}',\ -\theta_{2i}^*) \end{Bmatrix}$$

where $\theta_{1i} \in [-\pi, \pi]$ and $\theta_{2i}^* \in [0, \pi]$. To find the joint angles, the angle offsets must be compensated. Using (55) and table 3 we get:

$$q_{1i} = \begin{cases} -\theta_{1i} + q_{1i,offset}, & \text{for configuration 1} \\ \theta_{1i} + q_{1i,offset}, & \text{for configuration 2} \end{cases}$$

$$q_{2i} = \begin{cases} -\theta_{2i} - q_{2i,offset}, & \text{for configuration 1} \\ \theta_{2i} - q_{2i,offset}, & \text{for configuration 2} \end{cases}$$

Also, the results are wrapped in $[-\pi, \pi]$.

Checking feasibility of IK solutions

For each chain, there are two solutions sets, and thus there are 4 total combinations of the solution sets. The mathematical feasibility of the IK does not guarantee that the solutions are feasible in the real system. The solutions of the IK must respect the **joint angle limits** as well as result in **non self-colliding configurations**. Thus, each pair of solution sets is checked further.

Checking whether the joint limits are satisfied is trivial, and thus it will not be presented. Checks for self collision are presented below.

1. **Shank distance:** The distance of the shank joints must be adequate so that they do not collide. Using (42), shank collision occurs when :

$$\boxed{||\mathbf{P}_{c2} - \mathbf{P}_{c1}||_2 \leq D_{MARGIN}} \tag{60}$$

   D_{MARGIN} must be larger than the joint connection ($0.025m$) and is selected as 0.03.

2. **Hip clearance:** The hips[14] must not intersect. Using (45) to define $\mathbf{v}_{1i}$, the intersection point of the hips is found solving the following equation:

$$\mathbf{P}_{int} = {}^{MH}\mathbf{P}_{j11} + \lambda \mathbf{v}_{11} = {}^{MH}\mathbf{P}_{j12} + \kappa \mathbf{v}_{12} \tag{61}$$

   The following hold true:

   - For $\lambda = 1,\ \kappa = 1$, the intersection point coincides with *joint21* or *joint22* respectively, from the definition of $\mathbf{v}_{1i}$.
   - For $\lambda < 0,\ \kappa < 0$ the intersection point is behind the hips and for $\lambda > 1,\ \kappa > 1$ it is further than the hips.

[14] See figure 1.

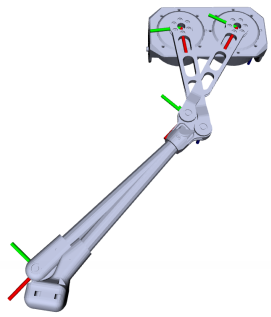
- For $1 > \lambda > 0,\ 1 > \kappa > 0$ the intersection point is alongside the hip 1 or 2 respectively.

For a collision to happen the following must be true:

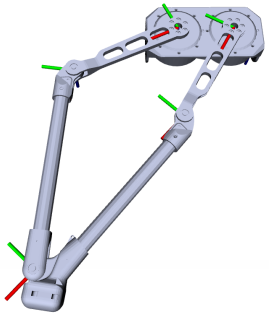
$$\boxed{\kappa \in [0, 1 + D_{CLEARANCE}] \ \ \textbf{AND} \ \lambda \in [0, 1 + D_{CLEARANCE}]} \tag{62}$$

$D_{CLEARANCE}$ is a margin that handles intersections that happen further than *joint2i* origin but are inside the radius of the mechanical joint. $D_{CLEARANCE} = D_{MARGIN}/2l_{1i}$

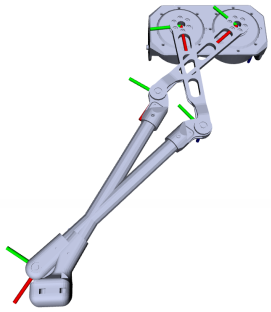
The results of the feasibility checks are showcased in an example in figure 9. In figure 9a the shank distance constraint is violated while in 9c the hip clearance constraint is violated. To store the feasibility information, a boolean matrix is created where each entry corresponds to a solution pair and its value (*true/false*) shows if it is feasible. The corresponding feasibility array is shown in table 5.



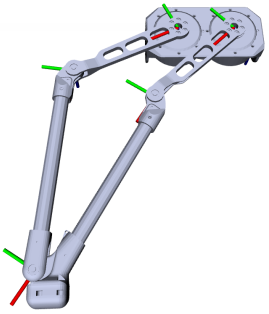
(a) **Chain1**: set 1, **Chain2**: set 1.



(b) **Chain1**: set 1, **Chain2**: set 2.



(c) **Chain1**: set 2, **Chain2**: set 1.



(d) **Chain1**: set 2, **Chain2**: set 2.

Figure 9: Example of feasible solutions generated by Inverse Kinematics without additional constraints.

Table 5: Calculated Feasibility Matrix example.

	chain 2 solution set 1	**chain 2 solution set 2**
chain 1 solution set 1	*false*	*true*
chain 1 solution set 2	*false*	*true*

The complete Inverse Kinematics procedure is presented below in Algorithm 4. It is evident that the IK does not return a specific solution and further logic must be employed to select the most appropriate solution. Two extra criteria are used. Firstly, solutions that result in a convex quadrilateral are preferred. If there are still multiple solutions, then the one with the smallest angle distance from the current state is used[15].

[15]Even with this distance criterion, if IK are used to plan a whole trajectory, a waypoint that minimizes a next step distance does not necessarily result in a Cartesian trajectory with minimum angle displacement.

Algorithm 4 Inverse Kinematics Algorithm.

function IK($\mathbf{P}_D$)
  if Not feasible **then** ▷ Conditions (50) and (58) do not hold
    return False
  end if
  Calculate $q_{MH},\ q_{1i},\ q_{2i}$. ▷ Using (52), (59) and (57).
  Store solutions for each chain in `SOL_ci` .
  Update `feasibility_matrix` . ▷ Using conditions (60) and (62).
  if `feasibility_matrix` is all false **then**
    return False
  else
    return True
  end if
end function

3.1.3 Transformation for the Front Right Leg

The leg dynamics and body kinematics are based on an updated model of the system that uses a different frame convention. The transformation from the kinematic frame convention to the convention employed by the robot is the following:

$$^{K}\boldsymbol{\theta} = \boldsymbol{\theta}_{offset,K} + \mathbf{D}_T\ ^{r}\boldsymbol{\theta}$$

where $^{K}\boldsymbol{\theta}$ is the angle in the *kinematics convention*, $^{r}\boldsymbol{\theta}$ is the motor angle in the actual robot, $\boldsymbol{\theta}_{offset,K}$ is the angle of the joints in the *kinematics convention* to reach the motor zero position and $\mathbf{D}_T$ is a diagonal transformation matrix that accounts for the positive direction of the motors[16].

For the front right leg, which is the one that will be used in the dynamics, the new joint limits are presented in table 6.

Table 6: Joint angle limits for front right leg in the robot convention.

	Lower Limit	**Upper Limit**
q_{MH}	$-\pi/2$	π
q_{11}	-1.2013	3.8803
q_{21}	-2.7504	1.2296
q_{12}	-1.2040	3.8776
q_{22}	-1.2222	2.6878

3.2 Body Kinematics

A relation that connects the leg frames to the quadruped frame is needed. The relevant frames of the legs and the torso are depicted in figure 10. Thus, a point can be transformed to the base $\{B\}$ frame from a leg frame $\{BL\}$ using :

$$^{B}\mathbf{T}_{BL} = \mathbf{T}(\mathbf{p}, \mathbf{0}) \tag{63}$$

where $\mathbf{p} = [DX,\ DY,\ DZ]^T$ represents the translation vector. The values for $\mathbf{p}$ for each leg are presented in table 7.

Table 7: Transform data for olympus' legs.

	Configuration	**DX** $[mm]$	**DY** $[mm]$	**DZ** $[mm]$
FR	1	143.3	-105	0
FL	2	143.3	105	0
RR	2	-143.3	-150	0
RL	1	-143.3	150	0

[16]Thus $\mathbf{D}_T$ contains $\{1, -1\}$ in the diagonal and $\mathbf{D}_T = \mathbf{D}_T^{-1}$.

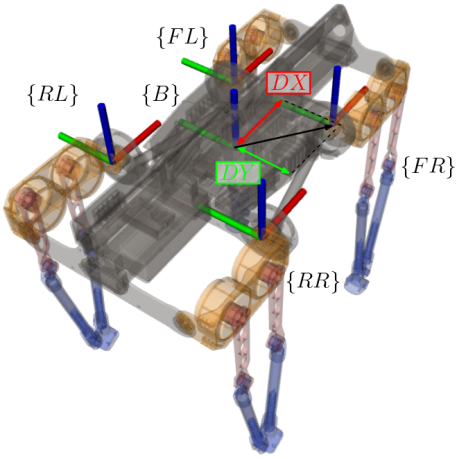


Figure 10: Body Transforms.

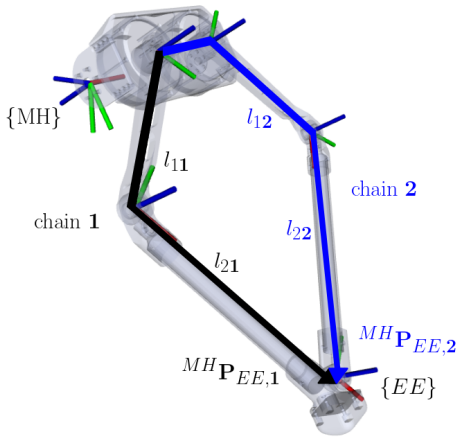


Figure 11: Definition of closure constraint.

3.3 Leg Dynamics

To formulate the Model Predictive Control (MPC) of the leg planner, a closed-form representation of the leg dynamics is essential. These dynamics are derived using the Euler-Lagrange method, as discussed in section 2.3. Initially, the unconstrained dynamics are obtained directly based on the dynamic and geometric properties included in the URDF. Subsequently, these dynamics are extended to integrate the chain closure constraint, with the procedure described in section 2.3.2. In this section, some details on formulating the closure constraint are presented. Verification of the analytical dynamic model is conducted by comparing it against the *Simulink* and Drake simulation models.

The holonomic constraint to ensure the closure of the mechanism is the following:

$$\mathbf{h} = {}^{MH}\mathbf{P}_{EE,1} - {}^{MH}\mathbf{P}_{EE,2} = \mathbf{0} \tag{64}$$

where ${}^{MH}\mathbf{P}_{EE,i}$ is the position of the end effector expressed with the state variables of the i-th kinematic chain. Its geometric interpretation is presented in figure 11. The constraint has the following[17] form:

$$\mathbf{h} = \begin{bmatrix} h_X \\ 0 \\ h_Z \end{bmatrix} = \mathbf{0}$$

as the two links have no degree of freedom in the y direction of the $\{MH\}$ frame. Thus the kinematic closure is ensured by two constraints, $h_X,\ h_Z$ that depend on $q_{11},\ q_{21},\ q_{12},\ q_{22}$. As discussed in section 2.3.2, $\mathbf{H} = \frac{\partial \mathbf{h}}{\partial \mathbf{q}}$ (where $\mathbf{h} = [h_X,\ h_Z]^T$) must be full rank[18]. Using a 200×200 grid for q_{11} and q_{12}, and using algorithm 3 to calculate the full state $\mathbf{q}$, it was verified that $\mathbf{H}$ is full rank in the whole workspace of the leg.

3.3.1 Dynamic Modelling Validation

To validate the analytical model[19], simulations were compared against the Simulink and Drake multibody models. The results of this comparison are illustrated in figure 12. It can be observed that the simulations are very close for a long simulation horizon. The properties and solvers of the simulation are presented in table 8. The simulation represents the response of the system,influenced solely by gravitational forces, with the following initial conditions:

$$[q_{MH,0},\ q_{11,0},\ q_{12,0}]^T = [0.5,\ 0.2,\ 0.3][rad]$$
$$[\omega_{MH,0},\ \omega_{11,0},\ \omega_{12,0}]^T = [0,\ 0,\ 0][rad/s]$$

[17] In the robot frame convention: ${}^{R}[\hat{\mathbf{x}}, \hat{\mathbf{y}}, \hat{\mathbf{z}}] = {}^{K}[\hat{\mathbf{x}}, \hat{\mathbf{z}}, -\hat{\mathbf{y}}]$.

[18] Adding the third constraint $h_y = 0$ or expressing the constraints in a different frame e.g. $\{MH\}$, results in depended constraints. Thus their jacobian is not full rank.

[19] As these are the dynamics that will be deployed in the controller, terms with magnitude smaller than $1e-3$, were discarded from the matrices of the dynamics. Their effect is indeed negligible, as seen in figure 12.

Table 8: Validation simulation parameters.

	solver	**Absolute Tolerance**	**Relative Tolerance**
matlab	ode15s	1e-7	1e-4
simulink	auto(ode15s)	1e-7	1e-4
drake	Runge-Kutta 3 (RK3)	-	1e-4

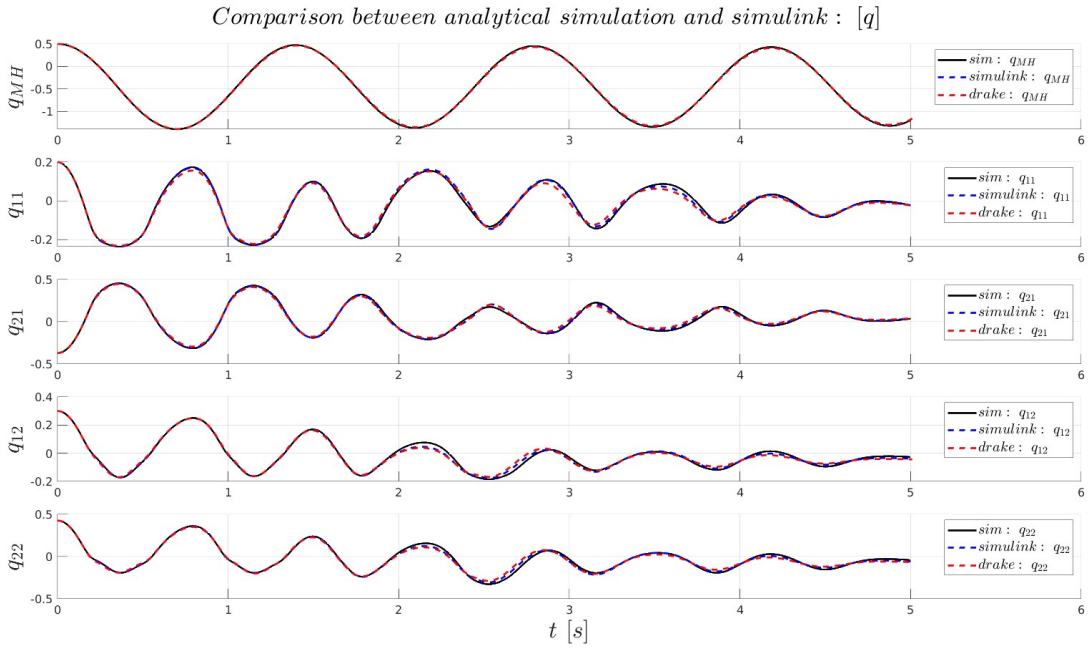


Figure 12: Simulation comparison between analytical and simulink model.

3.4 Workspace Constraints

The leg's movement is restricted by various workspace constraints, arising from potential self-collisions and interactions with the main torso. It is necessary to have a mathematical expression for these constraints. For that reason the procedure outlined in figure 13 was followed.

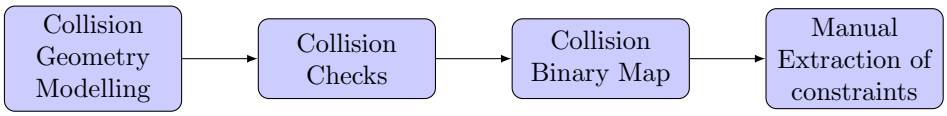
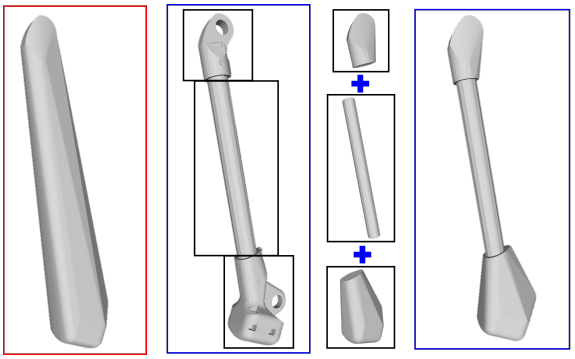
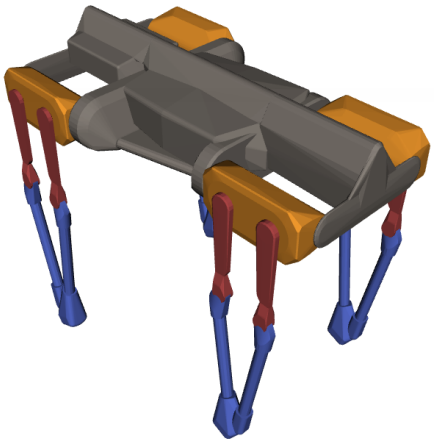


Figure 13: Constraint extraction procedure.

First, a suitable collision geometry must be created for the robot. The geometry is used both in the mathematical formulation of the constraints and in the simulation of *Olympus*. It should provide sufficient detail to capture all potential collisions while ensuring it does not unnecessarily restrict the robot's movement. At the same time, both matlab and drake handle only convex collision checking[20]. Therefore, the geometry must be decomposed into simpler convex shapes, resulting in a union of these shapes, as illustrated in Figure 14. This geometry was generated using meshlab[21].



(a) Example of decomposing the collision geometry of a link into convex shapes. The convex hull of the whole geometry is too coarse.



(b) Olympus Collision Geometry.

Figure 14: Details on the creation of the collision geometry.

Next, a grid search was conducted to identify the colliding configurations, represented in a binary map (0: Collision Free, 1: Colliding configuration). The detailed procedure is outlined in Algorithm 5. Using the binary map, mathematical expressions for the colliding configurations were extracted. Whenever possible, these expressions were formulated as polytopic state constraints, which take the following form:

$$\mathbf{C}\mathbf{x} < \mathbf{c} \tag{65}$$

If it is not possible to express the constraints using linear functions, then general non-linear constraints are used that take the form:

$$\mathbf{G}(\mathbf{x}) < \mathbf{g} \tag{66}$$

[20] See `https://www.mathworks.com/help/nav/ref/checkcollision.html` and `https://github.com/RobotLocomotion/drake/issues/20618`.

[21] `https://www.meshlab.net/`

Algorithm 5 Leg collision checking procedure.

procedure LEGCOLLISIONCHECK
  Set $N_{MH},\ N_{11},\ N_{12}$
  $\mathbf{A} = zeros(N_{MH},\ N_{11},\ N_{12})$
  for $k = 0$ **to** $N_{MH} - 1$ **do**
    for $i = 0$ **to** $N_{11} - 1$ **do**
      for $j = 0$ **to** $N_{12} - 1$ **do**
        $q_{MH} = (1 - k/(N_{MH} - 1))q_{MH,l} + (k/(N_{MH} - 1))q_{MH,u}$
        $q_{11} = (1 - i/(N_{11} - 1))q_{11,l} + (i/(N_{11} - 1))q_{11,u}$ ▷ l : lower limit, u : upper limit
        $q_{12} = (1 - j/(N_{12} - 1))q_{12,l} + (j/(N_{12} - 1))q_{12,u}$
        $\mathbf{q} =$ `state_estimation` $([q_{MH},\ q_{11},\ q_{12}]^T)$
        $\mathbf{A}(k, i, j) \leftarrow$ `collisionCheck`$(\mathbf{q})$ ▷ `collisionCheck` returns *true* if it detects collision.
      end for
    end for
  end for
end procedure

Below, each collision test and the corresponding constraints are presented.

Test 1: Self collisions. The resulting constraints are shown in figure 15, while their equations are presented below:

$$-q_{11} - q_{12} + 3.8020 > 0 \tag{67a}$$
$$-0.7284q_{11} - q_{12} - 0.2714 < 0 \tag{67b}$$
$$-1.3730q_{11} - q_{12} - 0.3726 < 0 \tag{67c}$$

Test 2: Collision with the main body. The resulting constraints are shown in figure 16 for selected q_{MH} values while their equations are presented below:

$$0.6708q_{11} - q_{12} - 1.9298 < 0 \tag{68a}$$
$$\mathcal{F}_{11}(q_{MH}) - q_{11} > 0 \tag{68b}$$
$$\mathcal{F}_{12}(q_{MH}) - q_{12} > 0 \tag{68c}$$

where $\mathcal{F}_{11},\ \mathcal{F}_{12}$ are smooth functions. $\mathcal{F}_{12}$ is a fifth degree polynomial and $\mathcal{F}_{11}$ is a function composed of the following (with $k_1 = 49.4,\ x_{0,1} = 0.9,\ k_2 = 30,\ x_{0,2} = 0.95$):

$$\sigma(x) = \frac{1}{1 + e^{-k(x - x_0)}},\ y_1 = -0.475x + 1.9,\ y_2 = -19.4x + 21.5$$

Their complete expressions are presented below:

$$\mathcal{F}_{11}(x) = \sigma_1 y_1 + (1 - \sigma_2)y_2$$
$$\mathcal{F}_{12}(x) = -0.8116x^5 + 8.9615x^4 - 38.6474x^3 + 81.1281x^2 - 83.6970x + 36.4564$$

The resulting nonlinear constraints are shown in figure 17.

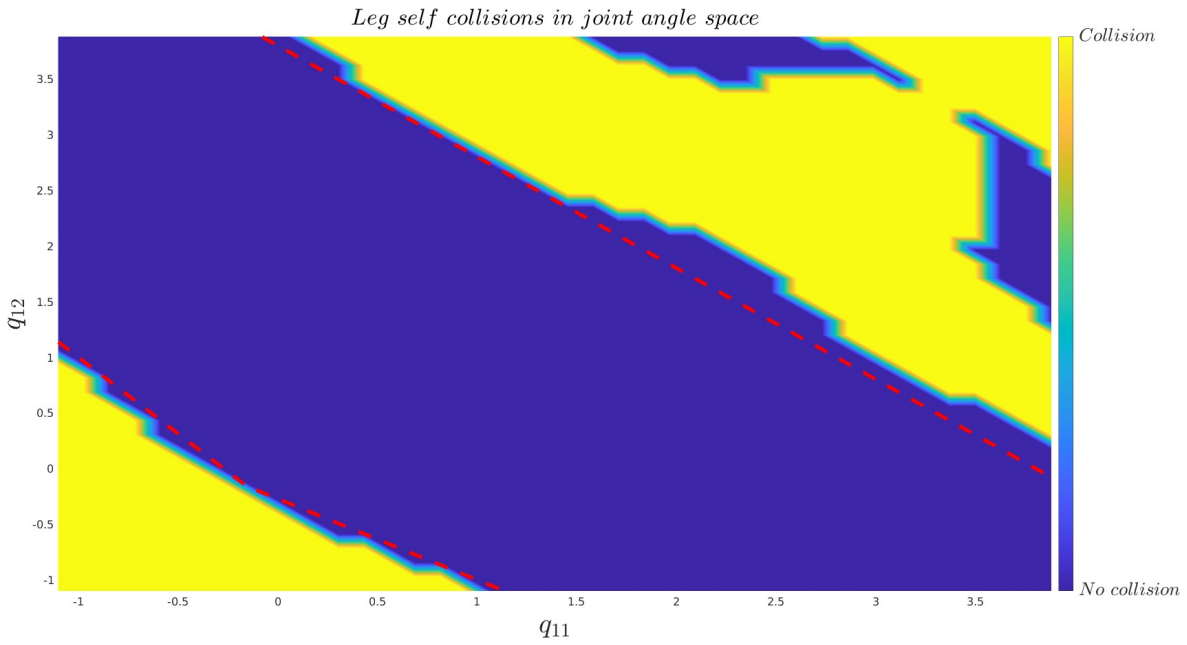


Figure 15: Leg workspace constraints resulting from checking leg self collisions.

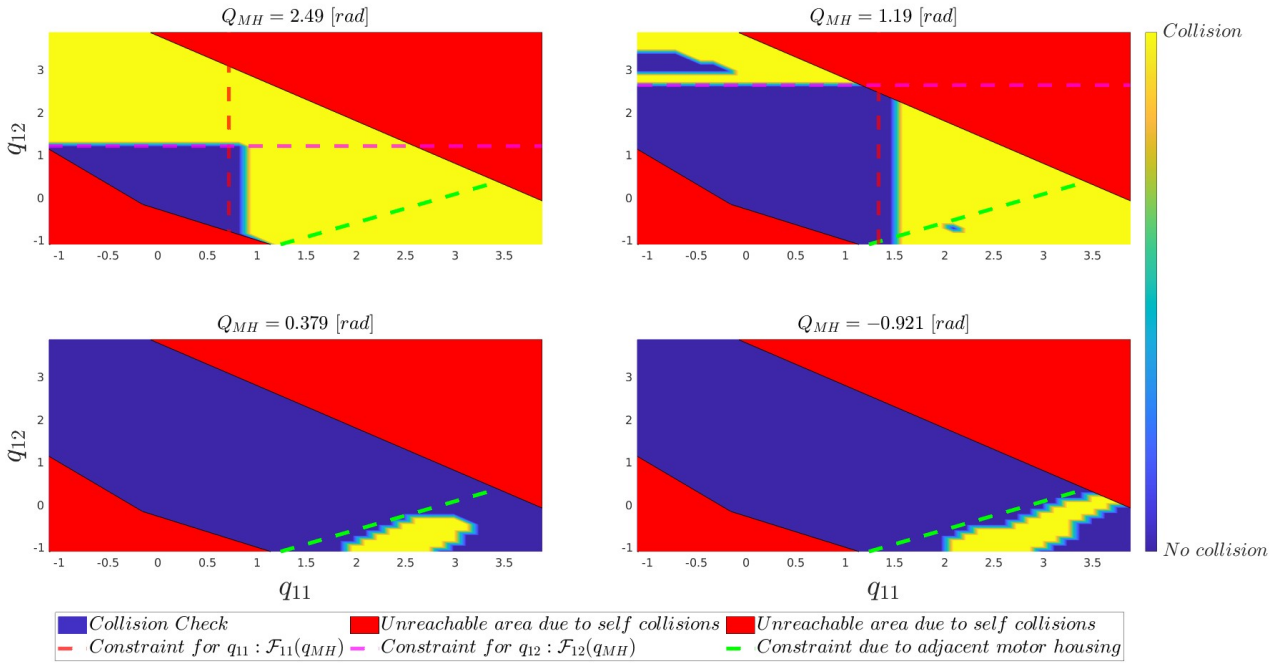


Figure 16: Leg workspace constraints resulting collisions with rest of the robot for selected q_{MH} values.

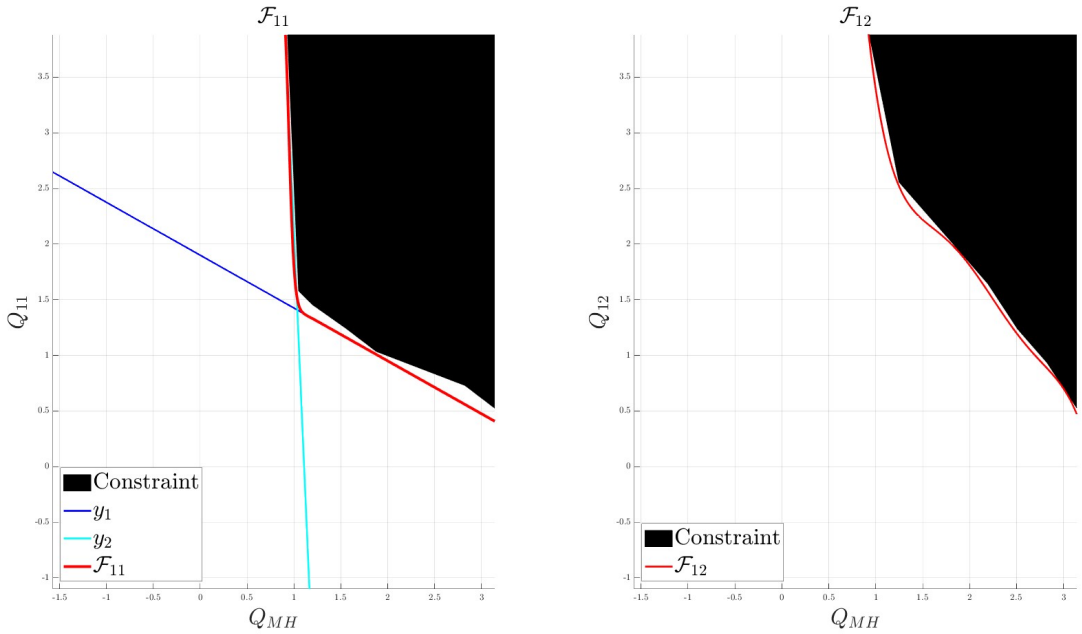


Figure 17: Nonlinear workspace constraints.

3.5 Modelling Implementation

There were two packages used to simulate the quadruped; simulink, used to prototype the MPC controller, and drake, used for the final verification simulations.

Some details that are shared across both simulating frameworks are the following:

- During a jumping phase, the robot is free falling. The system is analyzed from a reference frame attached to the robot, which is accelerating with an acceleration of $\mathbf{g}$. Consequently, the limbs do not experience acceleration relative to this body frame, and therefore, gravitational forces are not present in the reference frame used for the analysis. As only the orientation of the quadruped is of interest, and not its position, gravity is disabled in both simulations.
- Both frameworks do not explicitly handle close kinematic chains. A work-around is to add a virtual spring.

3.5.1 Simulink Implementation Details

The absence of gravity allows the use of a spherical joint to connect the quadruped to the simulation world. As the global position of the robot is of no interest, this joint does not introduce unnecessary degrees of freedom. At the same time, it encodes rotations using quaternions. So, the simulation does not suffer from gimbal lock and provides immediate feedback in quaternions for the controlling modules. To close the kinematic chain, a planar joint was used, as the kinematic chains move in the same plane by default. An overview of the simulation plant is shown in figure 18.

The simulation parameters, such as solver options and virtual spring stiffness and damping, are presented in table 9. The presence of the virtual spring, which had high stiffness to ensure the kinematic closure of the leg, motivated the choice of this particular solver, as it is suited for stiff problems.

Table 9: Simulink simulation parameters.

Solver	ode15s	**Relative Tolerance**	1e-3	K_{spring}	9e4
Max step size	1e-3	**Absolute Tolerance**	1e-6	D_{spring}	3e4

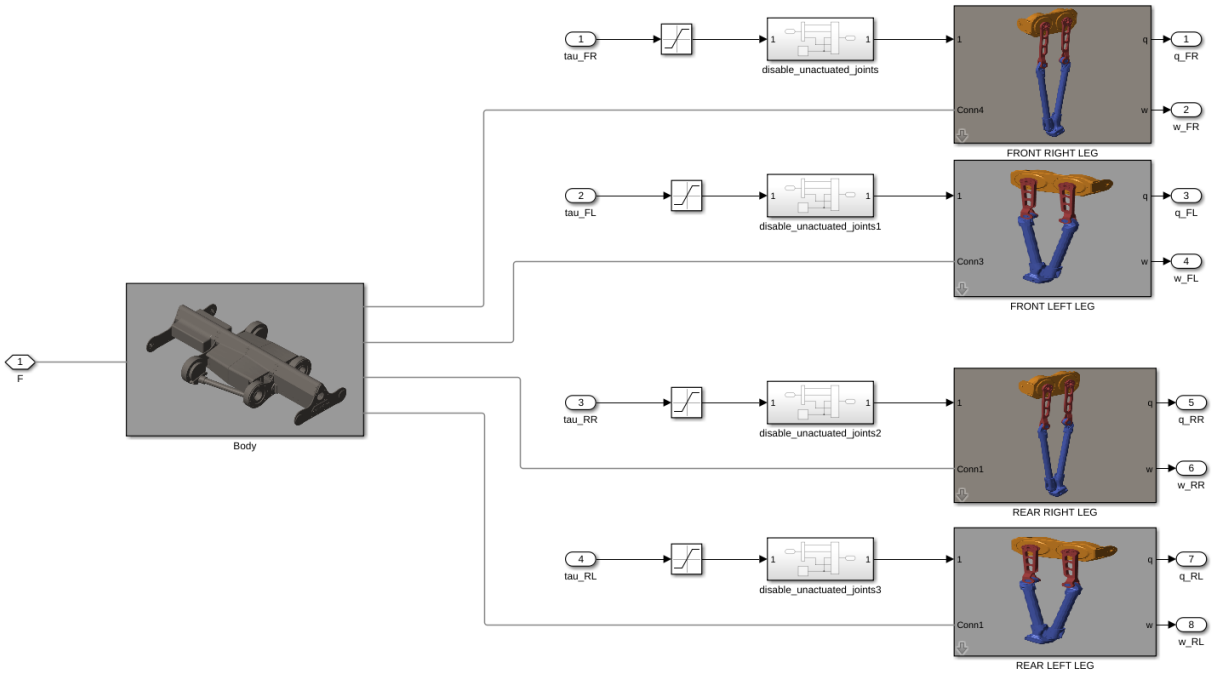


Figure 18: Model of olympus in simulink.

3.5.2 Drake Implementation Details

In drake, the torso is connected via a floating joint in the world, as it is the only joint that is parameterized using quaternions, and thus does not suffer from gimbal lock. To close the kinematic chain, a bushing joint was used. The simulation parameters, such as solver options and virtual spring stiffness and damping are presented in table 10. An overview of the simulation structure is shown in figure 18.

Table 10: Drake simulation parameters.

Solver	Runge Kutta 3 (RK3)	K_X	3e4	D_X	2.5e2
Max step size	1e-4	K_Y	0	D_Y	0
Relative Tolerance	1e-4	K_Z	3e4	D_Z	2.5e2

An overview of the simulation structure is depicted in figure 19. The system outputs are the torso and leg states and the actuation torques. The current implementation has integrated PID position controllers, which mimic the robot current control interface. Finally, the simulation can be launched with revolute body joints and the testbed, to mimic real life experiments.

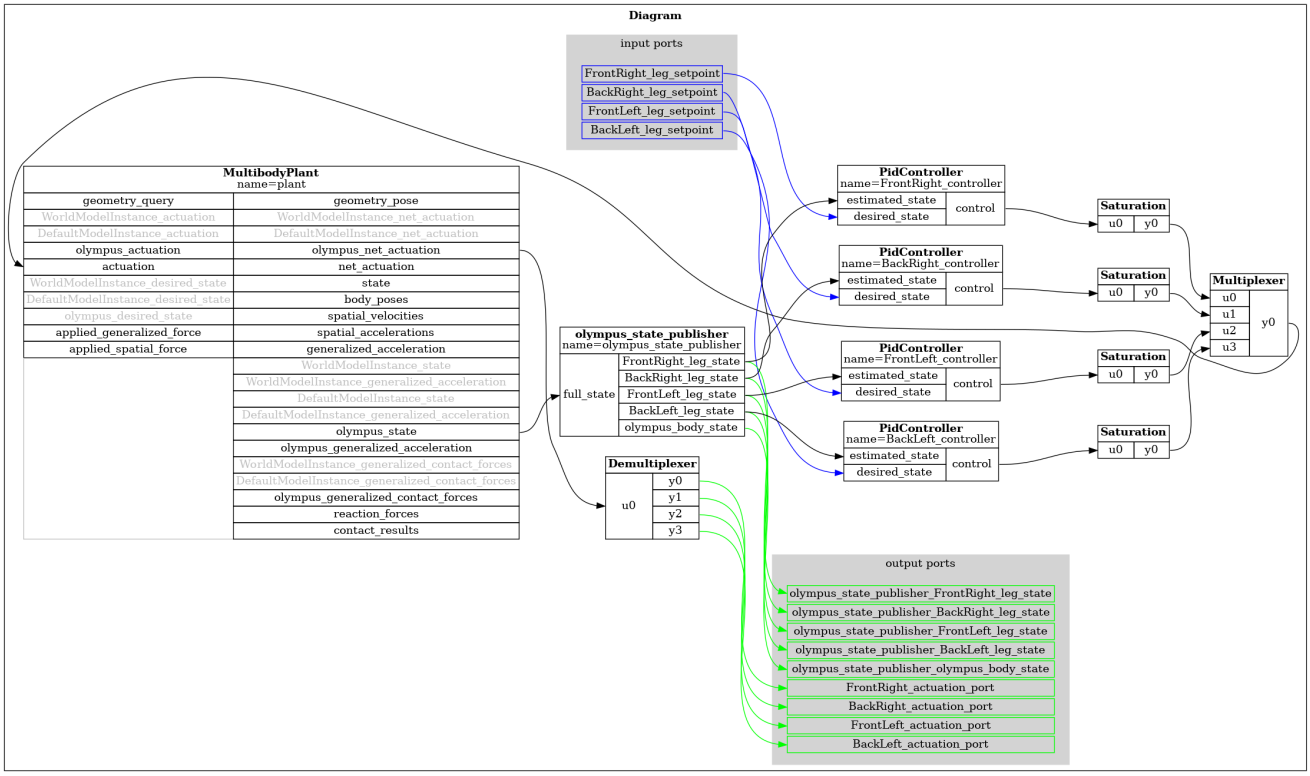


Figure 19: Structure of drake simulation

4. Control design

The goal of the control system is to stabilize the attitude of the quadruped's torso and reach the desired orientation q_D by moving the legs of the robot. The legs themselves are a complex dynamical system, with various workspace constraints (linear and nonlinear). Also, the relationship between the motor torques and leg movements, and the body-inflicted torques and the resulting rotational motion of the body, is not linear. For this reason, a hierarchical approach is used, a conceptual overview of which is shown in figure 20.

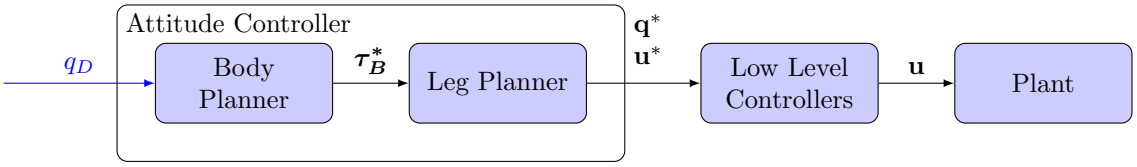


Figure 20: Control architecture conceptual overview.

Firstly, the torso trajectory is calculated, in the *Body Planner* module, along with the required body torques τ_B^*. At this stage, the robot is treated as a single rotating rigid body with fixed inertia[22]. Then, the resulting body torques are fed to the second module, the *Leg Planner*, which finds the corresponding leg movements to produce the desired body torques, while respecting the workspace limits. The full leg actuation torques $\mathbf{u}^*$ and desired joint angle trajectories $\mathbf{q}^*$ are then given as references to the low level controllers of the robot.

The planning modules, are designed using non-linear model predictive controllers (NMPC), in order to incorporate various input, task and workspace constraints as needed. The MPC is implemented using the acados open source framework, which was presented in section 2.5.4. To solve the Leg Planner MPC efficiently, additional modules are wrapped around the MPC. A resetting strategy is used to enable the controller to plan periodic trajectories online and a torque allocation method exploits the symmetries of the system to avoid collisions between legs and reduce computational cost by optimizing only one leg motion.

In this section, initially the *Body* and *Leg* planner modules will be analyzed. Next, the full controller architecture will be presented.

4.1 Body Planner

The Body Planner treats the robot as a single rotating rigid body with constant inertia (taken at the home position), and optimizes the torques that are acted upon it to rotate it to the specified orientation. The body planner is based on a nonlinear model predictive controller. The formulation of the MPC and the selection of appropriate parameters is presented below.

4.1.1 MPC Formulation

The MPC for the body planner has the following formulation:

$$\min_{\mathbf{x},\mathbf{u}} \int_0^{T_h} \left[\|\mathbf{f}_q\|^2_{\mathbf{Q}_q} + \|\boldsymbol{\omega}\|^2_{\mathbf{Q}_\omega} + \|\boldsymbol{\tau}_B\|^2_{\mathbf{R}} \right] dt + \underbrace{\|\mathbf{f}_{q,E}\|^2_{\mathbf{Q}_{q,E}} + \|\boldsymbol{\omega}_E\|^2_{\mathbf{Q}_{\omega,E}}}_{\text{Terminal cost terms}} + \mathbf{z}_{s,E}(\mathbf{s}_E) \tag{69}$$

so that

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \boldsymbol{\tau}_B) \tag{70a}$$
$$\mathbf{f}_{q,E} = \mathbf{0} \tag{70b}$$
$$\mathbf{x}_0 = \mathbf{x}(t) \tag{70c}$$
$$\boldsymbol{\tau}_B \in [\boldsymbol{\tau}_{B,l},\ \boldsymbol{\tau}_{B,u}] \tag{70d}$$

where:

- $\mathbf{x} = [\mathbf{q}^T,\ \boldsymbol{\omega}^T]^T = [w,\ q_x,\ q_y,\ q_z,\ \omega_x,\ \omega_y,\ \omega_z]^T$
- $\boldsymbol{\tau}_B = [\tau_{B,x},\ \tau_{B,y},\ \tau_{B,z}]^T$ is the body torque expressed in the body frame $\{B\}$.

  It was chosen not to have 4 $\boldsymbol{\tau}_B$, one for each leg, as with the current formulation, there would be multiple optimal solutions as the torque in each direction would split arbitrarily between each leg.

[22] This is a simplification, as the inertia changes as the legs move.

- The function $\mathbf{f}_q$ is the error metric (23) presented in (2.2.2). Using this metric, the shortest path is selected by the controller.
- $\mathbf{f}(\mathbf{x}, \boldsymbol{\tau}_B)$ is the dynamics constraint. The dynamics of the system are given by (25)
- The lower and upper values for the body torques are: $\tau_{B,i,u} = 10,\ \tau_{B,i,l} = -10\ [Nm]$ [12].
- The (slacked) terminal constraint (TC) ensures the solution is close to the desired orientation using the metric presented in (23). Setting the $q_E = q_D$ is not a correct formulation, as the deviation would be calculated treating quaternions as real valued vectors.
- $\mathbf{z}_E(\mathbf{s}_E)$ is the slack penalty function for the terminal constraint, with slack weights: $\mathbf{Z}_E = 0.1\mathbf{I}_3$
- $\mathbf{Q}_q = diag([100, 100, 100])$, $\mathbf{Q}_{q,E} = diag([1000, 1000, 1000])$
- $\mathbf{Q}_\omega = \mathbf{Q}_{\omega,E} = diag([10, 10, 100)$
- $\mathbf{R} = diag([10, 10, 10])$

To find the more suitable parameters, different variants of the above formulation were tested in closed loop re-orientation scenarios in simulink. The parameters[23] that were modified were: prediction horizon T_h, number of stages N, controller update rate T_s and simulation steps N_s, which are the internal simulation steps of the nonlinear MPC. Also, some variants do not have a terminal constraint, while others do not have a terminal cost.

The tests include 100 re-orientations scenarios (returning to $\mathbf{x}_{ref} = [1,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0]$) from a random initial orientation. The simulation settings are presented in table 11. The dynamic plant in simulink is a rigid body with inertia equal to the inertia of the robot in the default configuration. The results are presented in table 12. Settling time is the time it takes for the body to reach and stay within the specified threshold, which here was chosen as 3°.

Table 11: Settings for comparing body planner MPC.

Relative Tolerance	$1e-3$	**Max step size**	$1e-3$	$\mathbf{q}_0$	r,p,y $\in [-\pi, \pi]$
Absolute Tolerance	$1e-6$	$\Delta\theta$ **convergence**	3°	$\mathbf{w}_0$	**0**

The above formulation can be extended by is penalizing simultaneous torques at different directions (Cross Variant). To achieve this, the following nonlinear term is added to the cost function:

$$\mathbf{f}_{cross} = (K_c \boldsymbol{\tau}_x \boldsymbol{\tau}_y)^2 + (K_c \boldsymbol{\tau}_y \boldsymbol{\tau}_z)^2 + (K_c \boldsymbol{\tau}_z \boldsymbol{\tau}_x)^2 \tag{71}$$

Table 12: Comparison of body planner MPC performance with different parameters.

Settings	Variant 1	Variant 2	Variant 3	TC+Cost	Only TC	Cross Variant
T_h	5	2	2	2	2	2
N	50	20	4	4	4	4
T_s	0.1	0.1	0.5	0.5	0.5	0.5
N_s	1	1	5	5	5	5
Terminal Cost	yes	yes	yes	yes	no	yes
Terminal Constraint and initialization	no	no	no	yes	yes	yes
Results						
AVG(solution time) $[ms]$	1.86	0.467	0.406	0.46	0.42	1.63
MAX(solution time) $[ms]$	117	7.26	5.63	2.62	3.34	25.9
Converged /100	98	**100**	95	**100**	**100**	**100**
AVG(settling time) $[s]$	2.09	2.0104	2.3286	2.2667	2.3458	3.4114
STD(settling time) $[s]$	0.52	0.41509	0.59335	0.55475	0.57844	0.7467

Variant 2 is superior than variant 1 as there are fewer decision variables, resulting in faster and more reliable convergence. In variant 3, using a lower number of stages and a higher number of simulation steps, the same discretization of the dynamics is achieved, but the controller accounts for a less reactive leg controller that cannot instantaneously produce a torque in a different direction. The difference between variant 2 and 3 is showcased in figure 21, where the body planner was used with initial condition[24] $q_0 = [0.7011,\ 0.0923,\ 0.5610,\ 0.4305]^T$. It can

[23] The parameters are explained in section 2.4.3

[24] It is $rpy = [\pi/2,\ \pi/3,\ \pi/4]$

be observed that the overall trend for the torque is the same, but the MPC accounts for the lack of instantaneous torque delivery from the legs. Finally, it can be observed that variant 3 converges more reliably with the existence of terminal constraint.

The usefulness of extending the objective with (71) comes from the architecture of the control system, which is discussed in section 4.2.3. The performance of this formulation is showcased in table 12 and a representative result is shown in figure 21. It can be observed that only one direction has a significant torque reference at each instance. Also, it is a stable formulation, as all tests were successful. It should be noted that the increase in computing time is not a significant issue, as it remains minimal and allows for deployment at a maximum rate of 500Hz. However, the average increase in settling time by one second should be taken into account.

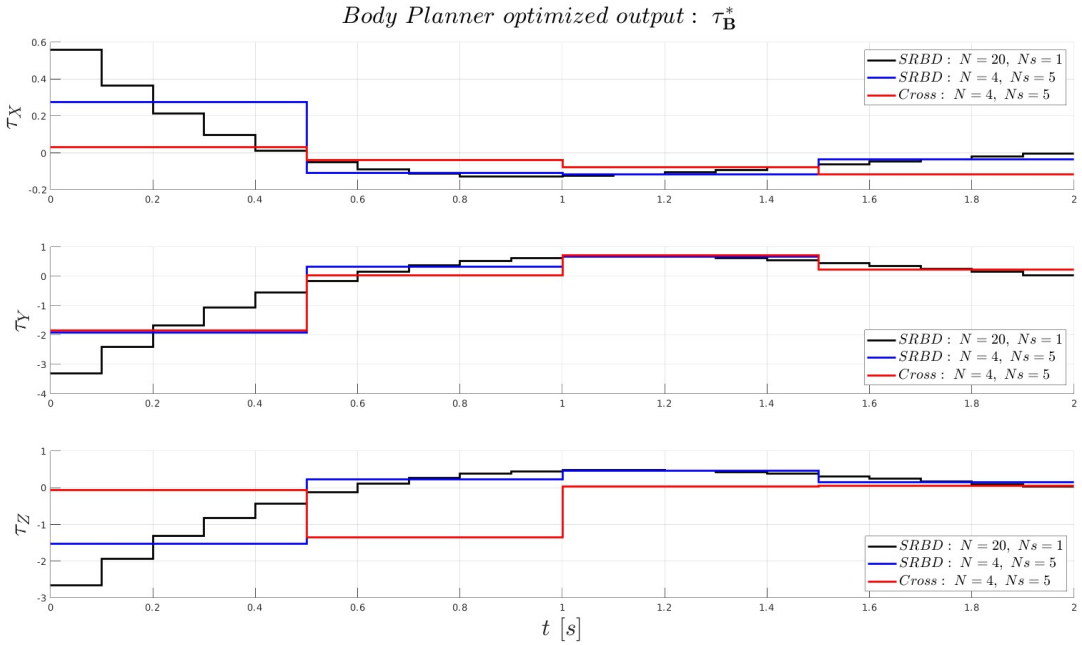


Figure 21: Body Planner optimized torques based on different formulations.

The results highlight that the body can be controlled to reach a desired orientation even with large sampling intervals. However, the stabilization once close to the desired value cannot be tested with that simulation, as the body changes its inertia when moving its legs.

4.2 Leg Planner MPC

The Leg Planner is responsible for finding the leg trajectories that generate the torques $\boldsymbol{\tau}_B^*$ from the Body Planner. It is based on a torque tracking MPC complemented by auxiliary modules designed to make the problem solvable in real-time. Initially, the formulation and tuning of the MPC are discussed, followed by an analysis of the additional modules.

4.2.1 Position Tracking MPC

Firstly, to test the capabilities of an MPC based planner for the leg, the controller was tested in a simple position tracking task. The formulation is presented below:

$$\min_{\mathbf{x},\mathbf{u}} \int_0^{T_h} \left[\|\mathbf{x} - \mathbf{x}_{ref}\|_{\mathbf{Q}}^2 + \|\mathbf{u}\|_{\mathbf{R}}^2 \right] dt + \|\mathbf{x}_E - \mathbf{x}_{ref,E}\|_{\mathbf{Q}_E}^2 + \mathbf{z}_{s,E}(\mathbf{s}_E) \tag{72}$$

so that

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}) \tag{73a}$$
$$\mathbf{x}_0 = \mathbf{x}(t) \tag{73b}$$
$$\mathbf{x}_N = \mathbf{x}_E \tag{73c}$$
$$\mathbf{x} \in [\mathbf{x}_l,\ \mathbf{x}_u] \tag{73d}$$
$$\mathbf{u} \in [\mathbf{u}_l,\ \mathbf{u}_u] \tag{73e}$$
$$\mathbf{C}\mathbf{x} \in [\mathbf{g}_l,\ \mathbf{g}_u] \tag{73f}$$
$$\mathbf{h}(\mathbf{x}) \in [\mathbf{h}_l,\ \mathbf{h}_u] \tag{73g}$$
$$\tag{73h}$$

where:

- $\mathbf{x}_{ref} = [\mathbf{q}_{ref}^T,\ \mathbf{0}^T]^T$. $\mathbf{q}_{ref}$ is the goal configuration for the position tracking task.
- $\mathbf{Q} = diag(\mathbf{Q}_q, \mathbf{Q}_{\dot{q}}),\ \mathbf{Q}_q = diag(50, 50, 50, 50, 50),\ \mathbf{Q}_{\dot{q}} = diag(5, 5, 5, 5, 5)$
- $\mathbf{Q}_E = 0.1\mathbf{Q}$
- $\mathbf{R} = 10\mathbf{I}_3$
- $\mathbf{f}(\mathbf{x}, \mathbf{u})$ are the closed chain dynamics (28) described in section 2.3.2.
- $\mathbf{C}\mathbf{x}$ are the linear (polytopic) workspace constraints (67,68a).
- $\mathbf{h}(\mathbf{x})$ are nonlinear constraints and include the closure constraint[25] (64) and the non-linear workspace constraints (68b, 68c).
- The terminal constraint is slacked with $\mathbf{Z}_{x,E} = 10\mathbf{I}_{10\times10}$. The slack variable weights for all the other constraints are presented in table 13, as they are the same with the torque tracking formulation which will be presented in section 4.2.2.

Table 13: Slack weights for Leg Planner MPC.

State Constraint	**Polytopic Constraints**
$\mathbf{Z}_x = 200\mathbf{I}_{10\times10}$	$\mathbf{Z}_p = 50\mathbf{I}_{4\times4}$
$\mathbf{z}_x = 10^3\mathbf{1}_{10\times1}$	$\mathbf{z}_p = 10^3\mathbf{1}_{4\times1}$
Closure Constraint	**Workspace Constraints**
$\mathbf{Z}_{h,closure} = 10\mathbf{I}_{2\times2}$	$\mathbf{Z}_{h,ws} = 10^2\mathbf{I}_{2\times2}$
$\mathbf{z}_{h,closure} = 10^2\mathbf{1}_{2\times1}$	$\mathbf{z}_{h,ws} = 10^3\mathbf{1}_{2\times1}$

The only hard constraints are the dynamics (73a) and input constraints (73e). Slacking all the other constraints, ensures the optimization problem is always feasible. Also, as discussed in section 3.3, the dynamics are well defined across the whole workspace, and thus the optimization problem is well posed.

To find suitable parameters for the MPC that ensure its convergence, some basic tests took place. The setup is as follows; the system is in a random initial position inside the workspace and tracks a similarly random[26] setpoint. The system starts from rest, meaning $\dot{\mathbf{q}} = \mathbf{0}$. The purpose of the initial test was to evaluate the impact of the constraints, and thus the parameters that changed were the constraints and their corresponding slack variables and the number of stages. The horizon time is set at $T_h = 1s$. A second test followed to tune the MPC.

The results for the initial test are shown in table 14. It is apparent that the constraints do not influence the solution performance that much. Removing them completely makes the solution $5ms$ faster on average. Still, in the unconstrained case, there are outliers with ten times the average solution time. Finally, the generated trajectories contain many more points outside of bounds, which can result in unwanted collisions in the real system.

However, the results above highlight the difficulty of the MPC to converge for large prediction horizons in an online setting. This observation is supported by a second set of tests, the tuning tests, where T_h and N were changed. The results are presented in table 15. It can be observed that with $N = 30$, when the horizon time is lower, the average solution time slightly decreases, but it does so in a much more repeatable way. The standard deviation of the solution time is halved. Also, with lower T_h, fewer sampling points are needed, which results in faster solution times. Thus, the key parameter that affects solution times is the prediction horizon.

[25] As discusses in section 2.5.4, acados does not support equality constraints directly. So the closure constraint is formulated as a

Table 14: Comparison of leg position MPC performance with different parameters for $T_h = 1s$.

Settings	High N	Low N	Normal N	Light slacks	Unconstrained
N	50	20	30	30	30
Linear constraints	yes	yes	yes	yes	no
nonlinear constraints	yes	yes	yes	lightly slacked	no
Results					
AVG(solution time) [ms]	85.2	41.9	54.3	50.7	44.5
STD(solution time) [ms]	35.2	37.6	35.9	39.4	30
MAX(solution time) [ms]	885	369	581	511	455
Converged /1000	999	986	998	994	997
Constraint violations	57	23	31	167	209

Table 15: Comparison of leg position MPC performance with different parameters for lower prediction horizons.

Settings					
N	30	30	20	30	15
T_h	1	1	1	0.75	0.5
Constraints	yes	no	yes	yes	yes
Results					
AVG(solution time) [ms]	49.1	44	31.3	45.5	22.6
STD(solution time) [ms]	38.4	38.4	14.8	**18.3**	**6.48**
MAX(solution time) [ms]	516	446	212	209	55.3
Converged /200	199	198	200	200	200
Constraint violations	8	54	2	13	7

The key takeaway is that the current model cannot be used to optimize large trajectories online. Therefore, the MPC must be wrapped with additional logic to track the required torque from the body planner. This additional logic is presented in sections 4.2.3 and 4.2.4. The MPC that is investigated for tracking the body planner torque in the next section has a short prediction horizon for that reason.

4.2.2 Torque Tracking MPC

The formulation for the torque tracking MPC is the following:

$$\min_{\mathbf{x},\mathbf{u}} \int_0^{T_h} \left[\|\boldsymbol{\tau}_B - \boldsymbol{\tau}_B^*\|^2_{\mathbf{W}_{track}} + \|\mathbf{u}_{2,3}\|^2_{\mathbf{W}_u} + \|\mathbf{x} - \mathbf{x}_{ref}\|_{\mathbf{Q}^2} \right] dt + \|\mathbf{x}_E - \mathbf{x}_{ref,E}\|^2_{\mathbf{W}_E} \tag{74}$$

with the same constraints as in the position control task (73), apart from a terminal constraint on state.

- The term $\|\boldsymbol{\tau}_B - \boldsymbol{\tau}_B^*\|^2_{\mathbf{W}_{track}}$ tracks the body torque that comes from the body planner. This torque is calculated using the following equation:

$$\boldsymbol{\tau}_{B->L} = \boldsymbol{\tau}_B = {}^B\mathbf{R}_{MH}\mathbf{u} = \begin{bmatrix} u_1 \\ cos(q_{MH})(u_2 - u_3) \\ -sin(q_{MH})(u_2 - u_3) \end{bmatrix} \tag{75}$$

- The term $\|\mathbf{u}_{2,3}\|^2_{\mathbf{W}_u}$ penalizes torques to smoothen the output of the controller.

- The terms $\|\mathbf{x} - \mathbf{x}_{ref}\|_{\mathbf{Q}^2}$ and $\|\mathbf{x}_E - \mathbf{x}_{ref,E}\|^2_{\mathbf{W}_E}$ guide the controller to specific setpoints in the workspace. Biasing the optimization with this term compromises the torque tracking capabilities but greatly helps the solver converge. More details about the selection of these reference points will be given in section 4.2.4.

To tune this MPC, 1000 tests were done per variation. A random point inside the workspace was set as initial condition $\mathbf{q}_0$. Some handpicked reference setpoints were chosen. These are the same as the ones used in the resetting algorithm presented in section 4.2.4. After some initial converging settings were found, the maximum

two sided inequality constraint with identical bounds.

[26] Both the initial and final setpoints are valid setpoints which satisfy all the constraints.

torque that can be tracked from the MPC was found. Having the maximum torque that can be tracked, random torque values within the maximum limits were generated as the input reference. As pitch and yaw desired moments can be contradictory, only one of these torques was requested each time. This is in accordance with the architecture detailed in section 4.2.3. The results of the testing are presented in the following table:

The following parameters were constant throughout the testing:

- Slack weights are the ones presented in table 13.
- $\mathbf{W}_{closure} = 0$ and $\mathbf{W}_u = 5\mathbf{I}_2$
- $\mathbf{W}_{track} = diag\left(\max(120\boldsymbol{\tau}/\left\|\boldsymbol{\tau}\right\|, [5,5,5])\right)$. This formulation tracks the main components of the desired torque while penalizing unwanted torque production.
- The initialization for the MPC is the initial state of the leg for all stages $\mathbf{x}_i = \mathbf{x}_0$ and zero input $\mathbf{u}_i = \mathbf{0}$.
- Prediction horizon: $T_h = 0.1s$

So the parameters that were varied in these tests are: the number of stages, the value of the *Levenberg-Marquardt* parameter, the value of $\mathbf{Q}_E$ and $\mathbf{Q}$.

Table 16: Comparison of leg torque tracking MPC performance with different parameters for $T_h = 0.1s$.

Settings	Variant 1	Variant 2	Variant 3	Variant 4
N	10	10	10	5
$\mathbf{Q}$	$\mathbf{0}$	$\mathbf{0}$	$D(0.1\mathbf{I}_5, 2\mathbf{I}_5)$	$\mathbf{Q}_{v3}$
$\mathbf{Q}_E$	$D(0.01\mathbf{I}_5, 0.02\mathbf{I}_5)$	$\mathbf{0}$	$\mathbf{Q}_{E,v1}$	$\mathbf{Q}_{E,v1}$
$\lambda_{Levenberg-Marquadt}$	0.075	0.075	0.075	0.05
Results				
AVG(solution time)	76.2	170	56.7	27.9
STD(solution time)	25	15.3	12.5	11.8
max weighted torque error	[0.2,6,13]%	[0,11.7,13]%	[0.6,13.4,21.7]%	[0.7,8.7,12.8]%
Converged	968/1000	**40/1000**	100/100	981/1000
Constraint violations	150	20	14	139

The following conclusions are drawn from the results above. First, reference setpoints are required to improve convergence and solution time. Also, even though stage weights on state tracking compromise torque tracking, they improve convergence and solution speed. Finally, the main factors which influence solution time are the number of stages and the chosen *Levenberg-Marquardt* parameter. To investigate the performance of the prediction horizon, the following tests were conducted:

Table 17: Comparison of leg torque tracking MPC performance with varying prediction horizon.

Settings	Variant 1	Variant 2	Variant 3	Variant 4
N	5	5	6	8
T_h [ms]	75	100	125	150
$\lambda_{Levenberg-Marquadt}$	0	0	0	0.015
Results				
AVG(solution time)	8.78	12.2	18.6	34.1
STD(solution time)	5.69	12.5	20.2	29.2
MAX(solution time)	43.6	88.8	105	139
max weighted torque error	[0.5,4.2,21.4]%	[0,9,19.8]%	[2.3,9.1,19,5]%	[5.6,9.6,27.6]%
Converged /100	100	99	98	96

It can be seen that the *Levenberg-Marquardt* parameter can be ditched when the number of stages is small, and the solution time is greatly improved. Also, increasing the prediction horizon, decreases the torque tracking capabilities as can be observed from the latest variant with $T_h = 150ms$.

4.2.3 Torque Allocation

The workspace of each leg interferes with the workspace of the other legs. To implement collision avoidance between different legs in the MPC, additional nonlinear constraints must be introduced. In fact, these constraints would use the planned trajectories of the other legs, which are not known a priori. Therefore, at each MPC step, an iterative approach is necessary, where each iteration uses the previously[27] optimized trajectory of all legs to formulate the constraints. This approach is costly and not suitable for online planning of trajectories. Also, knowledge from previous MPC updates cannot be used, as the update rate of the Leg Planner MPC is equal to its prediction horizon. Moreover, the MPC must be solved separately for each leg, further increasing the computational cost. Finally, it cannot be guaranteed that the legs will not enter in a configuration from which escape is either impossible or highly costly. The inability of the MPC to handle large horizons (even with simpler constraints) that was observed in section 4.2.1, enhances this problem.

To avoid this issue, a torque allocation strategy is employed that takes advantage of the following facts:

- Each leg has a similar range of motion[28]; The motor housing joint has a range of around 270° while the hip motors have a range of 291°.
- Each chain has identical link lengths, as seen in table 1 and similar inertial characteristics.
- The workspace constraints are almost identical. Differences exist only in the roll direction and in the workspace constraints that come from the collision of the front legs with the rear motor housing. Both of them can be handled with suitable offsets.

Using the above observations, the optimal trajectory of one leg, the front right one, can be *projected* into the other using simple linear transformations.

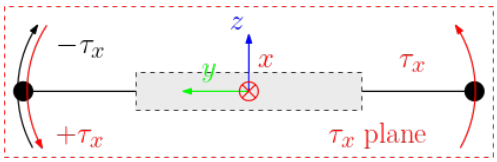
Same-side legs: The workspace of same-side legs interferes the most. Thus, to avoid same-side leg collisions, the rear legs mimic the movements of the front ones. This allocation strategy is implemented with the following transformation:

$$\begin{Bmatrix} q_{MH} \\ q_{11} \\ q_{12} \end{Bmatrix}_{RR} = \underbrace{\begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}}_{\mathbf{Pr}_R} \begin{Bmatrix} q_{MH} \\ q_{11} \\ q_{12} \end{Bmatrix}_{FR}, \quad \begin{Bmatrix} q_{MH} \\ q_{11} \\ q_{12} \end{Bmatrix}_{RL} = \underbrace{\begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}}_{\mathbf{Pr}_L} \begin{Bmatrix} q_{MH} \\ q_{11} \\ q_{12} \end{Bmatrix}_{FL} \tag{76}$$

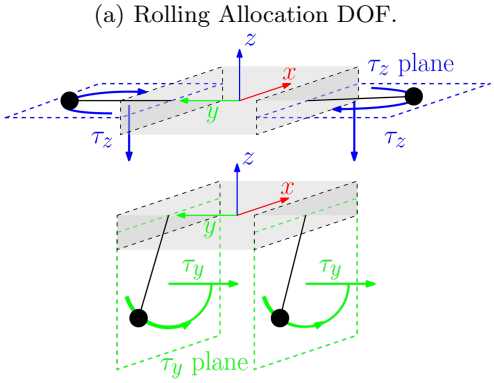
Opposite-side Projection: Given the relationship between same-side legs, only one transformation must be further specified; the transformation of the trajectories from the front right to the front left leg. From (75), the only actuator that produces rolling torques is the one actuating the hip. Actuating the hip joints produces pitch and yaw moments, depending on q_{MH}. Thus, the projection has two *degrees of freedom*; whether rolling moments will cancel out (by extending and retracting the legs of opposite sides in unison) or not, and whether the legs will produce pitch or yaw moments.

These degrees of freedom are shown in figure 22, where a schematic of the robot and its legs (shown as point masses) is depicted. The movement of the legs is indicated along with the induced torque in the torso.

As it can be seen in figure 22a, when opposite-side legs extend their legs outward and retract them inward from the sagittal plane (τ_y plane) simultaneously, the net rolling torque on the torso is close to zero. When one side retracts inward and the other extends outward, a net rolling moment in the torso is induced. In short, the **Rolling DOF** is associated with the movements of opposite-side limbs in the coronal plane.



(a) Rolling Allocation DOF.



(b) Pitch-Yaw Allocation DOF.

Figure 22: Allocation degrees of freedom (DOF).

[27] From the previous iteration, not previous MPC update.

[28] Defined as the length of the interval $[q_{i,L}, q_{i,U}]$ where $q_{i,L}, q_{i,U}$ are the lower and upper limit of the i-th joint. The joint ranges can be calculated from table 3

Producing pitch and yaw moments is more complicated as the legs must move near particular planes, as indicated by (75); near a parasagittal plane (τ_y plane) for pitch and near the transverse plane (τ_z plane) for yaw. Assuming that they do indeed move in the right plane for pitch and yaw torques, a particular mapping is still required to actually achieve the desired effect. As illustrated in figure 22b, when the legs are in the τ_y plane inducing a pitch moment in the torso requires synchronized movement of opposite-side legs from front to rear and vice versa. Conversely, in the τ_z plane for yaw, creating a yaw moment necessitates a leg from one side moving forward while a leg from the opposite moves backward. Therefore, the **Pitch/Yaw DOF** controls whether legs move forward and backwards in unison.

The **Rolling DOF** transformation is described in (77) while the **Pitch/Yaw DOF** transformation in (78).

$$q_{MH,FL} = \begin{cases} q_{MH,FR} - 35°, & \text{rolling mode} \\ -q_{MH,FR}, & \text{canceling roll} \end{cases} \tag{77}$$

$$\begin{Bmatrix} q_{11} \\ q_{12} \end{Bmatrix}_{FL} = \underbrace{\begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}}_{\mathbf{Pr}_{pitch}} \begin{Bmatrix} q_{11} \\ q_{12} \end{Bmatrix}_{FR}, \quad \begin{Bmatrix} q_{11} \\ q_{12} \end{Bmatrix}_{FL} = \underbrace{\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}}_{\mathbf{Pr}_{yaw}} \begin{Bmatrix} q_{11} \\ q_{12} \end{Bmatrix}_{FR} + \begin{Bmatrix} 30° \\ 30° \end{Bmatrix} \tag{78}$$

However, this projection couples all the leg movements. Depending on how this coupling is achieved, some torques cancel out. The advantage of opposite side projection is that by correctly specifying the projection, opposite side collisions are avoided and only one MPC is needed to produce leg reference trajectories. For these reasons, this strategy was employed. However, the legs can produce torques in a discrete way, which results in sub-optimal manoeuvres, as indicated by the cross variant in table 12.

So according to the selected projection settings, the controller can operate in four different modes:

1. **Roll Mode**: It uses the *rolling mode* and *pitch* transformations.
2. **Pitch Mode**: It uses the *cancel roll* and *pitch* transformations.
3. **Yaw Mode**: It uses the *cancel roll* and *yaw* transformations.
4. **Stabilization Mode**: It uses the *cancel roll* transformations and inherits the previous *pitch* or *yaw* projection setting.

The operating mode is selected based on the largest term of $\boldsymbol{\tau}_B^*$. The controller switches to stabilization mode once the robot's orientation is within a defined angular threshold of the target orientation.

4.2.4 Reset Algorithm

The short horizon of the leg planner MPC does not allow it to plan the whole motion on its own. When the leg reaches the boundary of the workspace while tracking a reference torque, reaching a resetting point from which it can efficiently resume the torque tracking, goes against its short term objective. Indeed, this resetting manoeuvre is most likely a movement that produces torque in the opposite direction. Thus, a resetting strategy must be wrapped around the MPC. Previous work used a similar resetting strategy, but using a heuristic way to produce the leg trajectories, treating the leg as a pendulum [12]. The developed algorithm is inspired by that work and takes into consideration the conclusions of section 4.2.2; the use of reference setpoints to improve the convergence of the torque tracking MPC.

Intuitively, the movement of the legs is separated into phases. There is a phase in the trajectory that produces the desired torque and another one where the leg goes to an advantageous position to continue tracking the torque in the feature. These are the *TORQUE* and *RESET* phases respectively. These phases are infused by two intermediate phases: *EXTENSION* and *CONTRACTION*. During these phases, the legs extend or contract in length, causing the center of mass of the entire leg to move farther from or closer to the body, thereby increasing or decreasing the induced torque on the torso from the limb's motion.

The resulting resetting algorithm functions as a Finite State Machine (FSM), where each state/phase is linked with a set of weights for the MPC, and a corresponding reference setpoint. These setpoints were selected manually, through experimentation. Transitions between states occur when the leg configuration reaches the specified setpoint within a defined accuracy, which itself is a parameter of the resetting algorithm.

The parameters of the FSM are listed below:

- A reference setpoint for the Leg Planner $\mathbf{q}_i$ (3 parameters [29]).
- Weight values for the Leg Planner (12 parameters). The weight values that change are $\mathbf{W}_{track}$ (3 values), $\mathbf{W}_u$ (1 value) and $\mathbf{Q}_{\mathbf{q}}$, $\mathbf{Q}_{\mathbf{q},E}$, $\mathbf{Q}_{\dot{\mathbf{q}}}$, $\mathbf{Q}_{\dot{\mathbf{q}},E}$ (8 values, 2 values each, one for q_{MH} and one for the rest of the states).
- A weight matrix $\mathbf{W}_i$ needed for the state transition function 79 (3 parameters[30]).
- A threshold value $\mathcal{T}_i$ needed for the state transition function (1 parameter).

The state transition function is the following:

$$\|\mathbf{q}(t) - \mathbf{q}_i\|_{\mathbf{W}_i} \leq \mathcal{T}_i \tag{79}$$

The FSM is presented in figure 23 while its geometric interpretation is shown in figure 24.

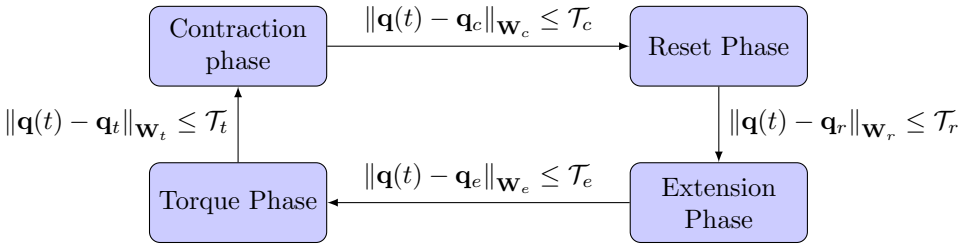


Figure 23: Finite State Machine of the resetting algorithm.

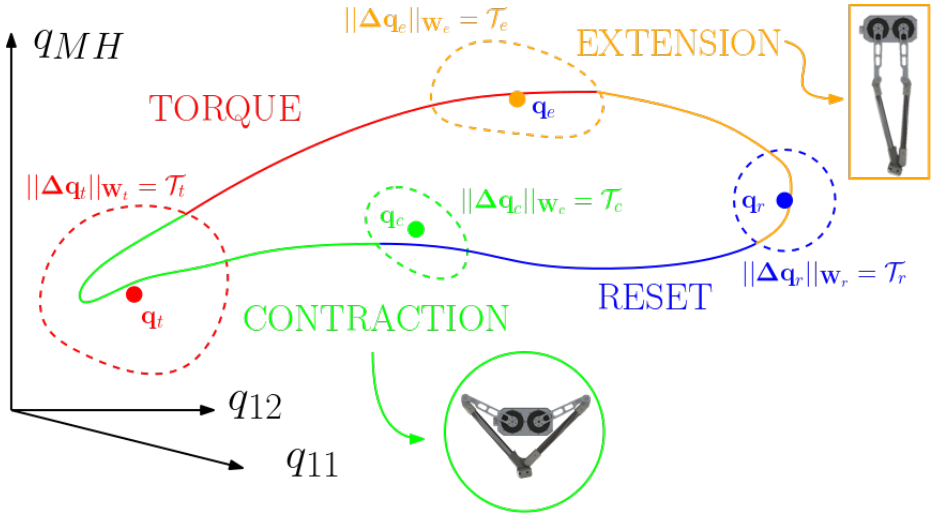


Figure 24: Resetting algorithm conceptual overview. Each colour is associated with a certain phase. When the leg is close to the phase reference setpoint within the defined accuracy, the FMS transitions to the next phase. The contracted and extended configurations of the legs are also being shown.

Each state has 21 parameters, which results in 84 parameters for the whole FSM. Due to the allocation algorithm, these parameters are different for each rotation mode. Thus, the whole controller has 252 parameters. These parameters were manually tuned to get satisfactory results.

The legs have one configuration where they are fully extended and one where they are fully contracted, thus contraction and extension phases were selected near these configurations. However, depending on the direction of the desired body torque $\boldsymbol{\tau}_B$, the torque and reset setpoints are interchanged. This introduces some anisotropy in the controller, which is observed in the simulation results presented in section 5.1.

[29] In reality it is 5 parameters as there are 5 joint angles. However, only 3 of them are independent. The other 2 are calculated (offline) using the state estimation algorithm 3.

[30] It is usually diagonal

It must be noted that it is crucial to detect the phase change when it happens, as some of the setpoints were near the workspace boundaries to increase the range of motion and the torque produced. Thus, in the actual implementation, in each control loop it is checked whether a phase change occurs.

4.3 Architecture Overview

The overall control architecture is shown in figure 25. The body planner takes the latest orientation and angular velocity estimates of the body $q_B(t), \boldsymbol{\omega}_B(t)$ and calculates the optimal body torques $\boldsymbol{\tau}_B^*$. The Leg planner reads the latest joint positions and velocities $\mathbf{q}(t),\ \dot{\mathbf{q}}(t)$ and latest virtual torques $\boldsymbol{\tau}_B^*$, checks whether there is a mode or phase change and calculates the optimal joint trajectories for the front right leg, which are then projected suitably in all the legs, through the Allocation Module. Finally, a tracking controller is responsible for tracking the desired joint trajectories $\mathbf{q}_D$ and torques $\mathbf{u}$. Here, a PID position controller is utilized, although more advanced tracking controllers[31] that accept state and input trajectories can also be employed.

In general, the attitude controller runs at 1kHz, in order for the phase transition checks of the FSM to take place. The Body Planner that was used was Variant 3 with terminal constraint and cost, as described in table 12, while the Leg Planner is based on Variant 2, presented in table 17, but utilizes different weights determined by the resetting algorithm. The Body and Leg planner update rate is 10Hz. The allocation module updates the reference every 20ms, as there are 5 stages in the MPC of the Leg Planner. However, if a phase change is detected, the leg planner recalculates the trajectory based on updated parameters. Its interrupting capability is indicated by the red colour in figure 25.

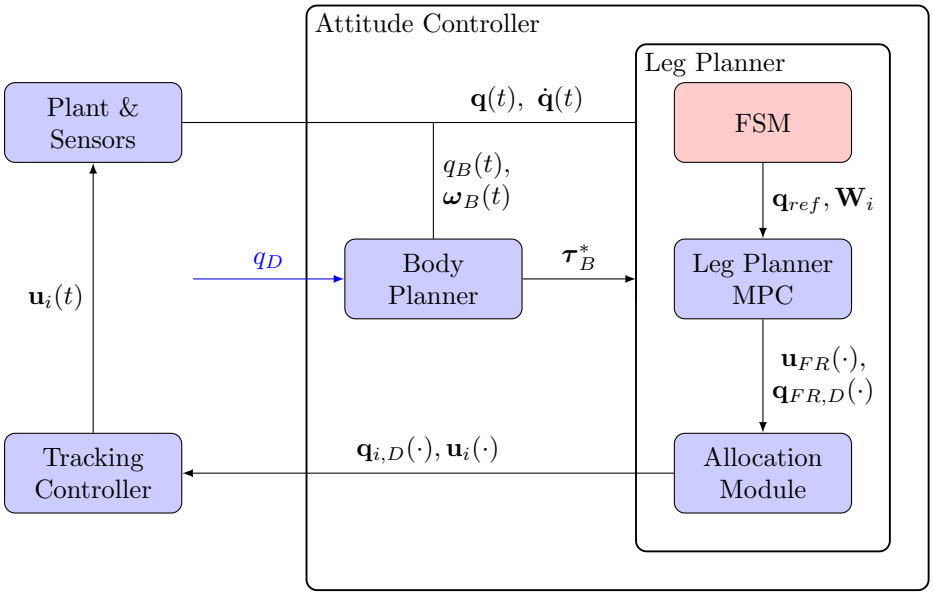


Figure 25: Controller Architecture.

[31] In the sense of a feed-forward PID: $u = u_{PID} + u_{MPC}$ like the one used in [34].

5. Results

This section is dedicated to simulation and experimental results. Initially, single axis reorientation simulations are presented, in order to quantify the performance of the controller. Next, the effect of adding extra mass in the paws, which is expected to increase the control authority of the system, is investigated along with the robustness of the proposed control to the mass of the torso. Finally, experimental results are showcased and compared with the simulation.

5.1 Simulation

To showcase the performance of the controller, it was tasked to stabilize 90° single axis turns. The setup of the simulations is the following:

- The robot starts with $q_0 = q_I$.
- A desired orientation q_D is given as reference to the attitude controller.
- Gravity is disabled to simulate the free falling conditions during jumping phase.
- The robot is connected with a floating joint into the world, meaning it is free to rotate in all degrees of freedom.
- For these simulations, the update rate of the body planner and leg planner is 0.1s. Thus, the position reference update rate, for the low level PID, is 50Hz.
- The simulation settings are listed in table 10.
- The convergence threshold was set to 5° as the controller generally performs wide movements and thus it is difficult to stabilize very precisely with the current formulation. Also, even if, through suitable projections, torques in unwanted directions are minimised, they are still present, which makes precise stabilisation even more difficult because the total angle is taken into account to enter the stabilisation mode.

5.1.1 Roll Step Response

The response of the system for a roll reference of ±90° is presented in figures 26 and 27.

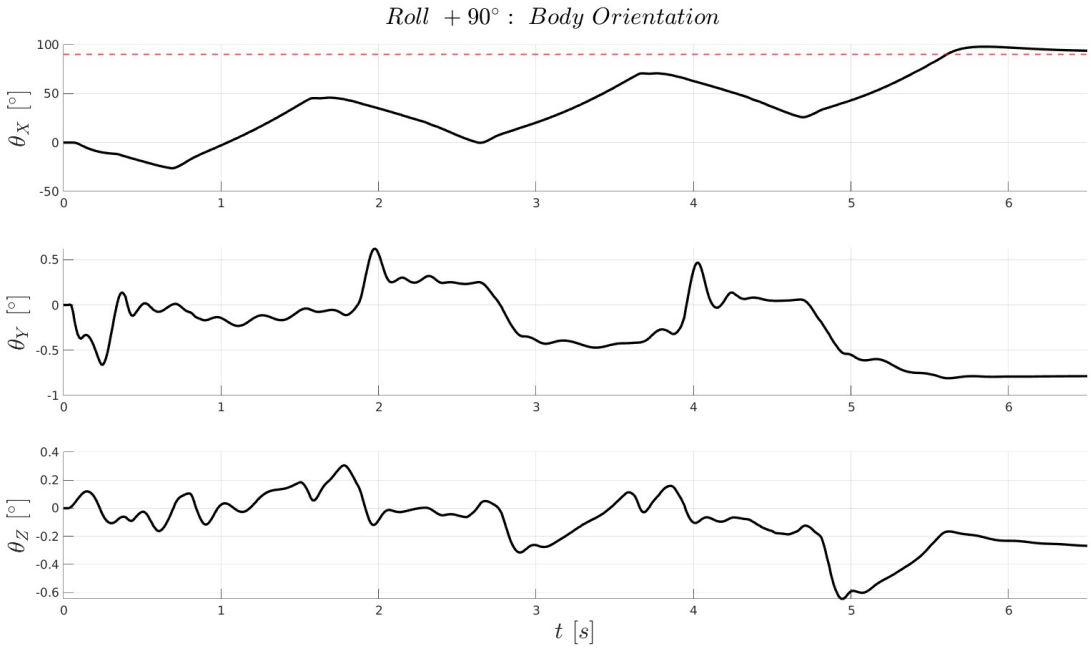


Figure 26: Simulation of response to +90° reference for roll.

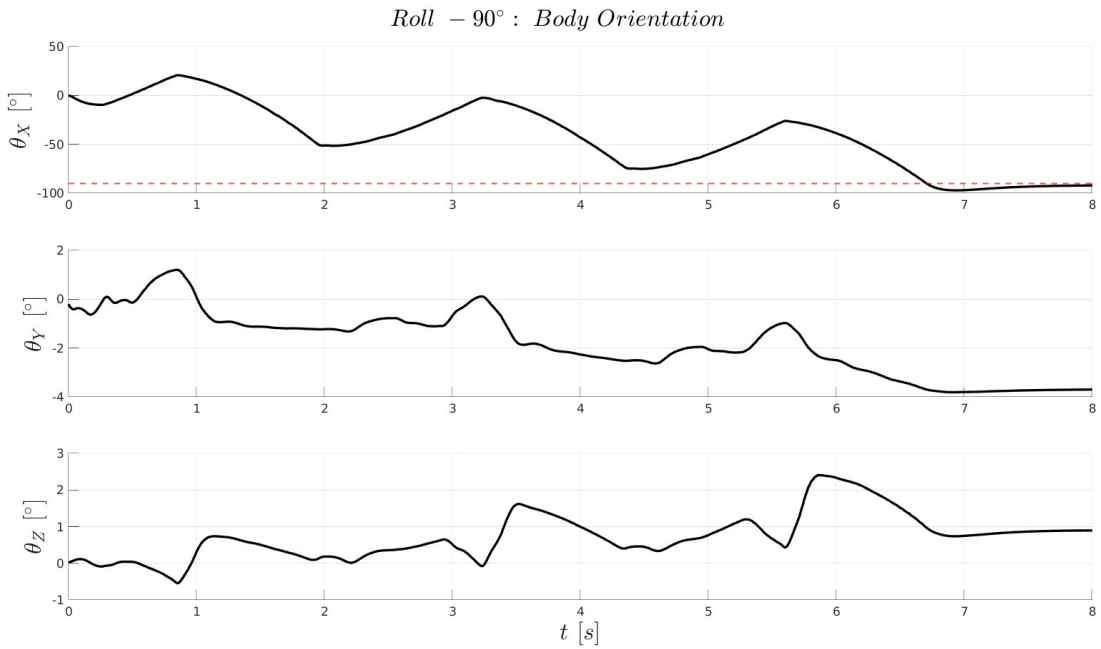


Figure 27: Simulation of response to $-90°$ reference for roll.

The results of the simulation are summarized in table 18. The orientation converges within the specified threshold in both situations. The solution time of the leg planner was well within the allowed time of 100ms between consecutive calls of the controller. Also, it can be observed that the pitch and yaw remain relatively unaffected, only because of the suitable projection of the optimized joint trajectories from the front right leg to the rest.

Table 18: Roll step reference simulation results.

	$+90°$	$-90°$		$+90°$	$-90°$
Steady State Error	3.9°	2.2°	**MPC solution time** ($\mu/\sigma/\max\ [ms]$)	13/4.2/25	12/6.1/55
Settling Time	6.3s	7.2s	$\bar{\omega}_x\ [°/s]$	14.9	12.8

5.1.2 Pitch Step Response

The response of the system for a pitch reference of $\pm 90°$ is presented in figures 28 and 29. The results of the simulation are summarized in table 19. The orientation converges within the specified threshold in both situations. The solution time of the leg planner was well within the allowed time of 100ms between consecutive calls of the controller. Also, it can be again observed that the roll and yaw remain relatively unaffected.

Table 19: Pitch step reference simulation results.

	$+90°$	$-90°$		$+90°$	$-90°$
Steady State Error	1°	0.6°	**MPC solution time** ($\mu/\sigma/\max\ [ms]$)	8.5/9/55	9/9/42
Settling Time	2.4s	4.4s	$\bar{\omega}\ [°/s]$	37.5	20.5

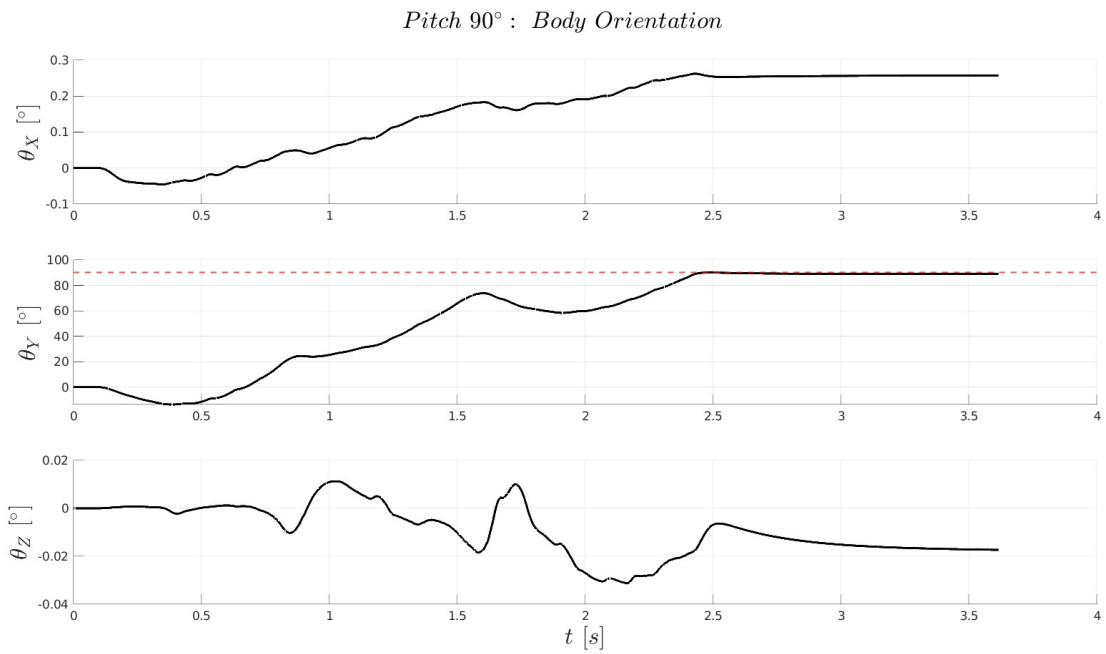


Figure 28: Simulation of response to +90° reference for pitch.

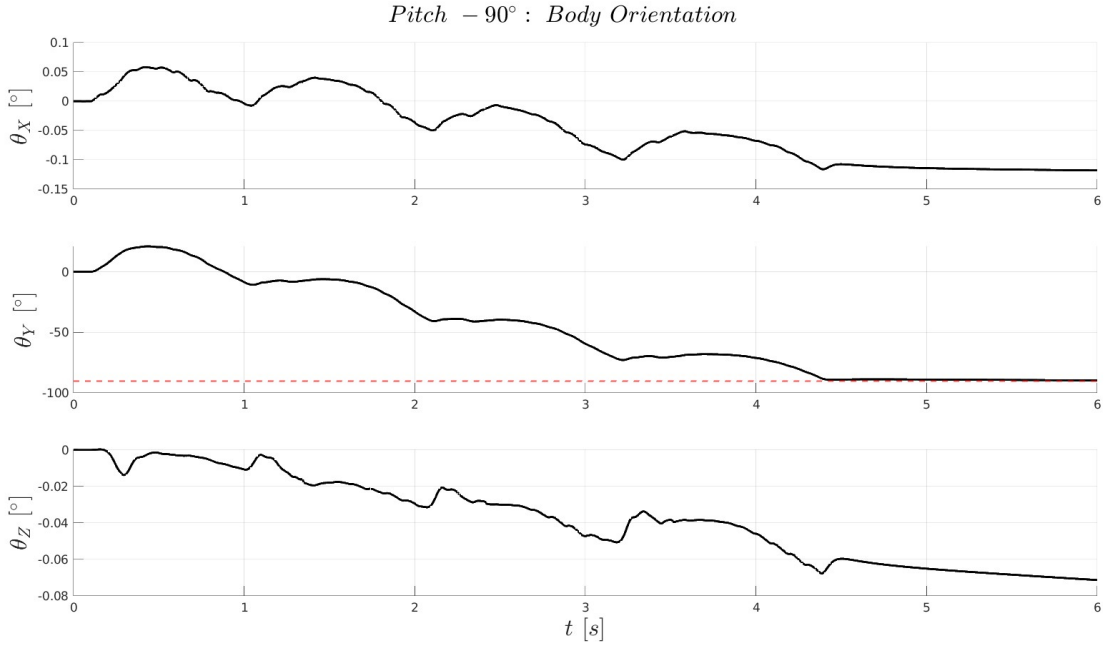


Figure 29: Simulation of response to −90° reference for pitch.

5.1.3 Yaw Step Response

The response of the system for a yaw reference of $\pm 90°$ is presented in figures 30 and 31.

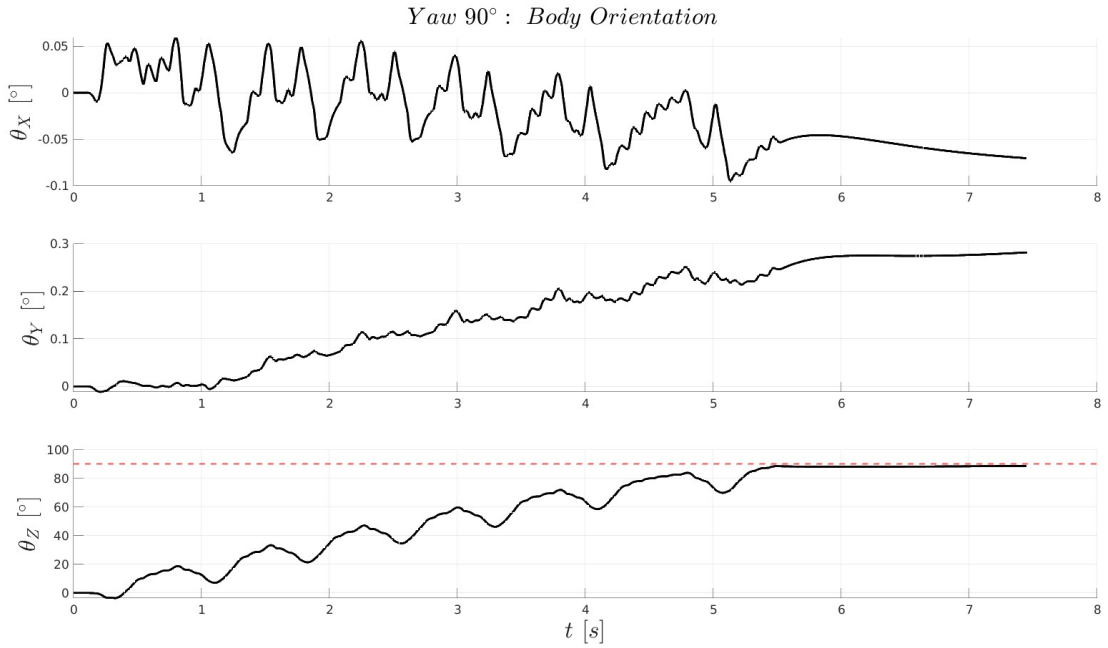


Figure 30: Simulation of response to $+90°$ reference for yaw.

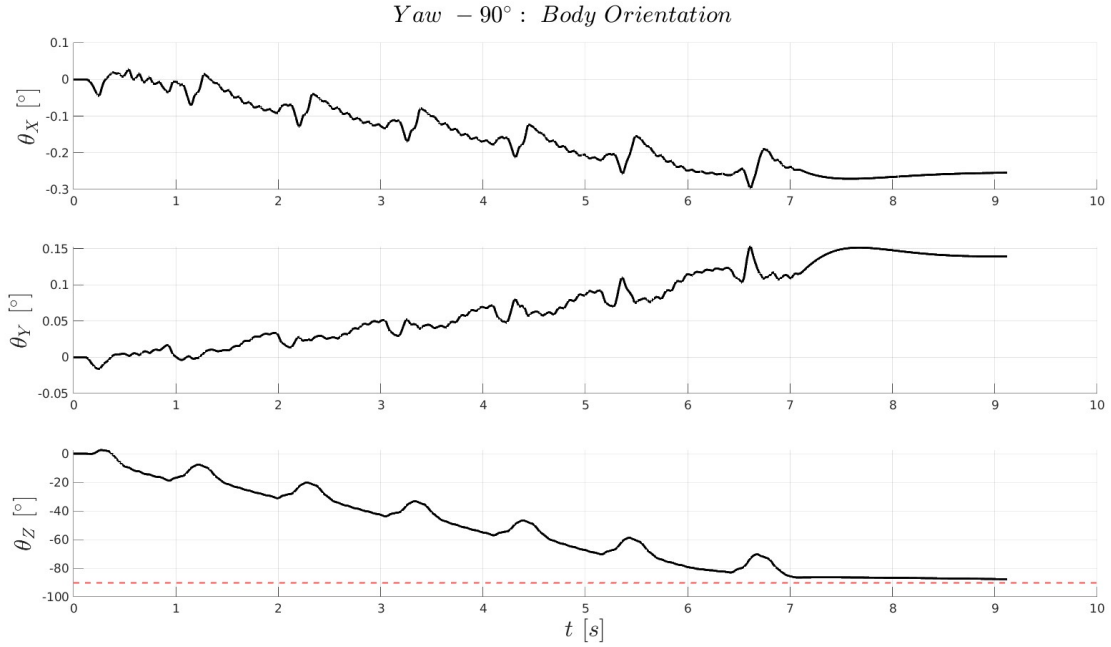


Figure 31: Simulation of response to $-90°$ reference for yaw.

The results of the simulation are summarized in table 20. The orientation converges within the specified threshold in both situations. The solution time of the leg planner was well within the allowed time of 100ms between consecutive calls of the controller. Also, it can be again observed that the roll and pitch remain relatively unaffected.

Table 20: Yaw step reference simulation results.

	$+90°$	$-90°$		$+90°$	$-90°$
Steady State Error	1.5°	2.1°	**MPC solution time** ($\mu/\sigma/\max$ $[ms]$)	6.6/4/19.4	5/4.4/16.5
Settling Time	5.5s	10.4s	$\bar{\omega}$ $[°/s]$	16.1	8.5

5.2 Ablation Study

This section is dedicated to an ablation study, where the effect of extra mass in the paws is investigated. Also, the robustness of the control method is investigated under parameter uncertainty, mainly the mass of the torso (eg. adding a bigger and heavier battery, different actuators or extra on board sensors and tools). The effect of these modifications on each direction is showcased in figures 32, 33 and 34.

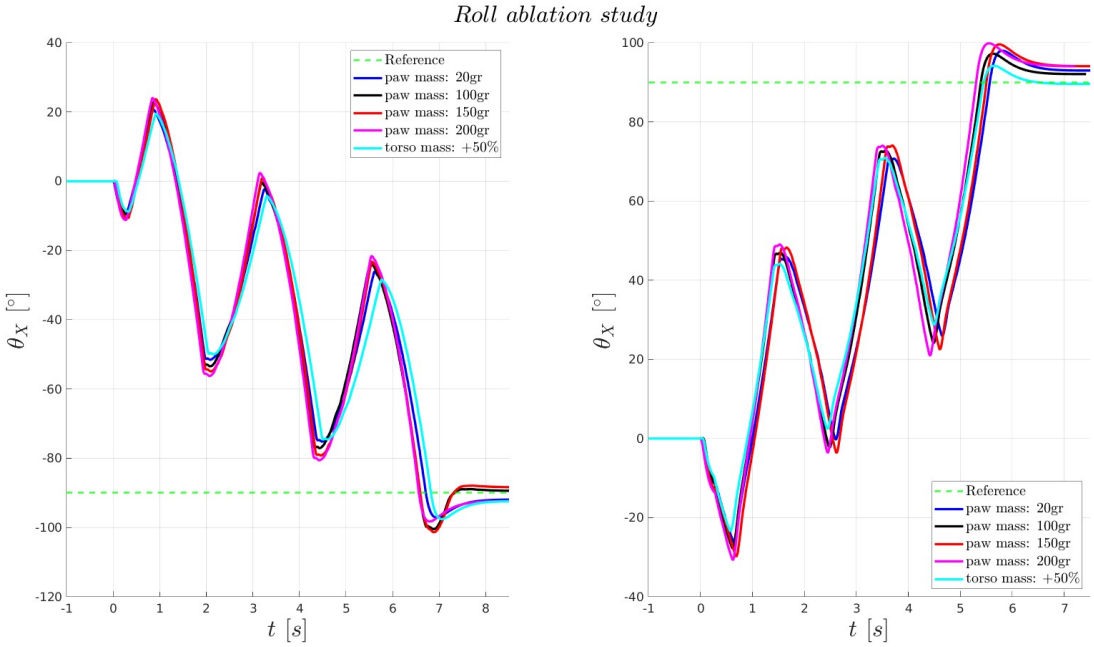


Figure 32: Roll Ablation Study.

First, it can be observed that the controller manages to stabilize the desired setpoint in all cases, demonstrating its robustness. Roll seems quite unaffected by changes in the mass. The roll moment of inertia is very small (one order of magnitude smaller than other directions), so it is expected that doubling the mass of the main body will not significantly change the turning ability of the robot. It is mainly the inertia of the legs that affects the controller's performance. Adding mass to the paws simultaneously increases the rolling inertia and control authority of the robot, so the result depicted in figure 32 is reasonable and is in accordance with previous results [12]. Pitch performance is greatly improved by adding extra mass to the paws. The main indication of improved performance is the displacement achieved per cycle, which steadily increases until 150gr. Increasing the torso mass has a negative effect on the turning speed of the robot, as expected. Finally, adding extra mass to the paws increases the performance of yaw, especially in the $-90°$ case.

From the simulations above, it can be observed that paws of mass 150 gr offer the biggest performance benefit, reducing convergence by 0.7 seconds in pitch and 2 seconds in yaw in the $-90°$ case. Further increase, can destabilize the controller as the movements become too violent to converge in the desired orientation. This can be observed in figure 34, for 200gr in the $-90°$ case. Also, the controller is quite robust to body mass changes.

Pitch ablation study

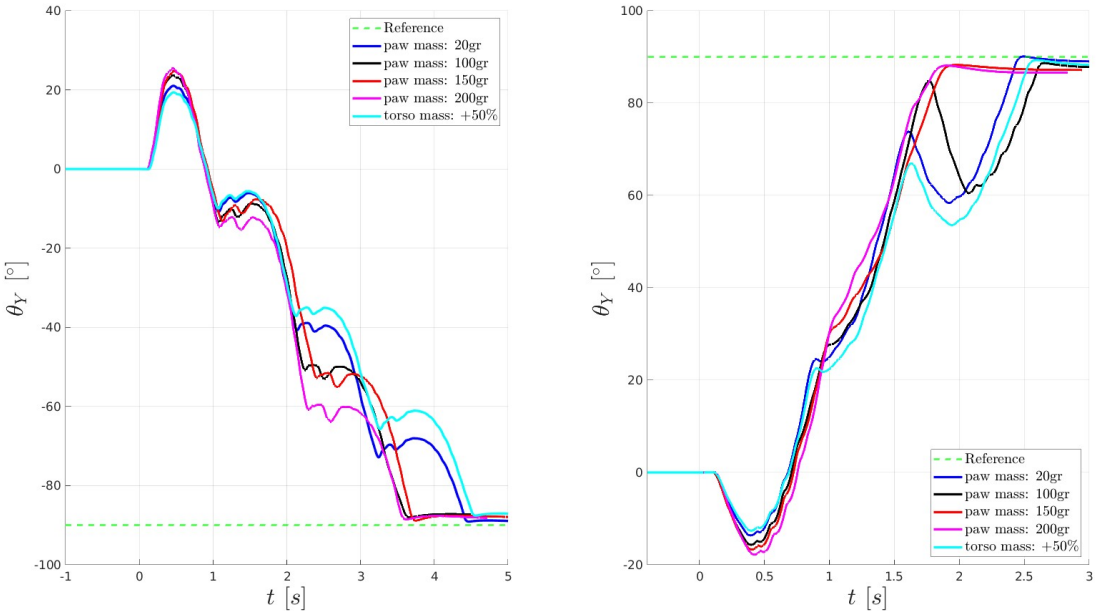


Figure 33: Pitch Ablation Study.

Yaw Ablation Study

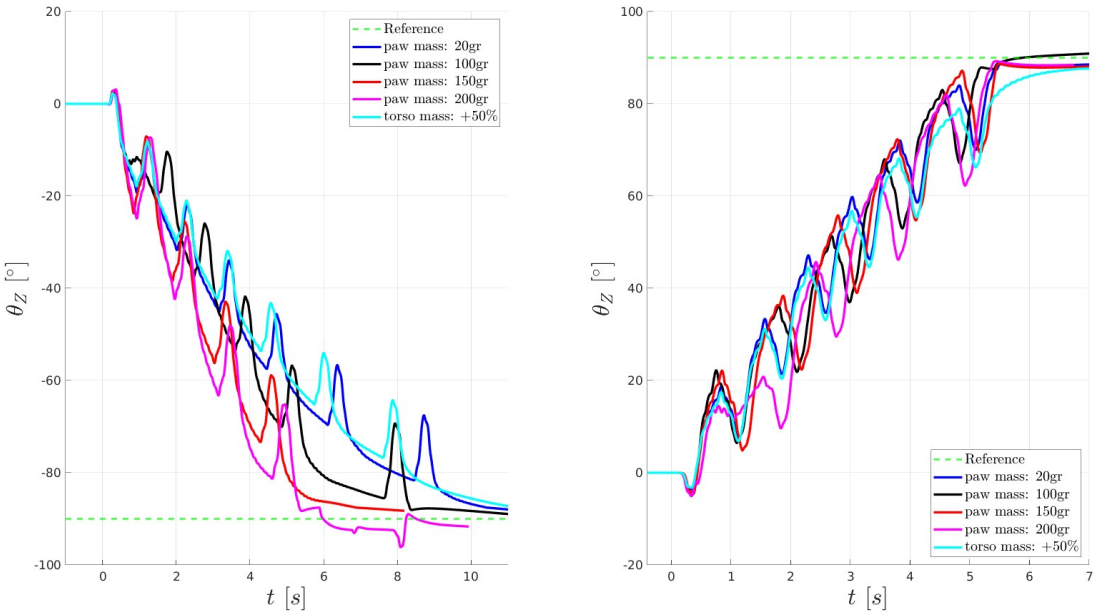


Figure 34: Yaw Ablation Study.

5.3 3D Reorientation

In this subsection, indicative results from arbitrary reorientations will be presented. The robot starts at an arbitrary orientation, which is specified using the Roll-Pitch-Yaw (rpy) convention, and returns to $q_D = q_I$. The results are showcased in figures 35, 36 and 37. To increase the control authority of the system, the mass of the paws is increased by 100gr.

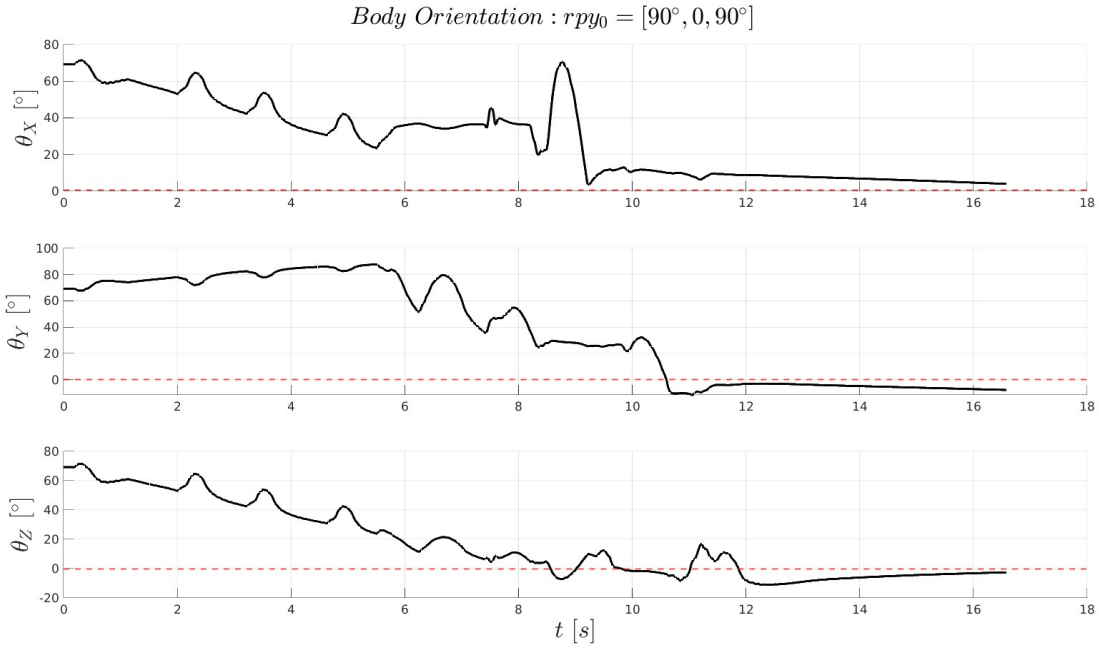


Figure 35: Reorientation simulation: $\mathbf{rpy}_0 = [\pi/2, 0, \pi/2]$

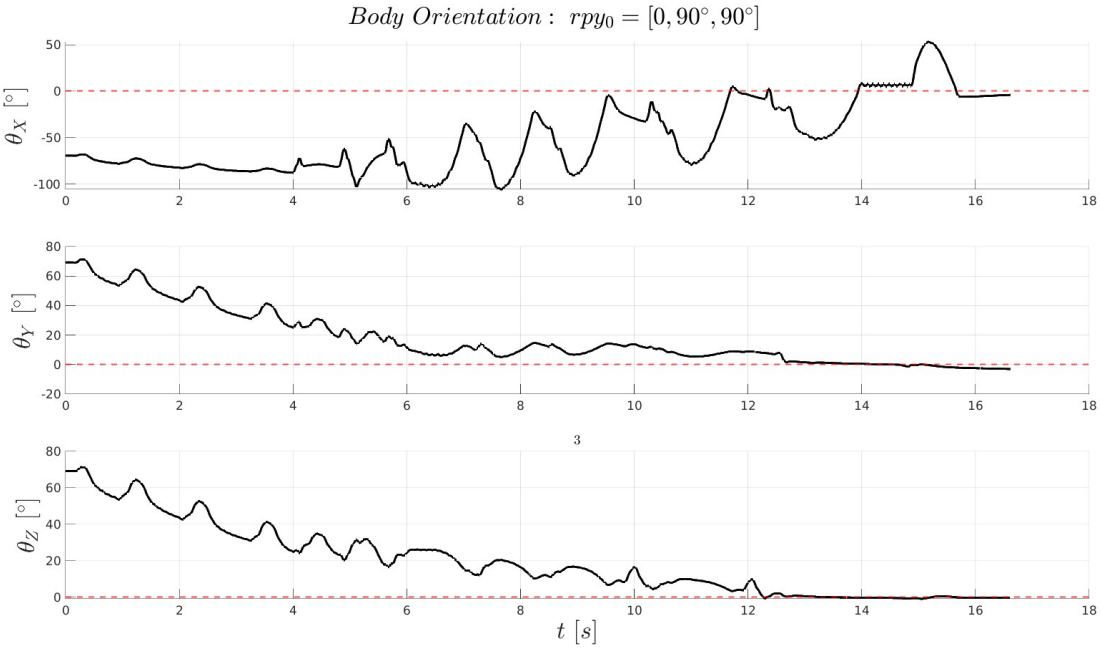


Figure 36: Reorientation simulation: $\mathbf{rpy}_0 = [0, \pi/2, \pi/2]$

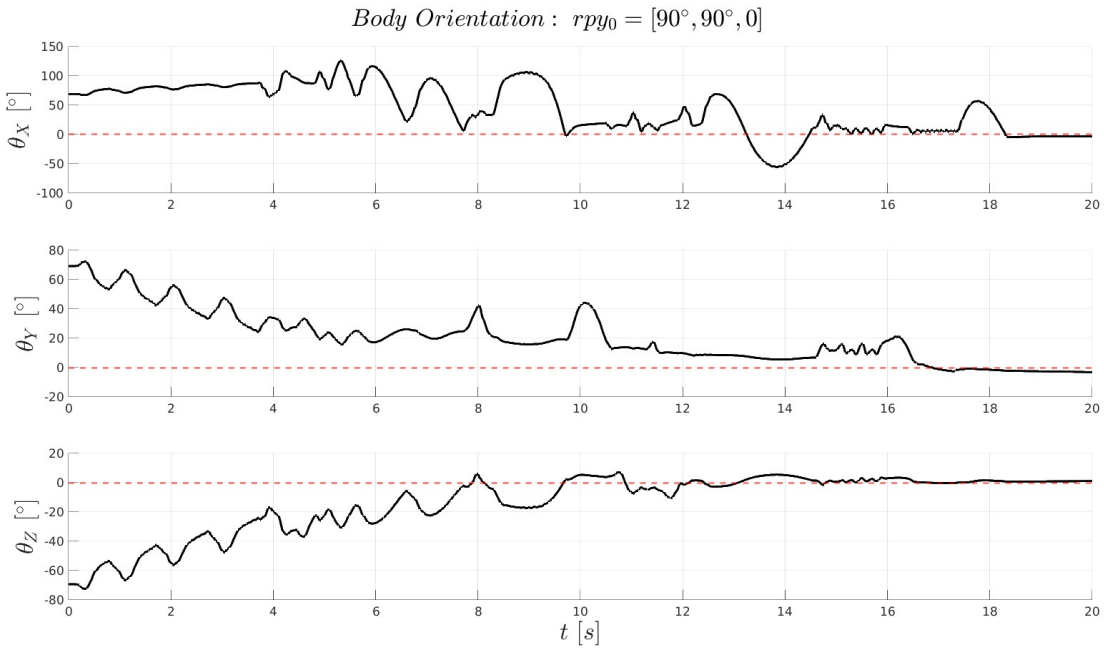


Figure 37: Reorientation simulation: $\mathbf{rpy}_0 = [\pi/2, \pi/2, 0]$

In all cases, it can be observed that the controller manages to stabilize the orientation within the specified accuracy of 7.5°. However, the settling time is very large; on average, 16.7s. The sub-optimal performance is mainly due to the manually selected hyper-parameters. Additionally, these parameters were tuned based on single-axis re-orientations, leading to over-fitting and thus poor generalization in 3D manoeuvres. Moreover, the separation of reorientation modes into discrete roll, pitch, and yaw manoeuvres, as per the allocation strategy discussed in 4.2.3, further degrades performance. This is supported by the cross-variant results in table 12. Also, because of the large range of motion of the leg manoeuvres, the controller fails to stabilize near the desired orientation, resulting in oscillatory behavior. This is the reason for the large stabilization threshold of 7.5°. Finally, when the controller is switching modes, it often introduces unwanted disturbances in other directions.

5.4 Experiment

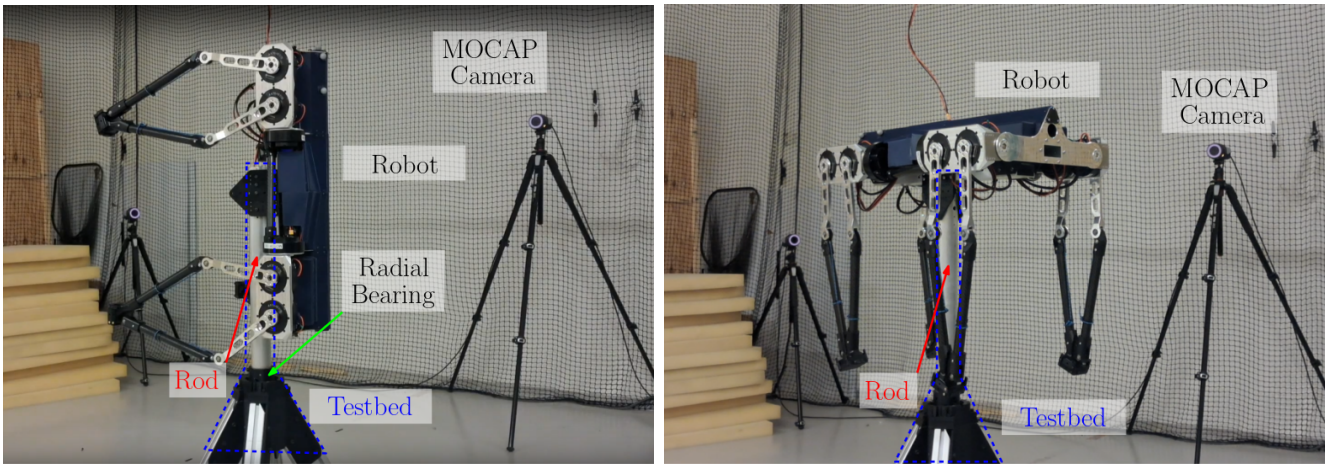
5.4.1 Experimental Setup

The experimental setup is depicted in figure 38. The robot is mounted on a testbed, which is essentially a rotating rod that rotates together with the robot. For each rotational degree of freedom, the robot was placed differently on the testbed, so that gravity was parallel to the axis of rotation. Orientation feedback was provided by the laboratory motion capture system. The controller, estimation and motor controller nodes were deployed on the on board computer, while the reference publisher and visualization were running on a different computer. Communication was established via Wifi. The structure is depicted in figure 39.

This setup ensures that gravity does not affect the rotation of the torso directly but rather acts as a disturbance to the system, affecting the link motions. Additionally, as the axis of the rod does not pass from the centre of gravity[32], some bending moments are induced in the bearings that affect their friction characteristics. Indeed, bearings are not designed for handling bending moments. To achieve better performance and overcome frictional phenomena, the experiments were conducted with extra 100 gr in the paws to provide the system with greater control authority. Also, the leg planner update rate was increased to 20Hz and only the latest angle reference was given in the low level controller. This reduced oscillations in the system and provided a more constant reference for the motors to track. Finally, no pitch experiments took place, as the presence of the rod severely limited the workspace of the leg.

For roll and yaw, 2 kinds of experiments took place. A response to a step input reference of $\pm 90°$ and a response to a changing reference (piecewise constant). The step responses were used to extract some quantitative results for the controller and verify the simulation environment.

[32]In yaw, the axis passes through the centre of mass when the legs are in their default configuration, but not necessarily when moving.



(a) Experimental setup for roll.

(b) Experimental setup for yaw.

Figure 38: Experimental setup.

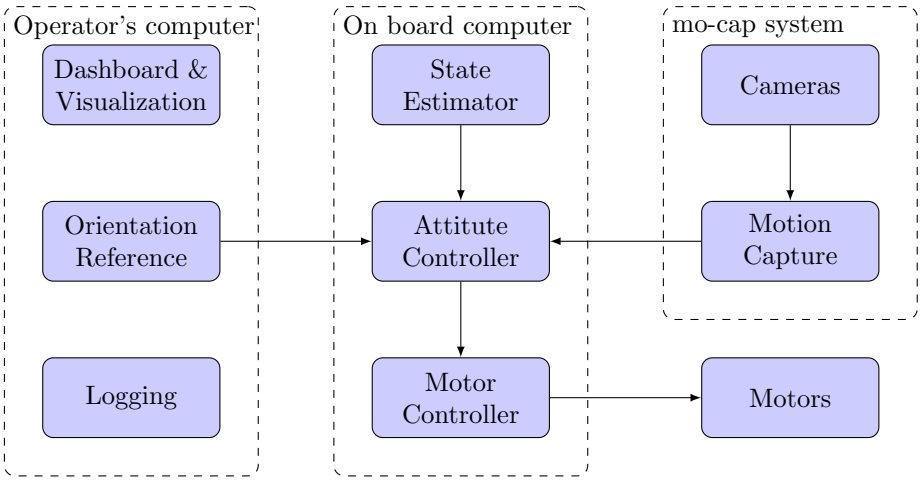


Figure 39: System architecture during the experiment.

The second experiment showcases the ability of the controller to track more complex manoeuvres. Also, in both roll and yaw, the last commanded displacement is greater than 180°, which showcases the ability of the controller to select the shortest distance. However, in the diagrams, the last commanded angle was shifted (+360°) to showcase the convergence of the controller.

5.4.2 Experimental Results

The step responses for roll are depicted in figure 40, while the tracking of the changing reference is shown in figure 41.

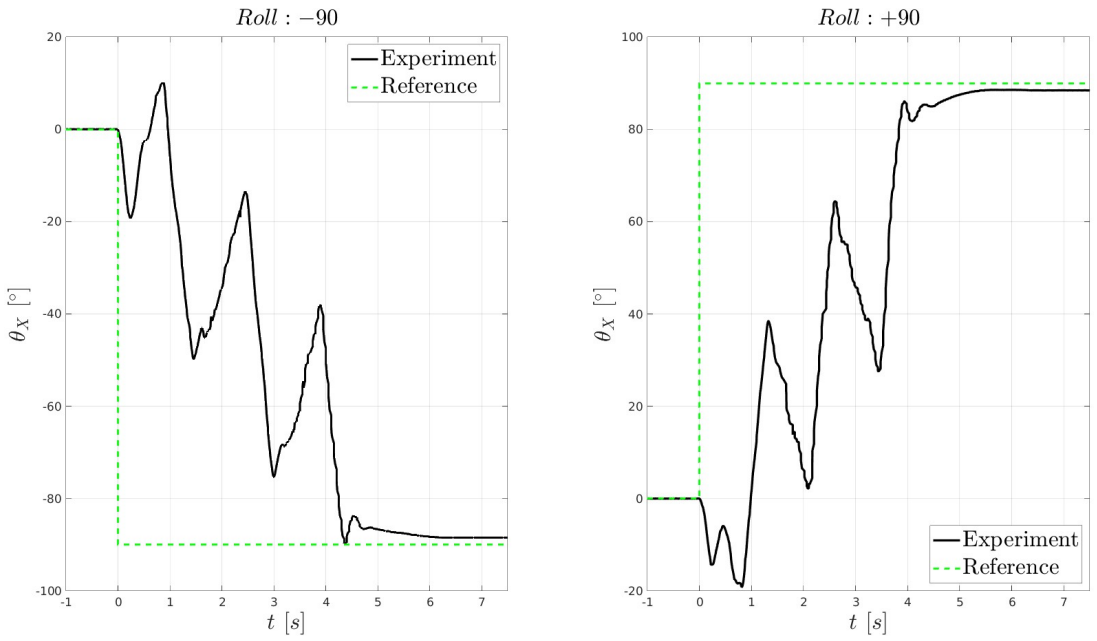


Figure 40: Response to a step input reference for roll.

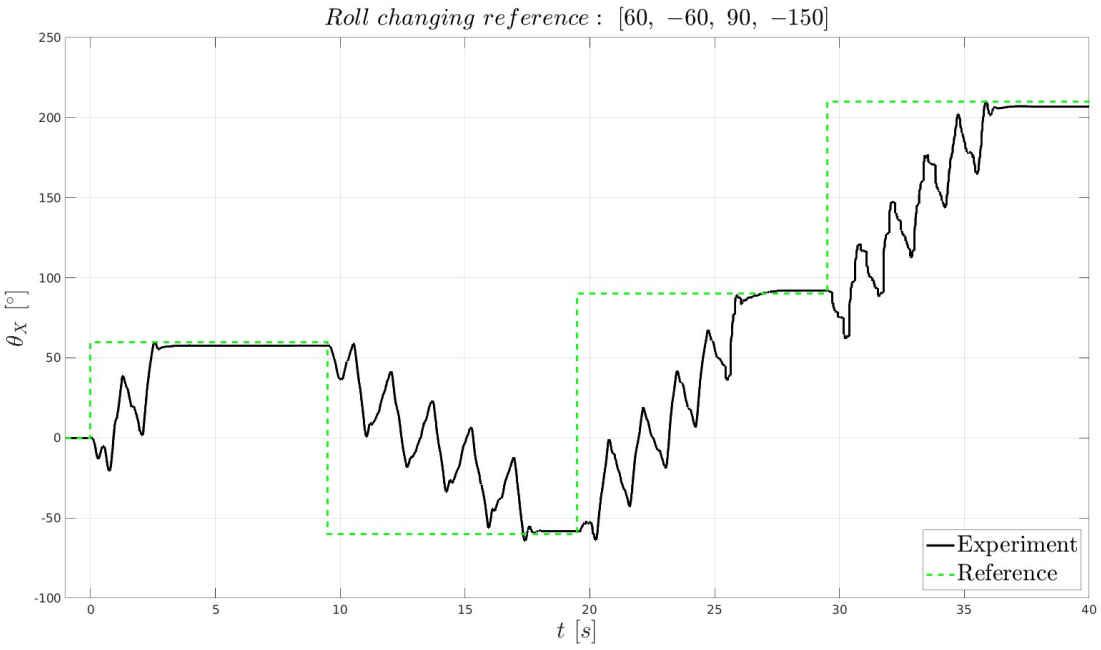


Figure 41: Response to a changing input reference for roll.

From the figures, it can be observed that the steady state error is around 1.5°, which is within the stabilization threshold specified for roll 5°. Also, the robot requires $5s$ to stabilize a 90° turn. In the second experiment, the controller manages to track a changing reference and finds the shortest path from 90° to −150° (or equivalently 210°), by continuing to turn in the positive direction, to minimize the distance metric that was introduced in section 2.2.2.

Similarly, the same types of experiments took place for yaw. They are showcased in figures 42 and 43.

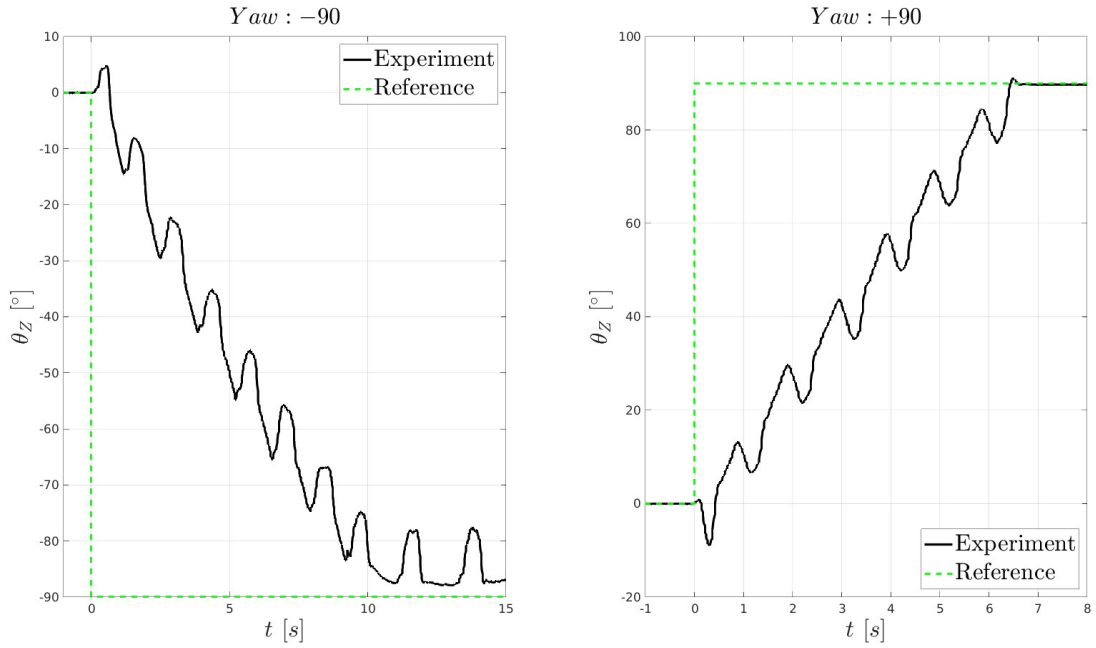


Figure 42: Response to a step input reference for yaw.

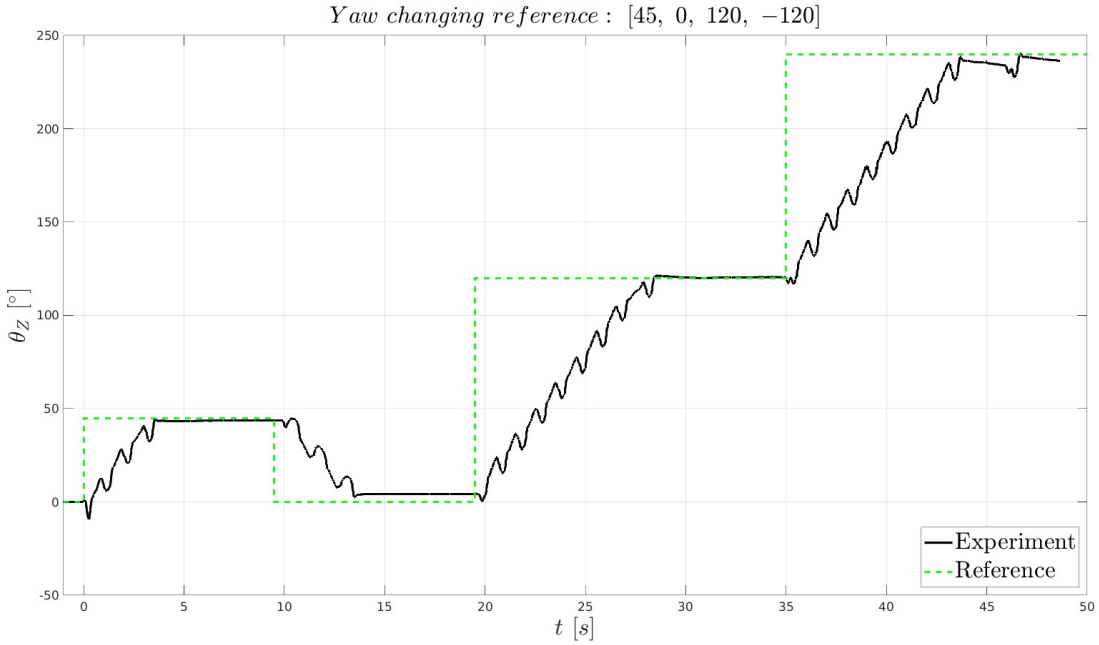


Figure 43: Response to a changing input reference for yaw.

From the figures, it can be observed that the steady state error is around 2.5° for the negative step reference and almost zero for the positive reference. Also, the convergence time is different between the two turning directions,

requiring 6.5s to stabilize $+90°$ and almost 12s for the negative direction. This can be attributed to the manual tuning of the controller, which has introduced some anisotropy in the controller, and the misalignment of the rod. Even though tuning of the testbest had taken place before the experiments, oscillations may have disturbed its alignment and introduced a local minima in the direction of the robot, which favoured positive motion. In the second experiment, the controller manages to track a changing reference and finds the shortest path from $120°$ to $-120°$ (or equivalently $240°$), by continuing to turn in the positive direction.

5.4.3 Comparison with Simulation

To validate the simulation model, the reference positions generated by the attitude controller in the experiment were replayed in a simulation that included the testbed and gravity was enabled in order to mimic the experimental conditions. The results for roll are shown in figure 44. It can be observed that the simulation has similar behaviour to the actual system. The playback took place for different damping coefficients for the testbed to investigate its effect. However, the observed deviations are independent of the damping coefficient. Therefore, the differences can be attributed to factors such as the imbalance of the rod, the presence of non-viscous friction (which is not modelled in the simulation), and differences in motor saturation limits.

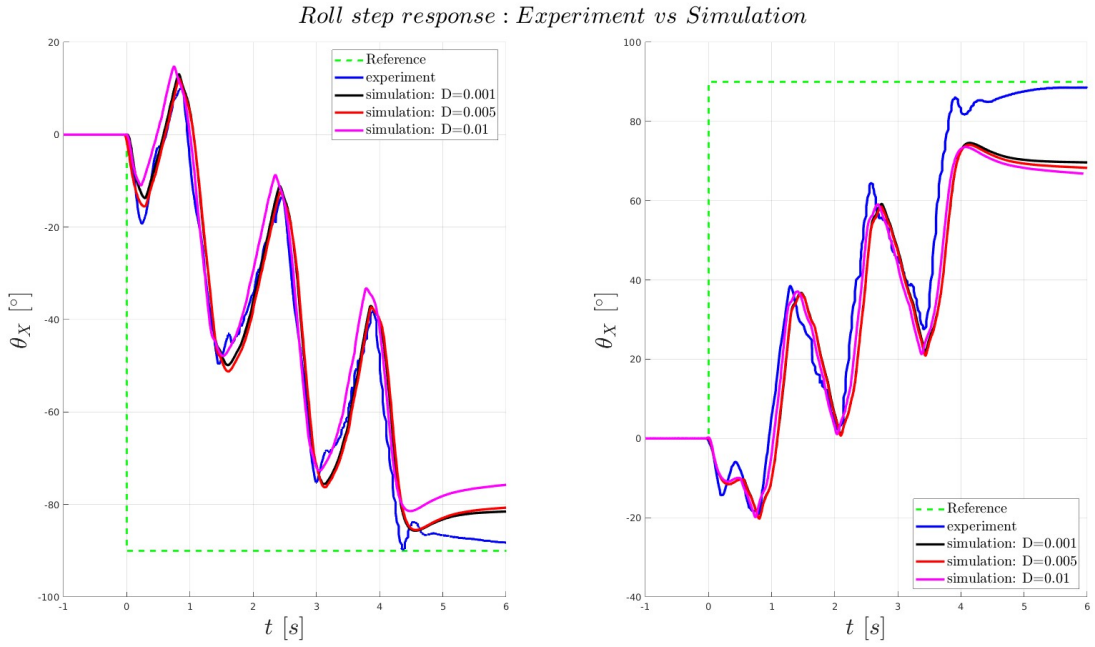


Figure 44: Experimental and simulation roll displacement using the experimental motor references.

The results for yaw are showcased in figure 45. Yaw presents a significant difference between the simulation and the experiment, especially the positive step response. This can be attributed to the following:

- The motor models in the simulation are not accurate enough. The actual motor controllers have different saturation for each gain term. This explains the non-smoothness of the displacement curves. Also, the motors in simulation are ideal torque sources.
- The rod was imbalanced. This can be observed as the difference between positive and negative turning directions is considerably greater. Imbalance in the rod can introduce an equilibrium point in the system, which biases the angular velocity towards a certain turning direction. Before testing, the testbed was tuned to avoid equilibrium points. But vibrations from previous tests, could re-introduce them.
- Non viscous friction moments are more prevalent in yaw, which has lower control authority than rolling motions. Figure 46 depicts the Stribeck curve, a comprehensive model of the tribological phenomena in lubricated contacts. It demonstrates that friction increases at low speeds, potentially hindering resetting motions where the robot's angular velocity is lower compared to forward movements.

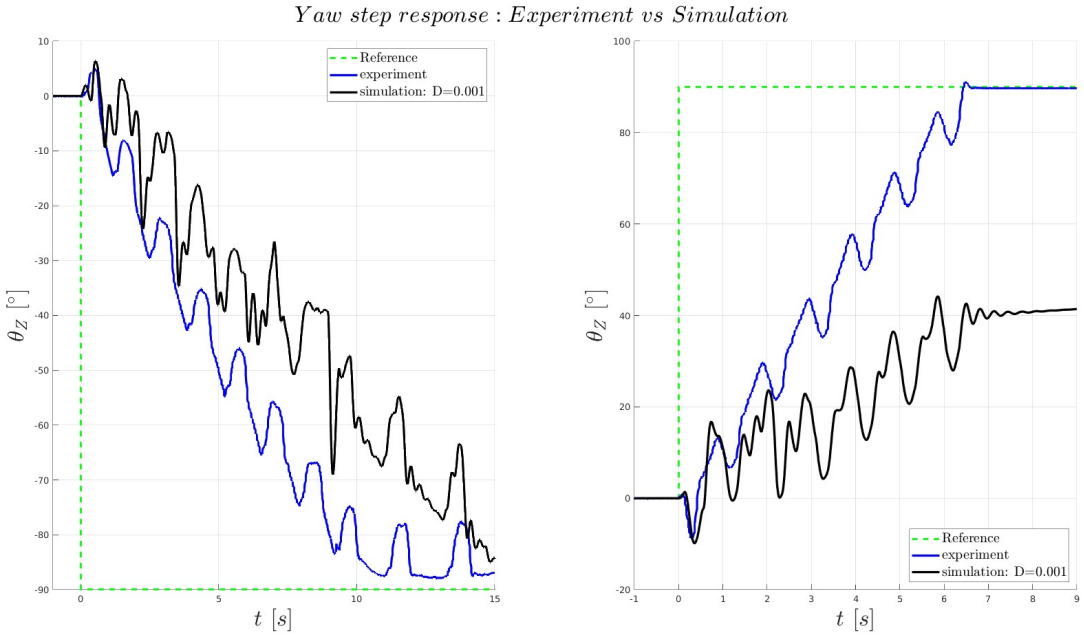


Figure 45: Experimental and simulation yaw displacement using the experimental motor references.

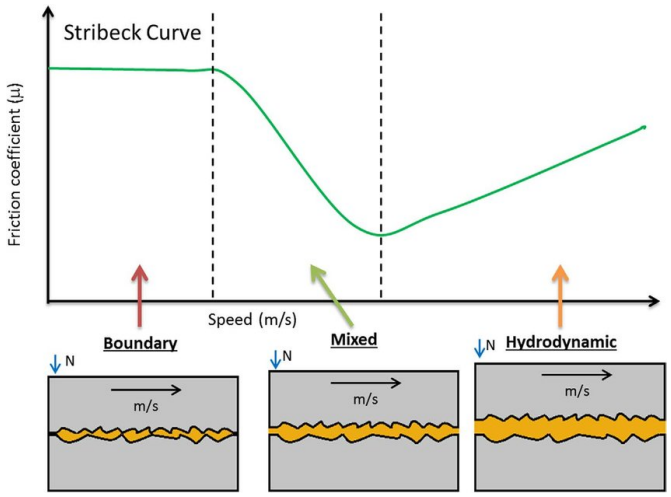


Figure 46: Stribeck curve.

Finally, in figures 47 and 48, experimental and simulation results are compared. In both cases, the desired reference is tracked successfully. However, the performance varies between the simulation and the experimental data, especially for yaw. This difference can be attributed to both different frictional phenomena and the performance sensitivity of the controller to various dynamical parameters. The controller manages to find periodic trajectories online, but these trajectories change depending on when the phase transition occurs.

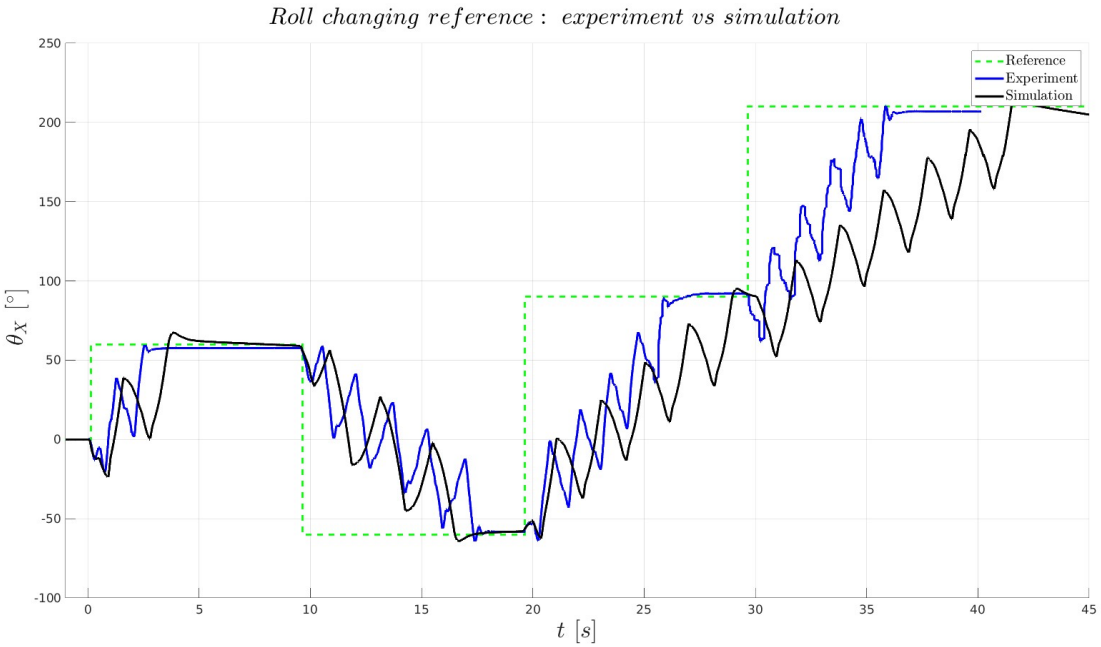


Figure 47: Comparison between experimental data and simulation for changing roll reference.

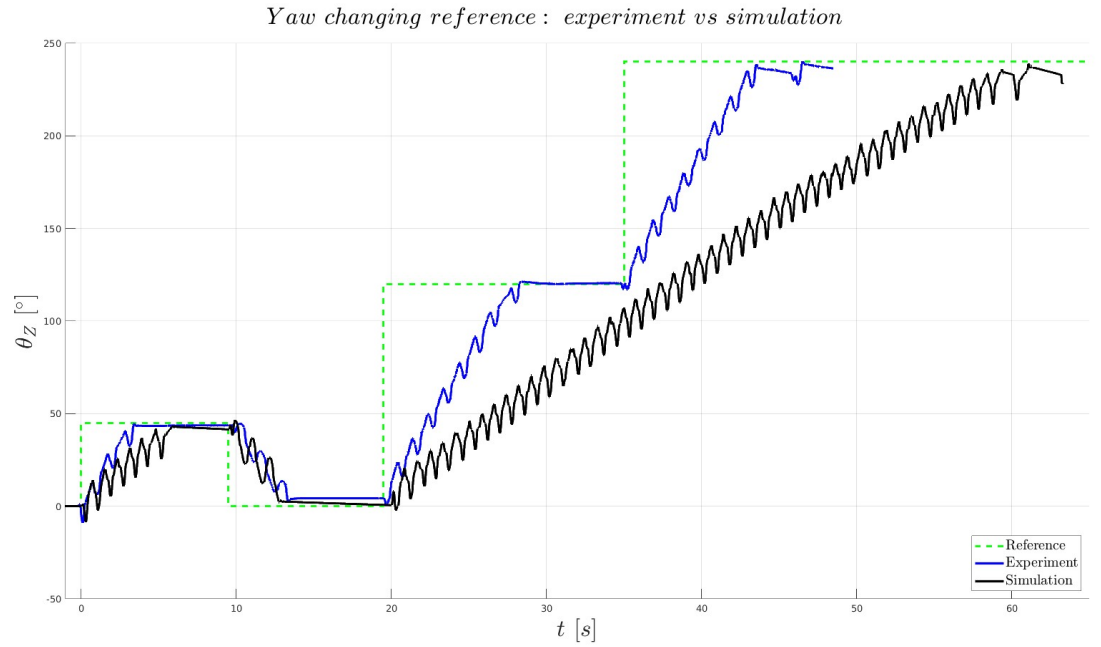


Figure 48: Comparison between experimental data and simulation for changing yaw reference.

6. Conclusions and Future Work

6.1 Conclusions

In this thesis, the modelling and attitude control of a jumping quadruped were investigated. A detailed kinematic and dynamic analysis led to the development of analytical models for the robot. Also, analysing the workspace led to the extraction of various operational constraints. A hierarchical model-based attitude controller was then developed. The controller first optimizes the torso trajectory to track arbitrary orientation references using a simplified rigid body model and virtual torques that are acted upon it. The quaternion parameterization and suitable metric allow the top-level controller to select the shortest trajectory to the desired orientation in less than 2 ms. The bottom module finds corresponding joint trajectories for only one leg to produce the virtual torques that result from the body planner while respecting various workspace, state and input constraints. A "resetting" strategy allows online computation of the joint trajectories and an allocation method projects the optimal trajectories to the other legs. The controller was evaluated both in simulation and in experiments.

One important outcome of the present work is the simulation framework, which is easily modifiable, documented and can be easily used to test reorientation scenarios both in free floating and experimental conditions. Also, it has the same API as the actual robot, rendering it a useful tool that can be used for further development of *Olympus*.

The proposed controller manages to stabilize the robot in single axis reorientation scenarios. However, the performance is suboptimal and worse than the state of the art [12, 10]. The proposed controller has a lot of tuning parameters; each orientation has 56 parameters. This makes achieving a stabilizing performance, let alone an optimal one, a challenge. In addition, while the complete dynamic model of the leg allows for accurate handling of workspace constraints, its complexity severely limits the horizon of the optimization, which in turn limits the optimality of the leg planner. Also, the current formulation of the leg planner is conflicted, as the optimizer tries to jointly track a setpoint reference and a desired torque. Due to the high nonlinearity of the problem, it is generally difficult to achieve a suitable compromise. Finally, the proposed controller struggles to stabilize arbitrary orientations. This is mainly due to the discretization of the turning manoeuvres in roll, pitch and yaw.

The experimental results showcase that the controller is applicable to a real system, as the robot managed to reorient itself and track changing references. At high velocities, the experimental results closely mimic the simulation. However, in slower velocities, such as the experiments in yaw, nonlinear frictional phenomena greatly affect the result.

6.2 Future Work

The work of this thesis showcased that model based controllers can stabilize the attitude of a jumping quadruped avoiding self collisions. However, the performance of the proposed controller must be improved in order to stabilize the quadruped's attitude during jumping. One direction could be the use of a hyperparameter optimization tool, such as *Optuna*[33] to find optimal MPC parameters, and increase the performance of the controller. Also, it is expected to make the tuning process easier. Additionally, the present formulation can be used to create a series of switching controllers with different setpoints and weights to increase performance near the reference orientation. Another direction, is to use only the adjacent leg projection described in section 4.2.3 and use two MPCs, one for each side, to allow the controller to track multi-directional torques. This is expected to increase the performance for arbitrary orientations, as hinted in table 12. Finally, it is worth investigating simpler models for the legs to co-optimize the the body trajectory and leg motions, and avoid using an allocation algorithm that in turn discretizes the reorientation motions.

Regarding the experimental aspect, the current simulation framework should be modified to model the frictional phenomena in the rod to allow for a more detailed comparison between simulation and experimental data. Also, more detailed motor models should be implemented [35, 36].

[33] `https://optuna.org/`

7. References

[1] A. D. et al., "Lunar and martian lava tube exploration as part of an overall scientific survey," 2013.

[2] J. A. Olsen and K. Alexis, "Martian lava tube exploration using jumping legged robots: A concept study," 2023.

[3] M. Kulkarni, M. Dharmadhikari, M. Tranzatto, S. Zimmermann, V. Reijgwart, P. De Petris, H. Nguyen, N. Khedekar, C. Papachristos, L. Ott, R. Siegwart, M. Hutter, and K. Alexis, "Autonomous teamed exploration of subterranean environments using legged and aerial robots," in *2022 International Conference on Robotics and Automation (ICRA)*, pp. 3306–3313, 2022.

[4] A. Bouman, M. F. Ginting, N. Alatur, M. Palieri, D. D. Fan, T. Touma, T. Pailevanian, S.-K. Kim, K. Otsu, J. W. Burdick, and A. akbar Agha-mohammadi, "Autonomous spot: Long-range autonomous exploration of extreme environments with legged locomotion," *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2518–2525, 2020.

[5] Y. Ding, A. Pandala, C. Li, Y.-H. Shin, and H.-W. Park, "Representation-free model predictive control for dynamic motions in quadrupeds," *IEEE Trans. Robot.*, vol. 37, pp. 1154–1171, Aug. 2021.

[6] H. Li and P. M. Wensing, "Cafe-mpc: A cascaded-fidelity model predictive control framework with tuning-free whole-body control," *ArXiv*, vol. abs/2403.03995, 2024.

[7] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Science Robotics*, vol. 5, no. 47, p. eabc5986, 2020.

[8] H. Kolvenbach, E. Hampp, P. Barton, R. Zenkl, and M. Hutter, "Towards jumping locomotion for quadruped robots on the moon," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5459–5466, 2019.

[9] N. Rudin, H. Kolvenbach, V. Tsounis, and M. Hutter, "Cat-like jumping and landing of legged robots in low gravity using deep reinforcement learning," *IEEE Transactions on Robotics*, vol. 38, no. 1, pp. 317–328, 2022.

[10] A. Spiridonov, F. Buehler, M. Berclaz, V. Schelbert, J. Geurts, E. Krasnova, E. Steinke, J. Toma, J. Wuethrich, R. Polat, W. Zimmermann, P. Arm, N. Rudin, H. Kolvenbach, and M. Hutter, "SpaceHopper: A small-scale legged robot for exploring low-gravity celestial bodies," 2024.

[11] J. A. Olsen and K. Alexis, "Design and experimental verification of a jumping legged robot for martian lava tube exploration," *2023 21st International Conference on Advanced Robotics (ICAR)*, pp. 452–459, 2023.

[12] A. Westre, J. A. Olsen, and K. Alexis, "Model predictive attitude control of a jumping-and-flying quadruped for planetary exploration," in *2024 IEEE Aerospace Conference*, pp. 1–12, 2024.

[13] H. Goldstein, C. P. Poole, and J. L. Safko, *Classical Mechanics*. Upper Saddle River, NJ: Pearson, 3 ed., June 2001.

[14] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modelling, Planning and Control*. Springer Publishing Company, Incorporated, 2010.

[15] A. Kehlenbeck, *Quaternion-based control for aggressive trajectory tracking with a micro-quadrotor UAV*. PhD thesis, University of Maryland, College Park, Jan. 2014.

[16] J. Solà, "Quaternion kinematics for the error-state kalman filter," *ArXiv*, vol. abs/1711.02508, 2015.

[17] D. Q. Huynh, "Metrics for 3D rotations: Comparison and analysis," *J. Math. Imaging Vis.*, vol. 35, pp. 155–164, Oct. 2009.

[18] Y.-B. Jia, "Quaternions." Com S 477/577 Notes, Dec. 8 2022. Available from: `http://example.com`.

[19] R. Tedrake, *Underactuated Robotics*. 2023.

[20] R. Featherstone, *Rigid body dynamics algorithms*. New York, NY: Springer, 2 ed., Nov. 2007.

[21] Patrick M. Wensing, M. Posa, Y. Hu, A. Escande, N. Mansard, and A. D. Prete, "Optimization-based control for dynamic legged robots," 2022.

[22] M. Kelly, "An introduction to trajectory optimization: How to do your own direct collocation," *SIAM Review*, vol. 59, no. 4, pp. 849–904, 2017.

[23] J. T. Betts, "Survey of numerical methods for trajectory optimization," *Journal of Guidance, Control, and Dynamics*, vol. 21, no. 2, pp. 193–207, 1998.

[24] M. Diehl, H. G. Bock, H. Diedam, and P.-B. Wieber, "Fast Direct Multiple Shooting Algorithms for Optimal Robot Control," in *Fast Motions in Biomechanics and Robotics*, (Heidelberg, Germany), 2005.

[25] T. A. Johansen, "Introduction to nonlinear model predictive control and moving horizon estimation." `https://api.semanticscholar.org/CorpusID:14712069`, 2011. [Accessed 29-06-2024].

[26] H. Ferreau, C. Kirches, A. Potschka, H. Bock, and M. Diehl, "qpOASES: A parametric active-set algorithm for quadratic programming," *Mathematical Programming Computation*, vol. 6, no. 4, pp. 327–363, 2014.

[27] G. Frison and M. Diehl, "Hpipm: a high-performance quadratic programming framework for model predictive control," 2020.

[28] M. P. Kelly, "Transcription methods for trajectory optimization: a beginners tutorial," 2017.

[29] A. Bemporad, "Model predictive control lecture notes: Linear mpc." `http://cse.lab.imtlucca.it/~bemporad/teaching/mpc/imt/1-linear_mpc.pdf`. [Accessed 29-06-2024].

[30] M. Quigley, "Ros: an open-source robot operating system," in *IEEE International Conference on Robotics and Automation*, 2009.

[31] R. Tedrake and the Drake Development Team, "Drake: Model-based design and verification for robotics," 2019.

[32] T. R. Institute, "Rethinking Contact Simulation for Robot Manipulation — medium.com." `https://medium.com/toyotaresearch/rethinking-contact-simulation-for-robot-manipulation-434a56b5ec88`. [Accessed 26-06-2024].

[33] R. Verschueren, G. Frison, D. Kouzoupis, J. Frey, N. van Duijkeren, A. Zanelli, B. Novoselnik, T. Albin, R. Quirynen, and M. Diehl, "acados – a modular open-source framework for fast embedded optimal control," *Mathematical Programming Computation*, Oct 2021.

[34] J. Norby, Y. Yang, A. Tajbakhsh, J. Ren, J. K. Yim, A. Stutt, Q. Yu, N. Flowers, and A. M. Johnson, "Quad-SDK: Full stack software framework for agile quadrupedal locomotion," in *ICRA Workshop on Legged Robots*, May 2022.

[35] I. Poulakakis, J. A. Smith, and M. Buehler, "Modeling and experiments of untethered quadrupedal running with a bounding gait: The scout ii robot," *The International Journal of Robotics Research*, vol. 24, pp. 239 – 256, 2005.

[36] I. Poulakakis, J. Smith, and M. Buchler, "Experimentally validated bounding models for the scout ii quadrupedal robot," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, vol. 3, pp. 2595–2600 Vol.3, 2004.

Appendix A - Euler-Lagrange Equations of Motion Extraction

General Methodology

For the derivation of the analytical dynamical equations, the Euler-Lagrange method was used. The process is as follows:

1. Calculate the Kinetic energy (T) of the system.
2. Calculate the Potential energy (U) of the system.
3. Calculate the Lagrangian: $\mathcal{L} = T - U$.
4. Find the following derivatives:

$$\frac{\partial}{\partial t}\left(\frac{\partial(\mathcal{L})}{\partial \dot{\mathbf{q}}}\right)$$

$$\frac{\partial(\mathcal{L})}{\partial \mathbf{q}}$$

where $\mathbf{q}$ is the vector of generalized coordinates. Here $\mathbf{q}$ is the joint angle vector. The equations of motion are given by:

$$\frac{\partial}{\partial t}\left(\frac{\partial(\mathcal{L})}{\partial \dot{\mathbf{q}}}\right) - \frac{\partial(\mathcal{L})}{\partial \mathbf{q}} = \mathbf{f} \tag{80}$$

where $\mathbf{f}$ is the generalized force. It contains:

- the coulomb friction: $\mathbf{F_s}\ sign(\dot{\mathbf{q}}) = diag\{F_{s,1}, ..., F_{s,i}, ..\} \cdot sign(\dot{\mathbf{q}})$
- damping forces: $\mathbf{D}\ \dot{\mathbf{q}} = diag\{D_1, ..., D_i, ..\} \cdot \dot{\mathbf{q}}$
- forces due to interactions with the environment: $\mathbf{J}^T(\mathbf{q})\mathbf{h}_e$, where $\mathbf{J}^T$ is the geometric Jacobian and $\mathbf{h}_e$ is the wrench vector from the end-effector to the environment
- motor inputs: $\boldsymbol{\tau}$

Usually, these equations are more useful in a matrix form, that takes the following form:

$$\mathbf{B}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) + \mathbf{D}\dot{\mathbf{q}} + \mathbf{F}_s sign(\dot{\mathbf{q}}) + \mathbf{J}^T(\mathbf{q})\mathbf{h}_e = \boldsymbol{\tau} \tag{81}$$

5. Collect the terms to get the matrix form of the equations.

Having the dynamics in the form of equation (81), one can easily simulate the system using equation (82). (This form is compatible with the matlab and acados solvers):

$$\frac{d}{dt}\begin{bmatrix}\mathbf{q}\\ \dot{\mathbf{q}}\end{bmatrix} = \begin{bmatrix}\dot{\mathbf{q}}\\ \mathbf{B}(\mathbf{q})^{-1} \cdot (\boldsymbol{\tau} - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} - \mathbf{G}(\mathbf{q}) - \mathbf{D}\dot{\mathbf{q}} - \mathbf{F}_s sign(\dot{\mathbf{q}}) - \mathbf{J}^T(\mathbf{q})\mathbf{h}_e)\end{bmatrix} \tag{82}$$

Regarding the friction and damping forces, usually only the joint static and viscous friction[34] is taken in account in robotics, and thus these forces can be calculated if the matrices $\mathbf{F}_s$ and $\mathbf{D}$ are defined. Both in drake and simscape, one can directly define these matrices. These forces will be ignored, except if stated otherwise. The interaction force is zero as long as there are no interactions, such as when the robot is moving in free space. Thus, initially, one has to find the $\mathbf{B}, \mathbf{C}, \mathbf{G}$ matrices.

Calculating the Lagrangian

The first step is the calculation of the Lagrangian. The following steps are needed:

- Definition of the transformation matrices ${}^{i-1}\mathbf{T}_i$.
- Loading of the geometric and inertial quantities of the leg, from the URDF.

[34] Friction forces that have to do with the end effector and the environment are modelled as an external wrench $\mathbf{h}_e$.

- Calculatation of the positions of the coordinate frame i with respect to the j frame[35].

$$ {}^{j}\mathbf{P}_i = {}^{j}\mathbf{J}_i(1:3,\ 4) = \begin{bmatrix}\mathbf{I}_{3x3} & \mathbf{0}_{3x1}\end{bmatrix} {}^{j}\mathbf{T}_i \begin{bmatrix}\mathbf{0}_{3x1} \\ 1\end{bmatrix} \tag{83} $$

- Calculatation of the orientation of the coordinate frame i with respect to another j frame

$$ {}^{j}\mathbf{R}_i = {}^{j}\mathbf{T}_i(1:3,\ 1:3) = \begin{bmatrix}\mathbf{I}_{3x3} & \mathbf{0}_{3x1}\end{bmatrix} {}^{j}\mathbf{T}_i \begin{bmatrix}\mathbf{I}_{3x3} \\ \mathbf{0}_{1x3}\end{bmatrix} \tag{84} $$

- Calculation of the positions of the center of mass of link i with respect to the 0 frame. From solidworks, we have ${}^{i}\mathbf{r}_{c,i}$ (the position of the center of mass of link i with respect to the coordinate frame of the link i).

$$ {}^{0}\mathbf{x}_{c,i} = {}^{0}\mathbf{P}_i + {}^{0}\mathbf{R}_i\ {}^{i}\mathbf{r}_{c,i} \tag{85} $$

- Calculation of the angular velocities of the coordinate frame i with respect to the 0 frame. These are the same as the angular velocities of the center of mass of the link i with respect to the 0 frame.

$$ {}^{0}\boldsymbol{\omega}_i = {}^{0}\boldsymbol{\omega}_{i-1} + {}^{0}\mathbf{R}_i\ [0,\ 0,\ \dot{q}_i]^T \tag{86} $$

- Calculation of the linear velocities of the coordinate frame i with respect to the 0 frame.

$$ {}^{0}\mathbf{u}_i = {}^{0}\mathbf{u}_{i-1} + \underbrace{{}^{0}\mathbf{u}_i^{i-1}}_{=\mathbf{0}} + [{}^{0}\boldsymbol{\omega}_{i-1}]^x\ {}^{0}\mathbf{R}_{i-1}\ {}^{i-1}\mathbf{P}_i \tag{87} $$

- Calculation of the velocities of the center of mass of link i with respect to the 0 frame.

$$ {}^{0}\mathbf{u}_{c,i} = {}^{0}\mathbf{u}_i + \underbrace{{}^{0}\mathbf{u}_{c,i}^{i}}_{=0} + [{}^{0}\boldsymbol{\omega}_i]^x\ {}^{0}\mathbf{R}_i\ {}^{i}\mathbf{r}_{c,i} \tag{88} $$

Having calculated these quantities, it is easy to write the Kinetic energy as:

$$ T = \sum_{i=1}^{3} \left[\frac{1}{2} m_i\ {}^{0}\mathbf{u}_{c,i}^T\ {}^{0}\mathbf{u}_{c,i} + \frac{1}{2}\ {}^{0}\boldsymbol{\omega}_i^T\ {}^{0}\mathbf{R}_i\ \mathbf{I}_i\ ({}^{0}\mathbf{R}_i^T)\ {}^{0}\boldsymbol{\omega}_i \right] \tag{89} $$

The potential energy is :

$$ U = -\sum_{i=1}^{3} m_i \mathbf{g}_0^T\ {}^{0}\mathbf{x}_{c,i} \tag{90} $$

where $\mathbf{g}_0$ is the gravity vector expressed in the base frame of the kinematic chain.

The Lagrangian is the $\mathcal{L} = T - U$

Getting the B,C,G matrices

Getting the derivatives of the Lagrangian can be done using the differentiation functions of the `symbolic` package of matlab.

Each equation concerning the i-th DOF has the following form:

$$ eq_i = \sum_j b_{i,j}(\mathbf{q})\ddot{q}_j + \sum_j c_{i,j}(\mathbf{q}, \dot{\mathbf{q}})\dot{q}_j + G_i(\mathbf{q}) \tag{91} $$

To get the matrices, the following process is followed for each degree of freedom i:

1. Get $b_{i,j}$ from the coefficients of $\ddot{q}_j$ in the equation of the i-th degree of freedom. This can be achieved as the polynomial of $\ddot{q}_j$, $p(\ddot{q}_j)$ has degree $deg(p(\ddot{q}_j)) \leq 1$. The latter fact can be observed from (91).

2. Update the equation:

$$ eq_i^{new,1} = eq_i - \sum_j b_{i,j}(\mathbf{q})\ddot{q}_j $$

[35]Indexing via matrix multiplication was done because `symfun` objects cannot be indexed using parenthesis indexing

3. Get $c_{i,j}$ from the coefficients of $\dot{q}_j$ in the **new** equation of the i-th degree of freedom $eq_i^{new,1}$. The polynomial of $\dot{q}_j$, $p(\dot{q}_j)$ has degree $deg(p(\dot{q}_j)) \leq 2$. This fact can be observed from (91). The polynomial is $p(\dot{q}_j) = p_0 + p_1\dot{q}_j + p_2\dot{q}_j^2$. So $c_{i,j}$ is obtained by :

$$c_{i,j} = p_1 + p_2\dot{q}_j$$

Generally, there are infinite[36] choices for the C matrix.

4. Update the equation:

$$eq_i^{new,2} = eq_i^{new,1} - \sum_j C_{i,j}(\mathbf{q}, \dot{\mathbf{q}})\dot{q}_j$$

5. The rest are the gravitational terms: $G_i(\mathbf{q}) = eq_i^{new,2}$

Getting the coefficients is done by the `coeffs` command of the `symbolic toolbox`. Getting the coefficients of the full polynomial for each degree of freedom is done by providing the *'All'* argument.

[36]Siciliano [14], section 7.2.1. For example, a term $k\dot{q}_1\dot{q}_2$ in the i-th equation can be distributed in the $\mathbf{C}$ matrix as follows:

$$c_{i,1} += \lambda\dot{q}_2$$
$$c_{i,2} += (k - \lambda)\dot{q}_1$$

where $\lambda \in \mathbb{R}$.