



POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Low level Whole-Body Controller for a quadruped robot - design and implementation

TESI DI LAUREA MAGISTRALE IN
AUTOMATION AND CONTROL ENGINEERING - INGEGNERIA
DELL'AUTOMAZIONE

Author: **Daniele Di Francesco**

Student ID: 991221

Advisor: Prof. Luca Bascetta

Co-advisors: Douglas W. Bertol

Academic Year: 2022-23

Abstract

Legged robots, mainly quadrupeds, saw a significant increase of interest by the scientific community in the last years due to their versatility, adaptability and potential for real-world application, even if they present several control challenges due to their intrinsic underactuation. The today state of the art robots, often equipped with sensors and manipulators, are able to perform complicated tasks also on rough and unpredictable terrains, notwithstanding this, there is wide room for improvement both from the control point of view and also regarding path planning and sensor fusion.

The reason behind this thesis is to design and simulate the full low level control scheme of a light and small quadruped robot, side by side with learning a control framework for its implementation. Namely, it is not taken into account the high level controller, the reason behind its movement, but the controller will address the problem of taking as input a velocity along the x-y plane and giving as an output the torque control for each of the joints. Along with this, the controller will take into account the robot stability, the non slipping condition of the legs and wrench optimization.

The proposed model and the low level whole-body controller have been successfully used to simulate tasks such as walking in different surface conditions and performing pushups. From the implementation point of view, the controller has been coded and the robot simulated and, analyzing the libraries used in the control framework, it has been proposed the use of a more effective and easy to use C++ library to compute the Rigid Body Dynamics algorithms.

Keywords: quadruped robot, whole-body controller, control framework

Abstract in lingua italiana

I robot quadrupedi hanno visto un significativo aumento di interesse da parte della comunità scientifica negli ultimi anni grazie alla loro versatilità, adattabilità e potenzialità di applicazione nel mondo reale, anche se presentano diverse sfide di controllo a causa della loro intrinseca sottoattuazione. I robot allo stato dell'arte, spesso dotati di sensori e manipolatori, sono in grado di svolgere compiti complicati anche su terreni accidentati e imprevedibili, tuttavia esistono ampi margini di miglioramento sia dal punto di vista del controllo sia per quanto riguarda la pianificazione del percorso e l'integrazione dei sensori.

Lo scopo di questa tesi è progettare e simulare lo schema di controllo a basso livello di un robot quadrupede leggero e di piccole dimensioni, parallelamente all'apprendimento di un framework di controllo per la sua implementazione. In altre parole, non viene preso in considerazione il controllore di alto livello, la ragione del suo movimento, ma il controllore di basso livello affronterà il problema di prendere in ingresso una velocità lungo il piano x-y e dare in uscita il controllo della coppia per ciascuno dei giunti. Inoltre, il controllore terrà conto della stabilità del robot, della condizione di non slittamento delle zampe e dell'ottimizzazione delle forze che agiscono sul tronco.

Il modello proposto e il controllore a basso livello sono stati utilizzati per simulare compiti come camminare in diverse condizioni di superficie ed eseguire flessioni. Dal punto di vista dell'implementazione, sono stati realizzati sia il modello che il controllore e, analizzando le librerie utilizzate dal framework, è stato proposto l'uso di una libreria C++ più efficace e facile da usare per calcolare gli algoritmi di dinamica del corpo rigido.

Parole chiave: robot quadrupedi, controllo a basso livello, framework di controllo

Contents

Abstract	i
Abstract in lingua italiana	iii
Contents	v
Introduction	1
1 State of the Art and related works	5
1.1 Trajectory planner and Low level Whole-Body Controller	8
1.2 Implementation Techniques and Algorithms	11
2 Robot model	13
2.1 Dynamic model	14
2.2 Linearized friction cone constraint	16
2.3 Trot gait	18
3 Control scheme	19
3.1 Reference Generator	19
3.2 Whole-Body Controller	21
3.2.1 Trunk Controller	22
3.2.2 Joint-Space PD	25
4 Implementation	27
4.1 The DLS framework	28
4.1.1 Periodic App	30
4.1.2 Robot library	31
4.1.3 Gazebo hardware	32
4.2 Robotlib	32
4.3 Rigid Body Dynamics algorithms	33

4.3.1	RobCoGen	34
4.3.2	Pinocchio	35
5	Simulations and results	39
5.1	Tuning parameters	39
5.1.1	Whole-Body Controller parameters	39
5.1.2	Joint-space PD tuning	40
5.2	Walking	41
5.3	Walking on a slippery surface	43
5.4	Walking on an inclined plane	45
5.5	Falling	47
5.6	Energetic analysis	49
6	Conclusions and further developments	51
	Bibliography	53
A	Euler angular velocities	57
B	Jacobian of a task	59
	List of Figures	61
	List of Tables	63
	List of Symbols	65
	Nomenclature	67

Introduction

In recent years, the field of robotics has witnessed remarkable advancements, particularly in the domain of legged locomotion. Among the myriad forms of legged robots, quadrupeds stand out for their versatility, adaptability, and potential for real-world applications across various domains, including search and rescue, exploration, and agriculture. Central to the functionality and performance of these quadrupedal robots is the development of sophisticated low level control strategies.

Legged robots offer a significant advantage over wheeled counterparts due to their capability to navigate intricate and unpredictable environments like forests, obstacles, and debris. Nevertheless, controlling legged robots presents intricate challenges related to underactuation, where the body is controlled indirectly through the legs, and the hybrid nature of the forces needed for motion generation. This complexity arises from the robot's necessity to establish and interrupt contact between its feet and the ground

This thesis delves into the realm of low level control mechanisms tailored specifically for robot quadrupeds, namely it will not consider the cause for the requested movement, leaving this job to a more high-level planner/controller. It will consider as inputs the desired gait, the linear velocities along the x,y plane or directly the joints position. The output of the controller will be as low as the desired torque for each joint.

This work is part of a collaboration between the UDESC (Universidade do Estado de Santa Catarina) of Joinville, Brazil and the DLS (Dynamic Legged System) laboratory of the Italian Institute of Technology of Genova, Italy. The research was carried out in the GASR (Grupo de Pesquisa em Automação de Sistemas e Robótica) laboratory of the UDESC university, a laboratory focused in control of mobile robots. There has also been strict contact with the DLS lab of Genova, already very well known in the world of quadruped's control mainly because of the design of an hydraulic quadruped robot called HyQ. The goal of DLS's research, along with designing robots, is the one of creating a universal framework for controlling quadrupeds. This framework, called DLS framework, would have many advantages with respect to other state of the art controlling frameworks such as ROS because its aim is to be more high level, giving pre-made templates of each

component of the control scheme in order to help the implementation of the controller.

Main contribution and results



Figure 1: Unitree Go1 robot of UDESC university

The work presented in this thesis summarize the research made at the UDESC university where the main goal has been to design a model and a low level whole-body controller to make a quadruped able to perform simple tasks such as walking in different directions or being able to perform pushups. This work has been done with the goal of implementing the final scheme on the hardware, therefore particular effort has been made to develop a full parametrized code and to also consider non-nominal conditions of the environment.

The hardware considered is a Unitree Go1 (Fig.1), a 12 *kg* robot with 12 joints. Despite being this the reference robot, the assumptions, the model and the controllers presented from now on can be referred to a wide range of similar robots. This was the main aim of the research, to be able to develop a way to quickly switch between models and controllers in the DLS2 framework environment. Therefore, the model and the low level controller presented here are made to be suitable for a wider range of similar hardware, considering dimensions, weight and actuated joints.

The work started with an extensive study of the state of the art solutions already present in literature along the study of the functionalities of the DLS2 framework. This includes a full knowledge of the programming environment, learning how to access and to edit each of the part of the framework such as the DDS system or the low level templates for the

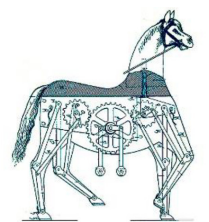
generators and the controllers. After that, a suitable dynamic model has been studied and a simple PID controller and a joint-space reference generator has been implemented to test very simple tasks. Finally, a full low level whole-body control scheme has been developed: considering an external velocity input that can be held by a gaming joystick, a reference generator computes the joint-space trajectories of each of the joint that will be tracked in a closed-loop with a PID controller. Along this, the desired wrench of the robot in the CoM is computed and tracked with the solution of a QP problem.

Thesis structure

The document is organized as follows: Chapter 1 contains literature research from the invention of legged robots up to the most performing quadruped robots actually on the market, the most advanced and up-to-date control schemes along with the current software and hardware used to code, simulate and implement the required algorithms. Chapter 2 proposes a dynamic model and the relative constraints used later in Chapter 3 where the whole control scheme is described. Chapter 4 describes the software and the libraries used to implement the low level controller. Chapter 5 analyzes the results of the experiments run in simulation and, finally, Chapter 6 concludes the document summarizing the work done and proposing new development possibilities.

1 | State of the Art and related works

In the realm of quadruped mobile robots (QMR)[2], the initial strides were made by Rygg in the 1890s with the creation of the horse machine (shown in Fig. 1.1), marking the first attempt at a quadrupedal walking device. However, it lacked controllability, disqualifying it as the inaugural QMR. The roots of modern autonomous QMRs can be traced back to the late 1960s with the development of the Phony Pony, also known as the Californian horse, by Frank and McGhee. This pioneering creation (also shown in Fig. 1.1) was the first QMR under computer control, making its debut walk in 1977. It quickly became a model for subsequent QMRs. The Phony Pony boasted 2 degrees of freedom (DOF) at each leg, powered by electrical motors, with an adaptive joint in the front enabling varied gaits like walking, crawling, and trotting while maintaining stability, albeit at a sluggish pace. Additionally, the robot's symmetry allowed for uniform control in both forward and backward movements, with indistinguishable hip and knee joints on all legs.[22]



(a) The horse machine



(b) Phony Pony

Figure 1.1: The horse machine of 1890 and the Phony Pony of 1977

Fast forward to modern days, Boston Dynamics introduced several advanced QMRs that have made this company stand as a front-runner in canine-QMR technology. In the year 2011, BD released the fastest QMR in the world named Cheetah, which can run up to 12.5 (m/s) speed and stop quickly. Cheetah QMR is designed with an articulated back that bends at each step like real animals, which increases the robot's trot and bound speeds.

The next generations of QMRs made by BD concentrates more on industrial and service

robotics applications; Spot (later named Spot Classic) is one of them. This QMR (shown in Fig. 1.2) with only 72.57 kg in weight is way smaller, quieter, and cheaper than the past series while operating more sophisticated algorithms with more agility and inspection. Spot, which is built based on pump-electric servo hydraulic cylinder actuators, is way quieter than its predecessors, and it can navigate over rough terrains as well as soft terrains (i.e. snowy, muddy, etc.), avoid obstacles, climb stairs and hills, and jump. The onboard control unit of this robot is based on a much more powerful PC, and five sensor modules all around the robot allow Spot to autonomously explore more complex environments while doing intelligent tasks such as going through narrow spaces in a wide range of conditions. Spot is mainly designed for the construction process, factory usage, and high-risk situation monitoring applications, although it can easily fit in other similar applications as well. SpotMini (shown in Fig. 1.2), is an upgraded Spot with the same flexibility and dynamics, but it is built based on quieter and cleaner actuators with less battery consumption. SpotMini, which is relatively smaller, can operate for about 90 min on battery. Recently, BD changed the name of the SpotMini to Spot and canceled the old version. There are also Spot robots of both versions that are equipped with a 5-DoF manipulator mounted at the back of the robot, allowing the robot to pick up objects, open doors, and perform other manipulator tasks.[12]



Figure 1.2: Spot classic and spot mini

The Italian Institute of Technology (IIT) introduced a large-size hydraulically and electrically actuated QMR named HyQ (shown in Fig. 1.3) that is designed for study purposes, aiming at highly dynamic motions across different types of terrains. The robot's size is 1 by 0.5 (m) with 0.98 (m) height in stand mode with a weight of about 80 kg. HYQ can jump 20 cm up and reach the speed of 1.8 (m/s) trotting. There are several well-detailed studies on HYQ mechanism and performance, for instance, a study on the leg mechanism of HYQ is available in, and a study on the HYQ leg stability as well as a compliance controller is published in literature[23]. HyQ2Max[24] is an upgraded version of HyQ with the same weight while having better robustness and reliability in its hardware components, besides, its joints can perform higher ranges of motions with relatively higher torques, that allow the robot to obtain larger foot workspaces. Moreover, all the key components

are put inside a trunk made of a strong aluminum alloy, which protects them from the impact of dirt and humidity as well as collision and external impacts. HyQ2Centaur is a HyQ2Max with a slight change in its trunk design equipped with two hydraulic arms mounted on its front top, which enables the robot with an ability of object manipulation, while MiniHyQ is a lightweight version of HyQ in $0.85 \times 0.35 \times 0.77$ m (LWH) dimensions and 25 kg weight with a reconfigurable leg mechanism. HYQReal[1] can perform different types of gaits (i.e. run, jump, hop, gallop, etc.), maintain its stability, and execute complex and heavy-duty tasks; for instance, in an experiment conducted by its designers, HYQReal pulls a small size airplane. HYQReal is 130 kg in weight, with a size of $1.33 \times 0.67 \times 0.9$ (m), capable of executing sophisticated heavy-duty tasks. This QMR is designed based on a smart hydraulic actuation mechanism developed by Moog Inc in collaboration with IIT, which enhances the robot's dynamics and compared to HyQ and HyQ2Max, the HYQReal is much more sophisticated in terms of autonomy.

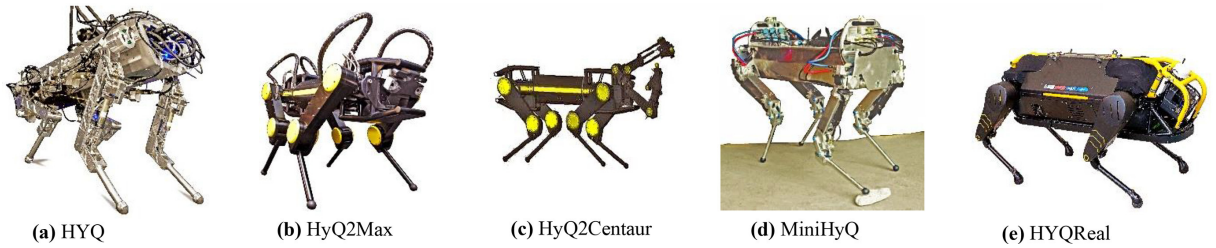


Figure 1.3: IIT QMRs

Unitree, designed by Shanghai University in collaboration with Wang[13] is a series of commercially available and highly dynamic robots that are built on out-runner brushless actuators (8108 motors) and equipped with efficient sensory modules and AI-based controllers for sophisticated industrial and educational applications. Go1 series (Air, Pro, and Edu versions) have 12 kg in weight and $0.59 \times 0.22 \times 0.29$ m in dimensions with the same appearance, but with different sets of sensory modules and control components. Go1-Air with 2.5 (m/s) max trot speed and Go1-Pro with 3.5 (m/s) speed can carry 3 kg of loads, while Go1-Edu with 3.7 to 5 (m/s) speed is capable of 5–10 kg payloads, and it has also a more powerful processor, Python API support, foot touch sensors, and 3D LIDAR sensor. Unitree has also AlienGo QMR, which is a small-size ($0.6 \times 0.35 \times 0.6$ m) and lightweight (21 kg) robot capable of reaching 1.67 (m/s) in speed, and it can carry 10 kg loads while performing highly dynamic motions, such as fast trotting, jumping, climbing, etc., for 2.5 - 4.5 h in one span.



Figure 1.4: Unitree Go1 and Unitree Aliengo

1.1. Trajectory planner and Low level Whole-Body Controller

The low level controller in a quadruped robot is responsible for executing the detailed motor commands that drive the individual joints and actuators to achieve desired movements. It plays a crucial role in converting high level commands or intentions, often generated by higher-level planning or control layers, into low level control signals that the actuators can understand and execute. The primary tasks of the low level controller in a quadruped robot include:

Motor Control:

The low level controller is responsible for regulating the position, velocity, or torque of each joint's motor. This involves generating control signals that command the actuators to move the joints to specific positions or follow specific trajectories.

Gait Generation:

Quadruped robots typically use a specific gait or walking pattern to move efficiently. The low level controller generates and coordinates the sequence of leg movements, ensuring a stable and effective gait. This may involve adjusting the timing and coordination of leg motions to adapt to different walking speeds or terrains.

Stability and Balance Control:

Ensuring stability is crucial for a quadruped robot to maintain balance during locomotion. The low level controller incorporates feedback from various sensors, such as gyroscopes and accelerometers, to adjust the joint motions in real-time and prevent the robot from falling.

Terrain Adaptation:

Quadruped robots often encounter different types of terrain, and the low level controller

must adapt the leg movements and joint control to handle variations in ground conditions. This includes adjusting the leg compliance and stiffness based on sensor feedback.

Compliance Control:

Compliance control allows the robot to interact with its environment in a flexible and adaptive manner. The low level controller adjusts the compliance of the joints to absorb shocks, navigate uneven surfaces, and respond to external forces.

Energy Optimization:

The low level controller may implement strategies to optimize energy consumption during locomotion. This involves adjusting the control signals to minimize power usage while maintaining effective movement.

Real-Time Sensor Fusion:

Integrating information from various sensors in real-time is crucial for effective low level control. This may include proprioceptive sensors (joint encoders) and exteroceptive sensors (cameras, lidar) to provide accurate feedback for adjusting motor commands.

Dynamic Locomotion:

Quadrupeds need to adapt their movements dynamically to navigate challenging environments. The low level controller calculates and adjusts joint trajectories to ensure the robot can handle dynamic changes in terrain or external disturbances. In summary, the low level controller acts as the interface between the high level planning or decision-making layers and the physical actuators of the quadruped robot. It is responsible for executing precise and coordinated movements to enable stable and effective locomotion in diverse and dynamic environments.

Regarding the problem of designing a controller able to compute the joint torques desired to correctly track the trajectory generated from the high level control scheme, many solutions can be adopted.

Starting from the quadruped model, since the goal of the robot is to achieve a dynamic locomotion on rough terrains, it is often discarded a full kinematic model, where the inertias are considered to be negligible. A solution is the use of the Single Rigid Body Dynamics, where only the trunk of the quadruped is considered to have a mass. This assumption is made depending on the actual hardware, for instance it has been shown [19] that this hypothesis holds while conducting experiments with a 87 kg HyQ robot, where the mass is highly concentrated on the trunk with respect to the legs.

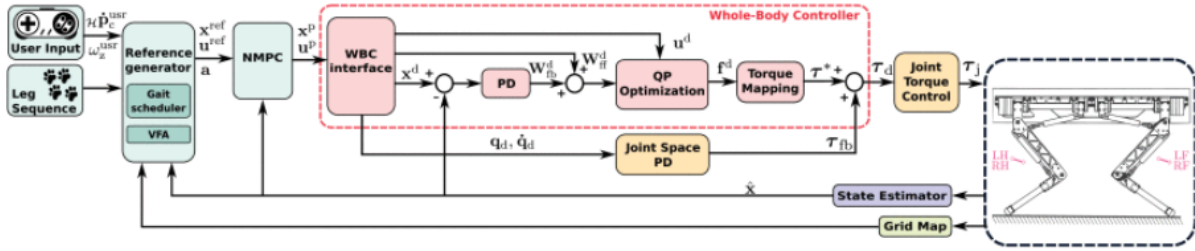


Figure 1.5: Typical complete low level control scheme, as showed in [19]

The most complete solution, but also more demanding with respect to the computational power, is the use of the Full Rigid Body Dynamics, where also the mass of the legs is considered. In particular this solution is crucial when addressing the problem of stability of robot manipulators [21], where there is a strong coupling between the manipulator, the trunk and the legs.

An innovative modeling approach is instead the Torso Dynamics [11] where initially a Full Rigid Body Dynamics system is considered, but then the rigid-body inertia matrix is divided based on the top 6 underactuated rows in the centroidal dynamics and the coupling between the legs and the trunk. To simplify the torso dynamics, the function to compute the angular and linear velocities is evaluated with zero inertial coupling forces from the joints. This simplification allows to consider the legs only on velocity level and removes joint accelerations from the formulation.

Regarding the trajectory planner, the most common solution is to adopt a Nonlinear MPC. Many different solutions can be adopted with respect to the formulation of the problem and particularly to the costs of the formulation. N. Rathod [19] introduces an innovative concept of mobility, defined as the attitude of the leg to arbitrarily change its foot position. The esteem of the optimal distance between hip and foot (mobility factor) is considered in this cost function, eliminating the need of specifying roll, pitch and height of the trunk. R. Grandia [11] instead proposes a solution with LIDARs and a terrain perception and segmentation algorithm to insert in the NMPC formulation a cost function with the optimal trunk orientation based on the terrain the robot has to walk on to.

Finally, the Whole-Body Controller is in charge to track the required ground forces by applying the correct torque to each joint of the quadruped. This problem is often solved with a PD and a Quadratic Programming optimization to map the required torques. Other solutions consider the implementation of an impedance control in a QP problem. In the case of a robot manipulator the derived closed loop system can be simplified as a

double mass-damper-spring system as shown in [21]. A very innovative solution of using a Nonlinear MPC to solve the low level control problem is shown in [17] where, through an explicit contact model for the feet and customized NMPC solvers, very good performances can be achieved with respect to a more classical QP controller. These experiments show an NMPC running at 190 Hz with half a second of time horizon that is able to outperform the state of the art by at least one order of magnitude.

1.2. Implementation Techniques and Algorithms

Regarding the control framework, it is widely used the Robotic Operating System that provides libraries and tools for tasks such as hardware abstraction, device drivers, communication between processes, and package management. Several packages and libraries within ROS2 are dedicated to robotic control. [25]

Another state of the art solution, but still under development, is the DLS2 framework maintained by the DLS lab of the Italian Institute of Technology of Genova. This middleware, tailored to control legged robots, is a containerized ROS-like framework developed in C++. It presents a slightly more high level environment to control legged robots with respect to ROS. Each node (called app) is taken from a template class that already defines some mandatory functions and the scheduling period. A DDS messaging system is also implemented such that can be easily accessed by all the apps.

Regarding the Rigid Body Dynamics libraries for implementing the control algorithms, RobCoGen [10] plays an important role in many implementations. Short for Robot Control Generator, it is a tool designed to generate efficient robot controllers. It focuses on generating customized controllers for various robotic systems, including quadruped robots. It features code generation for different control architectures and optimization for real-time performance. One of the drawbacks of RobCoGen is that needs a full detailed model to compute all the code that will be later imported in the framework.

To overcome this problem, a powerful tool is Pinocchio Robot Library [5] that instead takes as input only the URDF file and than it is able to provide the analytical derivatives of the main Rigid-Body Algorithms like the Recursive Newton-Euler Algorithm or the Articulated-Body Algorithm. For instance, taking as example an HyQ robot and a standard laptop equipped with an Intel Core i7 CPU @ 2.4 GHz, Pinocchio is able to compute the Inverse Dynamics algorithm in less than 1 μs . Crocoddyl [15] is instead tailored for Optimal Control problems of quadrupeds and humanoids. Based on Pinocchio, it computes optimal trajectories and feedback gains.

2 | Robot model

The purpose of this section is defining a model of the robot that can be valid for the kind of quadrupeds considered in this study, namely light and electric driven robots such as Unitree Go1 and Unitree Aliengo. Even though it will be performed a validation only through simulation, it is assumed to have the same sensors and encoders of these robot or at least to be able to esteem these states.

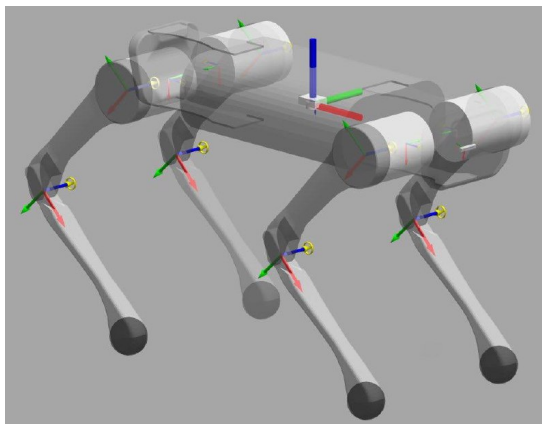


Figure 2.1: Home position and coordinate definition of the robot

The proposed model assumes that each leg has three degrees of freedom. The abduction/adduction joint contributes to the leg motion in the frontal plane. The hip and knee joint commonly relate to the leg motion in the sagittal plane. The robot can move in all directions with a walk or trot gait.

The whole-body dynamics model is built to analyze the dynamics characteristics during walking. In addition, moreover, it is hypotized to have contact sensors on the toes of each leg, which can sense the ground contact. Thus, the changes in external terrain and the position and posture of the robot body can be analyzed in combination with the IMU. Based on the perception of the external environment and itself, the current state can be comprehensively estimated.

2.1. Dynamic model

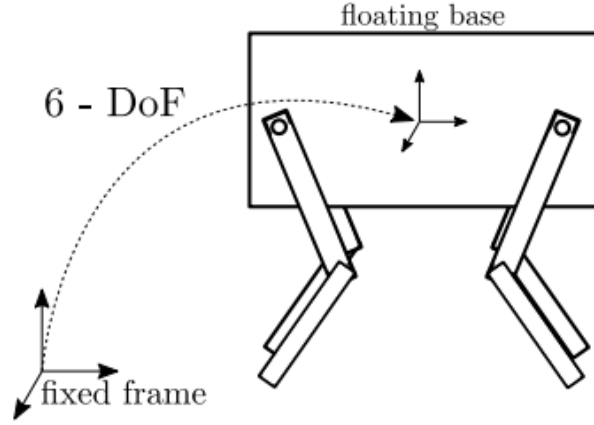


Figure 2.2: Fictitious 6 - DoF that connects fixed frame with the floating base.

To understand how our system changes over time, we need to create a mathematical model that considers how its components evolve. This model, known as a dynamic model, describes the system's behavior over time. For a robotic mechanism, we often use a connectivity graph, where each node represents a joint, including a fixed base or reference frame. If we're dealing with a mobile robot like a legged one, we include a special link called floating base. This adds six degrees of freedom (DoFs) to the system, on top of the existing degrees of freedom. So, if the system considered has n_s degrees of freedom (DoFs), adding the extra joint we have a $n'_s = 6 + n_s$. Importantly, these extra degrees of freedom provided by the 6-DoF joint don't restrict the system's movement but allow for both translation (3 DoF) and rotation (3 DoF).

The dynamic model for a generic n-DoF mobile system (floating base) can be described by a joint-space formulation

$$\mathbf{H}(\mathbf{q})\mathbf{v} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\mathbf{v} + \boldsymbol{\tau}_g(\mathbf{q}) = \boldsymbol{\tau} + \mathbf{J}^T \mathbf{f} \quad (2.1)$$

Where $\mathbf{H}(\mathbf{q}) \in \mathbb{R}^{n'_s \times n'_s}$ is the joint space inertia matrix. $\mathbf{v} = [\mathbf{v}_c^T \quad \dot{\mathbf{q}}^T]^T \in \mathbb{R}^{n'_s}$ is the vector of the generalized velocities where $\mathbf{v}_c = [\dot{\mathbf{c}}^T \quad \dot{\boldsymbol{\omega}}^T]^T \in \mathbb{R}^6$ is the floating base velocity and $\dot{\mathbf{q}} \in \mathbb{R}^{n'_s}$ represents the vector of joint velocities. $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) \in \mathbb{R}^{n'_s \times n'_s}$ is a matrix such that $\mathbf{C}\mathbf{v}$ is the vector of Coriolis and centrifugal terms, $\boldsymbol{\tau}_g(\mathbf{q}) \in \mathbb{R}^{n'_s}$ is the vector of gravity terms, $\boldsymbol{\tau} \in \mathbb{R}^{n'_s}$ is the vector of joint torques and $\mathbf{J}^T \mathbf{f} \in \mathbb{R}^{n'_s \times n'_s}$ is the term that accounts for external forces, given $\mathbf{J} \in \mathbb{R}^{n'_s \times n'_s}$ is the contact points Jacobian.

We can split (2.1) in two parts associated to the unactuated and the actuated parts of the system:

$$\begin{bmatrix} \mathbf{H}^u & \mathbf{H}^{ua} \\ \mathbf{H}^{au} & \mathbf{H}^a \end{bmatrix} \begin{bmatrix} \dot{\mathbf{v}}_c \\ \ddot{\mathbf{q}} \end{bmatrix} + \begin{bmatrix} \mathbf{C}^u \mathbf{v}_c \\ \mathbf{C}^a \dot{\mathbf{q}} \end{bmatrix} + \begin{bmatrix} \boldsymbol{\tau}_g^u \\ \boldsymbol{\tau}_g^a \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \boldsymbol{\tau} \end{bmatrix} + \begin{bmatrix} \mathbf{J}_u^T(\mathbf{p}) \\ \mathbf{J}_a^T(\mathbf{p}) \end{bmatrix} \mathbf{f} \quad (2.2)$$

Where $\mathbf{p} = [\mathbf{p}_1^T \ \dots \ \mathbf{p}_{nc}^T] \in \mathbb{R}^{3 \times nc}$ is the stacked vector of contact positions and nc is the number of contacts. The superscripts u , a , and ua stand for unactuated, actuated and the cross terms between unactuated and actuated parts, respectively.

To find how to employ this dynamics model, it can be rewritten in such a way that the body states, the forces, and the foothold locations are explicitly stated. Moreover, the contact Jacobians are computed using the feet positions. This dependency can be highlighted by stating it explicitly. Consider the rightmost term of equation (2.1), the unactuated part can be written as:

$$\mathbf{J}_u^T(\mathbf{p}) \mathbf{f} = \begin{bmatrix} \mathbf{S}(\mathbf{p}_1) & \dots & \mathbf{S}(\mathbf{p}_{nc}) \end{bmatrix} \begin{bmatrix} \mathbf{f}_1 \\ \vdots \\ \mathbf{f}_{nc} \end{bmatrix} \quad (2.3)$$

Where $\mathbf{S}(\cdot)$ stands for the skew-symmetric operator used to represent a cross product:

$$\mathbf{a} \times \mathbf{b} = \mathbf{S}(\mathbf{a}) \times \mathbf{b} \quad (2.4)$$

Each contact point $\mathbf{p}_i \in \mathbb{R}^3$ is a three-dimensional vector defined in a Cartesian space. From this relationship, it can be seen how the correct foothold placement is really important to correctly drive the system, as it has a direct influence on the inputs of the system. Since the goal is to control the base, the focus must be on solving the top part of (2.1). The unactuated part of the system can be seen as:

$$\mathbf{H}^u \dot{\mathbf{v}}_c + \mathbf{H}^{ua} \ddot{\mathbf{q}} + \mathbf{C}^u \dot{\mathbf{v}}_c + \boldsymbol{\tau}_g^u = \mathbf{J}_u^T \mathbf{f} \quad (2.5)$$

Assuming that the momentum given by the joint velocity is negligible and that the inertia remains close to the nominal one, it is possible to remove the dependency of the joint coordinates. The previous assumptions are reasonable if the trunk mass is considerably larger than the limbs (which is the case of most quadruped robots) or when the joint legs' velocities are small. For the purpose of this thesis the trot gait will be adopted that, under

controlled velocities, guarantees a non-dynamic locomotion. Under these assumptions, the well known Newton Euler equations appear, useful to describe the so-called Single Rigid Body Dynamics.

$$m\dot{\mathbf{v}}_c = m\mathbf{g} + \sum_{i=1}^4 \delta_i \mathbf{f}_i \quad (2.6)$$

$$\mathbf{I}\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times \mathbf{I}\boldsymbol{\omega} = \sum_{i=1}^4 \delta_i \mathbf{p}_{cf,i} \times \mathbf{c}\mathbf{f}_i \quad (2.7)$$

where m is the robot mass, $\dot{\mathbf{v}}_c \in \mathbb{R}^3$ is the CoM acceleration, \mathbf{g} is the gravitational acceleration, $\mathbf{f}_i \in \mathbb{R}^3$ is the ground reaction force at foot i , $\mathbf{I} \in \mathbb{R}^{3 \times 3}$ is the inertia tensor computed at the CoM frame origin, $\dot{\boldsymbol{\omega}} \in \mathbb{R}^3$ is the angular acceleration of the robot's base, $\mathbf{p}_{cf,i} \in \mathbb{R}^3$ is the distance between the CoM position $\mathbf{p}_c \in \mathbb{R}^3$ and the position $\mathbf{p}_{f,i} \in \mathbb{R}^3$ of foot i . The parameter $\delta_i = \{0, 1\}$ is introduced to define whether foot i is in contact with the ground and can therefore generate contact forces or not.

The robot dynamics can be expressed as the continuous-time state-space model:

$$\begin{bmatrix} \dot{\mathbf{p}}_c \\ \dot{\mathbf{v}}_c \\ \dot{\boldsymbol{\Phi}} \\ \dot{\boldsymbol{\omega}} \end{bmatrix} = \begin{bmatrix} \mathbf{v}_c \\ 1/m \sum_{i=1}^4 \delta_i \mathbf{f}_i + \mathbf{g} \\ \mathbf{E}'^{-1}(\boldsymbol{\Phi})\boldsymbol{\omega} \\ -\mathbf{I}^{-1}(\boldsymbol{\omega} \times \mathbf{I}\boldsymbol{\omega}) + \sum_{i=1}^4 \delta_i \mathbf{I}^{-1} \mathbf{p}_{cf,i} \times \mathbf{f}_i \end{bmatrix} \quad (2.8)$$

$$(2.9)$$

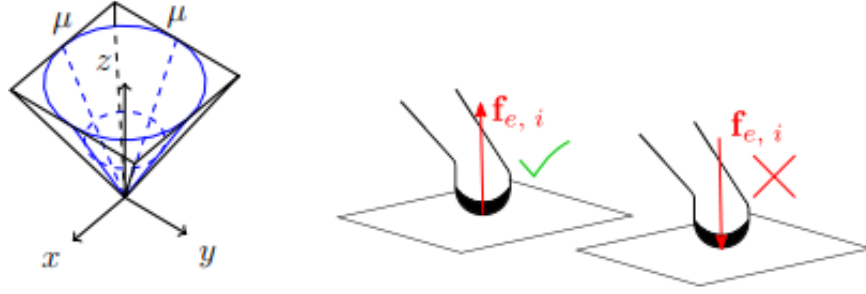
where \mathbf{v}_c is the CoM velocity of the robot. The robot base orientation is represented by the sequence of Z -Y -X Euler angles $\boldsymbol{\Phi} = (\phi, \theta, \psi)$ i.e., roll, pitch and yaw, respectively. The relation between the Euler Angles rates and the angular velocity is analyzed in Appendix A.

It is defined the state and control vectors as $\mathbf{x} = (\mathbf{p}_c, \mathbf{v}_c, \boldsymbol{\Phi}, \boldsymbol{\omega})$, and $\mathbf{u} = (\mathbf{f}_1, \dots, \mathbf{f}_4)$

2.2. Linearized friction cone constraint

In order to solve the dynamic problem it is necessary to guarantee the stability of the quadruped, namely, the non-slipping condition of the feet. Assuming an ideal foot with infinite stiffness (no deformation hypothesis) and an infinitely hard terrain (no penetration hypothesis), the force exchanged by a single foot with the ground is composed by tangential components ([x y] plane) and a normal component (along z direction). The

robot can exert forces at the contact points by actuating the hip and knee joints to generate a torque at its joints and then a force at the end points of its limbs. To include the friction constraints, a linearized Coulomb friction cone model is considered (shown in Fig. 2.3a). In this approximated model, the friction is assumed to be equal on both x and y directions.



(a) Friction cone approximation. Coulomb friction cone (blue) and its pyramidal approximation (black)

(b) Visualization of the unilaterality constraint. The forces depicted are GRFs, which means that are the forces that the ground exert on the robot.

Figure 2.3: The two dynamic constraints necessary to include in the problem formulation to guarantee the non-slipping condition of the contacting feet

The constraint can be translated with the following set of equations and inequalities:

$$\begin{cases} \mathbf{f} \leq \mathbf{f}_{max} \\ |f_x| \leq \mu f_z \\ |f_y| \leq \mu f_z \\ f_z \geq 0 \end{cases} \quad (2.10)$$

that can be summarized into a set of linear constraints such as:

$$\underline{\mathbf{d}} < \mathbf{Kf} < \bar{\mathbf{d}} \quad (2.11)$$

where

$$\mathbf{K} = \begin{bmatrix} (-\mu\mathbf{s} + \mathbf{t}_1)^T \\ (-\mu\mathbf{s} + \mathbf{t}_1)^T \\ (\mu\mathbf{s} + \mathbf{t}_2)^T \\ (\mu\mathbf{s} + \mathbf{t}_2)^T \\ \mathbf{n}_i^T \end{bmatrix} \quad \text{and} \quad \underline{\mathbf{d}} = \begin{bmatrix} -\infty \\ -\infty \\ 0 \\ 0 \\ f_{min} \end{bmatrix} \quad \text{and} \quad \bar{\mathbf{d}} = \begin{bmatrix} 0 \\ 0 \\ \infty \\ \infty \\ f_{max} \end{bmatrix} \quad (2.12)$$

being \mathbf{s} the surface normal and \mathbf{t}_i the tangential directions. f_{min} and f_{max} are, instead, design parameters for the robot, being the minimum and maximum reaction forces tolerated by the feet.

2.3. Trot gait

Another important configuration that must be considered while addressing the problem of modelling the movement of a quadruped is the gait. The gait of a robot quadruped refers to the specific pattern or sequence of leg movements it uses to move. There are several types of gaits that a robot quadruped can employ, each with its own advantages and disadvantages in terms of stability, speed, energy efficiency, and adaptability to different terrains.

Since the strong relationship with the animal world, several behaviours from different animals have been studied to fulfill the specific needs of robots in terms of linear velocity, stability and energy efficiency.

Concerning quadruped robots the most common gait is the trot. It is one of the most stable gaits[16] as the COM is always at the midpoint between the feet on the ground. A peculiarity of this gait is that with the limbs leaving and touching the ground at the same time, the robot's body tends to have a slight bounce, dropping between beats and bouncing back up as the next contact with the ground occurs[3].

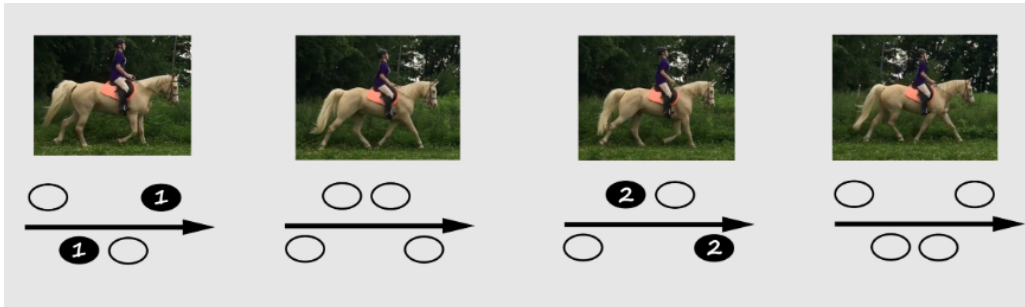


Figure 2.4: Foothold sequence of a horse trotting

3 | Control scheme

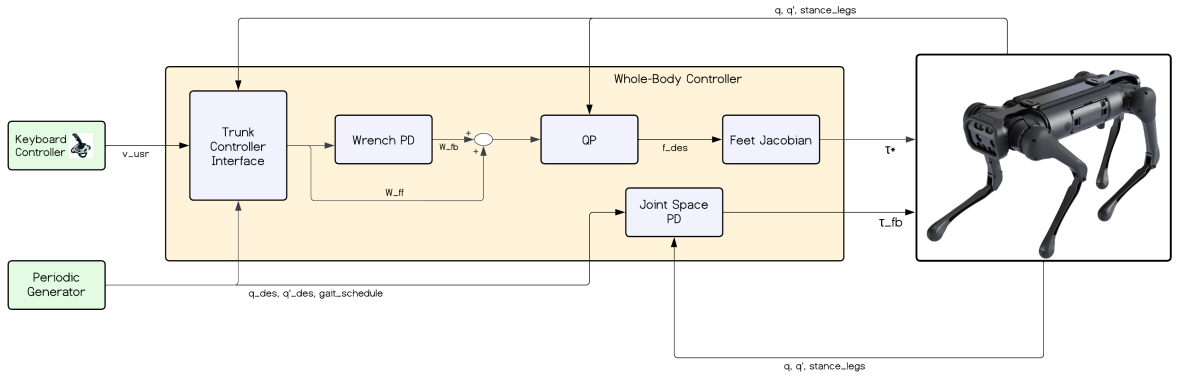


Figure 3.1: The proposed control scheme

3.1. Reference Generator

The used reference generator[19] is based on heuristics and it takes as inputs:

- The user commanded longitudinal and lateral CoM velocity ${}_{\mathcal{H}}\mathbf{v}_c^{\text{usr}} \in \mathbb{R}^2$ in the horizontal frame \mathcal{H}
- User commanded heading velocity $\omega_z^{\text{usr}} \in \mathbb{R}$
- Current pose of the robot (\mathbf{p}_c, Φ)
- Current feet position of the robot $\mathbf{p}_f \in \mathbb{R}^{12}$

and outputs:

- States $\mathbf{x}^{\text{ref}} \in \mathbb{R}^{n_x \times (N+1)}$
- sequence of the contact status $\boldsymbol{\delta} (\in \mathbb{R}^{4 \times N})$
- the sequence of the foot locations $\mathbf{p}_f (\in \mathbb{R}^{12 \times N})$

First the X-Y components of the total velocity $\mathbf{v}_c^{\text{ref}} \in \mathbb{R}^3$ are computed, which depend on

both $\mathbf{v}_c^{\text{usr}}$ and the X -Y components of the tangential velocity due to the heading velocity $\boldsymbol{\omega}^{\text{usr}} = (0, 0, \omega_z^{\text{usr}})$.

$$\mathbf{v}_{c,(x,y)}^{\text{ref}} = \mathbf{v}_c^{\text{usr}} + (\boldsymbol{\omega}^{\text{usr}} \times \mathbf{p}_c^{\text{ref}})_{(x,y)} \quad (3.1)$$

The X-Y CoM position $\mathbf{p}_{c,(x,y)}^{\text{ref}}$ is obtained by integrating the $\mathbf{v}_{c,(x,y)}^{\text{ref}}$ (in the world frame) with the explicit Euler scheme. The reference for the yaw $\dot{\psi}^{\text{usr}}$ is obtained by integrating the user defined yaw rate $\dot{\Phi}^{\text{usr}} = \mathbf{E}^{-1}(\dot{\Phi}^{\text{ref}})\boldsymbol{\omega}^{\text{usr}}$ (see Appendix A for the transformation between angular velocity and Euler rates). The reference for angular velocity, instead, coincides with $\boldsymbol{\omega}^{\text{usr}}$.

The sequence of contact status $\boldsymbol{\delta}$ and of footholds are computed by the gait scheduler and robocentric stepping strategy, respectively.

The gait scheduler is logically decoupled from the reference trajectory generation and determines if a leg is either in swing or in stance ($\boldsymbol{\delta}_i$) at each time instance for the entire gait cycle. Every time the reference generator is called, it extracts N points from the gait schedule starting from an index called gait counter. It keeps memory of the index of the gait schedule achieved by the previous call of the reference generator. The synchronization between the first point of a contact sequence $\boldsymbol{\delta}_{ik}$ computed by the reference generator and the actual contact state of the robot avoids the reference generator to compute a zero reference force while the leg is in stance and vice-versa

The choice of foothold is a key element in locomotion, since it deals with the kinematic limits of the robot. It is used an approach that continuously computes footholds consistent with the current position of the robot. To compute a foothold for a swinging leg i , it is considered its hip position \mathbf{h}_i instead of using the foot position at the moment of lift-off. In this way a disturbance acting on the robot or a tracking error occurred during a swing can be recovered in the following swing. For leg i , dropping the index to simplify the notation and defining the lift-off trigger as $l_k^{\text{tr}} = \delta_k \wedge \bar{\delta}_{k+1}$, the foot position is computed as:

$$\mathbf{p}_{f_{k+1}} = \begin{cases} \mathbf{p}_{f_k}^{\text{td}} & l_k^{\text{tr}} = 1 \\ \mathbf{p}_{f_k} & l_k^{\text{tr}} = 0 \end{cases} \quad (3.2)$$

The lift-off condition $l^{\text{tr}} = 1$ at instance k , \mathbf{p}_f is set equal to the touchdown point \mathbf{p}_f^{td} and it is kept constant until the next lift-off event occurs. The X-Y component of the touchdown point is given by:

$$\mathbf{p}_{f_k,(x,y)}^{\text{td}} = \mathbf{h}_k + \alpha T_{\text{sw}}^{\text{d}} (\mathbf{v}_c^{\text{usr}} + (\boldsymbol{\omega}^{\text{usr}} \times \mathbf{p}_{\text{bh}})_{(x,y)}) \quad (3.3)$$

The second term in 3.3 represents the step length which is computed with respect to the hip instead of the previous foot location. Parameter α is an empirically chosen scaling factor. Parameter T_{sw}^{d} is the default swing duration computed starting from and duty-factors and step frequency such as:

$$T_{\text{sw}}^{\text{d}} = (1 - D_f) / f_{\text{step}} \quad (3.4)$$

where D_f is the duty factor (the fraction of locomotion cycle that each leg spends in contact with the ground) and f_{step} being the frequency of the step. The distance between hip and center of the base is denoted by $\mathbf{p}_{\text{bh}} \in \mathbb{R}^3$

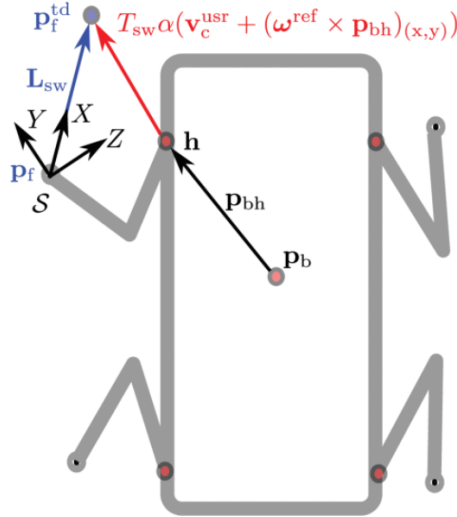


Figure 3.2: Representation of the robocentric stepping strategy and of the Swing Frame, located at the lift-off point

As shown in Fig. 3.1, then the reference generator outputs the desired joint position and velocity computed using inverse kinematics on the previously computed desired foothold position.

3.2. Whole-Body Controller

In this section it is described the WBC that tracks planned trajectories. The WBC first computes the feedback \mathbf{W}_{fb} and feedforward \mathbf{W}_{ff} wrench in world frame from the planned

trajectories and then finds the optimal set of GRFs \mathbf{f}^{ref} for the contacting feet to track the wrench. Finally, the WBC maps the GRFs into the joint torques $\boldsymbol{\tau}^*$. This joint torque along with low-impedance feedback torque $\boldsymbol{\tau}_{\text{fb}}$ results in the total torque $\boldsymbol{\tau}_{\text{d}}$ required by the low-level joint torque control block.

3.2.1. Trunk Controller

The goal of this module is to compute and track the desired wrench in the world frame ($\mathbf{W} = \mathbf{W}_{\text{fb}} + \mathbf{W}_{\text{ff}}$) outputting the previously mentioned set of joint torques $\boldsymbol{\tau}^*$. This module is run at 250Hz.

The design of the controller is based on the following assumptions. First, we assume that for the contacting feet Coriolis and centrifugal forces are negligible[9]: this is reasonable because in this context the robot moves slowly. Second, since most of the robot's mass is located in its base, we approximate the CoM and the average angular velocity of the whole robot with the CoM and the average angular velocity of the base. Third, since the platform has nearly point-like feet, we assume that it cannot generate moments at the contacts. Fourth, it is assumed that the GRFs \mathbf{f} are the only external forces acting on the system. Under these assumptions the linear acceleration of the CoM $\dot{\mathbf{v}}_c$ and the angular acceleration of the base $\dot{\boldsymbol{\omega}}_b$ can be expressed as functions of the c GRFs (i.e., where c is the number of stance feet):

$$m(\dot{\mathbf{v}}_c + \mathbf{G}) = \sum_{i=1}^c \mathbf{f}_i \quad (3.5)$$

$$\mathbf{I}_G \dot{\boldsymbol{\omega}}_b \simeq \sum_{i=1}^c (\mathbf{p}_{\text{com},i} \times \mathbf{f}_i), \quad (3.6)$$

where m is the total robot's mass, \mathbf{G} is the gravity acceleration vector, \mathbf{I}_G is the centroidal rotational inertia[18], $\mathbf{p}_{\text{com},i}$ is a vector going from the CoM to the position of the foot defined in an inertial world frame as in Fig. 3.3

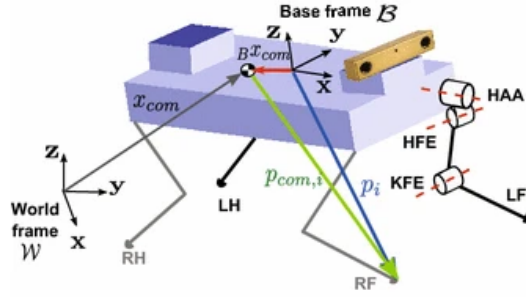


Figure 3.3: Used frame references in the controller and $\mathbf{p}_{com,i}$

Moving on to the actual controller, firstly the desired feedforward wrench in the base frame (${}_{\mathbf{b}}\mathbf{W}_{ff}$) is computed as follows:

$${}_{\mathbf{b}}\mathbf{W}_{ff} = -\alpha_{\mathbf{b}}\mathbf{G}m \quad (3.7)$$

where ${}_{\mathbf{b}}\mathbf{G} \in \mathbb{R}^6$ is the gravitational acceleration matrix in the base frame, m is the robot mass and α is an arbitrarily chosen spline factor. The aim is to send this wrench in feedforward to the controller to perform disturbance compensation. The gravitational force is accounted as a disturb and its action is predicted based on the knowledge of how it is directed on the base frame. In this way the WBC is performing gravity compensation, this process is held by the trunk controller interface shown in Fig. 3.1

The feedback wrench is then computed with respect to the horizontal frame. A horizontal frame is a reference frame whose xy plane is always horizontal (i.e. orthogonal to the gravity vector \vec{g}), such that the projection of its x axis on the horizontal plane is parallel to the same projection of the x axis of the robot (that is, the horizontal frame has the same yaw angle as the robot, with respect to the world frame). A horizontal frame can be attached to the robot (it is then said to be floating), or fixed somewhere in the environment, as illustrated in Fig. 3.4.

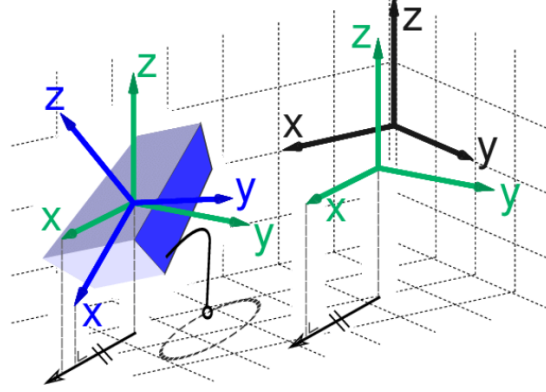


Figure 3.4: Horizontal reference frames (in green) and the robot frame (in blue-the parallelepiped represents the robot trunk; horizontal frames share the same yaw angle with respect to the world reference frame (in black)

After rotating the wrench vector with an adequate rotating matrix, the feedback desired wrench in the horizontal frame (${}_{\mathcal{H}}\mathbf{W}_{fb}$) is computed as the following by the wrench PD controller shown in Fig. 3.1.

$${}_{\mathcal{H}}\mathbf{W}_{fb} = \mathbf{K}_w \begin{bmatrix} \boldsymbol{\theta}_c^{\text{ref}} - \boldsymbol{\theta}_c \\ \mathbf{p}_c^{\text{ref}} - \mathbf{p}_c \end{bmatrix} + \mathbf{D}_w \begin{bmatrix} \mathbf{v}_c^{\text{ref}} - \mathbf{v}_c \\ \boldsymbol{\omega}_b^{\text{ref}} - \boldsymbol{\omega}_b \end{bmatrix} \quad (3.8)$$

where matrices $\mathbf{K}_w \in \mathbb{R}^{6 \times 6}$ and $\mathbf{D}_w \in \mathbb{R}^{6 \times 6}$ are diagonal matrices containing the proportional and derivative gains and they can be interpreted as impedances; the desired position and orientation of the base ($\boldsymbol{\theta}_c^{\text{ref}}$ and $\mathbf{p}_c^{\text{ref}}$), the desired velocities, linear and angular ($\mathbf{v}_c^{\text{ref}}$ and $\boldsymbol{\omega}_b^{\text{ref}}$) are provided from the reference generator 3.1 and expressed with respect to the horizontal frame \mathcal{H} . The actual position and velocities ($\boldsymbol{\theta}_c$, \mathbf{p}_c , \mathbf{v}_c and $\boldsymbol{\omega}_b$) are feedbacked from the plant and are also expressed with respect to the horizontal frame \mathcal{H} .

Given the desired value of the wrench in the world frame (\mathbf{W}_{fb}) obtained rotating the previously computed ${}_{\mathcal{H}}\mathbf{W}_{fb}$, we want to compute the desired GRFs \mathbf{f}_i . The matrix representation:

$$\underbrace{\begin{bmatrix} \delta_1 \mathbf{I} & \dots & \delta_4 \mathbf{I} \\ \delta_1 [\mathbf{p}_{cf,1} \times] & \dots & \delta_4 [\mathbf{p}_{cf,4} \times] \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} \mathbf{f}_1 \\ \vdots \\ \mathbf{f}_4 \end{bmatrix}}_{\mathbf{f}} = \underbrace{\mathbf{W}_{fb} + \mathbf{W}_{ff}}_{\mathbf{b}} \quad (3.9)$$

is derived from the simplified SRBD model assumed earlier and allows to map the desired

wrenches into GRFs. To compute the desired GRFs vector \mathbf{f}^d the following QP it is solved:

$$\mathbf{f}^d = \underset{\mathbf{f}}{\operatorname{argmin}} \quad \|\mathbf{A}\mathbf{f} - \mathbf{b}\|_{\mathbf{S}}^2 + \alpha \mathbf{f}^T \mathbf{T} \mathbf{f} \quad (3.10)$$

$$\text{s.t.} \quad \underline{\mathbf{d}} \leq \mathbf{K}\mathbf{f} \leq \bar{\mathbf{d}} \quad (3.11)$$

Matrices $\mathbf{S} \in \mathbb{S}_+^6$ and $\mathbf{T} \in \mathbb{S}_+^{12}$ are positive-definite weight matrices. Inequality 3.11 encodes the friction cone and unilateral constraints similarly as stated in Chapter 2.

The GRFs vector \mathbf{f} must be mapped into joint torques $\boldsymbol{\tau}^*$. This is done by exploiting the joint dynamics:

$$\boldsymbol{\tau}^* = -\mathbf{J}(\mathbf{q})^T \mathbf{f} + \mathbf{h}(\mathbf{q}, \dot{\mathbf{q}}) \quad (3.12)$$

where $\mathbf{J}(\mathbf{q}) \in \mathbb{R}^{n_u \times n}$ is the contact Jacobian and $\mathbf{h}(\mathbf{q}, \dot{\mathbf{q}})$ the vector of gravity/Coriolis terms in the leg joint dynamics, computed through inverse dynamics. The number of joints is denoted by n . The joint acceleration contribution it is neglected because it is very small with respect to the other terms.

3.2.2. Joint-Space PD

A 1 kHz Joint-Space PD (shown in Fig. 3.1 is put in cascade with the trunk controller before sending the desired torques to the robot. In this way, the desired trajectories of the swinging legs can be tracked and it is increased the robustness in case a foot loses contact with the ground. To compute the joint trajectories, inverse kinematics is required which in turn needs the swing trajectory \mathbf{p}_f^{sw} . A swing frame \mathcal{S} , whose X-axis is aligned with the vector that links lift-off and touchdown point (\mathbf{L}_{sw}), Y-axis is perpendicular to the X-axis of the swing frame and to the Z-axis of the world frame. Finally the Z-axis is such that \mathcal{S} is a counter-clockwise coordinate system. The origin of the swing frame \mathcal{S} coincides with the lift-off point. In this way the swing trajectory lies on the X-Z plane and we shape it as a semi-ellipse with \mathbf{L}_{sw} and H_{sw} as lengths of the axes:

$${}^s\mathbf{p}_f^{sw} = \begin{bmatrix} \frac{\mathbf{L}_{sw}}{2}(1 - \cos(\pi f_{sw} t_{sw})) \\ 0.0 \\ H_{sw} \sin(\pi f_{sw} t_{sw}) \end{bmatrix} \quad (3.13)$$

where t_{sw} is the time elapsed from the beginning of a swing and $f_s = 1/T_{sw}^d$ is the swing

frequency. After mapping \mathbf{p}_f^{sw} and $\dot{\mathbf{p}}_f^{sw}$ in world frame, \mathbf{q} and $\dot{\mathbf{q}}$ are evaluated via inverse kinematics.

The design of this controller is simple as it does not take into account gravity compensation and joint coupling. [14] Considering a single leg and a PD control without gravity compensation:

$$\boldsymbol{\tau} = \mathbf{K}_j(\mathbf{q}^{\text{ref}} - \mathbf{q}) - \mathbf{D}_j\dot{\mathbf{q}} \quad (3.14)$$

then the closed-loop dynamic equation becomes

$$\mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \boldsymbol{\tau}_g(\mathbf{q}) + \mathbf{D}_j\dot{\mathbf{q}} - \mathbf{K}_j\mathbf{e}_q = \mathbf{0} \quad (3.15)$$

Considering the positive definite function

$$V = \frac{1}{2}\dot{\mathbf{q}}^T\mathbf{H}(\mathbf{q})\dot{\mathbf{q}} + \frac{1}{2}\mathbf{e}_q^T\mathbf{D}_j\mathbf{e}_q + U(\mathbf{q}) + U_0 \quad (3.16)$$

where $U(\mathbf{q})$ denotes the potential energy with the relation of $\frac{\partial U(\mathbf{q})}{\partial \mathbf{q}} = \boldsymbol{\tau}_g(\mathbf{q})$ and U_0 is a suitable constant. Taking time derivative of V along the closed-loop dynamics gives:

$$\dot{V} = -\dot{\mathbf{q}}^T\mathbf{D}_j\dot{\mathbf{q}} \leq -\lambda_{\min}\mathbf{D}_j\|\dot{\mathbf{q}}\|^2 \quad (3.17)$$

In this case, the control system must be stable in the sense of Lyapunov, but it can not conclude that the regulation error will converge to zero by LaSalle's theorem. Actually, the system precision (the size of the regulation error vector) will depend on the size of the gain matrix \mathbf{K}_p in the following form:

$$\|\mathbf{e}_q\| \leq \|\mathbf{K}_j^{-1}\| g_0 \quad (3.18)$$

where g_0 is a constant bounding the torques. Hence, the regulation error can be arbitrarily reduced by increasing \mathbf{K}_j ; nevertheless, measurement noise and other unmodelled dynamics, such as actuator friction, will limit the use of high gains in practice.

The choice of this controller has been chosen, therefore, because it is not needed to achieve zero regulation error, also due to the fact that the role of gravity compensation it is led by the whole body controller.

4 | Implementation

As stated in the previous sections, the goal of this work is to test the proposed controller in a simulation that can be as close as possible to a real experiment, using real hardware. Here, it is described the implementation architecture and the libraries used to test and analyze the simulations. Extensive work has been made in this context, carefully exploring and analyzing different options.

The implementation work was never made having a single robot or a single model in mind. The main goal of this implementation framework is the ease of switching between different robot models and controllers. This will be the focus of the solutions adopted, making necessary very few steps to change to different robots, even not quadrupeds, assuming that the all the hypothesis made in Chapter 2 and Chapter 3 are still valid.

However in these specific cases, two robots have been explored, Unitree Aliengo and Unitree Go1.

Description	Data
RobotWeight	12 <i>kg</i>
Dimensions	645 x 280 x 400 <i>mm</i> (standing)
MaxSpeed	13.3 <i>km/h</i>
Number of Joints	12
MaxJointTorque	35.5 <i>NM</i>
MaxJointVelocity	26.5 <i>rad/s</i>

Table 4.1: Aliengo robot specifications

Description	Data
RobotWeight	12 <i>kg</i>
Dimensions	645 x 280 x 400 <i>mm</i> (standing)
MaxSpeed	13.3 <i>km/h</i>
Number of Joints	12
MaxJointTorque	35.5 <i>NM</i>
MaxJointVelocity	26.5 <i>rad/s</i>

Table 4.2: Go1 robot specifications

4.1. The DLS framework

DLS is a clustered and ROS-like framework where all the software related to simulation, analysis, robot model, control and reference generator is deployed. The different kind of layers that can be loaded are:

- Robot model
- Console layer
- Log layer
- Control layer
- Generator layer
- Hardware layer
- Estimation layer

They are divided into `layer` (loaded with the `-l` option) and `model` (loaded with the `-r` option) and can be loaded in the framework with the `dls` command as:

```
dls -llog -raliengo -lcontrol -lhardware -llog -lestimation
```

Each layer can be now be invoked to load the relative apps belonging to it. As it will be explained later, they are developed based on templates which define the parent classes and the necessary configuration to interact with the DDS. Apps are written in C++ and loaded after being built and installed through CMake.

To implement the full simulation environment, different apps have been developed and used:

- `AliengoLib` - Robot model - Contains the core of the model, its URDF and the library containing all the Rigid Body Dynamics algorithms (more about this in Section 4.3).
- `trunk_controller` - Control layer - It is part of the WBC, contains the wrench PD and the QP problem, runs at 250Hz and publishes on the `desired_torque` topic.
- `pid` - Control layer - It is actually a PD controller and runs at 1kHz, publishes on the `desired_torque` topic.
- `periodic` - Generator layer - Given a gait computes the joint trajectory, runs at 250 Hz and publishes on the `trajectory_generator` topic.
- `keyboard` - Generator layer - Takes velocity input through keyboard command.
- `gazebo_sim` - Hardware layer - Load the Gazebo simulation software.

In addition to this, the DLS framework manages the Data Distribution Service (DDS) middleware. Each app can read or publish in different topics. As a general guideline, it can be said that apps belonging to the generator layer reads from the `blindState` topic, that contains actual joint position and velocities, and publishes the new desired joint position and velocity on the `trajectoryGeneration` topic. The control layer reads from the `trajectoryGeneration` topic and publishes on the `desiredTorques` topic which is read from Gazebo and used to run the simulation. Finally, Gazebo publishes the actual joint values to the `blindState` topic and elaborated values of the actual state of the robot (such as trunk position, orientation and velocity) to the `baseState` topic.

Despite being these the standard topics used, it is as easy as creating a new message type to create and use a new topic. For instance, for debug and performances analysis issue, the `trunk_controller` publishes on the `trunk_controller_debug` topic the values of the desired GRFs and the actual forces acting on each foot estimated using inverse dynamics algorithms.

Before deep diving into the technical details of the apps, the following is an example overview of the commands needed to successfully run a simulation, starting from the load of the Gazebo environment up to starting the motion of the robot.

```
loadHardware gazebo_sim
loadModel aliengo
loadGenerator periodic
periodic::activatePeriodic
loadController pid
```

```
pid::activateController
periodic::goHome
freezeBase
loadGenerator keyboard
keyboard::activateKeyboard
loadController trunk_controller
trunk_controller::activateController
keyboardControl (it will open the keyboard console in the log terminal)
periodic::startMotion
stopMotion all
shutdown all
```

4.1.1. Periodic App

In DLS2 a `periodicApp` is a class that will contain main functions run periodically by the framework with a frequency rate defined in a configuration file. It is the case of the apps belonging to the Control, Generator and Estimation layer.

The creation of an instance of a `periodicApp` of any kind starts with the load of the robot library. This is done by the invocation of the `robotlib` function `openRobot` (more about `robotlib` in Section 4.2). The name of the robot is directly taken from the model that must have been previously loaded in the framework (an example of the robot loading is present in the previous section). This process is needed because now the app can call the robot library and use its modules. This becomes crucial when an app needs to use important functions such as inverse kinematics, inverse dynamics, inverse kinematics or direct kinematics. In this way the app will remain blind to the actual robot that it is controlling and therefore it can be coded in a general way just once. The robot specific parameters of the app, such as the PID gains, can be stored in a robot tailored configuration file and loaded based on the name of the robot loaded at the moment in the framework.

A `periodicApp` defines virtual functions that must be overridden by the child classes to achieve runtime polymorphism. This type of polymorphism is achieved by function overriding. Late binding and dynamic polymorphism are other names for runtime polymorphism. The function call is resolved at runtime in runtime polymorphism. In contrast, with compile time polymorphism, the compiler determines which function call to bind to the object after deducing it at runtime. These virtual functions defined in the class should be considered as the mandatory functions that must be coded.

The most significant virtual function is the `run` function, that is the one actually run at the defined frequency rate and must contain all the significant functions of the app. To better manage different functions inside `run` come very helpful the use of tokens that can be modified by the user via command line interface. As a general rule, the `run` function should load the desired parameters but should not do anything unless it reads the `active` token set to `True`. These tokens can also be used to edit on-the-run the parameters of the app, i.e. it can be coded a command to change the K_p values of a PID controller while the app is running.

```
trunk_controller::
* activateController
* ls
* pause
* setKdGain
* setKdGainByIndex
* setKpGain
* setKpGainByIndex
* shutdown
* where
```

(a) `trunk_controller` app commands before the activation

```
periodic::
* continue
* deactivatePeriodic
* goFold
* goHome
* init
* setKinematicAdjustment
* shutdown
* startMotion
* where
```

(b) `periodic` app commands after the activation

Figure 4.1: Examples of the command line interface commands available on two apps loaded in the DLS2 framework

4.1.2. Robot library

The robot library is loaded in the model layer of the DLS2 framework and it is specifically coded for a robot. In the main source file, a robot instance is created using the URDF file that must have been previously installed in the framework. The creation function reads the URDF and parse the different links into `robotLib` classes such as legs, arms and trunk.

The rest of the class serves as an interface between the RBD algorithm library (whatever it is) and the framework. As an interface, helpful functions can be coded. For instance, an inverse dynamics algorithm provided by the library can be exploited in several functions depending on the need, of the kind of inputs or of the kind of outputs. This could be particularly useful to directly code functions to compute gravity compensation at the wrench or gravity compensation at the joint torques. These functions will remain in the interface and therefore the other layers can just use them without being necessary any parsing of the data.

In the case of the use of Pinocchio (more about in in Section 4.3.2), the source file will

load the data and the model and the RBD algorithm functions will be invoked through the library. Instead, in the case of the use of RobCoGen (more about it in Section 4.3.1), this is also the place where all the code produced by the library will be stored. In this case it is very easy to substitute the model data with parameters because there is a direct access to the file. If the total mass of the robot depends on the hardware it is equipped with (such as cameras or sensors), changing the robot mass of the model is as easy as substituting the value with a variable without accessing again the URDF. Nevertheless, it must be said that is as risky procedure: other parameters, such as the inertia parameters, are strictly related into each other. It would be preferable to prepare several URDF versions based on the robot configuration.

4.1.3. Gazebo hardware

Gazebo is the simulation software used in the framework to perform the experiments. Its main function (as seen in the introduction of Chapter 4) is to load the robot model. This functions searches for a previously installed URDF file and loads it in the environment.

Despite being Gazebo actually a software, it is loaded in DLS2 with the hardware layer because this is the layer that should actually manage all the interactions with the real robot, the actual hardware. Namely, this means managing the hardware inputs (torque control) and outputs (reading the data from the sensors). In Gazebo, this can be done with custom made plugins that are loaded among with the URDF file of the robot.

The two custom made plugins loaded are `gazebo_base_state` and `gazebo_hw_sim`. The first just performs some direct and inverse kinematics to directly publish on the relative topic data about the trunk position, orientation and velocity (instead of publishing the blind state data which only takes into account joints).

The second, `gazebo_hw_sim` is an interface able to put in communication the robot and the hardware or, in this case, the robot and Gazebo. For instance, it translates the torque commands published on the topic in commands that Gazebo can understand and actuate.

4.2. Robotlib

As already stated before, the proposed locomotion control system is based on a modular and generic software framework that can be applied on robots of different morphologies, such as quadrupeds and hexapods. This software abstraction is achieved by Robotlib[6], a robot software interface that allows the same locomotion framework to run on many different robots. Robotlib provides a templated robot morphology that defines the hierar-

chical structure of joints and links. Additionally, it provides virtual and utility functions for the robot kinematics, dynamics, and Jacobians. Robot-specific libraries inherit the base robot interface to implement the particular morphology, kinematics, and dynamics of each different robot.

There are several advantages of using an architecture abstraction layer like Robotlib. For example, with Robotlib, robot-specific structures are hidden from the controllers and state estimators, making them more modular and easier to implement. The structure allows controllers and state estimators to be written only once (assuming that for the assumptions made are valid for all the kind of robots), and then the framework can dynamically load different robots. The abstraction layer also provides an easy way to switch backend libraries (as it is described in the Section 4.3) that compute the kinematics and dynamics without affecting the rest of the framework. Robotlib is written in C++17 to be fast and portable. It is compatible with the most adopted robotics libraries and is real-time safe.

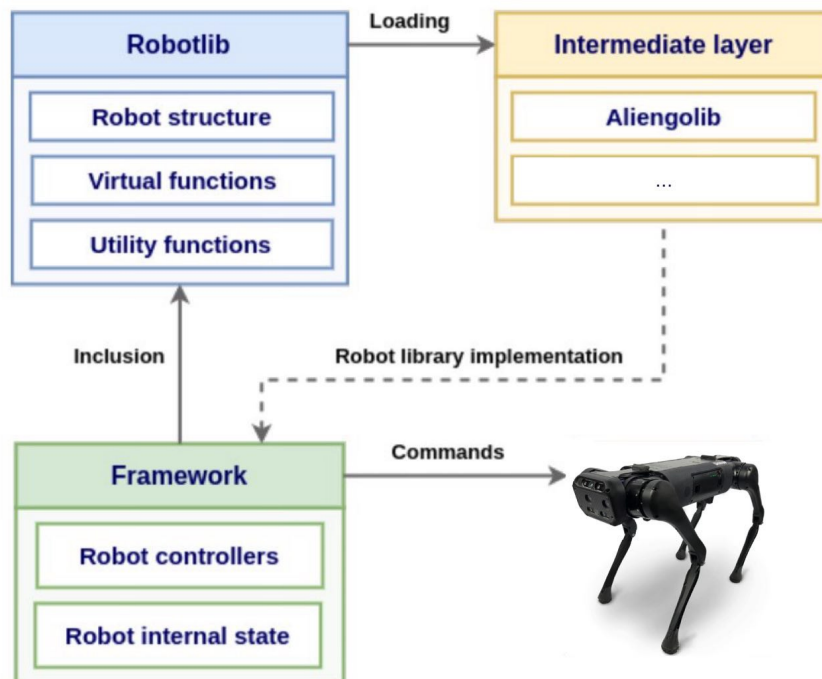


Figure 4.2: Robotlib interaction with the other modules of the proposed locomotion framework

4.3. Rigid Body Dynamics algorithms

As stated in the previous section, RobotLib provides the robot main structure. It is then crucial to use a library able to load certain parameters of the robot (such as frames,

mass for each link and inertias) and provide all the necessary functions of Rigid Body Dynamics.

The chose of this library happens to be crucial because it is the only time the actual robot parameters are inserted, the easier is to model a new robot the easier is to extend this framework to new hardware.

4.3.1. RobCoGen

The idea behind RobCoGen[10] is to relieve the user from the manual development of code which is crucial yet complex, necessary for most applications yet rarely the actual focus of the research: namely, state-of-the-art rigid body dynamics algorithms, coordinate transformation matrices and Jacobian matrices (i.e. kinematics algorithms).

RobCoGen allows its end users to deal only with high level information, the minimum amount required to define an instance of a problem, and then leave a computer program to deal with implementation details, by means of code generation. RobCoGen is based on three external Domain Specific Languages (Kinematics-DSL, Motion-DSL, Transforms-DSL). These DSLs can be thought of as specification languages, for the compact representation of coordinate transforms and the structure of articulated robots. The DSLs and generators that address the problem of coordinate transforms (the Motion-DSL and Transforms-DSL) constitute in fact an independent software. That is, one can generate the implementation of arbitrary coordinate transforms without any robot model, since coordinate transforms is an independent domain.

For this work, it is used RobCoGen to generate RBD code for the robot Aliengo. The input given to the library is composed by two seperate DSL files, one, `aliengo.dtdsl` containing the definitions of frames, transformations and jacobians and one, `aliengo.kinds1` containing the description of each joint and link with the respective parameters such as mass and inertias. The inertias and mass values are to be considered nominal to an Aliengo robot and have been provided by the DLS laboratory of the IIT. These values could also be taken from the robot's constructor URDF.

```

link LF_upperleg {
  id = 2
  inertia_params {
    mass = 5.877927
    CoM = (0.107987, -0.011459, -0.002121)
    Ix = 0.017230
    Iy = 0.144308
    Iz = 0.148583
    Ixy = -0.005410
    Ixz = 0.002550
    Iyz = -0.000097
  }
  children {
    LF_lowerleg via LF_KFE
  }
  frames {
    LF_upperlegCOM {
      translation = (0.107987, -0.011459, -0.002121)
      rotation = (0.0, 0.0, 0.0)
    }
  }
}

```

(a) Snippet of `aliengo.kinds1`

```

Robot Aliengo

Frames {
  fr_trunk
  , fr_LF_hip
  , fr_RF_hip
  , fr_LH_hip
  , fr_RH_hip
  , fr_LF_leg
  ...
}

Transforms {
  //for com jacobian computation
  base=fr_trunk, target=LF_hipassemblyCOM
  base=fr_trunk, target=RF_hipassemblyCOM
  ...
}

Jacobians {
  base=fr_trunk, target=LF_foot
  ...
}

```

(b) Snippet of `aliengo.dtds1`

Figure 4.3: Code snippets of the code used to generate the Aliengo RobCoGen library

Regarding the core of the dynamics algorithms, it is used the Composite Rigid–Body algorithm for the computation of the joint space inertia matrix, the Recursive Newton–Euler algorithm for inverse dynamics and the Articulated–Body algorithm for forward dynamics, all in the formulation using spatial vectors algebra[8]. These algorithms are the most efficient ones for the purpose. A robot model in the form of a document of the KinematicsDSL, allows to resolve the dependency of the algorithms on the model itself. That is, the generated implementation of a dynamics algorithm is no longer parametrized with the robot model, but implicitly embed the information of a specific robot. The output is then a ready-to-use C++ library to be implemented in the aforementioned `AliengoLib`. It contains all the necessary functions coordinate transform, direct and inverse kinematics and dynamics.

4.3.2. Pinocchio

Pinocchio[5] is a direct alternative to RobCoGen. In this work has been explored the possibility of substituting RobCoGen with Pinocchio with its pros and cons.

The main difference with RobCoGen is the strict separation between model and data. By model it is meant the physical description of the robot, including kinematic and possibly inertial parameters defining its structure. This information is held by a dedicated class which, once created, is never modified by the algorithms of Pinocchio. By data, it is meant all values which are the result of a computation. Data vary according to the joint configuration, velocity, etc... of the system. It contains for instance the velocity and

the acceleration of each link. It also stores intermediate computations and final results of the algorithms in order to prevent dynamic memory allocation. The important data class can be created directly via URDF, making, having in mind the ease of creating robot libraries, a great advantage with respect to RobCoGen. Pinocchio also deals for the geometric model of the robot, which in this case will not be used because this will be dealt directly by the simulation software through the URDF.

The results are promising, the URDF file can be directly obtained through the robot's constructor and, given the data class created from it, the rest of the library is the same for each robot. In addition to that, Pinocchio's function can be directly accessed via API, excluding the need of generating the library code. The mean computation time of all the main algorithms (RNEA for inverse dynamics, ABA for forward dynamics and CRBA for mass matrix) are in the order of $1 \mu s$.

The idea behind the implementation of Pinocchio in the robot library source files would be to code the needed algorithms in an external file and create a personal library to be used with every robot. With respect to RobCoGen, in the source robot library it would be just necessary to load the Pinocchio model and then each function could be called. It should not be needed to have all the code in each of the robot libraries.

The following is an example of inverse Kinematics run with Pinocchio after loading the data class:

Algorithm 4.1 Inverse Kinematics

- 1: The parameters of the algorithm are defined:
 - normTHR* to define when the desired pose is reached
 - iterationsTHR* to avoid infinite loops
 - DT* conversion rate for integration
 - λ damping factor
 - 2: **while** *True* **do**
 - 3: Compute forward kinematics
 - 4: Compute the error between the desired pose and the current one
 - 5: **if** $\text{norm}(\text{err}) < \text{normTHR}$ **then**
 - 6: Break
 - 7: **end if**
 - 8: **if** $\text{iterations} > \text{iterationsTHR}$ **then**
 - 9: Break
 - 10: **end if**
 - 11: Compute the Jacobian in joint frame
 - 12: Compute the Jacobian of task
 - 13: Compute the tangent vector employing the damping pseudo-inverse
 - $$v = J^T(JJ^T + \lambda I)^{-1}$$
 - 14: Compute the evolution of the configuration integrating the tangent vector
 - 15: **end while**
-

5 | Simulations and results

These simulations were performed using the model of the Aliengo robot of mass $m = 87kg$. The CoM is computed considering the mass of the individual link of the robot and the actual position of the links' CoM. The position of the links' CoM in their local frame is obtained from their CAD models. The feedback gains used in the WBC are $\mathbf{K}_w = \text{diag}(150, 150, 150, 0, 0, 500)$ and $\mathbf{D}_w = \text{diag}(100, \dots, 100)$. For the solution of the QP problem the weights $\mathbf{S} = \text{diag}(10, 10, 10, 5, 5, 10)$ and $\mathbf{T}_\tau = \text{diag}(5, 1, 0.2)$ are chosen and finally $\mathbf{K}_j = \text{diag}(15, \dots, 15)$ and $\mathbf{D}_j = \text{diag}(6, 3, 3, \dots, 6, 3, 3)$ for the joint-space PD. The friction coefficient of the ground is not estimated, but it is set with a nominal value of $\mu = 0.4$, a conservative choice to be more robust with respect to the actual surface.

5.1. Tuning parameters

Regarding the choice of the tuning parameters for each of the components of the control scheme, this section describes the logic behind them.

5.1.1. Whole-Body Controller parameters

From the simulations it can be noted that GRFs are always close to the cone boundaries. This is expected because, due to the quasi-static motions, gravitational components (mainly vertical) dominates in the body wrench, and using a regularization that minimizes the norm of the torques or of the forces leads to solutions that are close to the cone boundaries (for the actual task). To improve robustness it would be preferable[20] to have a solution where forces are close to the cones' normals. This is equivalent to penalizing the norms of the feet's forces in frames that are aligned with the contacts' normals. To achieve this the following block-diagonal weight matrix:

$$\mathbf{T} = \begin{bmatrix} \mathbf{t}_0 \mathbf{T}_{n_0} \mathbf{t}_0^T & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \mathbf{t}_c \mathbf{T}_{n_c} \mathbf{t}_c^T \end{bmatrix}, \quad (5.1)$$

where $\mathbf{t}_i = \begin{bmatrix} \mathbf{t}_{1_i} & \mathbf{t}_{2_i} & \mathbf{n}_i \end{bmatrix}$ is a rotation matrix whose columns are the coordinate axis of a frame aligned with the contact surface i . The weight matrix for each stance leg i is \mathbf{W}_i , where w_{1_i} and w_{2_i} are the weights used to penalize the tangential forces in the x and y directions. However, this method assumes to know the coordinate axis of the surface \mathbf{n}_i that, for the purpose of this work, we assume unknown.

Therefore another approach has been evaluated. The joint torque limits proved to be an important issue during our experiments. The respect of the joint-torque limits can be achieved through either the cost function or the inequality constraints. Even though this allows constraint violations, the first method has been used because the second one was computationally too expensive. The regularization term $\mathbf{f}^T \mathbf{T} \mathbf{f}$ can be defined in order to penalize joint torques rather than GRFs. This can be achieved by knowing the relationship between feet forces and torques: $\boldsymbol{\tau}^* = -\mathbf{J}_c^T \mathbf{f}$. Therefore to minimize $\boldsymbol{\tau}^{*T} \mathbf{T}_\tau \boldsymbol{\tau}^*$, with $\mathbf{T}_\tau \in \mathbb{R}^{3c \times 3c}$ being a diagonal positive-definite matrix, it is set:

$$\mathbf{T} = \mathbf{J}_c \mathbf{S}^T \mathbf{T}_\tau \mathbf{S} \mathbf{J}_c^T \quad (5.2)$$

where $\mathbf{S} = [\mathbf{0}_{n \times 6} \quad \mathbf{I}_{n \times n}]$ is a selection matrix that selects the actuated DoFs. This results in implicitly minimizing the torques of the stance-legs' joints.

Reminding that the feedback wrench PD is designed having in mind an impedance controller, the problem is to tune the task-space stiffness and damping matrix. Since it is required a very stiff robot on the z axis, looking at the last 3 elements of \mathbf{K}_w , it is present a much bigger value on the third element. The damping matrix \mathbf{D}_w is chosen such that the closed-loop system is stable. In addition to that a remark must be made on the hypothesis of this model: it has been considered that all the mass of the robot is concentrated in the trunk, according to the controller a big wrench component on the rotational x axis would cause a loss of stability. Instead this is not true because the trunk is linked to the legs that have mass and inertia, a little movement on that axis will not cause a loss of stability. Therefore the stiffness values of the rotational axis should be less than expected.

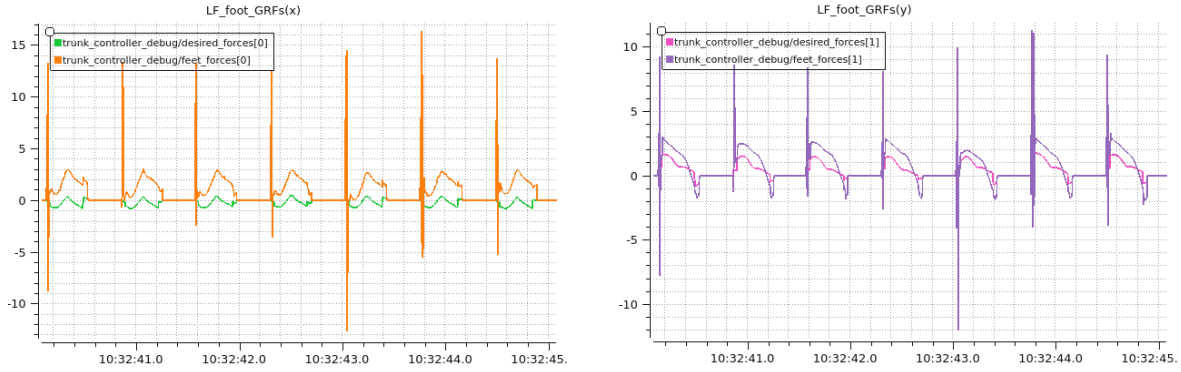
5.1.2. Joint-space PD tuning

The Joint-space PD tuning has been made in a simulation with the robot pinned in the air and no gravity. It has been used the Ziegler-Nichols method using the same parameters for all the joints, finding $\mathbf{K}_j = \text{diag}(15, \dots, 15)$ and $\mathbf{D}_j = \text{diag}(3, \dots, 3)$. Lately the derivative gain has been increased on the HAA joints because they are not extremely exploited in the motion and this improved the time response of those joints.

Notwithstanding the design of the regulator, the overshoot had a significant impact. The solution was found in a significant pre-filtering of the reference. This does not affect the open loop transfer function and therefore has no influence on the effect of the noises or stability. The filter was designed as a low-pass filter with a bandwidth of approximately 1 rad/s , obtained from a trade-off between the reduction of the overshoot and the consequent slowdown of the system.

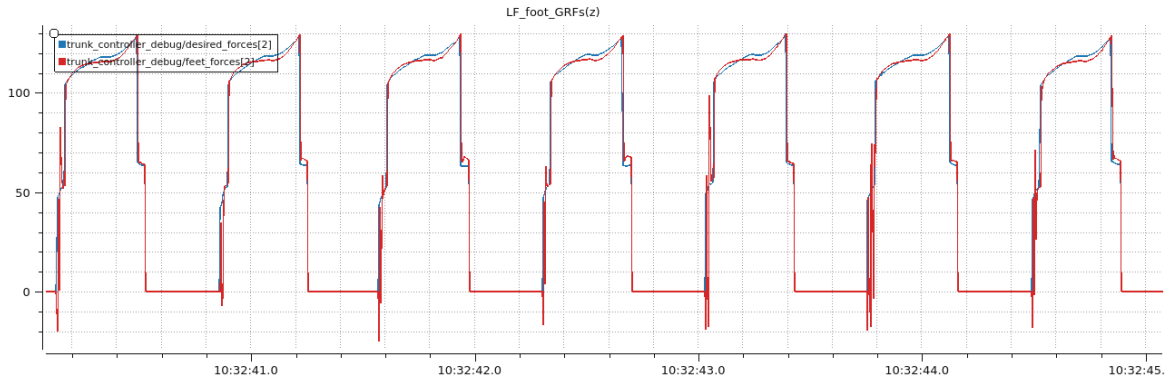
5.2. Walking

This is the nominal experiment. In a nominal situation (i.e. flat surface with friction coefficient $\mu = 1$) the Aliengo robot is controlled to move along the X-Y plane and also changing the heading velocity. This simulation has been used to understand the standard behaviour of the considered variables that have been taken as a benchmark to evaluate other simulations where there is an unexpected or a non-nominal situation.



(a) Desired GRF on the LF foot on the x axis (in green) and the actual reaction force

(b) Desired GRF on the LF foot on the y axis (in pink) and the actual reaction force



(c) Desired GRF on the LF foot on the z axis (in blue) and the actual reaction force

Figure 5.1: The desired GRFs and the actual ones during a locomotion along the x axis with $v = 0.2m/s$

As seen in Fig. 5.1 it is clear that there is a higher weight on the z axis when solving the QP problem. The tracking of the reaction force on the z axis is the best of the three and this guarantees a solid stability of the robot during the locomotion. Furthermore, in Fig. 5.2, instead, it is shown the relation between the force acting on the Z axis and the module of the tangential components. As stated in Chapter 4, according to the optimization strategy some considerations can be made. If the dots are too close to the cone boundary line the normalization optimization weight can be used. In this case, also because the floor surface normal it is not known, the torque minimization technique is used.

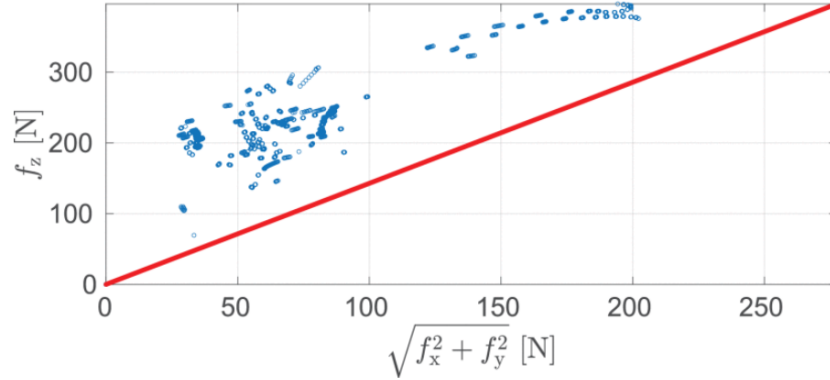
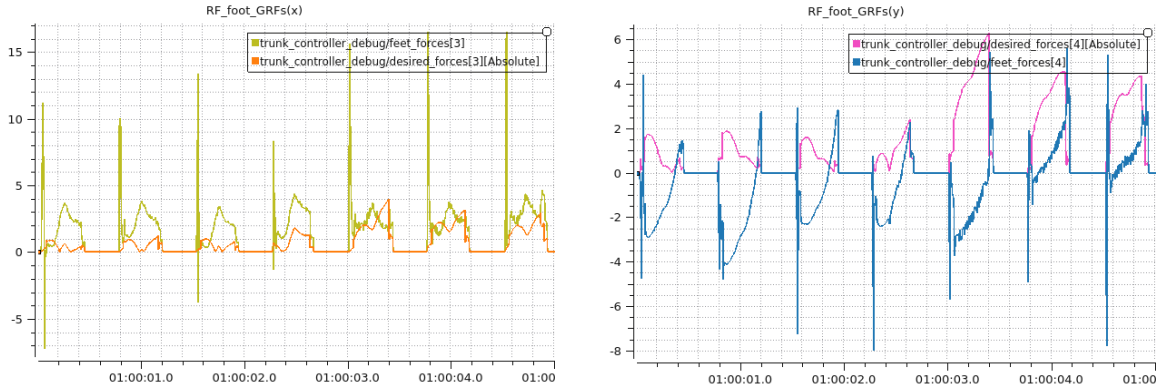


Figure 5.2: Normal force f_z versus tangential force of the LF leg. The red line is the cone bound μf_z .

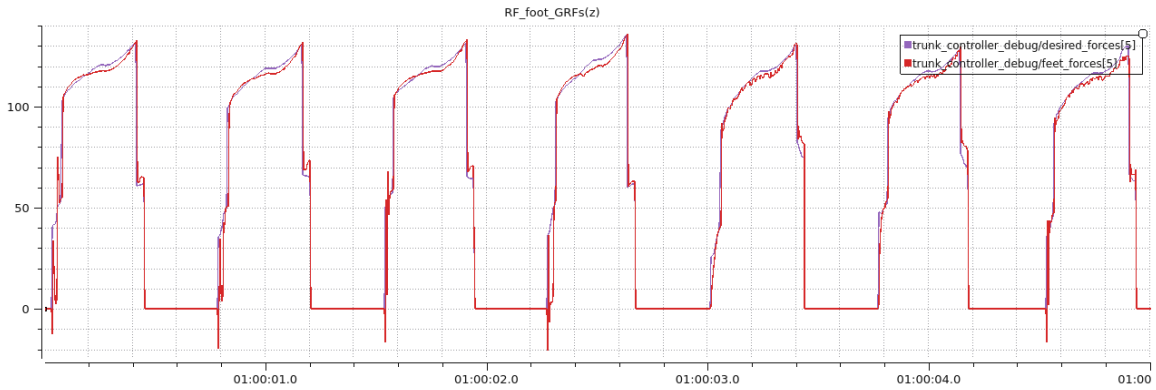
5.3. Walking on a slippery surface

In the environment in Gazebo it has been added an area with with a different friction coefficient ($\mu = 0.01$). The Aliengo robot is controlled to step on this area only with the right front foot. Reminding that the controller has been designed in a conservative way, hypothesizing a constant friction coefficient of 0.4, we expect a correct behaviour of the robot. The RF foot slips along the y axis, changing the orientation of the robot, but it successfully pass over the slippery surface. As seen in Fig. 5.3, when the event occurs, the desired forces on the x and y axis change in module, specially along the axis of the slippage (y axis). This tracking, as we know, is poor because it is lower in module than the z axis and also because it is less important with respect to the stability of the robot. The tracking of the z axis is instead very good, guaranteeing the accomplishment of the required task.



(a) Desired GRF on the RF foot on the x axis (in green) and the actual reaction force

(b) Desired GRF on the RF foot on the y axis (in pink) and the actual reaction force



(c) Desired GRF on the RF foot on the z axis (in purple) and the actual reaction force

Figure 5.3: The desired GRFs and the actual ones during a locomotion along the x axis with $v = 0.2m/s$. At 01:00:03 the right foot steps on a slippery surface with $\mu = 0.01$

The surface has been simulated to be slippery by Gazebo with the use of the several parameters provided by the software and defined as:

- **mu**: coefficient of friction along the first friction direction
- **mu2**: coefficient of friction along the second friction direction with the same value of **mu** in this case
- **fdir1**: 3-tuple unit vector specifying the direction of the first friction direction corresponding to **mu** in the collision local reference frame.

As shown in Fig. 5.4, the friction force limit is the product of the normal force and the non-dimensional coefficients **mu** and **mu2** in each friction direction.

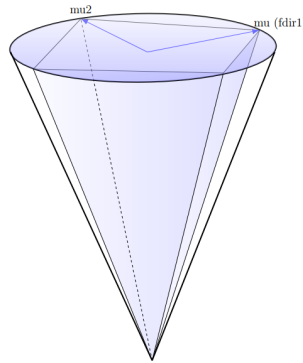
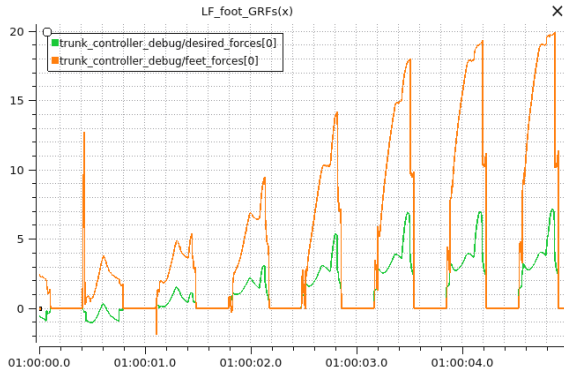


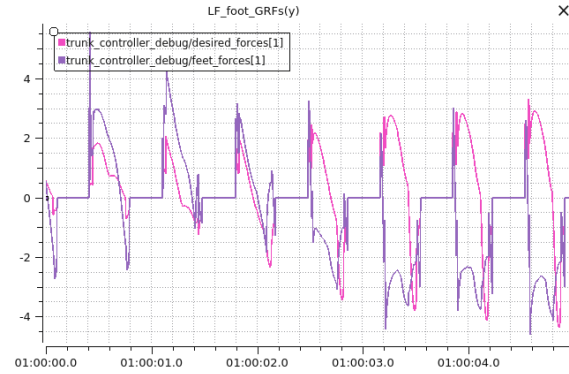
Figure 5.4: This picture shows how the friction cone is approximated with a pyramid model. The direction of $\mathbf{fdir1}$, μ and $\mu2$ are marked. Whenever a contact forms, the normal direction is decided and with a definition of $\mathbf{fdir1}$, contact coordinate frame can be easily constructed with the third $\mathbf{fdir2}$ as cross product of unit vector along normal and $\mathbf{fdir1}$ direction.

5.4. Walking on an inclined plane

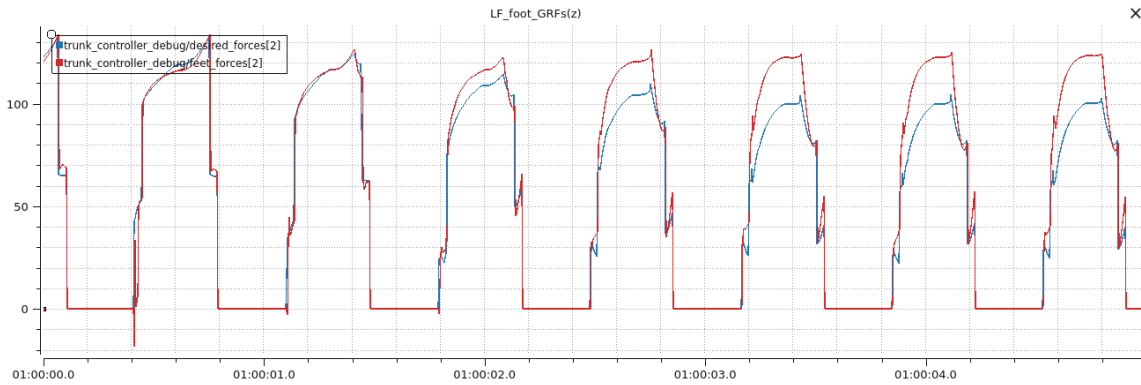
In this simulation the Aliengo robot is guided to climb a slightly inclined plane (13deg) on the x axis. It is interesting to see how the controller will deal with the difference of reaction forces between the front feet and the rear feet. From Fig. 5.8 it can be seen that there is a significant increase of the reaction force along the x axis which is not well tracked. At the same time on the z axis the reaction force acting on the foot are bigger than the desired one.



(a) Desired GRF on the LF foot on the x axis (in green) and the actual reaction force



(b) Desired GRF on the LF foot on the y axis (in pink) and the actual reaction force



(c) Desired GRF on the LF foot on the z axis (in blue) and the actual reaction force

Figure 5.5: The desired GRFs and the actual ones during a locomotion along the x axis with $v = 0.2m/s$. At 01:00:02 the robot starts climbing a plane inclined by 13 deg

Another interesting parameter that can be noticed from this simulation is the desired wrench given as input to the QP problem. Usually the wrench required on the z axis is almost the only one computed. In this simulation, instead, another component of the wrench arises: the desired wrench along the rotational y axis (as seen in Fig. 5.6). This is because the robot is inclined and the controller wants to put it back into a nominal orientation.



Figure 5.6: Desired wrenches of the Aliengo robot. The z axis wrench (in purple) is clearly the main component but, when the robot starts climbing the plane (at around 1:00:02) a much higher component on the rotational y axis (in red) can be appreciated

5.5. Falling

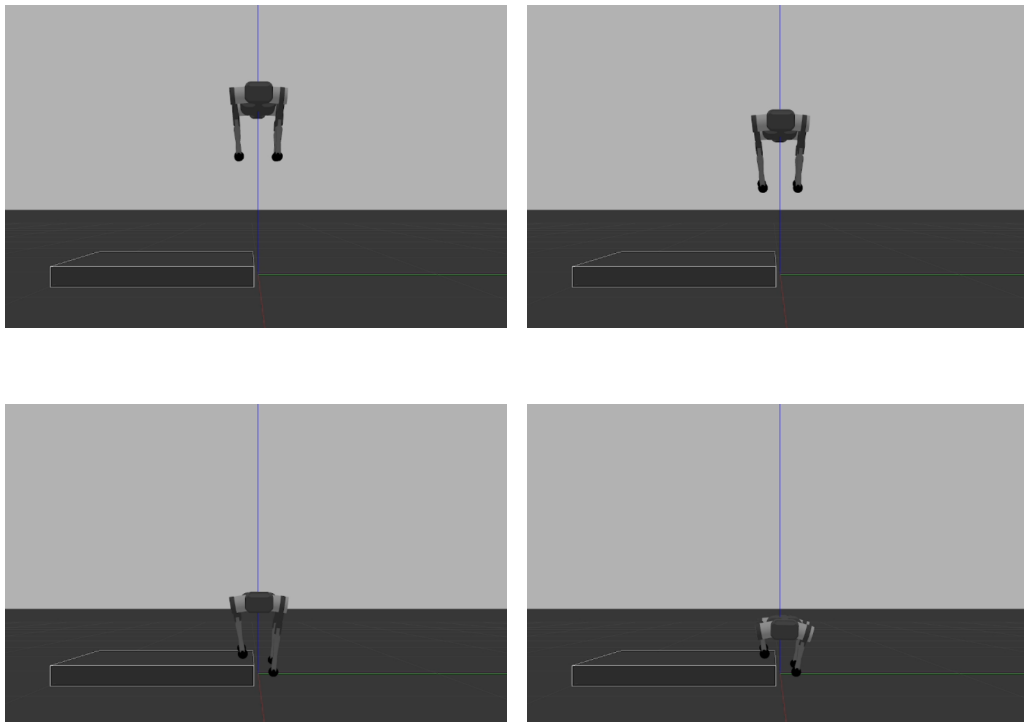
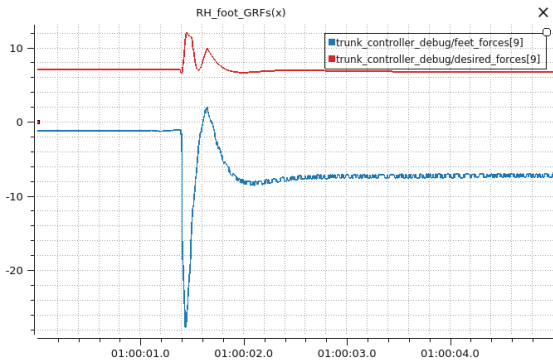
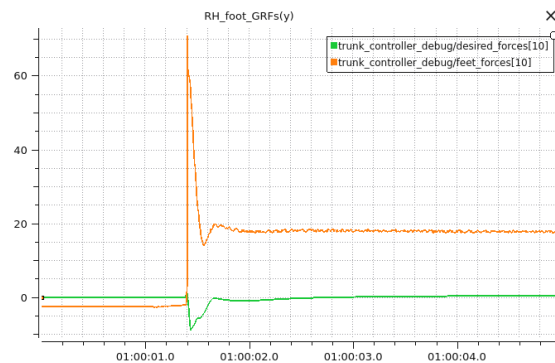


Figure 5.7: Falling process of the Aliengo robot running in a Gazebo environment

In this simulation the Aliengo robot is set free to fall from $1m$ height to the ground where there is a box $10cm$ tall made such that only the right feet of the robot will land on it. The controller is able to keep the robot standing even if, as shown in Fig. 5.8, the forces acting on the x axis and z axis on the feet standing on the box are significant.



(a) Desired GRF on the RH foot on the x axis (in red) and the actual reaction force



(b) Desired GRF on the RH foot on the y axis (in orange) and the actual reaction force



(c) Desired GRF on the RH foot on the z axis (in blue) and the actual reaction force

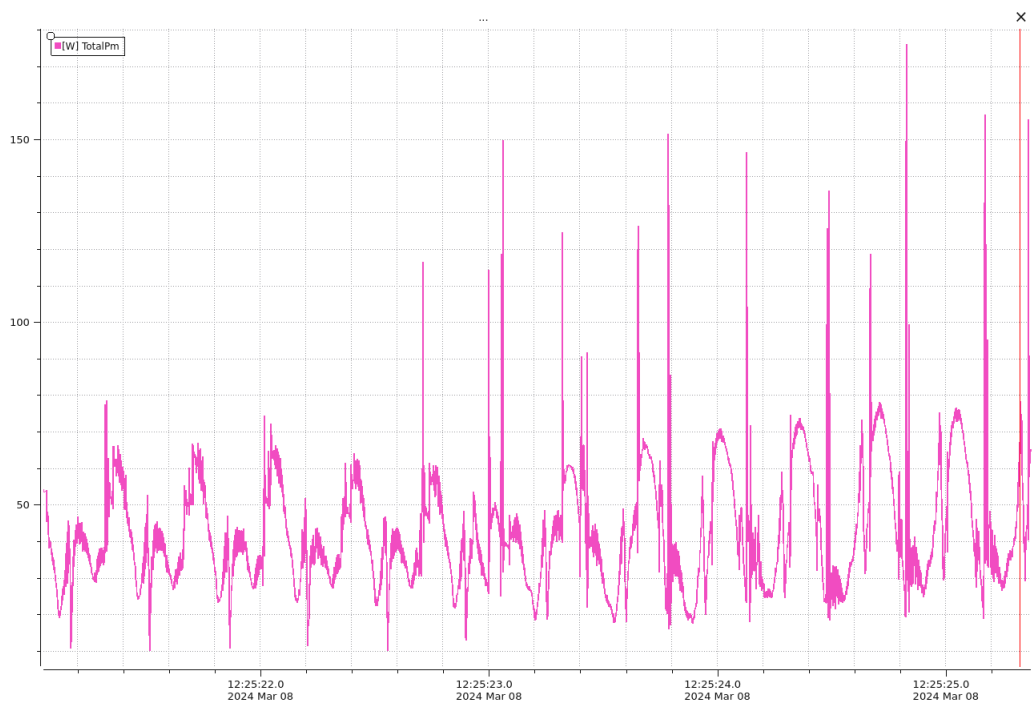
Figure 5.8: Desired GRFs versus the actual ones during a fall from $1m$. The RH foot is stepping on a $10cm$ box.

Each joint of the model have been set with a very k_p and k_d , namely the dynamically stiffness-equivalent coefficient and the dynamically damping-equivalent coefficient.

5.6. Energetic analysis



(a) Consumed power during a walk with $v_x = 0.5m/s$



(b) Consumed power during a walk on an inclined plane with $v_x = 0.5m/s$

Figure 5.9: The total power consumed by the robot in different situations.

Another key concern of robotic installations is their energy efficiency. Several experiments have been performed to understand and compute the power consumption of the Aliengo performing different tasks. In Fig. 5.9 it can be seen the mechanical power computed as the summation of the mechanical power of each of the 12 joints. Several spikes can be appreciated, coming from the computation of the desired torques and in the moment of contact of a foot with the ground. The experiment on the inclined plane show a much higher consume of power, this is probably caused by the highly non-nominal situation and probably, to correctly perform this task with a real hardware, some optimization regarding the energy should be done while solving the QP problem.

Little information is known about the electrical motors equipped by the Aliengo robot, a nominal efficiency a general brushless DC motor ($\xi = 0.85$) is considered to transform the mechanical power into electrical, the electrical power needed for a task is computed as:

$$P_e = P_m/\xi \quad (5.3)$$

being P_e the electrical power and P_m the mechanical power. The used energy can be defined as:

$$E = \int_t^{t_0} P_e dt \quad (5.4)$$

The obtained values in Table 5.1 describe the energy consumption during a gait cycle with a period of 0.8s. It can be noticed that there is a significant increase of energy consumption according to the walking speed. While walking on an inclined plane, instead, there is a significant increase of the desired torque, but also a significant (and un wanted) decrease of velocity. Therefore the actual energy consumption is not significantly higher than the nominal condition. If energy were a more strict condition, it could be taken into account in the QP problem.

Description	Data
Walking @ 0.2m/s	18 J
Walking @ 0.5m/s	35.5 J
Walking @ 1m/s	118 J
Walking on an inclined plane	41.5 J

Table 5.1: Energetic analysis on different tasks

6 | Conclusions and further developments

This research has demonstrated the effectiveness of a low level whole-body controller in managing the locomotion and stability of a quadruped robot. By directly manipulating the joint torques, the controller efficiently translates high level commands into precise motor actions, enabling the robot to navigate also non-nominal terrains and perform tasks with stability. Particular effort has been made also with respect to the actual implementation of the proposed solution, coding the control scheme and making it interact with a given robotic control framework such as DLS2.

Even if the controller is supposed to have no information about the environment such as terrain estimation and slippage estimation, it is able to accomplish the given task also in presence of an inclined plane and of an area with a slipping surface. In addition to that, a test of the stability of the robot in non-nominal condition is presented in the simulation where the robot is set free to fall from a considerable height and successfully land on a non flat surface.

While the current research has demonstrated the efficacy of the low level controller in controlling quadruped robots such as Unitree Aliengo, several avenues for future exploration remain.

These include the full implementation in the DLS2 framework of the Pinocchio library, which has been demonstrated to have better performances and an easier implementation setup with different robots. From the control point of view it would be interesting the addition of a state estimation layer in order to improve robustness. There is room for improvement of the reference generator, it could be designed an NMPC (Nonlinear Model Predictive Control) to produce better foothold reference to feedback the nominal ones computed by the actual generator; this would improve performances, specially regarding tasks such as walking on rough terrains.

Another important aspect not addressed by this thesis is the design of a high level controller, it would be interesting to test the proposed controller with commands not given

by a user but, for instance, by a local planner that tries to track a reference target in the environment.

Last but not least, the proposed controller and implementation method need an extensive effort to validate the functionality on real hardware, carefully tuning the design parameters to guarantee a robust closed-loop that makes the robot stable and able to reject disturbances.

Bibliography

- [1] *Brief introduction to the quadruped robot HyQReal*, June 2021. Zenodo. doi: 10.5281/zenodo.4782613. URL <https://doi.org/10.5281/zenodo.4782613>.
- [2] A study on quadruped mobile robots. *Mechanism and Machine Theory*, 190: 105448, 2023. ISSN 0094-114X. doi: <https://doi.org/10.1016/j.mechmachtheory.2023.105448>. URL <https://www.sciencedirect.com/science/article/pii/S0094114X23002197>.
- [3] AnimatorNotebook. A guide to quadrupeds' gaits - walk, amble, trot, pace, canter, gallop, 2022. URL <https://www.animatornotebook.com/learn/quadrupeds-gaits>.
- [4] S. Caron. Jacobian of a kinematic task and derivatives on manifolds, 2012. URL <https://scaron.info/robotics/jacobian-of-a-kinematic-task-and-derivatives-on-manifolds.html>.
- [5] J. Carpentier, G. Saurel, G. Buondonno, J. Mirabel, F. Lamiroux, O. Stasse, and N. Mansard. The pinocchio c++ library – a fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives. In *IEEE International Symposium on System Integrations (SII)*, 2019.
- [6] A. Dettmann, S. Planthaber, V. Bargsten, R. Dominguez, G. Cerilli, M. Marchitto, G. Fink, M. Focchi, V. Barasuol, C. Semini, and R. Marc. Towards a generic navigation and locomotion control system for legged space exploration. 06 2022.
- [7] J. Diebel. Representing attitude: Euler angles, unit quaternions, and rotation vectors. *Matrix*, 58, 01 2006.
- [8] R. Featherstone. A beginner's guide to 6-d vectors (part 1). *IEEE Robotics Automation Magazine*, 17(3):83–94, 2010. doi: 10.1109/MRA.2010.937853.
- [9] M. Focchi, A. del Prete, I. Havoutis, R. Featherstone, D. G. Caldwell, and C. Semini. High-slope terrain locomotion for torque-controlled quadruped robots. *Autonomous Robots*, 41(1):259–272, Jan 2017.

- [10] M. Frigerio, J. Buchli, D. G. Caldwell, and C. Semini. RobCoGen: a code generator for efficient kinematics and dynamics of articulated robots, based on Domain Specific Languages. *7(1):36–54*, 2016.
- [11] R. Grandia, F. Jenelten, S. Yang, F. Farshidian, and M. Hutter. Perceptive locomotion through nonlinear model predictive control, 2022.
- [12] E. Guizzo. By leaps and bounds: An exclusive look at how boston dynamics is redefining robot agility. *IEEE Spectrum*, 56(12):34–39, 2019. doi: 10.1109/MSPEC.2019.8913831.
- [13] J. Kim, D. Chung, Y. Kim, and H. Kim. Deep learning-based 3d reconstruction of scaffolds using a robot dog. *Automation in Construction*, 134:104092, 2022.
- [14] R. Lozano, A. Valera, P. Albertos, and S. Arimoto. Pd control of robot manipulators considering joint flexibility, actuators dynamics and friction. In *Proceedings of the 1997 American Control Conference (Cat. No.97CH36041)*, volume 5, pages 2638–2641 vol.5, 1997. doi: 10.1109/ACC.1997.611934.
- [15] C. Mastalli, R. Budhiraja, W. Merkt, G. Saurel, B. Hammoud, M. Naveau, J. Carpentier, L. Righetti, S. Vijayakumar, and N. Mansard. Crocoddyl: An Efficient and Versatile Framework for Multi-Contact Optimal Control. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.
- [16] X. Meng, W. Liu, L. Tang, Z. Lu, H. Lin, and J. Fang. Trot gait stability control of small quadruped robot based on mpc and zmp methods. *Processes*, 11(1), 2023. ISSN 2227-9717. doi: 10.3390/pr11010252. URL <https://www.mdpi.com/2227-9717/11/1/252>.
- [17] M. Neunert, M. Stäuble, M. Gifftthaler, C. D. Bellicoso, J. Carius, C. Gehring, M. Hutter, and J. Buchli. Whole-body nonlinear model predictive control through contacts for quadrupeds. *CoRR*, abs/1712.02889, 2017. URL <http://arxiv.org/abs/1712.02889>.
- [18] D. E. Orin, A. Goswami, and S.-H. Lee. Centroidal dynamics of a humanoid robot. *Autonomous Robots*, 35(2):161–176, Oct 2013. ISSN 1573-7527. doi: 10.1007/s10514-013-9341-4. URL <https://doi.org/10.1007/s10514-013-9341-4>.
- [19] N. Rathod, A. Bratta, M. Focchi, M. Zanon, O. Villarreal, C. Semini, and A. Bemporad. Mobility-enhanced MPC for legged locomotion on rough terrain. *CoRR*, abs/2105.05998, 2021. URL <https://arxiv.org/abs/2105.05998>.
- [20] L. Righetti, J. Buchli, M. Mistry, M. Kalakrishnan, and S. Schaal. Optimal dis-

- tribution of contact forces with inverse dynamics control. *International Journal of Robotics Research*, 32, 03 2013. doi: 10.1177/0278364912469821.
- [21] M. Risiglione, V. Barasuol, D. G. Caldwell, and C. Semini. A whole-body controller based on a simplified template for rendering impedances in quadruped manipulators, 2022.
- [22] M. E. Rosheim. *Robot evolution: the development of anthrobotics*. John Wiley & Sons, 1994.
- [23] C. Semini, N. G. Tsagarakis, E. Guglielmino, M. Focchi, F. Cannella, and D. G. Caldwell. Design of hyq – a hydraulically and electrically actuated quadruped robot. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, 225(6):831–849, 2011. doi: 10.1177/0959651811402275. URL <https://doi.org/10.1177/0959651811402275>.
- [24] C. Semini, V. Barasuol, J. Goldsmith, M. Frigerio, M. Focchi, Y. Gao, and D. G. Caldwell. Design of the hydraulically actuated, torque-controlled quadruped robot hyq2max. *IEEE/ASME Transactions on Mechatronics*, 22(2):635–646, 2017. doi: 10.1109/TMECH.2016.2616284.
- [25] Stanford Artificial Intelligence Laboratory et al. Robotic operating system. URL <https://www.ros.org>.

A | Euler angular velocities

It is employed the Z-Y-X convention [7] for the Euler angles sequence $\Phi = (\phi, \theta, \psi)^\top$ to represent the orientation of the robot base where, ϕ , θ and ψ are the roll, pitch and yaw, respectively. The angular velocity in inertial and CoM frame is related to the Euler angle rates with the following relations:

$$\begin{aligned}\boldsymbol{\omega} &= \mathbf{E}(\Phi) \dot{\Phi} \\ \boldsymbol{\omega}_{\text{CoM}} &= \mathbf{E}'(\Phi) \dot{\Phi}\end{aligned}$$

$\mathbf{E}(\Phi)$ and $\mathbf{E}'(\Phi)$ are the *Euler angle rates matrix* and conjugate *Euler angle rates matrix* respectively given by:

$$\begin{aligned}\mathbf{E}(\Phi) &= \begin{bmatrix} \cos(\theta) \cos(\psi) & -\sin(\psi) & 0 \\ \cos(\theta) \sin(\psi) & \cos(\psi) & 0 \\ -\sin(\theta) & 0 & 1 \end{bmatrix} \\ \mathbf{E}'(\Phi) &= \begin{bmatrix} 1 & 0 & -\sin(\theta) \\ 0 & \cos(\phi) & \cos(\theta) \sin(\phi) \\ 0 & -\sin(\phi) & \cos(\theta) \cos(\phi) \end{bmatrix}\end{aligned}$$

Remark: \mathbf{E} depends on pitch and yaw, whereas \mathbf{E}' on roll and pitch. Thus, the Euler angle rates $\dot{\Phi}$ is:

$$\begin{aligned}\dot{\Phi} &= \mathbf{E}^{-1}(\Phi) \boldsymbol{\omega} \\ \dot{\Phi} &= \mathbf{E}'^{-1}(\Phi) \boldsymbol{\omega}_{\text{CoM}}\end{aligned}$$

B | Jacobian of a task

In inverse kinematics, the control problem is to bring task residuals[4], also known as task errors, to zero. If it is denoted by $\mathbf{e}(\mathbf{q})$ the residual of a task when the robot is in configuration $\mathbf{q} \in C$, as well as $\mathbf{J}(\mathbf{q})$ the Jacobian of the task, inverse kinematics can be solved by integrating velocities $\dot{\mathbf{q}}$ computed as:

$$\dot{\mathbf{q}} = -\mathbf{J}(\mathbf{q})^{-1}\alpha\mathbf{e}(\mathbf{q}), \quad \alpha > 0$$

In this appendix, it is clarified the definition of the Jacobian of a kinematic task.

A kinematic task is fully defined by the residual function $\mathbf{e}(\mathbf{q})$ that should be brought to zero. The Jacobian of the task is then the Jacobian of this residual:

$$\mathbf{J}(\mathbf{q}) := \frac{\partial \mathbf{e}}{\partial \mathbf{q}}(\mathbf{q})$$

With this choice, the closed-loop behavior of integrating velocities $\dot{\mathbf{q}}$ such that

$$\begin{aligned} \dot{\mathbf{q}} &= -\mathbf{J}(\mathbf{q})\mathbf{q} \\ \dot{\mathbf{e}} &= \dot{\mathbf{q}} - \dot{\mathbf{q}}_{\text{cmd}} = -\alpha\mathbf{e} \end{aligned}$$

drives \mathbf{e} to zero exponentially with a characteristic frequency α .

Now the case of a pose task can be analyzed, where it is controlled the pose $\mathbf{T}_{0b} \in (\mathcal{SE}(3))$ of a robot frame b with respect to an inertial frame 0, and want to make it coincide with a target frame t defined by \mathbf{T}_{0t} .

The pose task can be defined as:

$$\mathbf{e}(\mathbf{q}) := {}_b\xi_0 = \mathbf{T}_{0t}^{-1} \ominus \mathbf{T}_{0b} = \log_6(\mathbf{T}_{0b}\mathbf{T}_{0t}) = \log_6(\mathbf{T}_{bt})$$

In this formula, the logarithm $\log_6 : \mathcal{SE}(3) \rightarrow \mathfrak{se}(3)$ maps poses (elements of the Lie group $\mathcal{SE}(3)$) to twists (elements of the corresponding Lie algebra $\mathfrak{se}(3)$). The operator \ominus is

called the right-minus. These concepts are introduced, for instance, in the first sections of the micro Lie theory (MLT) writeup.

Formally, this residual is a function $\mathbf{e} : \mathcal{C} \rightarrow \mathfrak{se}(3)$ from the configuration space \mathcal{C} (a manifold) to the Lie algebra $\mathfrak{se}(3)$. In what follows, it is defined in the local frame rather than in the world frame, following the same convention as both micro Lie theory and Pinocchio (where computed quantities are local by default).

The Jacobian of the pose task is, instead, defined by:

$$\mathbf{J}(\mathbf{q}) := \frac{\partial \mathbf{e}}{\partial \mathbf{q}}(\mathbf{q})$$

The practical formula is:

$$\mathbf{e}(\mathbf{q}) = -\mathbf{J} \log_6(\mathbf{T}_{tb}) \cdot \mathbf{e}_0(\mathbf{q}) \quad (\text{B.1})$$

It can be noticed that, contrary to the position task, the Jacobian of the pose task is not the same as the frame Jacobian $\mathbf{J}_{0b}(\mathbf{q})$. Rather, it is its image by the log-derivative $\mathbf{J}_{\log_6}(\mathbf{T}_{tb})$, where the function \mathbf{J}_{\log_6} is the right derivative of the logarithm:

$$\mathbf{J}_{\log_6}(\mathbf{T}) := \frac{\partial \mathbf{T}}{\partial \log_6(\mathbf{T})}$$

This function is available as `J_log6` in Pinocchio.

List of Figures

1	Unitree Go1 robot of UDESC university	2
1.1	The horse machine of 1890 and the Phony Pony of 1977	5
1.2	Spot classic and spot mini	6
1.3	IIT QMRs	7
1.4	Unitree Go1 and Unitree Aliengo	8
1.5	Typical complete low level control scheme, as showed in [19]	10
2.1	Home position and coordinate definition of the robot	13
2.2	Fictitious 6 - DoF that connects fixed frame with the floating base.	14
2.3	Dynamic constraints	17
2.4	Foothold sequence of a horse trotting	18
3.1	The proposed control scheme	19
3.2	Representation of the robocentric stepping strategy and of the Swing Frame, located at the lift-off point	21
3.3	Used frame references in the controller and $\mathbf{p}_{com,i}$	23
3.4	Horizontal reference frames (in green) and the robot frame (in blue-the parallelepiped represents the robot trunk; horizontal frames share the same yaw angle with respect to the world reference frame (in black)	24
4.1	Examples of the command line interface commands available on two apps loaded in the DLS2 framework	31
4.2	Robotlib interaction with the other modules of the proposed locomotion framework	33
4.3	Code snippets of the code used to generate the Aliengo RobCoGen library	35
5.1	The desired GRFs and the actual ones during a locomotion along the x axis with $v = 0.2m/s$	42
5.2	Normal force f_z versus tangential force of the LF leg. The red line is the cone bound μf_z	43

5.3	The desired GRFs and the actual ones during a locomotion along the x axis with $v = 0.2m/s$	44
5.4	This picture shows how the friction cone is approximated with a pyramid model. The direction of <code>fdir1</code> , <code>mu</code> and <code>mu2</code> are marked. Whenever a contact forms, the normal direction is decided and with a definition of <code>fdir1</code> , contact coordinate frame can be easily constructed with the third <code>fdir2</code> as cross product of unit vector along normal and <code>fdir1</code> direction.	45
5.5	The desired GRFs and the actual ones during a locomotion along the x axis on an inclined plane	46
5.6	Desired wrenches of the Aliengo robot. The z axis wrench (in purple) is clearly the main component but, when the robot starts climbing the plane (at around 1:00:02) a much higher component on the rotational y axis (in red) can be appreciated	47
5.7	Falling process of the Aliengo robot running in a Gazebo environment	47
5.8	Desired GRFs versus the actual ones during a fall	48
5.9	The total power consumed by the robot in different situations.	49

List of Tables

4.1	Aliengo robot specifications	27
4.2	Go1 robot specifications	28
5.1	Energetic analysis on different tasks	50

List of Symbols

Variable	Description
\mathbf{q}	vector of joints displacement (rad/s)
\mathbf{H}	joint space inertia matrix
\mathbf{C}	Coriolis and centrifugal term matrix
\mathbf{v}	vector of robot's generalized velocities (m/s)
\mathbf{v}_c	floating base (CoM) generalized velocities (m/s)
$\boldsymbol{\tau}$	vector of joint torques (N/m)
\mathbf{J}	matrix of contact point jacobian
\mathbf{f}	vector of GRFs
\mathbf{S}	skew-symmetric operator
\mathbf{I}	inertia tensor at CoM position
\mathbf{g}	vector of gravitational acceleration
δ_i	binary parameter for contact foot
\mathbf{p}_c	CoM position vector
$\boldsymbol{\phi}$	robot base orientation vector in Z-Y-X Euler angles (rad)
$\dot{\boldsymbol{\omega}}$	angular acceleration vector of the robot's base (rad/s ²)
\mathbf{f}	GRFs for the contacting feet
\mathbf{K}_w	proportional gain matrix of the WBC
\mathbf{D}_w	derivative gain matrix of the WBC
\mathbf{h}	vector of gravity/Coriolis terms for the leg joint dynamics
\mathbf{K}_j	proportional gain matrix of the joint space PD
\mathbf{D}_j	derivative gain matrix of the joint space PD
\mathbf{S}	QP cost function gain matrix
\mathbf{T}	QP cost function terminal term matrix
μ	friction coefficient

Nomenclature

Acronym	Description
IIT	Instituto Italiano di Tecnologia
DLS	Dynamic Legged System
GRF	Ground Reaction Force
SRBD	Single Rigid Body Dynamics
QP	Quadratic Programming
CoM	Center of Mass
PD	Proportional Derivative
HAA	Hip Abduction-Adduction
HFE	Hip Flexion-Extension
KFE	Knee Flexion-Extension

