Locomotion control of a lightweight quadruped robot

Pierre-Alexandre Léziart

Université Fédérale
Toulouse Midi-Pyrénées

THÈSE

En vue de l'obtention du
DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE
Délivré par l'Université Toulouse 3 - Paul Sabatier

Présentée et soutenue par

Pierre-Alexandre LEZIART

Le 17 octobre 2022

Contrôle de la locomotion d'un robot quadrupède léger

Ecole doctorale : **SYSTEMES**

Spécialité : **Robotique**

Unité de recherche :
LAAS - Laboratoire d'Analyse et d'Architecture des Systèmes

Thèse dirigée par
Philippe SOUERES et Thomas FLAYOLS

Jury

M. Marco HUTTER, Rapporteur
M. Ludovic RIGHETTI, Rapporteur
M. Claudio SEMINI, Examinateur
M. Philippe SOUERES, Directeur de thèse
M. Thomas FLAYOLS, Co-directeur de thèse
Mme Sophie TARBOURIECH, Présidente

Acknowledgments

First I would like to thank my two PhD supervisors Philippe Souères and Thomas Flayols for their continuous support during these 3 years. They were always available to talk with me, provide feedback and discuss about new ideas. Their involvement and skills played a crucial role in making this thesis what is is.

I also thank the referees for the time they spent reading this dissertation, and all members of the jury for accepting to review my work. I am glad that I have been able to exchange with these experts whose work guided my scientific developments.

My friends and the Gepetto team as a whole greatly contributed to making this PhD an enjoyable experience despite the inevitable ups and downs that come with a long term scientific project. For all our discussions, cakes, activities and positive atmosphere, I thank them too.

Finally, I would like to thank my parents for their support, not only for these 3 years but as far as I remember. This defense is merely a step on a long road and they helped me pave it down.

Contents

Chapter 1

Introduction

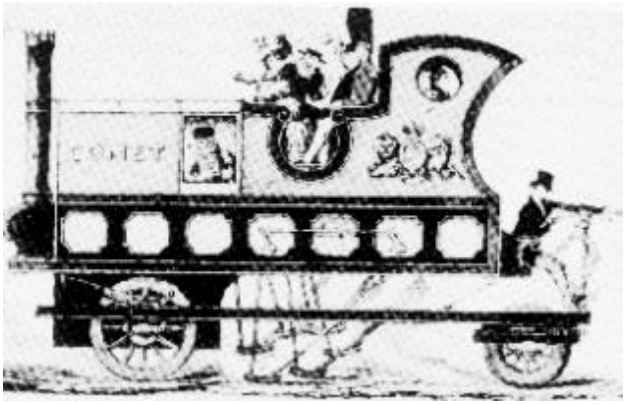
Legged robots occupy a special place in the robotics landscape. It might be because among all robots, they are by nature the closest to us due to their structure that reminds us of the animal world. Because of that, when we see them moving we sometimes feel a kind of empathy as we unconsciously perceive them as their natural counterpart. If not a proof, the comments under videos of recovery tests after pushes that jokingly complain about the "mean humans bullying the robot" are still an indication of a feeling shared by people which is absent when they see other kinds of robotic experiments, like crash tests of autonomous cars.

Scientists have been trying to understand the inner workings of legged locomotion for decades. Yet, due to its complexity, obtaining a robust autonomous locomotion in a wide range of situations is still an open challenge, even if increasingly impressive behaviors have been shown in recent years. Let us first discuss how the road of legged robotics has started to better understand how mature it has grown [ST07].

1.1 From scientific attraction to large-scale deployment

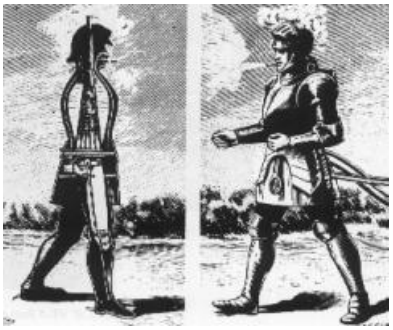
The very early stages of legged locomotion date back to the 15th century with Leonardo da Vinci designing the first anthropomorphic automaton. At the crossing between mechanical and anatomical studies, this armored knight was made of wood, leather and brass and could sit up, wave its arms and moves its head through a cable system [Ros94]. The first sketch of what could be called a legged vehicle dates from the 18th century with a carriage structure whose motion transmission was performed by a set of legs pushing on the ground [Thr85]. Then, a first step toward a proper formalization of legged locomotion was done in 1850 by the Russian mathematician Chebyshev who presented a model of kinematic linkage to move a body in a straight way based on alternating contacts with feet moving up and down [Rai86]. The first quadruped machine appears half a century later with a patent for the Mechanical Horse by Rygg in 1893, with pedals in a bike-like manner to power the stepping motion through a set of gears, even if there is no evidence it was ever built [Rai86]. The same year, Moore built his Steam Man, the earliest successful biped automaton. It was steam powered and could walk in circle with the guidance of a swing arm for stability reason [Ros94]. In a similar way than Rygg's mechanical horse, the baron of Bechtolsheim designed a striding wagon in 1913 but it was likely never built [SW89]. Finally, at the dawn of World War I, Thring developed a prototype of legged tractor as one of the first examples of what we call now hybrid legged machines (com-

bining legs and wheels) [Thr85]. Even though all these systems involved legs to produce movement, they still faced limitations so severe that they could not exploit the possibilities offered by legs to their full extent. Simple mechanisms and sets of gears imposed a fixed gait with predefined foot placements. Moreover, their controls could not perform any adaptation since it were completely blind to both machine and environment states. Some of these robots are presented in Fig. 1.1.

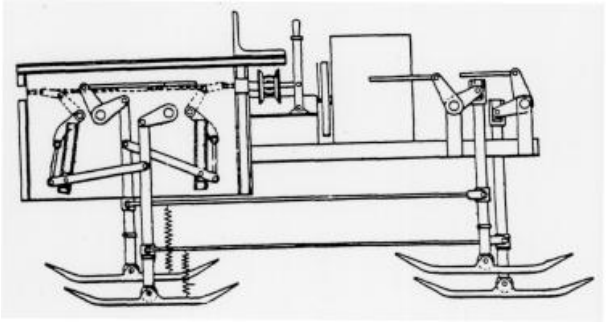


(a) Carriage

(b) Mechanical horse



(c) Steam man

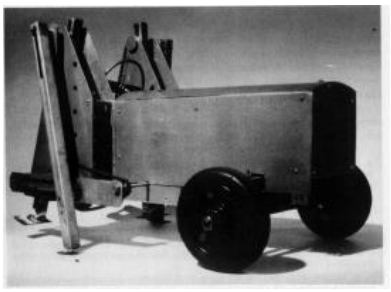


(d) Striding wagon

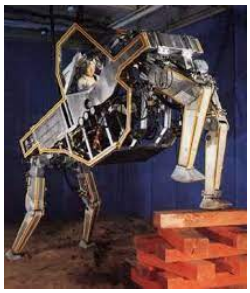
Figure 1.1: First legged machines before World War I.

Starting from the 1950s, several groups of researchers began to study walking machines in a systematic way. This lead to the development of increasingly refined systems in the following decades. Mosher's Walking Truck was built in 1968 as a vehicle controlled by a human driver using handles and pedals that were hydraulically connected to the four legs [Mos68]. At a completely different scale, a 15 000 tonnes legged machine was built in 1969 for use in an open-air coal mine. This Big Muskie had four hydraulic legs that lifted forwards or backwards all at once to raise the body and move it by one step [Cox70]. Back to a more human scale, McGhee developed a series of hexapod robots in the 1970s and 1980s using electric motors, digital control and on-board sensors [Ori+79; McG+85]. He also developed the Adaptive Suspension Vehicle in 1984, a 2 700 kg vehicle capable of carrying a person using hydraulically-actuated pantograph-shaped legs [SW89]. Another hexapod called Odex was designed in the 1980s for the power plant inspection. It could be equipped with a manipulator and was able to move inside man-made structures, including narrow passages [Bar87]. Although all these machines were one step further in terms of capabilities compared to what we have seen previously, they were limited to walks in laboratories or in controlled conditions. They could not benefit yet from the advantages of legged locomotion in rough terrains due to limited understanding

of gaits and lacking advancements in terms of leg coordination control and mechanical design. Some of these robots are presented in Fig. 1.2.

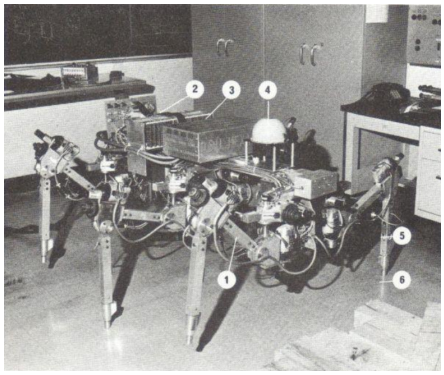


(a) Walking tractor

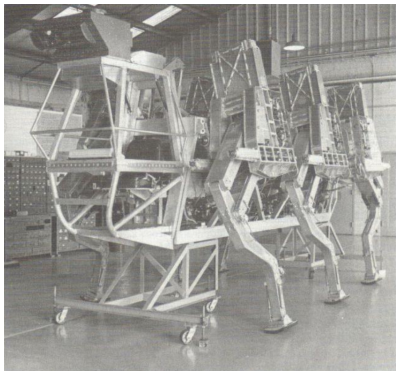


(b) Walking truck

(c) Big Muskie



(d) McGhee hexapods



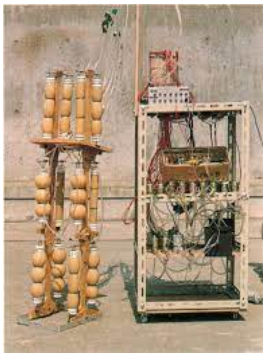
(e) Adaptive Suspension Vehicle

(f) Odex

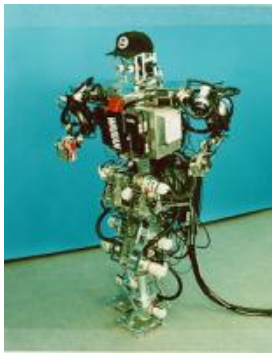
Figure 1.2: Increasingly refined legged machines after the 1950s.

Thanks to improved actuation technologies, innovative control loops and a deeper theoretical understanding, the legged machines that followed succeeded in overcoming the limits of their predecessors and laid the foundations of modern robotics. From 1967 to the 2000s, the Waseda University has shown a long history of computer controlled biped robots, such as the WAP-1 that could playback taught movements using artificial rubber muscles that were pneumatically actuated [LT07]. Their WL-10 RD biped that came later in the 1980s achieved a quasi-dynamic gait as it could briefly unbalance itself by leaning forwards to transfer support from one foot to another [Tak+85]. These developments culminated in 1999 with the WABIAN humanoid robot capable of moving while transporting loads with its arms as well as dancing in a dynamic way by waving its arms and hips [Yam+99]. Still in Japan, the Tokyo Institute of Technology has been developing a series of quadruped robots since 1976 with their Titan series, such as the Titan VIII that could use one of its legs as a manipulator arm with the possibility of being equipped with different end effectors [AH96]. Monopods, single legged robot whose locomotion is performed through hops, also heavily contributed to the field despite their apparent simplicity. With a series of monopods in the 1980s and 1990s, Raibert achieved for the first time a highly dynamic motion with a robotic system, going up to 2.2 m/s with the Pogo-stick [Rai86]. A monopod even went in space with the Hopper PrQP-F that collected scientific data on Phobos in 1998 [Kem98]. A new innovative subtype of legged robot makes its appearance in the 1990s with McGeer's passive walkers that could walk without any kind of actuation thanks to carefully designed mechanical structures that made use of gravity or an initial impulse to keep their motion going [McG+90]. The ASIMO biped presented by Honda in 2000 marked the culmination of 20th century robotics with a

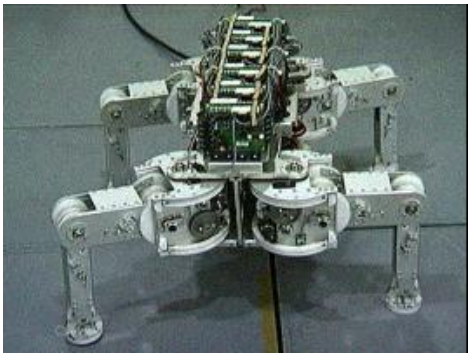
1.2 meter high fully autonomous humanoid robot with 26 degrees of freedom, electrically actuated and able to detect obstacles with vision, climb stairs and carry 0.5 kg with each hand [Sak+02]. Compared to hopping monopods or multilleged robots, performances of biped robots have advanced slower as they are more demanding in terms of control due to their unstable vertical posture [KV02]. The performances kept getting better and better over the years with initiatives such as the Humanoid Robotic Programme HRP [Kan+04]. Fig. 1.3 shows some of these robots.



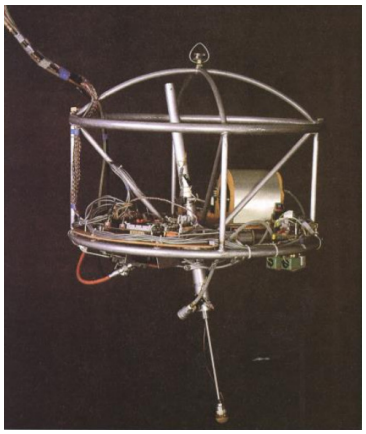
(a) WAP-1



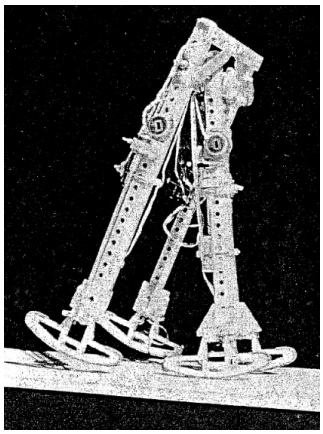
(b) WABIAN



(c) Titan VIII

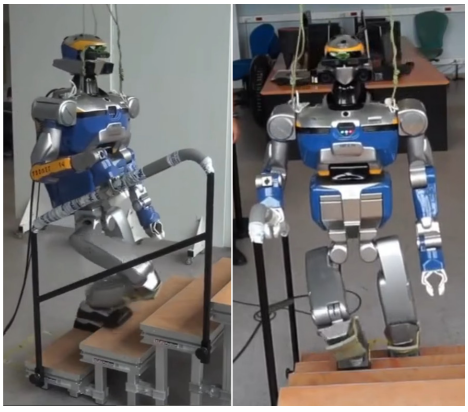


(d) Pogostick



(e) Passive walker

(f) Asimo

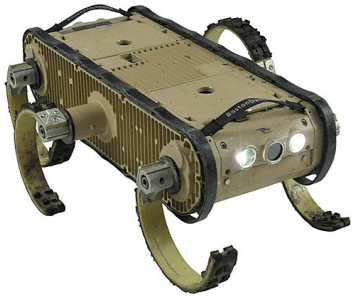


(g) HRP-2 robot of the Gepetto team climbing stairs [Car+16]

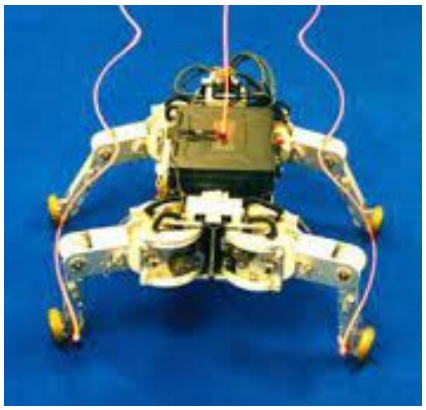
Figure 1.3: Improvements in actuator technologies lead to impressive legged robots.

At the fringes between legged robotics and wheeled locomotion, the concepts of Wheel-With-Legs [Vel+98] (straight rods attached on the rim of a rotating axis) and Whegs [Alt+01] (a contraction of wheel-leg for curved appendages attached on rotating axis) were explored in the 2000s as a way to combine the advantages of both structures. The goal was to go beyond the intrinsically limited locomotion speed of legs while having better traction capabilities than classic wheels. Despite their mechanical simplicity these robots were able to move in rough terrains and climb bigger obstacles than they could do with wheels of the same diameter. Likewise, hybrid legged robots associate wheels and legs as well, but this time as two different locomotion modes with the possibility to switch from one to another. They typically have actuated wheels at the tip of their legs. On even surfaces, they benefit from the efficient wheeled locomotion, yet they can still climb stairs or navigate in irregular terrains by locking their wheels and walking as a more classic legged robot would [MKK02]. See some prototypes in Fig. 1.4.

(a) Whegs-I



(b) RHex-0



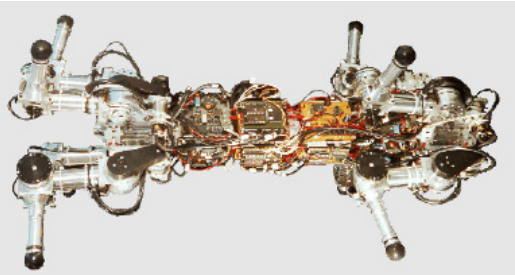
(c) Roller-Walker

(d) AZIMUT

Figure 1.4: Innovative robots combining wheeled and legged locomotion.

With the rise of increasingly robust and well-controlled robotic systems, the first considerations for commercial applications of legged robots outside pure research started to appear. A wide range of fields could make use of such systems when locomotion has to be performed in complex environments that can not be tackled with more traditional wheeled approaches. Inspection is one such task. A legged system can replace a human operator in cases where the target cannot be reached easily, such as the pipe inspection robot MORITZ that could work even with slopes, curves or pipe junctions [ZP03]. They can also assist in search and rescue operations in rugged and/or aquatic environments such as the Asguard robots which uses compliant whegs to climb rubble and swim [EGK08].

Patrolling harsh and dirty locations like offshore platforms [Geh+21] or mines [Tra+22] is also a possibility. Alongside inspection, legged robots can assist humans in load-carrying duties in rough natural terrains [Rai+08]. For now, quadrupeds take most of the spotlight due to the robustness offered by their four legs compared to bipeds. Some of these robots are presented in Fig. 1.5.



(a) MORITZ

(b) Asguard IV

(c) ANYmal for mine exploration

(d) BigDog

Figure 1.5: Fist considerations for commercial applications of legged robots.

As we have seen in recent years, thanks to the successive efforts of hundreds of researchers that tried to tackle the challenges of legged locomotion, legged machines went from a mere scientific curiosity at the dawn of the 18th century [Thr85] to a technological expertise that reached the early stages of large-scale commercial deployment [Dyna; Dynb; Rob; Zur].

1.2 A dual relation with biology: inspirations and applications

Understanding how legged animals move so well has been a long-lasting fascination, especially in light of how difficult it is to reproduce such performances with artificial machines. Studying animals can be an inspiration to grasp the inner workings of motion generation and transpose them to the robotics field [MSW20]. The resulting theoretical

advancements can sometimes be applied back to the natural world, like for the reconstruction of the gait of extinct species based on skeletal data [HWM20]. Back in Greek antiquity, Aristotle was already studying the shape and bone structure of the horses legs to understand animal motion [Nus+85]. In the second half of the 18th century, Muybridge used stop-motion photography to document running animals like cats, dogs, horses and humans to better understand their gait [Muy87]. To go further, Hoyt studied how horses have a preferred gait pattern for various ranges of locomotion speed to keep their cost of transport at a minimum [HT81].

Efforts have been made to develop robots that mimic animals as best as possible. Among all the animals that have been copied, insects are likely the most popular due to the intrinsic stability offered by their hexapod or octopod locomotion combined with a structure that is close to the ground and thus lead to a low center of mass. The stick-insects or cockroaches are especially used as a model because of their ability to move skillfully on irregular terrains despite their very simple kinematic structure [Cru+91]. Over the years, plenty of prototypes were built, with various degrees of complexity both in terms of mechanical structure and control. Mimicking the natural world did not stop at the structure but also involved new kinds of actuators like McKibben artificial muscles arranged in pairs for the antagonistic principle [Ker+04]. [Bai+01] went even further in the biomimetism with Sprawlita, an hexapod robot with a visco-eslastic structure and pneumatic actuators that relied on leg compliance to achieve a passive stabilization similar to the one observed in insects.

Animal behavioral studies also led to innovative control scheme such as the subsumption architecture, a particular type of behavior-based control, which was applied to a small hexapod in 1989 [Bro89]. A final striking example of bio-inspiration is the climbing robot Stickybot whose force control strategy worked in conjunction with its gecko-like fingers to maintain a sufficient level of adhesion [Kim+07].

1.3 Challenges of locomotion with legged robots

As we have seen previously with the history of legged robotics and the way researchers take inspiration from the natural world to try to understand the motion generation that comes with it, legged locomotion is still a challenge that is far from being completely solved. Several key characteristics of this kind of locomotion require carefully designed mechanical and control architectures to be able to perform tasks in a meaningful way.

First, legged robots have to move around without being directly attached to the ground so they cannot follow arbitrary motion commands. Their dynamics is an underactuated problem in the sense that their base, sometimes called free-flyer, has 6 degrees of freedom (3 for position, 3 for orientation) that are not actuated, with no way to directly regulate their orientation in the world for instance. As such they have to be controlled indirectly through the motion of their actuated appendages. This has to be done for task purpose, like going to a target location or orienting the body in a particular way, but also to keep balance. Most quadrupeds and bipeds are only dynamically stable when walking in non-conservative ways so they require careful control to stay upright. A conservative way of moving would be for instance to only swing one leg at a time with a quadruped, so that the zero moment point always remains in the support polygon formed by the three remaining legs, or to move in a slow quasi-static manner with a biped to keep it under the sole of the only foot in contact. Besides, the control scheme cannot be limited to kinematics as it can be the case with wheeled robots. Dynamics has to be taken into account to keep

balance. To do so, the controller has to coordinate multiple degrees of freedom, and in some cases properly handle redundancy when there are more controllable degrees of freedom than state variables. This especially tends to be true for humanoid robots with a high number of actuators, by opposition to quadrupeds which are often only equipped with the 12 actuators they need to place their point feet as desired in space with respect to their body. Finally, legged robots are by nature highly nonlinear systems with complex relationships between joint motor commands and robot posture. Because of that, some traditional control approaches that rely on linearity cannot be applied, at least without simplifying assumptions.

1.4 Thesis statement and organization of the manuscript

This thesis contributes to the locomotion of legged robots by developing a control architecture able to exploit the dynamical capabilities of a lightweight quadruped capable of acrobatic movements. The use of complementary filters allows a straightforward sensors fusion for the estimation of the robot state. Binary matrices make it possible to handle contact sequences in a generic way to modify the gait pattern on the fly. This information can then be used to determine footstep locations online using a small set of heuristics. By reasoning on a prediction horizon a centroidal model predictive control can then find out which forces should be applied at contact locations to follow a reference state trajectory and handle disturbances. Next, a whole-body controller translates desired contact forces and swinging feet trajectories into joint trajectories and feedforward torques. Finally, an impedance controller provides feedback torques based on the difference between the desired and current joint positions and velocities to obtain the commands sent to the robot. The modularity of the architecture allows to easily augment some aspects of the scheme or to replace them to test out other methods, as it will be shown several times in this thesis.

I will try to formulate the inner workings of this baseline architecture and how it can be augmented to improve its capabilities. In a second part, I will then display the possibilities offered by this architecture, both in terms of performances and modularity, through simulations and deployments on a real quadruped robot. These demonstrations will enable to experimentally qualify the performances and the feasibility of the presented control scheme.

This thesis is organized in four parts. The first one provides a reviews of legged robots locomotion and introduces the robot platform and the framework of the study. It includes two chapters. Chapter 2 presents a literature overview of the approaches used for legged robots locomotion, based on which our objectives will be positioned. Chapter 3 provides a general introduction to the Solo quadruped that was regularly used over the thesis, as well as the Open Dynamic Robot Initiative project that comes with it.

The second part of the thesis describes the various elements of the control scheme. It is structured in 6 chapters. Chapter 4 provides a brief overview of the nominal control scheme of the quadruped. Chapter 5 describes the two approaches that were implemented to perform sensor fusion for position and velocity estimations. Chapter 6 presents the way the gait is structured and handled as well as two extensions to get an adaptive gait and to perform contact detection respectively. Chapter 7 introduces the heuristics used to choose footstep locations online and how polynomial interpolation can generate trajectories for the swinging feet. Chapter 8 describes the various formulations of centroidal model predictive controller that were implemented to determine the forces that should be applied at contact locations, either to move on the ground or for jumping. Chapter 9 presents the

two whole-body control approaches that were tested out to translate the decisions of the model predictive control and the foot tracking tasks into commands for the quadruped, along with several improvements.

The third part of the thesis presents different results in which the control architecture was deployed. It is structured in 6 chapters. Their presentation follows the chronological order of their development, reflecting the opinions we had at each respective step. Chapter 10 presents the first application of the baseline architecture for trotting at low speed. Chapter 11 presents a comparison of the variants of model predictive controllers introduced in Chapter 8 with the improvements of the whole-body architecture explained in Chapter 9. Chapter 12 showcases how the control architecture can be augmented to use information about complex environments to carefully place its feet. Chapter 13 highlights some preliminary results in simulation using the contact detection of Chapter 6 and the jumping trajectories of Chapter 8. In Chapter 14, an end-to-end deep-learning-based approach is deployed on the Solo quadruped to serve as a form of comparison with the presented model-based architecture.

Finally, a last part presents some conclusions and perspectives of this thesis work.

1.5 Related publications

The work carried out in this thesis has led to the following publications:

- Pierre-Alexandre Léziart, Thomas Flayols, Felix Grimminger, Nicolas Mansard, and Philippe Souères. "Implementation of a Reactive Walking Controller for the New Open-Hardware Quadruped Solo-12". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2021, pp. 5007–5013

- Thomas Corbères, Thomas Flayols, Pierre-Alexandre Léziart, Rohan Budhiraja, Philippe Souères, Guilhem Saurel and Nicolas Mansard
  Thomas Corbères et al. "Comparison of predictive controllers for locomotion and balance recovery of quadruped robots". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2021, pp. 5021–5027

- Pierre-Alexandre Léziart, Thomas Corbères, Thomas Flayols, Steve Tonneau, Nicolas Mansard, and Philippe Souères. "Improved Control Scheme for the Solo Quadruped and Experimental Comparison of Model Predictive Controllers". In: *IEEE Robotics and Automation Letters (RA-L)* (2022)

- Fanny Risbourg, Thomas Corbères, Pierre-Alexandre Léziart, Thomas Flayols, Nicolas Mansard, and Steve Tonneau. "Real time footstep planning and control of the Solo quadruped robot in 3D environments". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2022

- Médéric Fourmy, Thomas Flayols, Pierre-Alexandre Léziart, Nicolas Mansard, and Joan Solà. "Contact Forces Preintegration for Estimation in Legged Robotics using Factor Graphs". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. 2021, pp. 1372–1378

- Michel Aractingi, Pierre-Alexandre Léziart, Thomas Flayols, Julien Perez, Tomi Silander, and Philippe Souères. "Controlling the Solo12 Quadruped Robot with Deep Reinforcement Learning". Submitted to Autonomous Robots. 2022

Chapter 2

Locomotion of quadruped robots: a literature review

Over the last decades a wide range of methods have been developed to perform dynamic locomotion with legged robots. Some of them are bio-inspired and rather intuitive, like central pattern generators, while others have a strong theoretical aspect to achieve formal proofs of stability, like hybrid zero dynamics. These techniques can often be applied to legged robots with various amount of legs, from monopods to octopods. To keep this bibliography to a manageable level, the following description of the state-of-the-art will be limited to developments that have been applied on a quadruped robot.

2.1 Model-based control

2.1.1 CPG-and-reflex based methods

Central pattern generators (CPG) and reflex-based methods are a prime example of how studying the inner workings of motion generation in the natural world can lead to the development of control techniques in robotics. Back in 1994, Taga [Tag94] studies biped locomotion and reveals how locomotor movements emerge as a limit cycle when using a neural rhythm generator. Simulations are used to show that this motion generation approach can adapt in real time to a changing environment and remains stable to some extent, even with delays in the actuation. Taga [Tag95] further strengthens this approach later with the construction of a human musculo-skeletal system centered on 7 pairs of neural oscillators. Although initially applied to biped locomotion, what is now widely known as a central pattern generator is extended to all kinds of legged locomotion in the following years, including quadrupeds. Tsujita [TTO01] presents an adaptive gait pattern control for quadrupedal locomotion that relies on nonlinear oscillators with mutual interactions. The signal from touch sensors located on the feet tunes the phase differences between oscillators which lead to the emergence of an adaptive gait pattern according to the robot state and the environment. Such touch sensors are also used by Righetti [RI08] in a more complex CPG architecture. Using insights from dynamical system theory, generic networks of coupled oscillators are assembled to independently control the swing and stance phases of limbs. This allows to generate various gaits by merely modifying oscillation parameters. As we have seen, CPGs can be deployed on their own to achieve robust locomotion on uneven ground, yet they can also be combined with other

control approaches. Ajallooeian [Aja+13] proposes a modular controller for quadruped locomotion that mixes CPGs with virtual model control for posture control. Virtual springs are attached to the base and generate virtual forces to correct the base attitude. Similarly, Barasuol [Bar+13] combines a CPG-inspired trajectory generation for swinging feet with a null-space based attitude control for the trunk. Proper footholds are found with the principle of instantaneous capture points to naturally counteract disturbances. This approach is augmented a few months later by Focchi [Foc+13] who implements a local elevator reflex so that the robot can reactively overcome high obstacles. He does so by modifying the parameters of the CPG-based online trajectory generation to increase foot clearance. Fukui [Fuk+19] links a simple vestibular sensory feedback (body tilt) to CPGs. The phase difference between the four legs is then changed to autonomously transition from walking to trotting, and then to galloping depending on the speed and the disturbances (obstacles, pulls, additional weights). Massi [Mas+19] applies an evolutionary algorithm to find the best parameters for a CPG-based control architecture. The oscillators are coupled with PID feedback controllers and cerebellar-inspired feedforward controllers. Unlike most CPG controllers, Suzuki [Suz+21] does not use inter-oscillator coupling but rather sensory coupling through bidirectional feedback between the legs and the base. The phases of all oscillators are thus not modified by their states but rather by the torques of the actuators (legs and actuated spine). This sensory feedback mechanism works well to adapt to unexpected bodily damage, such as one of the joint getting stuck in a fixed position. To sum things up, CPG-and-reflex based methods achieve distributed control through a set of oscillators that generates smooth trajectories. By nature, the resulting movements reach a limit cycle that is robust against disturbances. However, there are less mathematical tools to study them than other model-based methods, and no clear design methodology yet.

2.1.2 Heuristic control methods

As their name indicates, heuristic control methods relies on well-defined heuristics to achieve robust locomotion. In 1986, Raibert [RCB86] applies to quadrupeds the virtual leg control method he previously deployed on monopeds. If the movements of the four legs are coordinated, the quadruped gait can be mapped into a gait with a single virtual leg so that the one-leg algorithms can be used. This approach is rather simple to implement with no complex models and can produce highly dynamic motion, yet it requires powerful actuators and has no analytical proof of stability.

Another heuristic approach is the one initially developed for bipeds by Pratt [PDP97]. It creates virtual elements to keep the robot upright and have it move forwards. Then it computes the necessary torques so that the actuators replicate the effect of those virtual elements. This virtual model control is reused by Gehring [Geh+13] to control a small quadruped robot. A set of heuristic terms is defined to choose footsteps location while motion control relies on low-level joint position tracking and virtual forces that should be applied to the base to control the posture. Similarly, Winkler [Win+15] attaches virtual springs and dampers to a HyQ trunk on one side and the desired trajectory on the other one to pull the robot toward the desired state. They generates virtual forces and torques that are then converted into desired floating-base accelerations and ultimately translated into torques through inverse dynamics. Virtual model control has several advantages: it is an intuitive way of designing a controller, as the virtual elements pull the robot to the goal, it does not need an accurate model of the environment and is robust against disturbances. However, one has to make sure that the virtual forces can actually be generated by the

actuators of the robot.

Finally, Chevallereau [Che+03] presents the hybrid zero dynamics approach which provides theoretically-sound control algorithms for walking, running and balancing with a biped. It has the advantage of having one of the most complete theoretical foundation among locomotion techniques, with analytical proofs of asymptotic stability for walking and running gaits on the basis of scalar Poincaré return maps [Str18]. Yet it is not so easy to understand compared to previous approaches. Liu [LSP15] implements such an approach in simulation to perform a bounding gait with a quadruped robot with point feet. In the same way than virtual leg control was applied to quadrupeds even though it was designed for monopeds, here the model is planar so that the two pairs of front and hind legs are each considered as a single leg, thus returning to a biped case.

2.1.3 Trajectory-based methods

The main idea of trajectory-based methods is to design walking kinematic trajectories and use dynamic equations to test and prove that locomotion is stable. Those trajectories were initially designed by trial-and-error or from human recordings but the increase of computational capabilities now makes it possible to optimize them online. One of the most used stability criterion for such methods is the zero moment point, also known as ZMP, that is the point on the ground at which the net moment of the inertial forces and the gravity forces has no component along the horizontal plane [VB04]. It can be roughly considered as the projection of the point around which the robot is rotating. Locomotion is stable if the ZMP remains within the support polygon over time (it can briefly go out). Another criterion that can be used instead for robot with planar feet is the foot rotation indicator (FRI) point [Gos99], which is a point on the foot/ground contact surface where the net ground reaction force would have to act to keep the foot stationary. Thus, the FRI point must remain within the support polygon during motion to ensure no foot rotation. As foot rotation is an indication of postural instability, it should be carefully treated in a dynamically stable walk and avoided altogether for statically stable walks. The position of the center of pressure (CoP), point of application of the ground reaction forces vector, can be used as a stability indicator as well [SB04]. Popovic compares these three criterions (ZMP, FRI, CoP) in [PGH05]. Such trajectory-based approaches provides a well-defined methodology for proving stability, which is well-suited for expensive robots that should never fall. However, it can be time-consuming to define proper trajectories if done by hand and requires an excellent knowledge of the robot dynamics and of the environment. Even with theoretically valid trajectories, additional online control to handle disturbances is needed for movements that are not overly conservative.

Kalakrishnan [Kal+11] presents a control architecture for locomotion over rough terrain with a body trajectory optimizer based on the ZMP criterion. For maximum stability, only one leg moves at a time so that the ZMP can be kept at all time within the support polygon of the remaining three feet. Ugurlu [Ugu+13] proposes a CoP-based center of mass trajectory generator to synthesize reliable trot-walking locomotion cycles which are smooth, continuous and feasible. Along with an active leg compliance controller, it achieves a repetitive dynamic walk on an uneven surface. De Viragh [Vir+19] adopts a linear formulation of the ZMP criterion to generate feasible trajectories for a quadruped robot with actuated wheels on its feet, combining driving, walking and turning. This formulation is exploited by a quadratic programming solver which tries to keep the ZMP inside the support polygon. Tiseo [TVM19] generates center of mass trajectories by con-

sidering a quadruped as two bipeds connected with each other (front pair of legs connected with hind pair) on the basis of a linear inverted pendulum model. The center of mass is constrained within the support polygon for stability.

2.1.4 Planning methods

Planning methods put a particular emphasis on the planning aspect of legged locomotion. They use privileged information about the robot (models) and/or the environment (local map of the surroundings) to reason over a prediction horizon in order to take the best control decisions for a given set of criterion. Based on an accurate 3D model of the terrain, Kolter [KRN08] generates height and collision maps that allow to extract several local features of the ground such as the slope or the maximum height. This leads to the creation of a foot cost map through a linear combination of the features and of the collision map. Future footsteps can ultimately be planned along the desired body path by solving an optimization problem. Havoutis [Hav+13] uses a depth camera to reconstruct the height map of the ground in front of the robot from a point cloud. That way the quadruped can avoid stepping near sudden surface height changes (like steps) by computing the surface gradient and rather prefers to step on flat spots along the body path. Similarly, Mastalli [Mas+15] reconstructs a local height map of the environment in front of the robot using a depth camera which is then converted into a reward map, with large flat areas having a high reward and the edges of height changes having a low one. A body action planner then computes a sequence of body actions that maximize the cross-ability of future footsteps. The action plan is found online by searching over a graph built using a set of predefined body movement primitives. Winkler [Win+15] provides more details on how the planned sequence of footsteps is processed to generate a body trajectory that ensures that the robot is dynamically stable. The position of the ZMP is estimated by modeling the robot as a cart-table. Aceituno-Cabezas [Ace+17] later extends this control framework by introducing a mixed-integer convex formulation to plan simultaneously contact locations, gait transitions and motion. Farshidian [Far+17] proposes an optimal planning and control framework for quadrupedal locomotion. He deploys a multi-level optimization approach based on dynamic programming to find both the optimal times for contact switches and the optimal continuous control inputs to perform the motion. The approach can handle wide gaps where the robot cannot put its feet, forcing the robot to leap over the hole. Fernbach [Fer+17] presents a kinodynamic contact planner of legged robots. The planning pipeline can generate trajectories connecting two states of the robot (start and end locations for instance) while accounting for the state-dependent centroidal dynamic constraints inherent to legged robots. It exploits a 3D map of the environment to synthesize collision-free motions that respect the reachable workspace of the effectors and can include jump phases. Zhang [ZH18] investigates a single-image footsteps and route planner for legged robots based on perspective and tilt-corrected color and depth images. To do so, they deploy a convolutional neural network that processes the image to find the best footholds and handholds (it also works for climbing walls) after a training using human expert knowledge. Based on the quality of the observed holds, it then outputs a rough route to cross the terrain. Fankhauser [Fan+18] proposes a real-time motion planning pipeline that reconstructs an elevation map of the environment based either on depth camera or LIDAR data. He exploits the full motion range of the robot by optimizing its pose along the footstep selection. The planner continuously re-plans the motion to handle disturbances and dynamic environments, yet its prediction horizon is limited to

the immediate next step. Finally, Brandao [BFH19] showcases a locomotion planner in an outdoor industrial environment. His main planner has access to several controllers and sub-planners and decides which ones to apply on each section of the route to the goal in order to maximize objectives. It can for instance choose to trot on a clear flat ground to save energy while switching to a planner that carefully plan footsteps in complex situations like stepping stones. To sum things up, planning methods offer ways to handle very complex terrains that require careful footholds, yet they depends on accurate maps of the ground, with the risk of degraded performances if it is reconstructed online in the case of poor sensory inputs.

2.1.5 Inverse Dynamics, optimization and predictive control

Inverse dynamics allows to design controllers in task space, as opposed to joint space. Instead of reasoning in terms of joint states, one can define tasks in Cartesian space such as placing feet at given positions or keeping the base horizontal while tracking a reference velocity. These tasks are then translated by inverse dynamics into commands at the joint level. However, practical applications of floating-base inverse dynamics can be hindered as it depends on precise dynamics models. Buchli [Buc+09] presents an approach that avoids the need to know the contact forces by computing analytically correct inverse dynamics torques in the reduced dimensional null-space of the constraints. This is done by computing an orthogonal decomposition of the constraint Jacobian that appears next to the vector of contact forces in the equation of the dynamics. Combining inverse dynamics with some sort of model predictive control (MPC) is a well-spread approach to optimize the state and control of the robot over a prediction horizon. It can either directly use a full model of the robot with the equation of the dynamics, which is called whole-body predictive control, or a reduced model for computational reason such as centroidal model predictive control. The output of predictive control with a reduced model is often reprocessed down the line with an instantaneous whole-body controller to abide by the full dynamics of the robot. For instance, Bledt [BWK17] performs a policy-regularized MPC with a lumped-mass model with massless legs. Both contact forces and contact locations are optimized over an horizon of one gait period using heuristics for regularization. A lower level whole-body controller then handles the tracking. This work manages to stabilize a wide variety of gaits (bounding, trotting, galloping) in simulation. A similar two-stages architecture is used by Bellicoso [Bel+17; Bel+18] to first optimize the center of mass trajectory over the incoming steps to respect the ZMP stability criterion. Then, a whole-body controller tracks the desired motion of the floating base and swinging legs by reasoning on a hierarchy of tasks to find the contact forces to be applied on the ground, which are ultimately converted into joint torques. This approach is further extended in [Bel+19] to perform tasks with a six degrees of freedom robotic arm installed on the trunk of the quadruped. On the opposite, Neunert [Neu+18] proposes a single-stage architecture with a whole-body nonlinear model predictive control. He optimizes contact locations and timings along the full body dynamics over a half-a-second prediction horizon to perform trotting, squat jumps and forward jumps. The feat of reaching real-time performances with a whole-body MPC is made possible through multiple software engineering techniques such as auto-differentiation and multi-threading using a multi-shooting solver. Di Carlo [Di +18] uses a centroidal MPC to find the contact forces that should be applied at contact points over the prediction horizon to follow a reference velocity. Assumptions and simplifications of the full body model allow to formulate the

problem as a convex quadratic programming optimization. Inverse dynamics is still used at the low level for the control of the swinging legs. The resulting architecture demonstrates robust locomotion with several gaits (trotting, pronking, bounding, gallop, flying trot) and the same set of gains and weights. Kim [Kim+19] refines this approach by keeping the same centroidal MPC but extending the whole-body control to include a strict hierarchy of tasks. These tasks generate position, velocity and accelerations commands for the base and the joints through a series of projection in the null-space of tasks of higher priority. The final contact forces are then optimized by a quadratic program considering the reaction forces found by the MPC and the base and joint accelerations outputted by the task hierarchy while satisfying the equation of the full body dynamics. The whole architecture achieves high speed dynamic locomotion with aerial phases. Fahmi [Fah+19] presents a passive whole-body control for quadruped that optimizes body and joint accelerations as well as the ground reaction forces with a quadratic program. Virtual springs and dampers are used for trunk and swing leg control tasks, the reference being given by a higher-level planner that optimizes footstep locations and center of mass trajectory based on a reconstructed height map of the environment. The controller is said passive in the sense that the total energy stored in the controller is bounded from below and its derivative is less than or equal to the rate of energy injected by the control. The architecture achieves robust locomotion over a wide range of terrains (slopes, gaps, stairs) with crawling and trotting gaits. Villarreal [Vil+20] combines a convolutional neural network that continuously evaluates the terrain in search of safe footstep locations with a centroidal model predictive controller that optimizes contact forces at said locations. The wrench exerted by the legs during swing phases is taken into account in the MPC, which distinguishes it from usual centroidal MPCs which neglect leg inertia. Hamed [HKP20] proposes an event-based MPC that computes the optimal center of mass trajectories for a reduced-order linear inverted pendulum model. Here, events define the switch from one continuous domain to another, in other words the contact switches of the performed trotting gait. The linear inverted pendulum nature of the MPC has the advantage of enabling a formal asymptotic stability analysis. Full body dynamics and optimal trajectory tracking are ensured by a QP-based virtual constraints controller. Fawcett [Faw+22] investigates the use of data-enabled predictive control to capture nonlinear information about the classic lumped-mass model while avoiding linearization. It does so through a data collection phase which leads to the construction of Hankel matrices that contain implicit information about the dynamics of the system. Those are then used in a predictive control framework to plan trajectories for the center of mass and ground reaction forces. Optimal trajectories are finally passed to a low-level QP-based nonlinear controller for whole-body motion control using virtual constraints. Indoor and outdoor walks demonstrate the robustness of the approach. Finally, to go beyond simplified models and kinodynamic constraints that are often non-differentiable, Mitchell [Mit+20] captures a statistical representation of feasible joint configurations to form a structured latent space. Through the use of semantic indicators and learned classifiers, constraints are made differentiable and performing motion optimization amounts to finding a trajectory in this latent configuration space. The architecture is successfully deployed to perform a walking gait on a quadruped robot. Both Fawcett [Faw+22], Mitchell [Mit+20] and other approaches in recent years start to bridge the gap between more traditional control viewpoints and novel machine learning perspectives.

2.2 Data-based control

In the last decade, the rise of machine learning has been an increasingly hot topic for legged locomotion. It opens a whole new paradigm where, instead of being formulated by hand, the locomotion model is learned from data either online or during a training phase. It potentially allows to capture effects that would be hard to model by hand or which would be untractable for the considered model-based architecture, like non-linearity or non-convexity. Data-based models can go from full end-to-end networks with sensors as inputs and joint commands at output, or be limited to a small part of a broader control architecture. For instance, Pontón [PFB14] deploys machine learning as part of an architecture that contains classic inverse kinematics, inverse dynamics and trunk stabilization. The goal is there to optimize the robustness and energy efficiency of a trotting gait through a variable impedance policy for the four legs. The final policy learned at discrete speeds is then generalized for the whole speed range of the quadruped with a Gaussian process for interpolation. This approach is later extended by Heijmink [Hei+17] by learning not only the impedance profile over time, but also the gait parameters and gains of the trunk controller to further diminish the energy consumption of the quadruped when trotting over a set of terrains. Gehring [Geh+16] presents a learn-through-practice process to automatically fine tune the parameters of a model-based state-feedback controller. The training is done in simulation to avoid damaging the robot. Yet, an accurate model of the compliant actuation system of the quadruped is implemented to keep the sim-to-real gap as small as possible. In the end, the optimization results were directly applicable to the real platform. On the contrary, Tan [Tan+18] adopts an end-to-end approach that directly learns target joint position to perform trotting and galloping gaits. The training is done in simulation, with a particular care given to the actuator model and a simulation of control latency, again with the intent to narrow the reality gap. The policy is ultimately deployed on a real quadruped. Sun [Sun+18] investigates biologically inspired controllers by applying learning to a CPG-based architecture. The controller includes decoupled CPGs (one per leg) with sensory feedback and neural reflex mechanisms. The network is ultimately able to generate emergent quadruped locomotion and to adapt its self-organized gait to the presence of step-like obstacles on the ground. Instead of traditional neural networks, Lee [LP19] explores the use of a time-dependent genetic algorithm to improve the convergence performance of functions over time compared to real-coded genetic algorithms. They optimizes the shape of the feet trajectories as well as the parameters of the impedance controller of the joints. The quadruped is then able to handle flat, sloped and slightly irregular terrain in simulation. Magana [Mag+19] presents a planning architecture that uses vision to reconstruct a height map of the environment and to choose online the best landing positions for the feet. The choice of these positions was previously guided by heuristics but is now done with a convolutional neural network that processes the discrete height map. The goal is to replicate the performance obtained with complex heuristics, but with a computational cost hundred of times lower for real-time purpose. Similarly, Klamt [KB19] leverages a discrete height map of the environment with a convolutional neural network to replace the cost function of a navigation task that could require important tuning efforts. After training, the network outputs the cost of each cell of the height map, which is exploited by a search algorithm to find the best path to the goal location. We have already seen how motion primitives could be used by a trajectory based control to generate a quadrupedal walking gait. Instead of considering a given set of primitives, Singla [Sin+19] proposes to learn them. A deep reinforcement learning architecture first

learns to perform several walking gaits in simulation. From them kinematic motion primitives are extracted through principal component analysis. The trajectory based controlled finally synthesizes walking gaits by reconstructing joint trajectories from the motion primitives. Back to an end-to-end learning approach, Jain [JIC19] implements a hierarchy of two neural networks to perform a path following task on a flat ground. The high-level policy network receives the position and orientation of the quadruped as inputs and outputs latent commands and a duration during which the low-level policy should follow these commands. Based on these latent commands and proprioceptive data, the low-level policy issues target joint positions. After training, the robot can adapt to new paths thanks to a steering behaviour that automatically emerges in latent space. Jain further develops this approach in [JIC20] by replacing the position and orientation inputs of the high-level network by depth camera images so that the network learns the latent commands to navigate curved cliff and maze environments. Bhattacharya [Bha+19] proposes a end-to-end learning as well using a single deep neural network with the intent to perform bounding with an active spine. While the policy learns the target joint angle for the spine, it does not do so for the legs. The action space consists instead of the feet positions expressed in polar coordinates with respect to their associated shoulder so that the five-bar leg mechanisms never encounter a singularity. Finally, Ji [Ji+22] investigates the use of two concurrent networks: a locomotion network to ouput joint target positions and an estimation one to estimate the robot state such as base velocity, foot height and contact probability. Training is done in simulation to exploit privileged data from the simulator for the estimation network. Once deployed on real hardware, the quadruped is able to cross rough terrains such as hills, slippery areas and bumpy roads.

The next chapter presents the Solo-12 quadruped robot and the collaborative open framework in which it was developed.

Chapter 3

The Open Dynamic Robot Initiative

3.1 A project toward open robotics

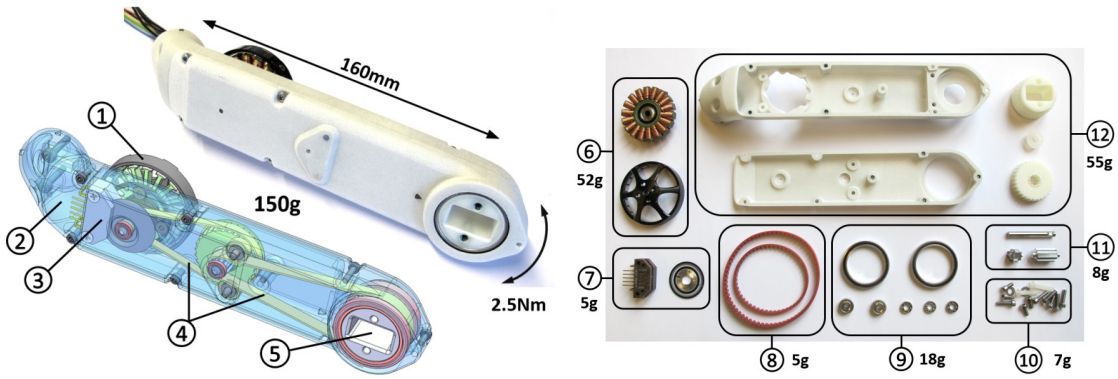
The very early days of what will be known later as the Open Dynamic Robot Initiative started back in 2016 at the Max-Planck Institute for Intelligent System [MPI]. The first investigations aimed to explore the possibilities offered by a wide range of small electric motors and find the best trade-off between their characteristics. This was done using a closed-source Texas Instruments evaluation board [TXA]. That was done alongside others developments intended to test the performance and mechanical design of compact reduction techniques using timing belts and 3D printing for the casing and other plastic parts.

The ODRI project itself started in 2019 as a collaboration between the Motion Generation and Control Group [MGC], the Dynamic Locomotion Group [DLG] and the Robotics Central Scientific Facility [RCS] at the Max-Planck Institute for Intelligent System [MPI], the Machines in Motion Laboratory [MML] at New York University's Tandon School of Engineering [NYU] and the Gepetto Team [GT] at the Laboratory of Architecture and Analysis of Systems LAAS-CNRS [LAA]. This project originated in an effort to build a low cost and low complexity actuator module using brushless motors that can be used to build different types of torque-controlled robots with mostly 3D printed and off-the-shelves components. This module, and extensions, can be used to build legged robots or manipulators, with the intent to provide the community with reliable platforms that can be easily maintained and repaired and could benefit from numerous contributions in their development.

All hardware, schematics, electronics, firmware and software are fully available under the BSD-3-clause license. The entry point of the resources made available is [ODR]. Both a paper describing the actuator module and the first quadruped design [Gri+20] and a paper describing the TriFinger Manipulator Platform and real-time reinforcement learning experiments [Wüt+20] have been published in 2020.

The actuator module consists of a brushless motor connected to a 9:1 dual-stage timing belt transmission to be able to output a reasonable peak torque and high velocity without going over-the-top. The actuator can output 2.7 Nm joint torque at 12 A. Keeping a low transmission ratio ensures sufficient transparency to enable accurate torque control through motor current measurements alone as well as offering enough reversibility for the actuator to absorb and dampen impacts. This would potentially even allow to send energy back to the batteries, contrary to the actuators with high reduction ratio found

on big quadruped or humanoid robots whose gearboxes are way less reversible and can even break for strong impacts. A high-resolution optical encoder and a 5000 count-per-revolution coding wheel mounted on the motor shaft provide joint position measurements. With such an incremental encoder, a short initially procedure is needed at startup to locate the index of the wheel, contrary to absolute encoders. The whole actuator weights 150 g for a segment length of 16 cm. Except the motor shaft and timing belts, all parts are either 3D printed or can be bought off-the-shelf. An assembled view of the actuators and of its individual parts is shown in Fig. 3.1.



(a) Assembled actuator module (b) Component overview

Figure 3.1: Brushless actuator module (a) assembled, and (b) individual parts. Brushless DC motor (1), two-part 3D printed shell structure (2), high resolution encoder (3), timing belts (4), and output shaft (5). Brushless motor (6), optical encoder (7), timing belts (8), bearings (9), fasteners (10), machined parts (11) and 3D printed parts (12). Figures extracted from [Gri+20].

Rather than using the TI evaluation board we evoked earlier, new open-source driver boards have been developed to execute dual motor torque control with a mass reduced by a factor of six (13 g) and a volume by a factor of 10 (23 cm^3), as shown in Fig. 3.2.

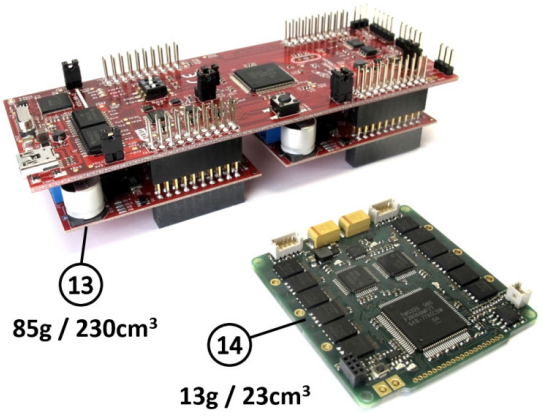


Figure 3.2: TI Evaluation Board (13) compared to the open-source micro-driver developed by ODRI (14). The built-in microprocessor can run an impedance controller at 10 kHz for better performances compared to a controller done at 1 kHz by the control computer. Figure extracted from [Gri+20].

The drivers can run an onboard impedance controller at 10 kHz and operate at motor voltages up to 40 V. The driver boards are managed by a single master board which han-

dles communications with the control computer, either wired or wireless. An overview of the control system is presented in Fig. 3.3.

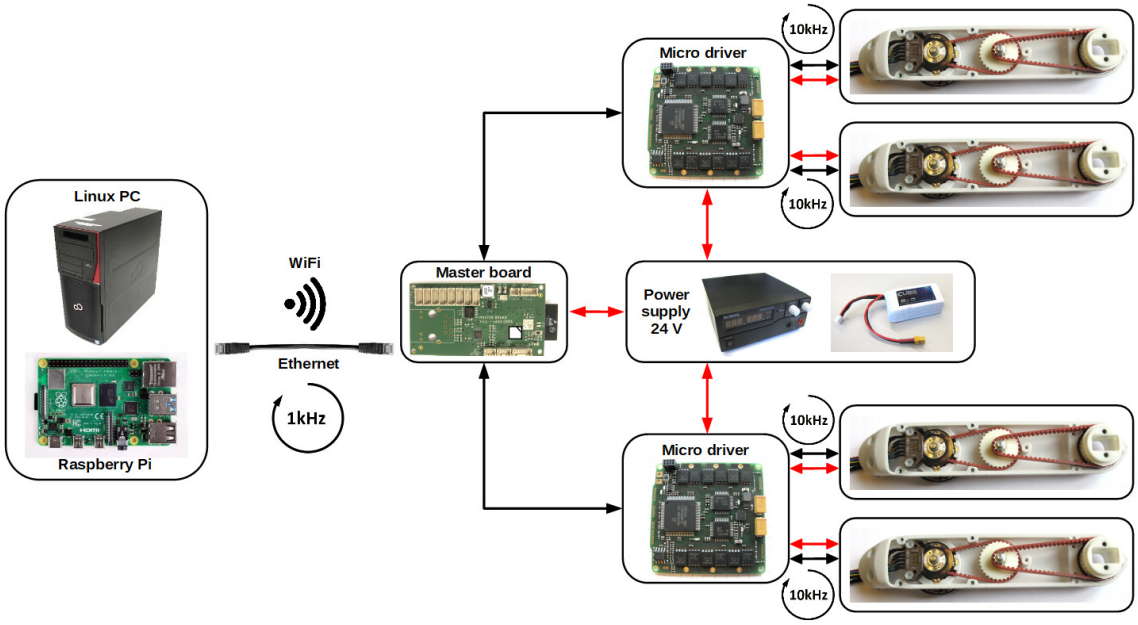
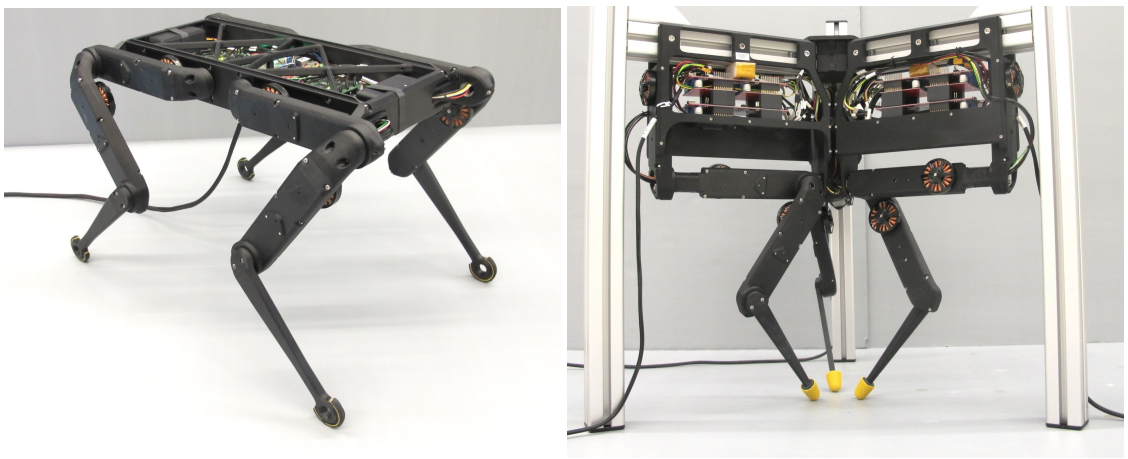


Figure 3.3: Control system overview. An ethernet cable or WiFi can connect a control computer to the master board at 1 kHz. The master board manages all micro drivers that each control 2 actuators at 10 kHz. Power supply can either be wired or on-board with batteries.

As we hinted at before, the two first platforms built in the framework of ODRI were the Solo-8, a lightweight (1.9 kg) quadruped with 8 degrees of freedom (2 per leg), and the TriFinger, a manipulator platform with 3 identical finger modules with 3 actuators each. Both platforms are shown in Fig. 3.4.

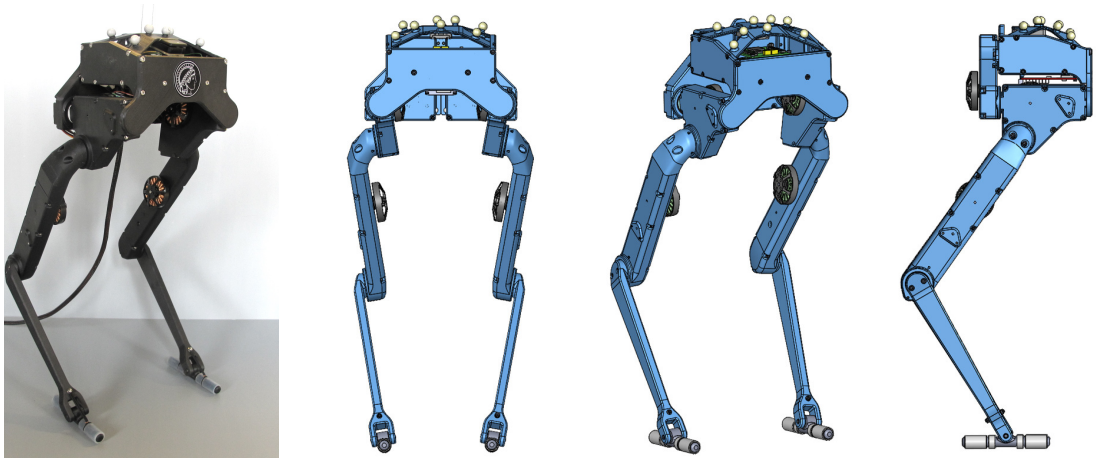


(a) Solo-8 quadruped (b) TriFinger Manipulator Platform

Figure 3.4: Both platforms highlight the possibilities offered even with standardized actuation modules, either for legged robotics or for object manipulation. Figures extracted from [ODR].

Further developments in 2020 led to a new version of the Solo quadruped called Solo-12 due to its 12 actuators (3 per leg), which will be described in more details in the next section. After working on a more compact version of the actuator by placing the timing belts in a different way, a new biped platform was made available in 2021. It weights

1.34 kg with 3 actuators per leg and a passive ankle joint. As shown in Fig. 3.5, the mechanical design keeps the same modular philosophy: the hip abduction-adduction and hip flexion-extension are both the same triangle-shaped actuators while the legs are just an elongated version of Solo's ones. Since the biped has no degree of freedom for the leg rotation, the robot cannot control its yaw angle. The passive ankle joint with a foot-like end-effector aims to help stabilize the yaw orientation of the robot with a line contact, contrary to what we would get with one of Solo's feet (point contact). It uses the same onboard electronics than the other ODRI platforms, with a master board to communicate with the control computer and microdrivers that command 2 actuators each.



(a) Biped (held by a string) (b) CAD model of the biped

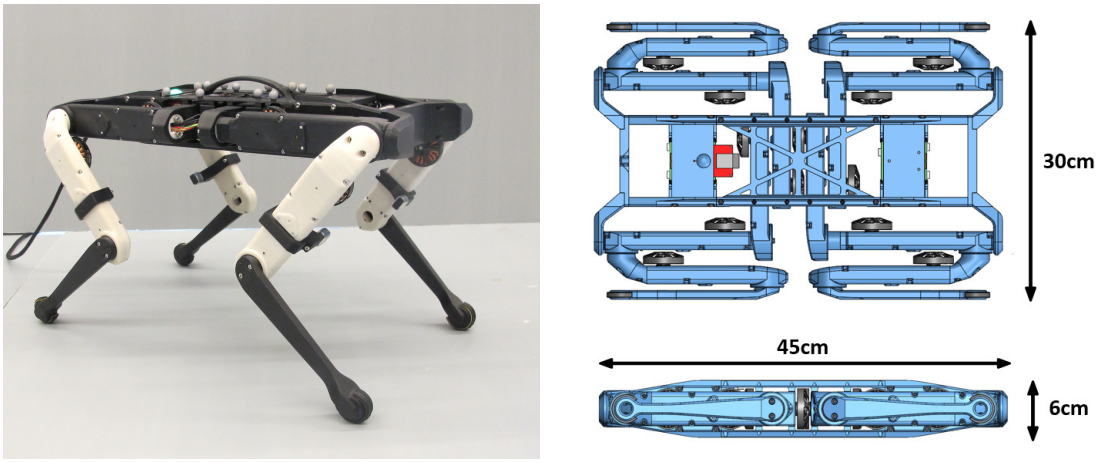
Figure 3.5: Design of the 6-DoF biped robot. The reflective markers are used for base estimation with motion capture. Figures extracted from [ODR].

3.2 The Solo-12 quadruped

Solo-12 is an upgraded version of Solo-8 with 1 additional actuator per leg. It is otherwise very structurally similar to its predecessor, as seen in Fig. 3.6. Compared to Solo-8 which had only 2 actuators per leg for the hip flexion-extension and the knee and could thus only move its feet in a vertical plane, Solo-12 gets rid of this limitation thanks to hip abduction-adduction. This greatly expands the feet workspace and thus the robot capabilities as it can now move sideways and better stabilize itself. It also opens more possibilities for the control of the base orientation. The internal electronics remains similar to the one of Solo-8, with only two additional micro-drivers to handle the four added actuators. The workspace of each leg is highlighted in Fig. 3.7.

An autonomy upgrade designed in 2021 allowed the robot to operate without power and Ethernet wires. Two lithium batteries, one at the front, one at the back, provide the required 24 V. Space is made on top of the robot to attach a small computer board like a Raspberry Pi to perform onboard computations. A custom power management board safely manages the battery packs and can record energy consumption data (voltage, current, power). The fully autonomous Solo-12 we used at LAAS for experiments is shown in Fig. 3.8.

Preliminary tests with a replay of a hand-made jumping trajectory reveals the explosivity of the Solo platform, with the base going up to 1.06 m from the ground. As shown



(a) Default configuration (b) CAD model and dimensions

Figure 3.6: Design of the 12-DoF version of the Solo quadruped. Figures extracted from [ODR].

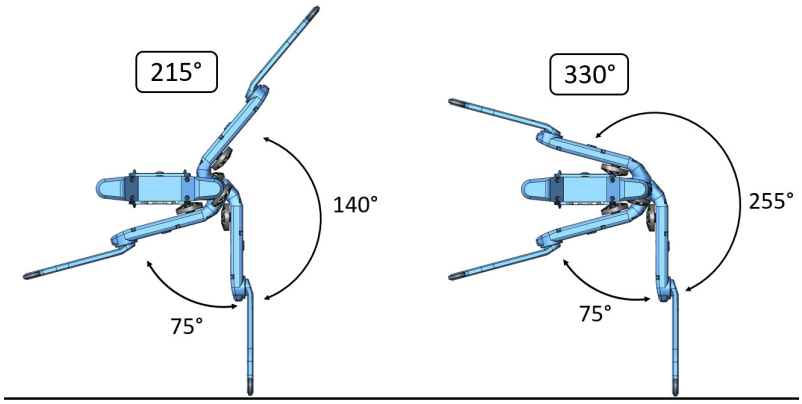


Figure 3.7: The standard range of motion of the hip abduction-adduction is 215 deg. It extends up to 325 deg if the hip flexion-extension is moved as well. Figure extracted from [ODR].

in Fig. 3.9, these experiments were done with Solo-8. It remains to be seen how high Solo-12 can jump with the added weight of 4 actuators, likely depending on how well the abduction-adduction of the legs can be used for jumping. The Solo platform has promising dynamical capabilities, now all we need is a control architecture to exploit them, which will be the subject of the rest of this thesis.

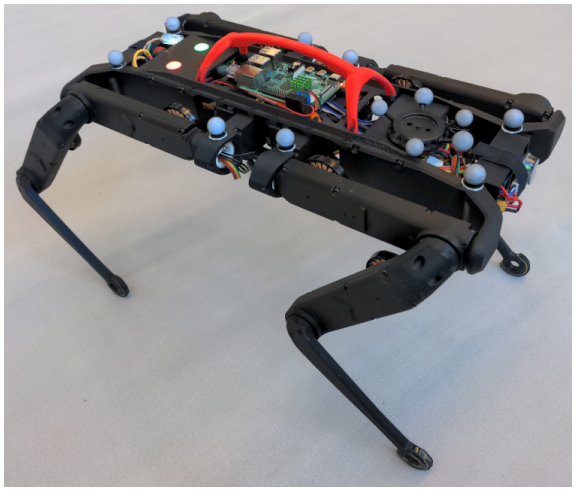


Figure 3.8: The autonomous version of Solo-12 equipped with motion capture markers. Note that the knees are reversible by design: here the hind knees are reversed compared to Fig. 3.6a.

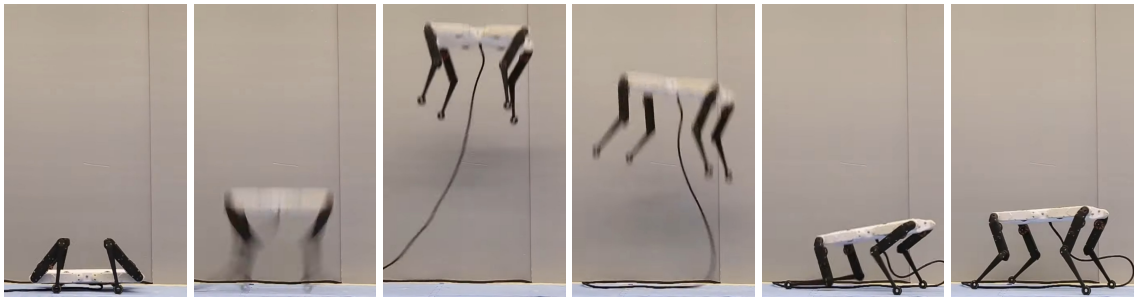


Figure 3.9: Snapshots of a vertical jump with Solo-8. Figures extracted from [JMP].

Part I

Control Architecture

Chapter 4

Overview of the control architecture

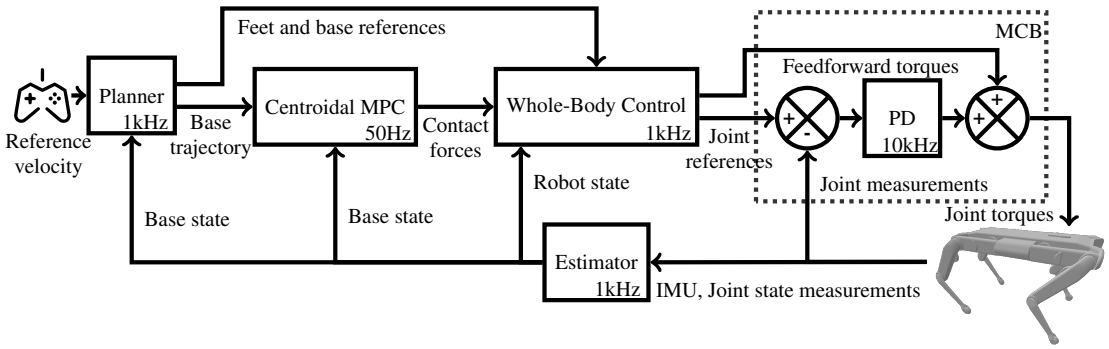


Figure 4.1: Nominal reactive walking control architecture. The low-level proportional-derivative (PD) controller is directly performed by the motor control board (MCB) of the robot.

This chapter provides a brief overview of the nominal control scheme of the quadruped, displayed in Fig. 4.1, to which we will refer when describing how the main control blocks have evolved over the duration of this thesis. This overview outlines the role of each of these blocks to make it easier for the reader to get a general grasp of how the control architecture is structured before digging deeper into technical developments in the following chapters.

The main goal of this control architecture is to track a reference base velocity specified either by a user or a higher level controller. The only other inputs that the architecture receives are the sensors measurements that come from the joint encoders and the inertia measurement unit. The controller processes these inputs to ultimately output torque commands for the 12 actuators of Solo-12. To do so, the computations are split into 5 control blocks that communicate with each other and that we will describe thereafter.

- **Estimator**

The aim of the estimator is to reliably estimate the state of the robot and give us access to quantities that are not directly measured by the joint encoders and the inertia measurement unit. With the encoders, we can get accurate joint positions as well as slightly noisy joint velocities by using finite difference. The IMU includes an accelerometer and a gyroscope to output linear acceleration, angular velocity and orientation of the robot body. With this set of sensors, only the position and linear velocity remain to be assessed through

other ways. To do so, the estimator exploits sensor fusion by using either a cascade of complementary filters or a Kalman filter. If feet in contact are assumed to be immobile, we can use forward kinematics with joint positions and velocities to get an estimate of the base velocity. Forward geometry also outputs an estimate of base position with respect to the feet. The low-accuracy non-drifting assessments coming from forward kinematics and geometry are merged with highly-accurate IMU acceleration measurements that would drift when integrated alone, hence resulting in a good quality non-drifting estimates of base position and linear velocity.

- **Planner**

The role of the planner is to manage to rhythmic motion of the legs to make the robot follow the base reference velocity. Discretized contact sequences are used to characterize the succession of contacts between the feet and the ground. In other words, the planner manages the sequences of stance and swing phases for each foot that lead to a cyclic pattern of footsteps for locomotion. During swing phases, feet have to be guided from their current position on the ground to the next one. So the planner regularly has to chose at which location each foot should land at the end of its respective swing phase. To do so, it uses several heuristic terms to determine the target locations on the ground based on the estimated and reference base velocities. These locations are of particular importance because pushing on the ground is the only way for the quadruped to interact with its environment, balance itself and move around. Finally, while feet in contact must stay immobile on the ground, polynomial interpolation is used to generate reference trajectories in position, velocity and acceleration to guide swinging feet from their current position to their next target on the ground.

- **Model predictive control**

Model predictive controllers can generate motion in real time by predicting the behavior of the robot over a prediction horizon. That way, locomotion decisions are taken by considering the future evolution of the system and incoming events (contact locations and timings, disturbances, environment, ...). To reduce computational complexity, our MPC uses a centroidal model of Solo-12. Since quadruped robots tend to have lightweight limbs, most of their mass is localized in their trunk and, as such, centroidal dynamics can provide an appropriate approximation of their whole-body dynamics. First, a reference state trajectory of the base is generated from the reference base velocity given as input to the control architecture. Then, along with the footstep locations decided by the planner and the estimated state of the base, this reference trajectory is given as input to the centroidal MPC. Its goal is to output desired contact forces that should be applied on the ground to track the reference. Several approaches of increasing complexity are implemented to solve this optimal control problem. Due to its computational cost, the MPC only runs at 50 Hz in a process parallel compared to the main control loop at 1 kHz. For this reason, MPC inputs and outputs are only refreshed once every 20 iterations.

- **Whole-body control**

While the centroidal model of the MPC forgets all notion of joints to consider the robot as a single lumped mass, in practice, the robot is a poly-articulated system with multiple joints that have to be controlled. Thus, the role of the whole-body controller

is to convert the desired contact forces provided by the MPC for feet in stance phase and the reference feet position, velocity and acceleration given by the planner for feet in swing phase, into torque, position and velocity commands that are sent to the low-level impedance controller. Ideally, this conversion would have been done by using inverse dynamics. Yet, we will present the practical reasons that led us to use a combination of task-oriented inverse kinematics (IK) and quadratic programming (QP) for real-world deployment. In fact, a QP problem is solved to find a compromise between tracking the joint acceleration commands of the IK, taking into account the MPC contact forces and respecting the equation of dynamics.

- **Low-level impedance controller**

The low-level impedance controller is the last interface between our control architecture (1 kHz) and the motor boards that drive the joints with a high-frequency current loop (10 kHz). It provides feedback torques based on the difference between the desired and current joint positions and velocities, that are added to the feedforward torques of the whole-body controller to obtain the final torque commands sent to the motors. The impedance controller directly runs on the robot control board at 10 kHz with references from the control loop refreshed at 1 kHz.

Chapter 5

State estimation

Contents

The state of a system consists of a minimal set of parameters that allows to completely describe its dynamics over time. When its initial state is known and a given command is applied, the trajectory of the system is defined in an unique manner. On a quadruped robot, the state is typically composed of the joint angles and velocities together with the base position, orientation, linear and angular velocities. As a body reference point, the center of mass (CoM) is sometimes preferred over the geometric center of the base depending on control strategies. Some of those quantities may not directly be measurable due to their physical nature (the CoM is a virtual point) or to limited sensor data. For instance, proprioceptive sensors are insufficient for measuring the robot position in the world. Those variables might however be estimated through other means. For instance, forward geometry provides an estimate of the CoM position, while sensor fusion can indirectly reconstruct the position in the world. The task of estimation can thus be summed up as finding the robot state given available measurements.

In our case, the state of the quadruped has been defined as a configuration vector $\mathbf{q}$ which includes base position $\mathbf{q}_{lin}$, orientation $\mathbf{q}_{ang}$ and joint angles $\mathbf{q}_a$, plus a velocity vector comprising base linear velocity $\dot{\mathbf{q}}_{lin}$, angular velocity $\dot{\mathbf{q}}_{ang}$, and joint velocities $\dot{\mathbf{q}}_a$. Subscripts $_u$ and $_a$ denote the underactuated (free-flyer) and actuated (joints) parts of the state respectively.

$$\mathbf{q} = \begin{bmatrix} \mathbf{q}_u \\ \mathbf{q}_a \end{bmatrix} = \begin{bmatrix} \mathbf{q}_{lin} \\ \mathbf{q}_{ang} \\ \mathbf{q}_a \end{bmatrix} \quad \text{Configuration vector} \tag{5.1}$$

$$\mathbf{q}_{lin} = \begin{bmatrix} x & y & z \end{bmatrix}^T \quad \text{Base position} \tag{5.2}$$

$$\mathbf{q}_{ang} = \begin{bmatrix} \phi & \theta & \psi \end{bmatrix}^T \quad \text{Base orientation (roll, pitch and yaw angles)} \tag{5.3}$$

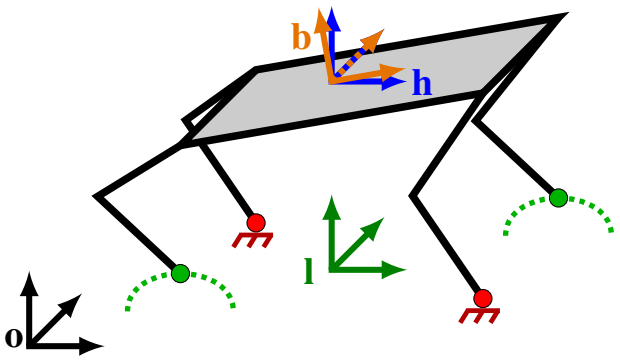


Figure 5.1: Four frames are used throughout the control architecture: **w**orld, **l**ocal, **h**orizontal and **b**ase frames.

Depending on the needs, $\mathbf{q}_{ang}$ is sometimes expressed as a quaternion instead of Euler angles. We mostly use the Euler representation in the architecture even if there are a few back and forth with the quaternion representation when calling dynamics libraries that requires it. As a result, the size of the configuration vector varies between 18 and 19 for our 12 degrees of freedom (DoF) Solo quadruped.

$$\dot{\mathbf{q}} = \begin{bmatrix} \dot{\mathbf{q}}_u \\ \dot{\mathbf{q}}_a \end{bmatrix} = \begin{bmatrix} \dot{\mathbf{q}}_{lin} \\ \dot{\mathbf{q}}_{ang} \\ \dot{\mathbf{q}}_a \end{bmatrix} \quad \text{Velocity vector} \tag{5.4}$$

$$\dot{\mathbf{q}}_{lin} = \begin{bmatrix} \dot{x} & \dot{y} & \dot{z} \end{bmatrix}^T \quad \text{Base linear velocity} \tag{5.5}$$

$$\dot{\mathbf{q}}_{ang} = \begin{bmatrix} \dot{\phi} & \dot{\theta} & \dot{\psi} \end{bmatrix}^T \quad \text{Base angular velocity} \tag{5.6}$$

Contrary to the configuration vector, the size of the velocity vector is always 18 in our case as both the derivative of quaternion and Euler representations lead to angular velocities around the x, y and z axes.

Three frames will be mainly used throughout the control architecture: an absolute world frame, a horizontal frame located at the center of the robot body, with its two first axes in the horizontal plane and aligned in yaw with the body, and a base frame located at the center of the body and aligned with it. The first axis of the base frame points in the forward direction of the body while the second one points laterally to the left. World, horizontal and base frames are noted with o, h and b respectively. A fourth frame, noted the local frame l, is used for the model predictive controller and is the equivalent of the horizontal frame at ground level (see Fig. 5.1). Translation vectors $\mathbf{T}$ and rotation matrices $\mathbf{R}$ between these frames are thus defined as:

$$^o\mathbf{T}_l = \begin{bmatrix} x & y & 0 \end{bmatrix}^T \tag{5.7}$$

$$^o\mathbf{R}_l = \mathcal{R}(0, 0, \psi) \tag{5.8}$$

$$^l\mathbf{T}_h = \begin{bmatrix} 0 & 0 & z \end{bmatrix}^T \tag{5.9}$$

$$^l\mathbf{R}_h = \mathcal{R}(0, 0, 0) \tag{5.10}$$

$$^h\mathbf{T}_b = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \tag{5.11}$$

$$^h\mathbf{R}_b = \mathcal{R}(\phi, \theta, 0) \tag{5.12}$$

$\mathcal{R}(\phi, \theta, \psi)$ denotes the 3 by 3 matrix that applies a rotation in roll, pitch and yaw.

The Solo quadruped is equipped with two kinds of sensors for estimation purpose: an Inertial Measurement Unit (IMU) attached to the body and incremental encoders at each joints. Those proprioceptive sensors do not allow to directly measure the whole state of the robot. Incremental encoders provide a measurement of joint angles $\mathbf{q}_a$ as well as joint velocities $\dot{\mathbf{q}}_a$ through finite difference. The IMU includes an accelerometer and a gyroscope to output linear acceleration $\ddot{\mathbf{q}}_{lin}$, angular velocities $\dot{\mathbf{q}}_{ang}$ and orientation $\mathbf{q}_{ang}$. An in-built Kalman filter automatically debiases those quantities with respect to the gravity vector and provides linear acceleration without gravity.

With this set of sensors, only the position $\mathbf{q}_{lin}$ and linear velocity $\dot{\mathbf{q}}_{lin}$ remain to be assessed through other ways.

5.1 Sensor fusion for position and velocity estimation

The goal of sensor fusion is to indirectly assess base position $\mathbf{q}_{lin}$ and linear velocity $\dot{\mathbf{q}}_{lin}$ as they are not directly measurable by the set of sensors available on the quadruped robot. If feet in contact are assumed to be immobile, we can use forward kinematics with joint position $\mathbf{q}_a$ and velocity $\dot{\mathbf{q}}_a$ measured by the encoders to get an estimate of the base velocity. We define the set $\mathcal{M}$ of indexes of feet that have been in contact for long enough to assume they are properly set on the ground:

$$\mathcal{M} = \{i \in \{1,2,3,4\} \text{ s.t. } i\text{-th foot in contact for more than } m_{ctc} \text{ iterations}\} \tag{5.13}$$

The position ${}^b\mathbf{T}_i$, orientation ${}^b\mathbf{R}_i$ and linear velocity $\dot{\mathbf{x}}_i$ of the i-th foot in its own frame is evaluated by considering an immobile base with moving joints. Then, we get an estimate of the base velocity from the average of all velocities obtained for each feet in $\mathcal{M}$.

$$\forall i \in \mathcal{M},\ {}^b\mathbf{T}_i, {}^b\mathbf{R}_i, \dot{\mathbf{x}}_i = FK(\begin{bmatrix} 0 \\ 0 \\ \mathbf{q}_a \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ \dot{\mathbf{q}}_a \end{bmatrix})_{i-th\ foot\ frame} \tag{5.14}$$

$${}^b\dot{\mathbf{q}}_{lin}^{FK,B} = \frac{1}{n(\mathcal{M})} \sum_{i \in \mathcal{M}} \left({}^b\mathbf{T}_i \times \dot{\mathbf{q}}_{ang} - {}^b\mathbf{R}_i \dot{\mathbf{x}}_i \right) \tag{5.15}$$

where FK stands for forward kinematics, ${}^b\dot{\mathbf{q}}_{lin}^{FK,B}$ denotes the linear velocity of the robot body estimated by FK, expressed in base frame and at point B (center of body). A foot radius compensation term can be used to take into account the fact that the contact is not punctual but instead rolls on the edge of the round foot as the leg moves (see Fig. 5.2). With r the foot radius and $\dot{q}_{HFE,i}, \dot{q}_{Knee,i}$ the joint velocities of the Hip Flexion-Extension (HFE) and Knee of the i-th leg, the previous estimation becomes:

$${}^b\dot{\mathbf{q}}_{lin}^{FK,B} = \frac{1}{n(\mathcal{M})} \sum_{i \in \mathcal{M}} \left({}^b\mathbf{T}_i \times \dot{\mathbf{q}}_{ang} - {}^b\mathbf{R}_i \dot{\mathbf{x}}_i + \begin{bmatrix} (\dot{q}_{HFE,i} + \dot{q}_{Knee,i})\mathsf{r} \\ 0 \\ 0 \end{bmatrix} \right) \tag{5.16}$$

Similarly, we can get an estimate of the base position with respect to the contact points with the ground using forward geometry, to which we add the average of the contact locations in world frame ${}^o\mathbf{T}_i$ (refreshed when they are deduced from the known position of the base with respect to (w.r.t) the world ${}^o\mathbf{T}_b$ and the position of the feet w.r.t the base

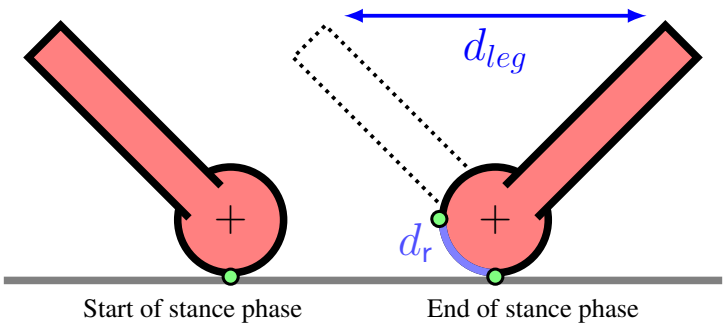


Figure 5.2: By default the forward kinematics will only take into account the traveled distance d_{leg} based on joint angles since the foot frame is located at the center of the foot, without any consideration for foot radius. However, in practice an additional distance d_r has been traveled due to the foot rolling on the ground as the leg changes its angle. This means the base moves slightly faster than the forward kinematics thinks depending on foot radius and angular velocity of the foot, which is taken in (5.16).

${}^{b}\mathbf{T}_i$):

$$\forall i \in \mathcal{M},\ {}^{h}\mathbf{T}_i = FG(\begin{bmatrix} 0 \\ \mathbf{q}_{ang} \\ \mathbf{q}_a \end{bmatrix})_{i-th\ foot\ frame} \tag{5.17}$$

$${}^{o}\mathbf{q}_{lin}^{FG} = \frac{1}{n(\mathcal{M})} \sum_{i \in \mathcal{M}} \left({}^{o}\mathbf{T}_i - {}^{h}\mathbf{T}_i \right) \tag{5.18}$$

There is no compensation of the feet radius here as contact locations are considered to be at the center of the feet whose positions w.r.t the base are tracked by our controller.

By combining information coming from the encoders with contact locations and the orientation given by the IMU, we now have estimates of the base position and linear velocity (${}^{o}\mathbf{q}_{lin}^{FG}$ and ${}^{o}\dot{\mathbf{q}}_{lin}^{FK,B}$ respectively). These estimates are not highly accurate due to how they are computed: elasticity in the mechanical structure of the robot leads to model inaccuracy during forward geometry and kinematics, use of finite difference to get $\dot{\mathbf{q}}_a$ and assumption of immobile contact that might not be well respected at all time. However, they have the advantage that their average value do not drift, as opposed to pure acceleration or velocity integration. The goal is now to merge this information with the highly-accurate linear acceleration of the IMU to benefit from the best of both worlds. To be merged, the various quantities have to be brought to the same location. As it is easier to change the point at which a velocity is expressed, rather than an acceleration, the merging will happen at the IMU location, which is off-centered on the body but well-aligned.

The base orientation is known thanks to the IMU so we can express the estimated velocity ${}^{o}\dot{\mathbf{q}}_{lin}^{FK,B}$ at the IMU location, noted as I, as well as the estimated position of the IMU through ${}^{o}\mathbf{q}_{lin}^{FG}$:

$${}^{o}\mathbf{R}_b = \mathbf{R}(\mathbf{q}_{ang}) \tag{5.19}$$

$${}^{o}\dot{\mathbf{q}}_{lin}^{FK,I} = {}^{o}\mathbf{R}_b \left[{}^{b}\dot{\mathbf{q}}_{lin}^{FK,B} + {}^{b}\mathbf{T}_I \times {}^{b}\dot{\mathbf{q}}_{ang} \right] \tag{5.20}$$

$${}^{o}\mathbf{q}_I^{FG} = {}^{o}\mathbf{q}_{lin}^{FG} + {}^{o}\mathbf{R}_b\, {}^{b}\mathbf{T}_I \tag{5.21}$$

Then, a cascade of two complementary filters or a single Kalman filter merge at IMU location the information coming from the FG estimated position, the FK estimated veloc-

ity and the IMU acceleration, in world frame:

$$ {}^{o}\mathbf{q}_I^{filt}, {}^{o}\dot{\mathbf{q}}_{lin}^{filt,I} = filter({}^{o}\mathbf{q}_I^{FG}, {}^{o}\dot{\mathbf{q}}_{lin}^{FK,I}, {}^{o}\mathbf{R}_b {}^{b}\ddot{\mathbf{q}}_{lin}^{IMU}) \tag{5.22} $$

These filters are further described in Section 5.2 and Section 5.3. Then, we can switch back to a velocity expressed at the center of the base in base frame and to the position of the center of the base in world frame:

$$ {}^{b}\dot{\mathbf{q}}_{lin}^{filt,B} = {}^{b}\mathbf{R}_o \left[{}^{o}\dot{\mathbf{q}}_{lin}^{filt,I} - {}^{b}\mathbf{T}_I \times {}^{b}\dot{\mathbf{q}}_{ang} \right] \tag{5.23} $$

$$ {}^{o}\mathbf{q}_{lin}^{filt} = {}^{o}\mathbf{q}_I^{filt} - {}^{o}\mathbf{R}_b \, {}^{b}\mathbf{T}_I \tag{5.24} $$

The final estimated configuration and velocity vectors are then obtained by stacking the previously estimated position ${}^{o}\mathbf{q}_{lin}^{filt}$ and linear velocity ${}^{b}\dot{\mathbf{q}}_{lin}^{filt,B}$, with the orientation ${}^{o}\mathbf{q}_{ang}$ and angular velocity ${}^{b}\dot{\mathbf{q}}_{ang}$ measured by the IMU, and the joint positions $\mathbf{q}_a$ and velocities $\dot{\mathbf{q}}_a$ measured by the encoders.

$$ {}^{o}\mathbf{q}_{lin}^{filt} = \begin{bmatrix} {}^{o}\mathbf{q}_{lin}^{filt} \\ {}^{o}\mathbf{q}_{ang} \\ \mathbf{q}_a \end{bmatrix} \quad {}^{b}\dot{\mathbf{q}}_{lin}^{filt} = \begin{bmatrix} {}^{b}\dot{\mathbf{q}}_{lin}^{filt,B} \\ {}^{b}\dot{\mathbf{q}}_{ang} \\ \dot{\mathbf{q}}_a \end{bmatrix} \tag{5.25} $$

5.2 Cascade of complementary filters

A cascade of two complementary filters can be used to fuse position, velocity and acceleration data coming from different sources. In essence, a complementary filter processes information from two inputs, one that is highly accurate but slowly drifts and another one with lower accuracy, yet without drift. It combines them to get the best of both worlds by filtering the first input with a high-pass filter to retrieve its accurate high-frequency variations and the second one with a low-pass filter to only keep the non-drifting low-frequency components. In our case, for velocity estimation, a pure integration of the highly accurate IMU linear acceleration would convey quite well the instantaneous changes of the linear velocity but will inevitably drift slowly over time. On the contrary, the velocity estimated through forward kinematics has poor accuracy to fast variations due the assumption of non-moving punctual contacts that is not always perfectly respected, joint elasticity and the noise of joint velocity estimates (done through a finite difference of the position of encoders). Yet its average value does not drift. The same can be said for the base position estimate through forward geometry, which does not drift but is less accurate in the short term than the integration of the velocity outputted by the first complementary filter. The architecture of the whole filter is described in Fig. 5.3.

To define the effect of the complementary filter cf in a generic way, let's denote by $\mathbf{d}$ a quantity trusted at low frequency, by $\dot{\mathbf{d}}$ its derivative trusted at high frequency and by $\tilde{\mathbf{d}}$ the filtered output quantity. $\mathbf{d}_{HP}$ and $\mathbf{d}_{LP}$ are respectively the internal state of the high-pass and low-pass filters of the complementary filter. $\alpha \in [0, 1]$ is a coefficient which tunes the trust in both high and low frequency components. Δt is the time step of the whole filter.

$$ \tilde{\mathbf{d}} = cf(\mathbf{d}, \dot{\mathbf{d}}, \alpha) \qquad \text{Effect of the filter} \tag{5.26} $$

$$ \mathbf{d}_{HP}^{+} = \alpha(\mathbf{d}_{HP} + \Delta t \dot{\mathbf{d}}) \qquad \text{High-pass filtering} \tag{5.27} $$

$$ \mathbf{d}_{LP}^{+} = \alpha \mathbf{d}_{LP} + (1 - \alpha)\mathbf{d} \qquad \text{Low-pass filtering} \tag{5.28} $$

$$ \tilde{\mathbf{d}} = \mathbf{d}_{HP}^{+} + \mathbf{d}_{LP}^{+} \qquad \text{Merging filtered quantities} \tag{5.29} $$

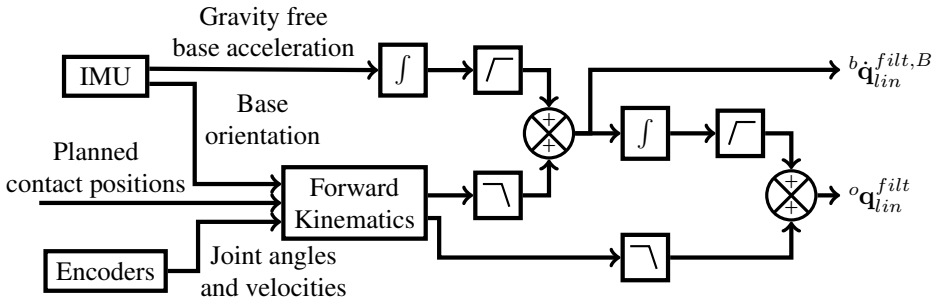


Figure 5.3: A dual cascaded complementary filter provides base position and velocity estimate while being easy to tune and simple to implement.

If $\alpha = 1$, the trust is put entirely on the high frequency quantity. On the contrary, only the low pass frequency is considered when $\alpha = 0$. In practice, a trade-off is often made between both quantities with $\alpha \in [0.9, 1[$ so that the average value of the output remains close to the one of the non-drifting input while the short term variations follow those of the highly-accurate drifting input.

From a signal processing point of view, α is directly correlated with the cut frequency of both filters. $\alpha = 0$ amounts to a 0 Hz cut frequency since only the high-pass affects the outpout. On the opposite, $\alpha = 1$ is equivalent to an infinite cut frequency with a complete disregard to the high-frequency quantity.

If we convert (5.28) into its representation in the z-domain, we get the following transfer function:

$$\mathcal{H}(z) = \frac{\alpha}{1 - (1 - \alpha)z^{-1}} \tag{5.30}$$

For this transfer function the α value that results in a 3 dB cut-off frequency f_c is:

$$\alpha = -\beta + \sqrt{\beta^2 + 2\beta} \quad \text{with} \quad \beta = 1 - \cos(2\pi f_c \, \Delta t) \tag{5.31}$$

While we initially chose a constant value of α over the whole gait cycle for the sake of simplicity, we eventually had to change to a dynamic α to improve the quality of the estimation that worsened near contact switches. The estimation of base velocity through forward kinematics heavily relies on the assumption of an immobile contact point. This assumption is challenged near contact switches as contacts with the ground are often not properly established immediately after landing. This is either due to tracking imperfections or slight slipping when the foot is not well set on the ground and the controller tries to apply ground reaction forces with the newly enabled contact. As a consequence, only IMU data is used around contact switches ($\alpha = 1$). For each foot, we assumed that the middle of the stance phase is when the immobile assumption has the most chance to be respected. So the trust in forward kinematics is progressively increased as the contact goes on. Then it is decreased as the next swing phase gets closer, as shown in Fig. 5.4. The same dynamic value α_{dyn} is used for the $(\dot{x}, \dot{y}, \dot{z})$ components of the velocity filter, with $\alpha_{vel} = [\alpha_{dyn} \;\; \alpha_{dyn} \;\; \alpha_{dyn}]^T$. Such a use of dynamic α value has already been exploited on other robotic system, for instance by [Net+09] for a wheeled robot with an online computation of α depending on assessments of terrain conditions and irregularities.

The forward geometry used for the estimation of the position of the base appeared to be less sensitive to this phenomenon. So constant α values were used over the whole gait cycle, with $\alpha_{pos} = [0.995 \;\; 0.995 \;\; 0.9]^T$ for the (x, y, z) components of the position filter.

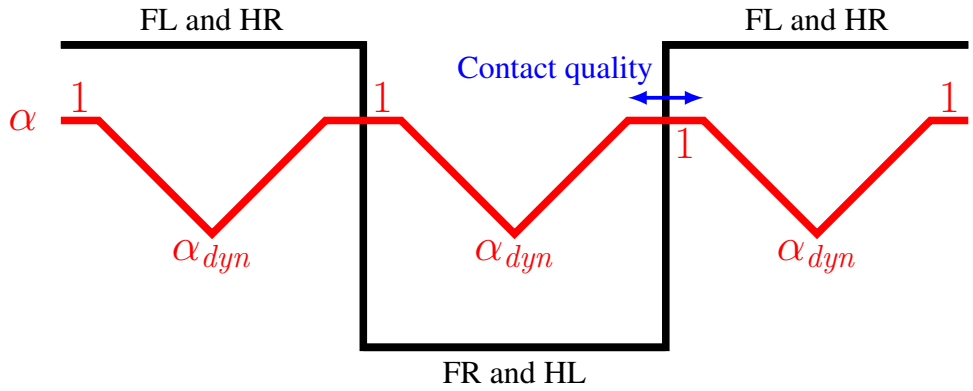


Figure 5.4: The values of α vary over the gait sequence to reflect the trust in forward kinematics. Around contact switches, the quality of the contact is uncertain both for the feet that were already in stance or the ones that just entered it. So α is set to 1 to only trust the IMU. As the contact is better established, the trust in forward kinematics increases, so α decreases. The width of the trust region around contact switches can be adjusted depending on the roughness of the terrain.

5.3 Kalman filter

Contrary to the cascade of two complementary filters which handle position and velocity estimates sequentially, a single Kalman filter can process the two of them at the same time in an optimal way. In this context, IMU acceleration is used as the control vector to make a prediction based on the physical system, while forward kinematics and geometry are used as measurements that correct the prediction. Inaccuracies are represented by process and observation noises.

The state vector $\mathbf{x}$ of the Kalman filter is of size $n = 6$ as it contains the 3 components of the base position and linear velocity. The control vector $\mathbf{u}$ is of size $m = 3$ for the 3 components of the IMU linear acceleration. The observation vector $\mathbf{z}$ is of size $o = 6$ for the 3 components of the base linear velocity and base position estimated through forward kinematics and forward geometry respectively. The observation matrix $\mathbf{H}$ used to correct the prediction done by integrating IMU acceleration and avoid drifting is of size $o \times n$.

$$\mathbf{A} = \begin{bmatrix} I_3 & \Delta t I_3 \\ 0_3 & I_3 \end{bmatrix} \tag{5.32}$$

$$\mathbf{B} = \begin{bmatrix} \frac{1}{2}\Delta t^2 I_3 \\ \Delta t I_3 \end{bmatrix} \tag{5.33}$$

$$\mathbf{H} = I_6 \tag{5.34}$$

The process and observation noises are modeled by the covariance of the process noise $\mathbf{Q}$ of size $n \times n$, the covariance of the observation noise $\mathbf{R}$ of size $o \times o$ and the *a posteriori* estimate covariance $\mathbf{P}$ of size $n \times n$. A normally distributed noise with cross-correlation for both process and observation is assumed.

$$\mathbf{P} = I_6 \tag{5.35}$$

$$\mathbf{Q} = \begin{bmatrix} (\frac{1}{2}\Delta t^2 \sigma_{acc}^2)^2 I_3 & 0_3 \\ 0_3 & (\Delta t \sigma_{acc}^2)^2 \end{bmatrix} \tag{5.36}$$

$$\mathbf{R} = \begin{bmatrix} \frac{1}{2}\Delta t^2 I_3 \\ \Delta t I_3 \end{bmatrix} \tag{5.37}$$

Prediction step

The predicted *a priori* state estimate $\hat{\mathbf{x}}_{k|k-1}$ at time k given observations up to and including time $k-1$ can be computed with $\hat{\mathbf{x}}_{k-1|k-1}$ the *a posteriori* state estimate at time $k-1$ given observations up to and including time $k-1$, $\mathbf{A}_k$ the state-transition model, $\mathbf{B}_k$ the control-input model and u_k the control vector at step k:

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{A}_k \hat{\mathbf{x}}_{k-1|k-1} + \mathbf{B}_k \mathbf{u}_k \tag{5.38}$$

$$= \begin{bmatrix} \mathbf{I}_3 & \Delta t \mathbf{I}_3 \\ 0_3 & \mathbf{I}_3 \end{bmatrix} \hat{\mathbf{x}}_{k-1|k-1} + \begin{bmatrix} \frac{1}{2}\Delta t^2 \mathbf{I}_3 \\ \Delta t \mathbf{I}_3 \end{bmatrix} \mathbf{u}_k \tag{5.39}$$

The predicted *a priori* error covariance $\mathbf{P}_{k|k-1}$ at time k given observations up to and including time $k-1$ can then be computed, with $\mathbf{Q}_k$ the covariance of the process noise at time k:

$$\mathbf{P}_{k|k-1} = \mathbf{A}_k \mathbf{P}_{k-1|k-1} \mathbf{A}_k^T + \mathbf{Q}_k \tag{5.40}$$

$$\mathbf{P}_{0|0} = \mathbf{I}_6 \tag{5.41}$$

$$\mathbf{Q}_k = \begin{bmatrix} (\frac{1}{2}\Delta t^2 \sigma_{acc}^2)^2 \mathbf{I}_3 & 0_3 \\ 0_3 & (\Delta t \sigma_{acc}^2)^2 \mathbf{I}_3 \end{bmatrix} \tag{5.42}$$

with σ_{acc} the acceleration variance. Here we assume that uncontrolled forces lead to acceleration perturbations that are normally distributed without cross-correlation.

Correction step

To correct the prediction with measurements, the innovation $\tilde{\mathbf{i}}_k$ or measurement pre-fit residual at time k, with $\mathbf{H}_k$ the observation model which maps the true state space into the observed space, is defined as follows:

$$\tilde{\mathbf{i}}_k = \mathbf{z}_k - \mathbf{H}_k \hat{\mathbf{x}}_{k|k-1} = \mathbf{z}_k - \mathbf{I}_6 \hat{\mathbf{x}}_{k|k-1} \tag{5.43}$$

The associated innovation (or pre-fit residual) covariance $\mathbf{S}_k$ at time k with $\mathbf{O}_k$ the covariance of the observation noise at time k expresses as follows, assuming that position and velocity estimates with forward geometry and kinematics have a normally distributed error without cross-correlation:

$$\mathbf{S}_k = \mathbf{O}_k + \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^T \tag{5.44}$$

$$= \begin{bmatrix} \sigma_{pos}^2 \mathbf{I}_3 & 0_3 \\ 0_3 & \sigma_{vel}^2 \mathbf{I}_3 \end{bmatrix} + \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^T \tag{5.45}$$

with σ_{pos} and σ_{vel} the position and velocity variances.

The optimal Kalman gain $\mathbf{K}_k$ for the innovation at time k is then:

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}_k^T\mathbf{S}_k^{-1} \tag{5.46}$$

A state estimate is then corrected using the innovation and the optimal gain. We get the updated *a posteriori* state estimate $\hat{\mathbf{x}}_{k|k}$ at time k given observations up to and including time k:

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k\tilde{\mathbf{i}}_k \tag{5.47}$$

Finally, we compute the updated *a posteriori* estimate covariance $\mathbf{P}_{k|k}$ at time k given observations up to and including at time k:

$$\mathbf{P}_{k|k} = (\mathbf{I}_6 - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k|k-1} \tag{5.48}$$

In a similar way than for the cascade of complementary filters, the trust in the IMU acceleration and base linear velocity estimated by forward kinematics can be tuned in real time depending on the contact switches. This can be done for a Kalman filter by tweaking the coefficents of the covariance of the observation noise $\mathbf{O}_k$. Setting σ_{vel} to infinite (or at least a high enough value) near contact switches would nullify the impact of the FK-estimated velocity on the updated *a posteriori* state estimate $\hat{\mathbf{x}}_{k|k}$. That way, only the IMU acceleration would impact the velocity estimated through the prediction step.

5.4 Base velocity filtering

The linear velocity of the base can be further filtered to remove periodic oscillations due to the cyclic nature of the gait. This can be done by using an averaging filter over the span of a gait period, that will remove all oscillations whose frequency is a multiple of the gait period, at the cost of introducing a delay of one gait period, which can be acceptable to improve stability when a fast reaction to state evolution is not required. A short study of this filter is shown in Fig. 5.5. For instance, this filtered velocity is used for foot placement decisions, as it will be explained further in Chapter 7.

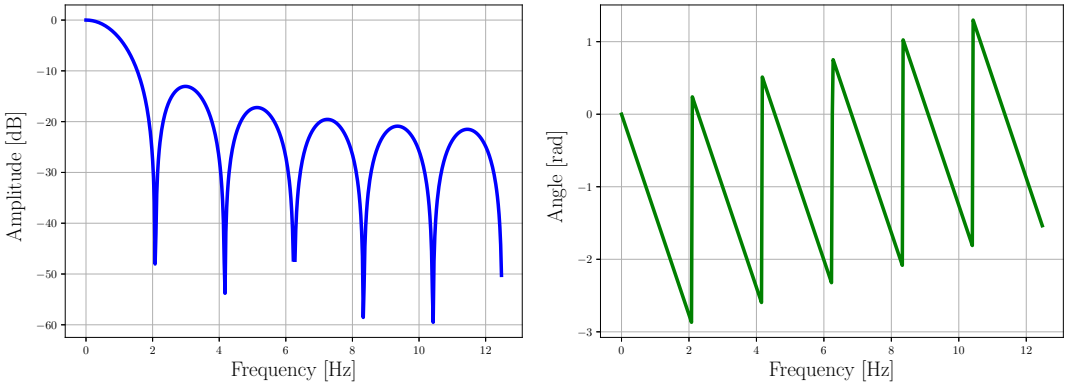
The resulting linear velocity ${}^b\dot{\mathbf{q}}_{lin}^{winfilt,B}$ is defined as follows, with N the number of samples in the averaging window, in this case the number of time steps in a gait period:

$${}^b\dot{\mathbf{q}}_{lin}^{winfilt,B} = \frac{1}{N}\sum_{k=0}^{N-1} {}^b\mathbf{q}_{lin,t-k}^{filt,B} \tag{5.49}$$

In our control architecture, the model predictive controller whose goal is to track a reference base velocity (both linear and angular) runs at 50 Hz, as shown in Fig. 4.1. To avoid undesired temporal aliasing effects in the velocity control, the estimated base velocity given to the MPC is filtered to respect the Nyquist–Shannon sampling criterion (no frequency higher than half the sampling period in the signal). So, a first order low-pass filter with a cutoff frequency of 15 Hz is used. A margin of 10 Hz with respect to the limit of 25 Hz has been chosen because a first order filter does not have an attenuation profile as steep as the one of higher order filters. The filtered underactuated velocity ${}^b\mathbf{q}_{u,t}^{15Hz,B}$ at time t is then:

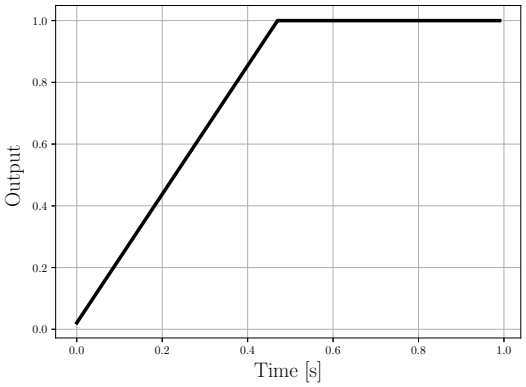
$${}^b\mathbf{q}_{u,t}^{15Hz,B} = (1-\alpha)\,{}^b\mathbf{q}_{u,t-1}^{15Hz,B} + \alpha\,{}^b\mathbf{q}_{u,t}^{filt,B} \tag{5.50}$$

with $\alpha = 0.09$ by applying (5.31) with $f_c = 15$ Hz and $\Delta t = 1$ ms.



(a) Amplitude response

(b) Phase response



(c) Step response

Figure 5.5: Responses of an averaging filter to various frequencies and a step input when tuned to perform a moving average over a gait period of 0.48s. As expected, the filter greatly attenuates all oscillations whose frequency corresponds to the gait period (-48 dB at $\approx$ 2 Hz) or multiples of that frequency. As seen on the step response, the introduced delay is equal to one gait period.

5.5 Reference velocity integration

The position of the robot in world frame is not an observable quantity with the set of sensors we use, which are limited to encoders and IMU. The position estimate in the world inevitably suffers from drift as we have no way to directly measure it. Thanks to the complementary filter or the Kalman filter that merges velocity integration with forward geometry, it is slower than with a pure velocity integration, but it drifts nonetheless. The forward geometry provides an estimate of the base position with respect to the contact points, which is then used to locate newly established contacts, used to locate the base and so on. Errors in position estimate that are slowly introduced in that loop have no way to be detected and corrected.

Yet this control architecture focuses on tracking a reference base velocity given by a user or a higher level controller, so position in world frame is not even required per se as long as the quadruped moves at the correct velocity in its local frame. The various control blocks of the architecture could all work in base frame only. Of course, working in a world frame can be convenient due to the simplicity of handling an immobile frame compared to a constantly moving one. Working in such a frame for convenience should not lead to a degradation of the control behaviour due to the use of an unnecessary badly-estimated quantities. For this reason, we define another “ideal” position of the robot in world frame

alongside the one estimated by the filters. This position results from the integration of the reference base velocity given as input to the robot. For now, the reference base velocity is limited to the horizontal plane: only forward, lateral and yaw (rotation about the vertical axis) reference velocities are non-zero. Roll, pitch and vertical ones are always 0. At each control loop, the new position ${}^{o}\mathbf{q}_{u,k} = [{}^{o}x_k \;\; {}^{o}y_k \;\; 0 \;\; 0 \;\; 0 \;\; {}^{o}\psi_k]^T$ of the robot in this ideal world is as follows, with ${}^{h}\mathbf{q}^{\star}_{u,k} = [{}^{h}x^{\star}_k \;\; {}^{h}y^{\star}_k \;\; 0 \;\; 0 \;\; 0 \;\; {}^{h}\dot{\psi}^{\star}_k]^T$ the base reference velocity:

$$ {}^{o}\psi_k = {}^{o}\psi_{k-1} + {}^{h}\dot{\psi}^{\star} \, \Delta t \tag{5.51} $$

$$ \begin{bmatrix} {}^{o}x_k \\ {}^{o}y_k \end{bmatrix} = \begin{bmatrix} {}^{o}x_{k-1} \\ {}^{o}y_{k-1} \end{bmatrix} + \begin{bmatrix} \cos({}^{o}\psi_k) & -\sin({}^{o}\psi_k) \\ \sin({}^{o}\psi_k) & \cos({}^{o}\psi_k) \end{bmatrix} \begin{bmatrix} {}^{b}\dot{x}^{\star} \\ {}^{b}\dot{y}^{\star} \end{bmatrix} \tag{5.52} $$

When locating the quadruped that way, ${}^{o}x$, ${}^{o}y$ and ${}^{o}\psi$ are perfect quantities, while ${}^{o}z$, ${}^{o}\phi$ and ${}^{o}\theta$ are measured ones (with the complementary or Kalman filters for ${}^{o}z$ and the IMU for ${}^{o}\phi$ and ${}^{o}\theta$). There is no issue going back and forth between this world frame and the horizontal frame for convenience as ${}^{o}\mathbf{T}_h$ and ${}^{o}\mathbf{R}_h$ are perfectly known.

5.6 Conclusion

In this chapter we have described a way to reliably estimate the full state of a quadruped robot using only encoders and IMU, despite the fact that they do not allow to directly measure all quantities of interest. To do so, we leveraged sensor fusion using a cascade of complementary filters or a Kalman filter to combine information that would not lead to satisfactory results independently. The low-accuracy non-drifting assessments coming from forward kinematics have been merged with highly-accurate IMU acceleration measurements that would drift when integrated alone, hence resulting in a good quality non-drifting estimates of base position and linear velocity. As a result, we have shown that even simple estimators and a few number of sensors can provide state estimates that are good enough to obtain state-of-the-art quadruped locomotion.

The next chapter focuses on how the gait is managed in our control architecture, with details on transitions between different kinds of gait, with the aim to achieve a real-time adaptive gait, and on an implementation of contact detection to avoid relying on predefined contact timings.

Chapter 6

Gait

Contents

Contact sequences are used to characterize a succession of contacts between the feet and the ground for an animal or robot in motion. Plenty of contact sequences exist to define the way legged animals move around, each with its own advantages and drawbacks in terms of speed, stability, cost of transport, etc. A contact sequence is often associated with a gait such as trot, gallop, bound (see Fig. 6.1). For animals, adapting their gait to their speed is a key mean to reach an optimized energy cost of transport to go from one place to another, as shown in Fig. 6.2.

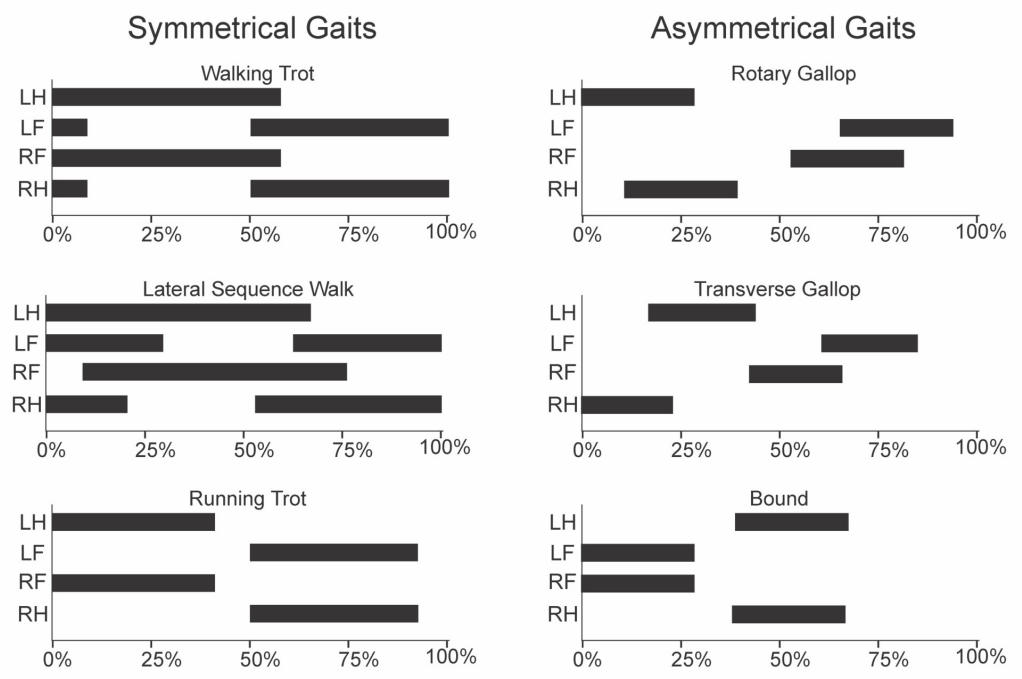


Figure 6.1: Examples of contact sequences over the span of one period of gait for quadrupeds. Each black area characterizes a moment during which the associated foot is touching the ground (stance phase). L and R stand for left and right, F and H stand for front and hind.

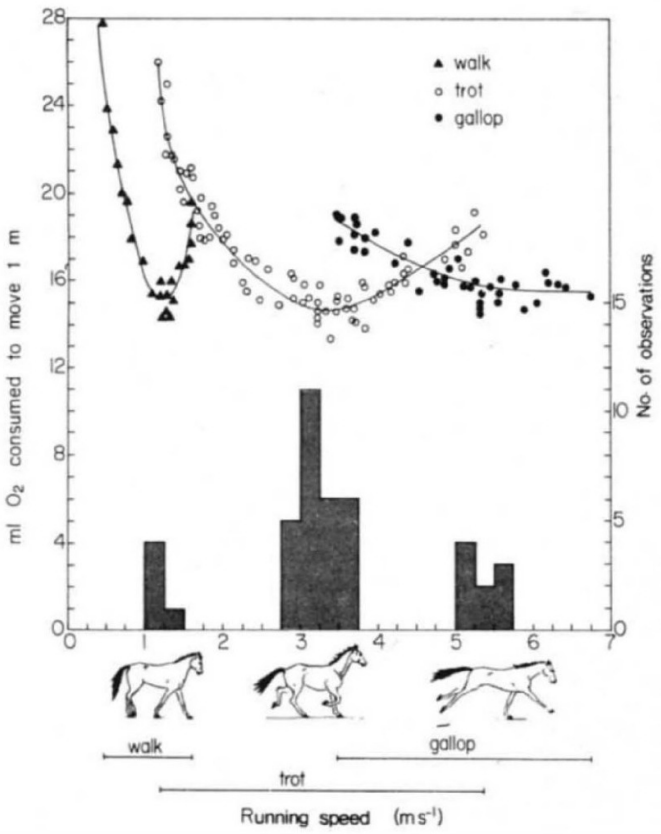


Figure 6.2: Horses have a preferred gait pattern for various ranges of locomotion speed to keep their cost of transport at a minimum (here measured through the quantity of oxygen consumed to move 1 meter). Figure extracted from [HT81].

6.1 Gait structure

By nature, contact sequences are continuous phenomena. Without particular limitations, there is no constraint to when swing and stance phases can start and end. However, from a discrete control point of view, such continuous quantities might be inconvenient to handle as computations would have to properly take into account contact switches that happen between control steps. For this reason, contact sequences can be discretized for a given time step (see Fig. 6.3) so that they can be more easily processed by discrete algorithms.

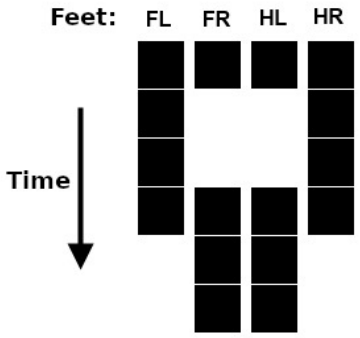


Figure 6.3: Discretized contact sequence for a quadruped (walking trot).

A discretized contact sequence can be stored in a matrix which contains zeroes (feet in the air) and ones (feet in contact with the ground). For instance, for the walking trot of Fig. 6.3, the discretized gait matrix $\mathcal{G}$ along 6 time steps can be defined as follow, where $\forall i \in \{1, \dots, 6\}$, $\mathcal{G}(i, j) = 1$ if foot $j \in \{1, 2, 3, 4\}$ is touching the ground during the i-th time step, $\mathcal{G}(i, j) = 0$ otherwise:

$$\mathcal{G} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix} \tag{6.1}$$

Past, current and future gait sequences all contain information that can be useful for control purpose, such as computing the total duration of a contact phase that is not completely included in the current gait matrix. In that case, we would either need to remember the past or know which contact phases are planned in the future. For this reason, information about the past, current and future gaits are stored in three different matrices: $\mathcal{G}_p$, $\mathcal{G}_c$ and $\mathcal{G}_f$ respectively. These matrices have 4 columns, each one containing contact information for the front left (FL), front right (FR), hind left (HL) and hind right (HR) feet in that order. Each row corresponds to one discretization time step. The first row of $\mathcal{G}_c$ is the current contact status for all feet.

To move the gait forwards, rows are shifted one step upwards, with the first row of $\mathcal{G}_c$ going at the end of $\mathcal{G}_p$ and the first one of $\mathcal{G}_f$ going at the end of $\mathcal{G}_c$. If we assume the gait is kept the same, then the first row of $\mathcal{G}_f$ also becomes the last one to keep the cycle going (see Fig. 6.4).

The size of the current gait matrix $\mathcal{G}_c$ has been chosen with the model predictive control in mind: the length of $\mathcal{G}_c$ is also the one of the prediction horizon of the MPC. That

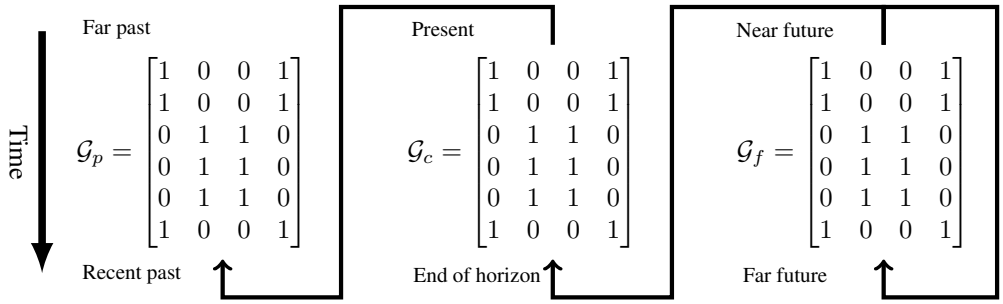


Figure 6.4: Transfers of rows between the $\mathcal{G}_p$, $\mathcal{G}_c$ and $\mathcal{G}_f$ matrices to move the gait forwards. The first row [1 0 0 1] of $\mathcal{G}_c$ expresses the current status of feet, here the FL and HR feet are in contact with the ground while FR and HL are in swing phase. This phase is planned to last 2 time steps. Then, a contact switch happens with FR and HL entering stance phase while FL and HR go in swing phase during 3 time steps. Here the robot is performing a trotting gait with a period of 6 time steps and it will keep doing so for the immediate future, as seen in $\mathcal{G}_f$.

way, the contact status of feet at the i-th time step of the horizon is stored in the i-th row of $\mathcal{G}_c$. It allows to get at first glance the phases the MPC will consider for optimization. Past $\mathcal{G}_p$ and future $\mathcal{G}_f$ gait matrices have the same size than $\mathcal{G}_c$ for simplicity as well. As previously seen in Fig. 4.1, the MPC runs at 50 Hz in a parallel loop while the main loop of the controller (Estimator, Foostep planner, WBC) runs at 1 kHz. This means the gait matrices are not as finely discretized as the main loop, with each row corresponding to 20 iterations. For instance, if the prediction horizon is equal to one gait period with $T = 0.48s$, then $\mathcal{G}_c$ will have 24 rows (for the 24 time steps of 20 ms in the horizon).

6.2 Gait transition

Gait transition can happen when reacting to external disturbances or environmental changes to better handle a new situation by adopting a gait with more fitting characteristics in terms of velocity or stability. We cannot blindly modify the current gait matrix $\mathcal{G}_c$ since it could lead to infeasible scenarios, such as having a foot in a middle of its swing phase suddenly being commanded to be in contact at the next time step, which would not leave enough time for the robot to react properly. We can instead modify without risk the future gait matrix. It is safe to do so because it has no direct effect on the control: the content of $\mathcal{G}_p$ is progressively inserted into $\mathcal{G}_c$ one step at a time.

Directly modifying $\mathcal{G}_c$ could also degrade the robot behaviour through the decisions of the MPC since the contact forces it has outputted during the past time steps based on the contact sequence in $\mathcal{G}_c$ might be inappropriate for the new contact sequence. If the state evolves as expected, the desired contact forces $\mathbb{F}$ for a prediction horizon of N time step would be:

$$\begin{array}{rrcl} \text{at time } k: & \mathbb{F}^k & = & [\ \mathbf{F}_0^k \quad \mathbf{F}_1^k \quad \mathbf{F}_2^k \quad \dots \quad \mathbf{F}_{N-3}^k \quad \mathbf{F}_{N-2}^k \quad \mathbf{F}_{N-1}^k\ ] \\ \text{at time } k+1: & \mathbb{F}^{k+1} & = & [\ \mathbf{F}_0^{k+1} \quad \mathbf{F}_1^{k+1} \quad \mathbf{F}_2^{k+1} \quad \dots \quad \mathbf{F}_{N-3}^{k+1} \quad \mathbf{F}_{N-2}^{k+1} \quad \mathbf{F}_{N-1}^{k+1}\ ] \\ & & \approx & [\ \mathbf{F}_1^k \quad \mathbf{F}_2^k \quad \mathbf{F}_3^k \quad \dots \quad \mathbf{F}_{N-2}^k \quad \mathbf{F}_{N-1}^k \quad \mathbf{F}_{N-1}^{k+1}\ ] \end{array}$$

with $\mathbf{F}_i^k$ the contact forces desired for the four feet at the i-th time step of the prediction horizon at time k. Thus, there is a consistency between the forces planned at time step k

and $k+1$. However if the contact sequence is modified, then $\mathbb{F}^{k+1}$ may be greatly different from $\mathbb{F}^k$, which is a problem because when the MPC planned his forces at step k, it did so considering it would be able to apply them during the whole prediction horizon. The applied forces may put the robot in a bad situation for the new contact sequence. With the nominal behaviour of the controller this effect is minimized because the new sequence is injected one step at a time from $\mathcal{G}_f$ to $\mathcal{G}_c$. That way the MPC can progressively adapt to the new gait as it is inserted at the end of its horizon.

Not having to carefully modify $\mathcal{G}_c$ by modifying $\mathcal{G}_f$ instead comes at the price of a delay since its new content will only be fully taken into account after one duration of the prediction horizon, which will be less than half a second in practice.

For instance, let's consider the initial situation of a trotting gait switching to a four-stance phase:

$$\mathcal{G}_p = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix} \quad \mathcal{G}_c = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix} \quad \mathcal{G}_f = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \tag{6.2}$$

After 3 time steps, the new gait is now included in the second half of the prediction horizon:

$$\mathcal{G}_p = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix} \quad \mathcal{G}_c = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \quad \mathcal{G}_f = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \tag{6.3}$$

The transition is finished after 6 time steps, the complete duration of prediction horizon:

$$\mathcal{G}_p = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix} \quad \mathcal{G}_c = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \quad \mathcal{G}_f = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \tag{6.4}$$

6.3 Adaptive trotting gait

By default, the gait of the quadruped is not modified by its current state: gait type and frequency remain the same, regardless of the situation. As stated earlier, each gait has its own pros and cons in terms of stability, velocity and cost of transport. So a predefined gait pattern can hardly be optimal in all situations. Hence, a refinement of the control architecture would be to adapt this pattern (trotting in our case) to improve both the robot behavior and the energy consumption.

6.3.1 Heuristic variation

A first approach could be to use a heuristic to modify the gait frequency in real-time according to the robot velocity. For trotting, the faster the gait, the easier the base is kept stable since the pair of diagonally opposed feet in contact with the ground switches more often. A higher frequency reduces the duration of swing phases and thus limits the natural tilt of the base since the robot cannot apply momentum along the axis that passes through its only two punctual contacts. In fact, it has a double beneficial effect as the base tilts during a shorter duration and the robot is able to correct that tilt more often as the contact pair switches. However, more frequent swing phases also mean feet have to move regularly from one contact location to another one, which can lead to a higher power consumption. So the goal is to properly chose the gait frequency to be fast enough for stability purpose, but not unnecessarily fast to limit energy consumption.

If we decide that the gait frequency $f_{\mathcal{G}}$ varies linearly according to the velocity with a null frequency when the reference velocity is 0 (no need to move the feet so the robot stays in a permanent four-stance phase), then we would only have to determine the slope η of the velocity to frequency relation:

$$f_{\mathcal{G}} = \eta \, \| {}^{b}\dot{\mathbf{q}}^{\star}_{lin} \|_2 \tag{6.5}$$

6.3.2 Reinforcement learning

Another possible approach for determining the contact sequences comes from the field of machine learning with a model-free reinforcement learning (RL) method for adapting the timings of the stance and swing phases for each foot. In collaboration with Michel Aractingi, PhD studen co-advised by LAAS-CNRS and Naver Labs, we worked on applying an hybrid RL/model-based approach to control Solo-12. This work is thoroughly presented in [Ara+21]. Thanks to the modularity of the architecture, the learned agent can seamlessly augment the control pipeline.

Based on an initial predefined trotting, the policy can adapt the gait period and the duration of both stance and swing phases within a period, and therefore the potential overlap of those phases between the feet. These modifications can lead to variations of the hard-coded trot that have different properties in terms of speed, energy consumption, reactivity and robustness. Although the network could directly control all the coefficients of $\mathcal{G}_c$, it would create an intractable action space with most combinations corresponding to infeasible gaits. The network would have to experimentally learn the constraints of the robot dynamics, which would greatly slow down the process. Instead, it uses the fact that the locomotion is periodic to consider the gait matrix $\mathcal{G}$ through parameterized oscillation functions.

For each foot, a base oscillation is defined as $\bar{\Gamma}(t, \tau_0, \tau_1)$ where t is the considered time, $\tau_0^i < \tau_1^i$ are the timings at which contact switches occur for the i-th foot and T is the oscillation (i.e gait) period:

$$\bar{\Gamma}(t, \tau_0^i, \tau_1^i, T) = \begin{cases} 0 & \text{if } \tau_0^i < (t \mod T) < \tau_1^i \\ 1 & \text{otherwise} \end{cases} \tag{6.6}$$

"mod " refers to the modulus operation to periodically reset the time to zero according to the period T. An oscillation function $\Gamma^i(t) : \mathbb{R}_+ \to \{0, 1\}$ is then defined to determine the future contact state of the i-th foot as a function of time t. The oscillation

is parameterized by two switch timing parameters τ_s^i and τ_c^i which indicate the beginning of the swing and stance phases of the i-th foot respectively:

$$\begin{aligned} \forall i \in \{1,2,3,4\},\ \Gamma^i(t,\tau_s^i,\tau_c^i) = \ & \delta_{1,\mathcal{G}_c(0,i)}\ \bar{\Gamma}(t,\tau_s^i,\tau_c^i,T^i) \\ & + \ \delta_{0,\mathcal{G}_c(0,i)}\ \left(1-\bar{\Gamma}(t,\tau_c^i,\tau_s^i,T^i)\right) \end{aligned} \tag{6.7}$$

where the period T^i is defined as $T^i = \max(\tau_s^i, \tau_c^i)$ and $\delta_{i,j}$ refers to the Kronecker delta, i.e $\delta_{i,j} = 1$ if $i = j$, 0 otherwise.

The network relies on a 4×2-dimensional continuous action space to control these oscillations, noted $\mathcal{A} = \{a^1, a^2, a^3, a^4\}$ with $\forall i \in \{1,2,3,4\}, a^i = (\Delta\tau_s^i, \Delta\tau_c^i) \in \mathbb{R}^2$. These actions define the displacement of contact switches with respect to the ones of the nominal trotting gait, $\tau_n^i = (\tau_{s,n}^i, \tau_{c,n}^i)$, whose period is set at the start of the training. That way the network learns deviations from the nominal values rather than the nominal values directly, which reduces the exploration space. Finally, all the new coefficients of $\mathcal{G}_c$ can be obtained with $\mathcal{G}_c(k,j) = \Gamma^i(\frac{k}{Nf_\mathcal{G}}, \tau_n^i + a_t^i)$, where $i \in \{1,2,3,4\}$ and $k \in \{1,\dots,N\}$ with N the number of rows in $\mathcal{G}_c$ (i.e the number of time steps in the prediction horizon of the MPC).

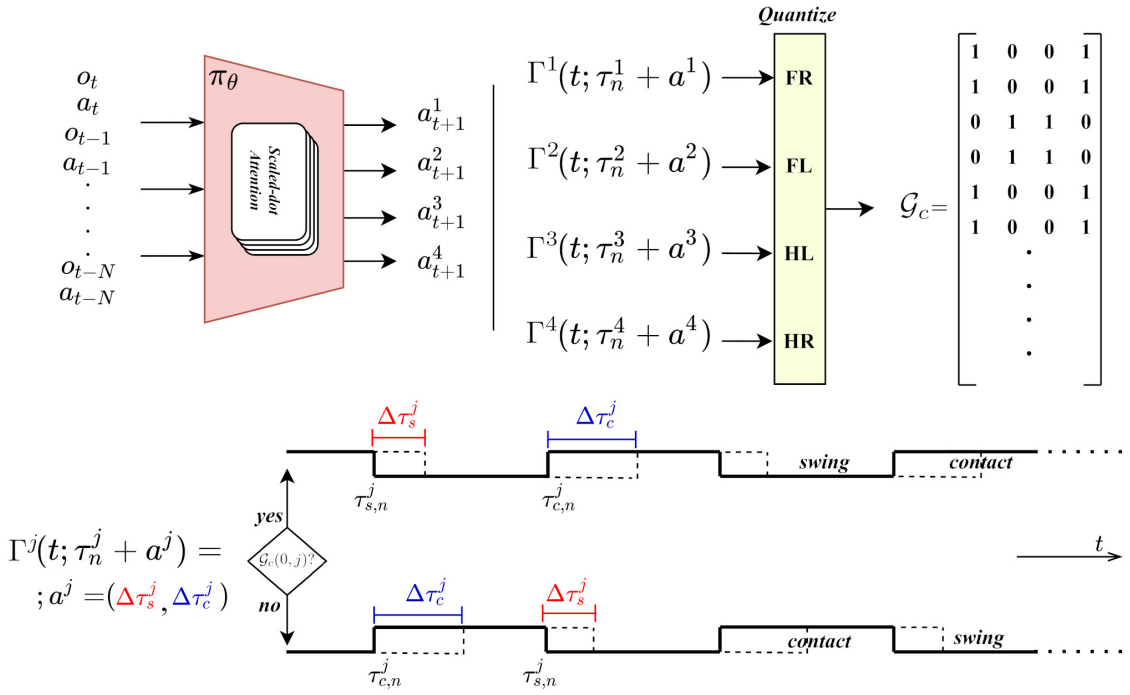


Figure 6.5: The displacements a^i of the nominal contact timings τ_n^i modify the nominal contact sequence of each foot, highlighted in bold in the lower part of the figure. The dotted lines represent the timings of contact switches after they have been shifted by the policy, depending on the current contact status of the foot. The choice yes/no in the bottom half of the figure depend on the current status of the considered foot $\mathcal{G}_c(0,j)$. Figure extracted from [Ara+21].

This policy runs at 10 Hz as a trade-off to give it enough time to execute an action and receiving useful learning feedback while keeping enough reactivity to adapt the gait in a meaningful way. During training, it tries to minimize the energy consumption as well as the velocity tracking error. The network takes as inputs (also called observation o_t) the base height and orientation, base linear and angular velocities, joint angles and velocities,

feet positions, current and past gait contact sequences. It keeps in memory the inputs and actions over the past 8 iterations to take advantage of the sequence of observation-action pairs thanks to self-attention layers. Fig. 6.5 summarizes how the architecture affects $\mathcal{G}_c$ through displacements of contact timings $\Delta\tau$.

The policy was successfully trained and deployed in simulation to adapt a nominal trotting gait for different reference velocities. For low velocities, the policy decreases the gait frequency, making the stepping slower. On the contrary, for high velocities the frequency is increased, thereby making the stepping faster which helps stabilizing the base and following the reference velocity. It performs a trade-off between gait frequency and energy consumption to follow the velocity command with low error without moving feet uselessly fast. At zero velocity, an optimal energy saving policy is reached with a static gait where all feet are in contact with the ground. Examples of the resulting gaits, due to the policy adaption of the nominal trotting, are shown in Fig. 6.6.

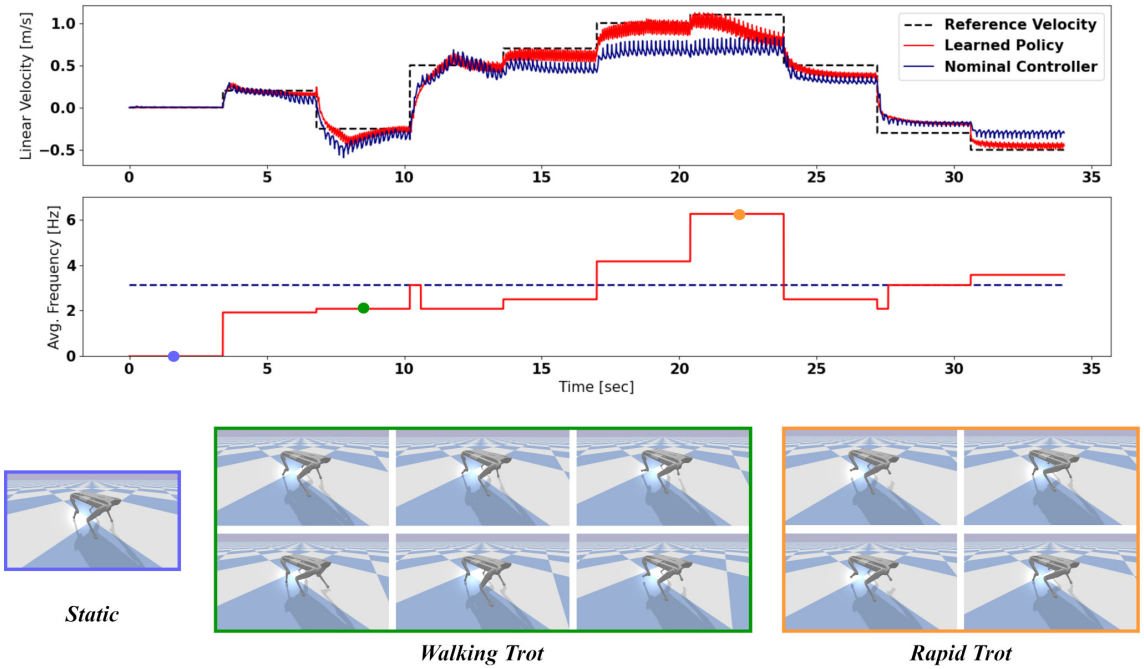


Figure 6.6: (Top) The policy is able to get the robot closer to the desired reference in most cases by adapting the gait frequency. (Middle) The blue dashed line represents the nominal trotting frequency ($\approx$ 3 Hz) while the red line indicates the average frequency of all legs after policy adaptation. The frequency of stepping is slowed down and sped-up according to the reference velocity to get a trade-off between energy consumption and tracking performance. (Bottom) Snapshots of the achieved gaits at different levels of the run, corresponding to the blue, green and orange dots in the middle graph. Figure extracted from [Ara+21].

6.4 Contact detection

Until now, no online contact detection was performed. All contacts were enabled or disabled based on a predefined contact timings that we considered as trustfull. For a given foot $i \in \{1, 2, 3, 4\}$, if $\mathcal{G}_c(0, i) = 0$ and $\mathcal{G}_c(1, i) = 1$, then it meant the foot would enter contact with the ground at the next time step. On the opposite, if $\mathcal{G}_c(1, i) = 0$, then contact would not be expected. This approach works well for trotting indoors on a flat ground

without obstacles, since the swinging feet tracking is good enough so that the mismatch with the moment contacts really happen does not hamper the motion. On uneven grounds, contact mismatches have more chance to be significant enough to be detrimental to the quality of the behavior, especially if the ground is slippy, like wet grass, or when the robot walks on obstacles. Because the controller runs with the assumption of a flat ground, if the target position of a swinging feet is in a depression, the foot will stop in the air a few centimeters above the ground, then it will hit the floor as the leg suddenly extends due to the activation of the contact. On the contrary, if it hits an obstacle, like a stair, earlier than expected, the controller will not react and will try to drive the foot downwards into the stair to reach the expect position at floor level. This is detrimental to the locomotion as it creates unexpected ground reaction forces.

Online contact detection would allow to detect if a contact happens too early or too soon instead of relying on the predefined contact sequence. The content of $\mathcal{G}_c$ would then be changed accordingly:

$$\mathcal{G}_c = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \xrightarrow[\text{with first foot}]{\text{Early contact detection}} \mathcal{G}_c = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \tag{6.8}$$

$$\mathcal{G}_c = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix} \xrightarrow[\text{with first foot}]{\text{Late contact detection}} \mathcal{G}_c = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix} \tag{6.9}$$

In (6.8), the first foot was supposed to touchdown in two time steps. However, early contact was detected for this foot so the controller decided to start the stance phase earlier by switching the two first rows to 1. On the opposite, in (6.9), the first foot was supposed to be in contact, yet no contact was detected. Thus, the controller supposed the contact was late by switching the first row to 0. The second row kept its value as we assumed the contact might actually happen at the next time step.

To perform this online detection, we focused on three quantities that we consider to be well representative of whether a contact has occurred. In fact, we do not want to always enable a contact immediately after the foot touched the ground, as the foot may not be well-set on the ground with some remaining velocity. The first quantity is the estimated ground reaction force at the considered foot. It is part of the equation of the robot dynamics:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{b}(\mathbf{q}, \dot{\mathbf{q}}) = \tau + \mathbf{J}_c^T(\mathbf{q})\mathbf{f} \tag{6.10}$$

with $\mathbf{M}$ the generalized mass matrix, $\mathbf{b}$ the gravitational and nonlinear forces, τ the vector of efforts including joint torques, $\mathbf{J}_c$ the contact Jacobian and $\mathbf{f}$ the ground reaction forces. Assuming low joint inertias, the ground reaction forces can be estimated as follows:

$$\mathbf{f} = -(\mathbf{J}_c^T)^\dagger(\tau - \mathbf{b}) \tag{6.11}$$

with $\{\cdot\}^\dagger$ the Moore-Penrose pseudo-inverse operator. Even if the estimation is rough, we do not need high accuracy, as we are merely checking whether there are ground reaction

forces instead of their exact value. Using the configuration and velocity vectors $\mathbf{q}, \dot{\mathbf{q}}$, we can asses the feet spatial velocities $\mathcal{V}(\mathbf{q}, \dot{\mathbf{q}})$ (in inertial frame). On the other hand, the predefined gait schedule in $\mathcal{G}_c$ provides expected timings around which contacts are supposed to happen due to feet motion.

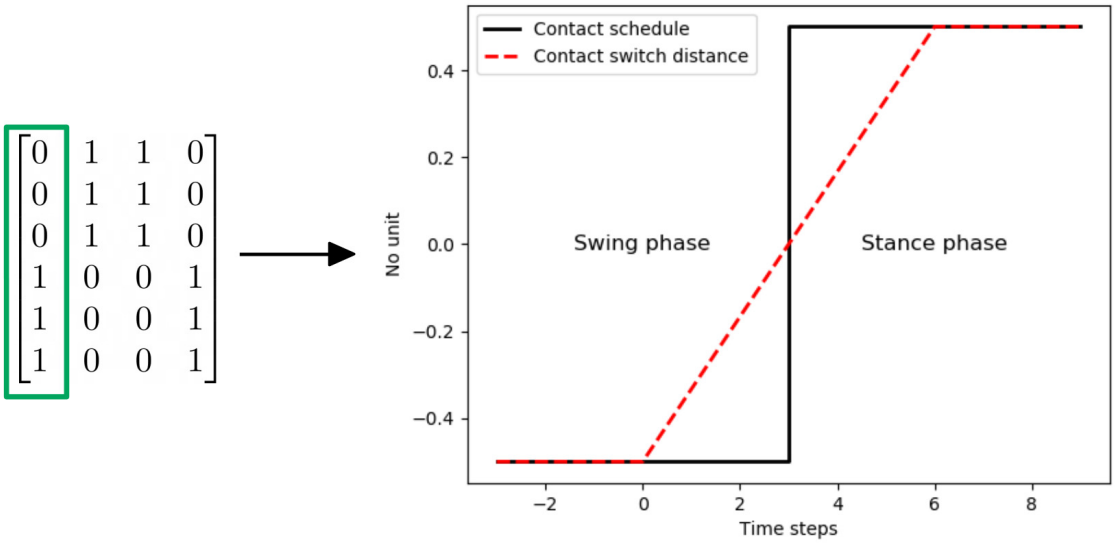


Figure 6.7: The distance from the contact switch is negative before it happens, then positive so that we can differentiate before/after contact, which we would not be able to do with an absolute distance.

We need metrics based on these three quantities (contact forces, spatial velocity, expected timing) to take a decision about the state of the contact. The normal component of the estimated ground reaction force provides a relevant estimate of the contact state since it should theoretically be equal to 0 N when the feet in swing phase. In practice we can still have small non-zero values due to the simplified formula (6.11) we are using. When a contact is well-set on the ground, we can expect the foot to be immobile so the norm of the spatial velocity can be used as an indicator of the contact quality. If the terrain is not excessively rough, contacts can be expected to happen a short moment before or after their predefined timing so the distance from it (in number of time steps) can be used (see Fig. 6.7).

Combining these 3 metrics refine the decision process instead of relying on a single quantity that can be misleading. For instance, when the robot base is immobile, the spatial velocity of a swinging foot will be close to 0 m/s at the apex of its trajectory. That could lead to a contact detection if the velocity metrics was the only one being used. This false positive can be easily avoided by merging this information with the force or timing metrics. Probabilistic approaches offer a straightforward way to merge these metrics. Based on the probability density function of a standard normal distribution $\phi(\rho)$, with ρ a metric, we can define the cumulative distribution function $\Phi(\rho)$:

$$\phi(\rho) = \frac{1}{\sqrt{2\pi}} e^{-\frac{\rho^2}{2}} \tag{6.12}$$

$$\Phi(\rho) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\rho} e^{-\frac{\rho^2}{2}} dt \tag{6.13}$$

There is no analytical solution to this integral but we can use the numerical approxi-

mation of the error function erf for a normal distribution of mean 0 and variance 0.5:

$$erf(\rho) = \frac{2}{\sqrt{2}} \int_0^{\rho} e^{-t^2} dt \tag{6.14}$$

Knowing that $\Phi(0) = 0.5$ by symmetry of the standard normal distribution, we can then express Φ according to erf:

$$\Phi(\rho) = \frac{1}{2} \left(1 + erf(\frac{\rho}{\sqrt{2}}) \right) \tag{6.15}$$

In the general case of a normal distribution with mean μ and variance σ, the same reasoning results in the following probability density function P:

$$P(\rho, \mu, \sigma) = \Phi(\frac{\rho - \mu}{\sigma}) = \frac{1}{2} \left(1 + erf(\frac{\rho - \mu}{\sigma\sqrt{2}}) \right) \tag{6.16}$$

With a normal distribution for each metrics, this leads us to three pairs of parameters to tune: $(\mu_{\mathbf{f}}, \sigma_{\mathbf{f}})$, $(\mu_{\mathcal{V}}, \sigma_{\mathcal{V}})$ and $(\mu_{\mathcal{G}}, \sigma_{\mathcal{G}})$. If we assume the contact has no particular reason to happen late or early on a rough terrain, then $\mu_{\mathcal{G}} = 0$ and there are 5 remaining parameters to tune (see Fig. 6.8). In the end, the contact probability $P_{ctc} \in [0, 1]$ is obtained by multiplying the probabilities of all metrics:

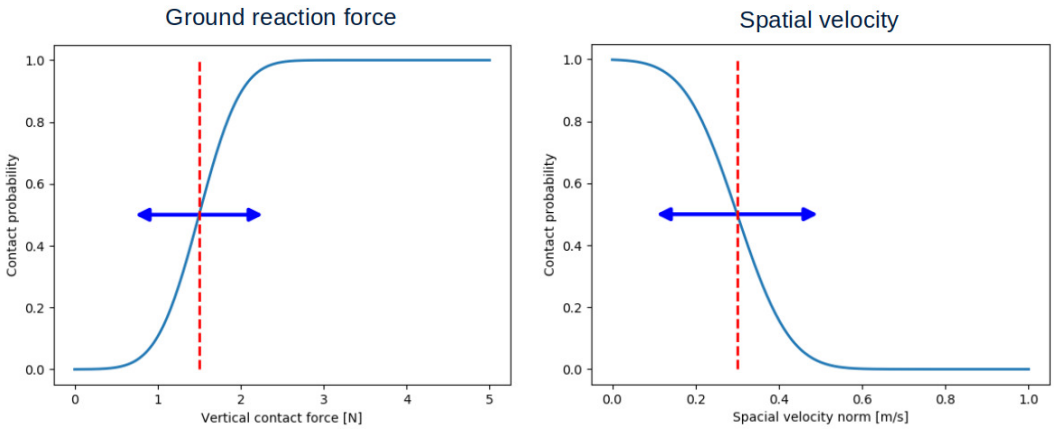
$$P_{ctc}(\mathbf{f}, \mathcal{V}, \mathcal{G}_c) = P(\mathbf{f}_z, \mu_{\mathbf{f}}, \sigma_{\mathbf{f}})\ P(\|\mathcal{V}\|_2, \mu_{\mathcal{V}}, \sigma_{\mathcal{V}})\ P(\mathcal{G}_c, \mu_{\mathcal{G}}, \sigma_{\mathcal{G}}) \tag{6.17}$$

The final decision is made based on a threshold value $P_{th} \in [0, 1]$.

6.5 Conclusion

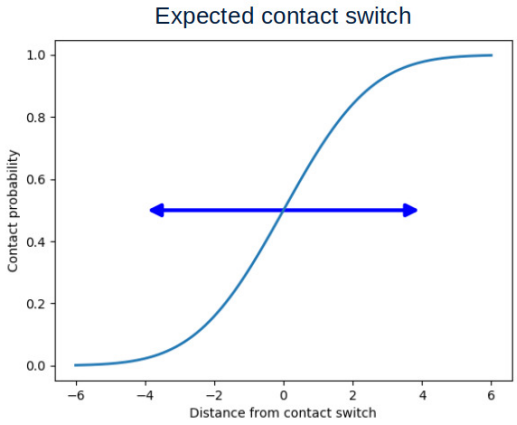
In this chapter we have described how the gait, i.e the sequence of stance and swing phase for each foot, is managed to obtain a cyclic pattern of footsteps for the locomotion of the Solo quadruped. Past, current and future gait matrices allow seamless transition between different kinds of gaits by letting the whole architecture progressively adapt to the new desired gait. While the gait follows a predefined trotting pattern by default, we presented two ways to adapt its characteristics online, either with a heuristic or through reinforcement learning. Finally, we described the implementation of an online contact detection to detect and react to contact timing mismatches than can happen when moving on rough terrain. With it, the predefined gait pattern is not completely trusted anymore and a probabilistic approach checks if contacts occurred before enabling them.

The next chapter focuses on how the choice of contact locations are made in our control architecture, with details on the heuristics that are used, on the trajectory generation to drive the foot from one location to another during swing phases, and on a way to augment the footstep decision process to use privileged information about the environment.



(a) Ground reaction force

(b) Spatial velocity



(c) Expected contact switch

Figure 6.8: A normal contact force equals to 0 Nm means a low probability of contact while a spatial velocity equals to 0 m/s means a high one, hence why those curves looks inverted. The normal distribution for contact timing is centered at a distance of 0 since it corresponds to the expected timing. Red dotted lines highlight the $\mu_{\mathbf{f}}$ and $\mu_{\mathcal{V}}$ values while the blue arrows illustrates the effect of $\sigma_{\mathbf{f}}$, $\sigma_{\mathcal{V}}$ and $\sigma_{\mathcal{G}}$ on the width of the transition between probabilities of 0 and 1.

Chapter 7

Footstep Planner

Contents

The gait is characterized by a sequence of stance and swing phase for each foot. During swing phases, feet have to be guided from their current position on the ground to the next one. So the control architecture regularly has to chose at which location each foot should land at the end of its respective swing phase. The locations of footsteps are of particular importance because pushing on the ground is the only way for the quadruped to interact with its environment, balance itself and move around.

7.1 Footsteps heuristics

Before generating a trajectory in the air that will be tracked by feet in swing phase, the way to determine the target locations on the ground must be defined. We decided to keep a simple heuristic since it was shown to work well in [Kim+19]. It is based on the Raibert's heuristics that was initially applied on a monopod and tries to keep the contact point centered under the shoulder in average, in a similar way to what is done with inverted pendulums.

7.1.1 Footstep locations in horizontal frame

The chosen footsteps planner is limited to the horizontal plane of the floor since we made the assumption that the floor is flat. It continuously outputs the footstep locations

of each foot ${}^{h}r_i^{\star}, \forall i \in \{1,2,3,4\}$ in the horizontal frame depending on the current and reference velocities of the body.

$$\forall i \in \{1,2,3,4\},\ {}^{h}r_i^{\star} = r_{sh,i} + r_{sym} + r_{fb} + r_c \tag{7.1}$$

- $r_{sh,i}$ defines the default foothold location of i-th foot when the quadruped is immobile and without velocity reference. This location is the projection on the ground of the shoulder joint associated with this foot. We neglect the roll and pitch angles of the base to use constant values (projection obtained when the base is horizontal) so r_{sh} is as follows for the Solo quadruped, with $r_{sh,i}$ the i-th column:

$$r_{sh} = \begin{bmatrix} 0.195 & 0.195 & -0.195 & -0.195 \\ 0.150 & -0.150 & 0.150 & -0.150 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{7.2}$$

- r_{sym} is a term that is used to maintain the current horizontal linear velocity of the robot by moving the footstep location in the direction the robot is currently moving. It is defined as:

$$r_{sym} = \frac{T_{stance}}{2} {}^{h}\dot{\mathbf{q}}_{lin} = \frac{T_{stance}}{2} \begin{bmatrix} {}^{h}\dot{x} \\ {}^{h}\dot{y} \\ 0 \end{bmatrix} \tag{7.3}$$

with T_{stance} the duration of the next incoming stance phase. This symmetry term is added to r_{sh} to make the gait symmetric with respect to the shoulders when moving. If the base moves forwards at speed $\dot{\mathbf{q}}_{lin}$ then with only r_{sh}, feet would land under their associated shoulder and would spend the whole stance phase "behind" the shoulder as the base keeps moving forwards while the foot in contact does not move. On the contrary, with this term, feet land at a position r_{sym} with respect to the shoulder, in the direction of the motion. That way if the base keeps a constant velocity, the contact will be right under the shoulder at half the stance phase and at a distance -r_{sym} with respect to the shoulder at its end. It ensures the contact is centered on the shoulder on average for balance reason. For an inverted pendulum, this would lead to identical landing and leaving angle.

- r_{fb} is the feedback term for the linear velocity. It slightly modifies the position of footholds to help the quadruped reduce the difference between its current velocity ${}^{h}\dot{\mathbf{q}}_{lin}$ and the target reference velocity ${}^{h}\dot{\mathbf{q}}_{lin}^{\star}$. It is defined as follows:

$$r_{fb} = k\,({}^{h}\dot{\mathbf{q}}_{lin} - {}^{h}\dot{\mathbf{q}}_{lin}^{\star}) = k \begin{bmatrix} {}^{h}\dot{x} - {}^{h}\dot{x}^{\star} \\ {}^{h}\dot{y} - {}^{h}\dot{y}^{\star} \\ 0 \end{bmatrix} \tag{7.4}$$

This feedback term is added to the previous ones to make it easier for the robot to reach the reference velocity. The only way the quadruped can interact with its environment is by pushing on the ground with its feet (it cannot pull). As per Newton's second law, if the quadruped wants to move in a given direction it has to apply a force in the inverse direction. So the feedback term makes it easier to do that by shifting the desired location of footsteps in the inverse direction of the velocity error $({}^{h}\dot{\mathbf{q}}_{lin}^{\star} - {}^{h}\dot{\mathbf{q}}_{lin})$. For instance if the robot is not moving fast enough forwards then the feedback term will slightly shift the footsteps backwards so that it's easier to push on the ground backwards and as a result to increase its forward velocity. If the robot is pushed to the right by an external force with the reference velocity going forwards then the feet will move on the right to push on the

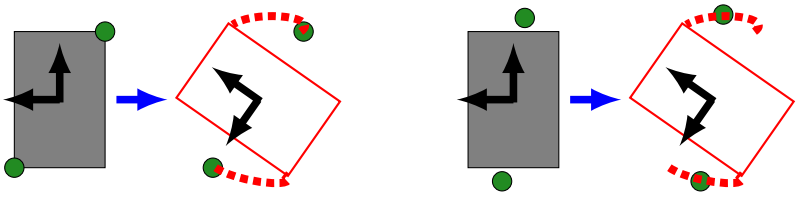
ground in that direction and reduce the lateral velocity. The value of this coefficient k is set to 0.03 m/(m/s), which was the value used by [Kim+19].

- The last term r_c, called centrifugal term, is related to rotation, in our case along the vertical axis. It is defined as follows:

$$r_c = \frac{1}{2}\sqrt{\frac{h}{g}}{}^h\dot{\mathbf{q}}_{lin} \times {}^h\dot{\mathbf{q}}^\star_{ang} = \frac{1}{2}\sqrt{\frac{h}{g}} \begin{bmatrix} {}^h\dot{y}\ {}^h\dot{\psi}^\star \\ -{}^h\dot{x}\ {}^h\dot{\psi}^\star \\ 0 \end{bmatrix} \tag{7.5}$$

A centrifugal term is added to the footstep planner to make it easier to compensate the centrifugal effect when the robot is turning about the vertical axis by adjusting the location of footsteps accordingly. The $\frac{1}{2}\sqrt{\frac{h}{g}}$ coefficient is associated with the use of an inverted pendulum model. The ${}^h\dot{\mathbf{q}}_{lin} \times {}^h\dot{\mathbf{q}}^\star_{ang}$ part of r_c shifts the feet in the direction of the centrifugal force that appears when moving in the horizontal plane while rotating along the vertical axis. It helps the robot keep moving by making it easier to compensate this force by pushing on the ground in that direction. For instance, if the robot is moving forwards while turning left then the centrifugal effect is applied sideways in the right direction. $\dot{\mathbf{q}}_{lin} \times \dot{\mathbf{q}}^\star_{ang}$ with $\dot{\mathbf{q}}_{lin} = [1\ \ 0\ \ 0]^T$ and $\dot{\mathbf{q}}^\star_{ang} = [0\ \ 0\ \ 1]^T$ results in $[0\ \ -1\ \ 0]^T$ which shifts the feet in the right direction in horizontal frame, as expected. Note that the angular velocity that is used is the reference one and not the current one. It could also be seen as a way to help the quadruped reach the reference angular velocity in a way similar to what the feedback term does for the linear velocity. If $\dot{\mathbf{q}}_{ang} = [0\ \ 0\ \ 0]^T$ and $\dot{\mathbf{q}}^\star_{ang} = [0\ \ 0\ \ 1]^T$ then using $\dot{\mathbf{q}}_{lin} \times \dot{\mathbf{q}}_{ang}$ would lead to a zero r_c. It might be more difficult to start turning without having the feet shifted sideways, compared to $\dot{\mathbf{q}}_{lin} \times \dot{\mathbf{q}}^\star_{ang}$ which shifts the feet to the right.

Remark: The shoulder term $r_{sh,i}$ can be replaced by $\mathcal{R}_z(\frac{T_{stance}}{2}{}^h\dot{\psi})r_{sh,i}$ so that the rotation of the body is taken into account in a similar way than what the symmetric term r_{sym} does with the linear velocity to keep footsteps centered on the shoulder in average (see Fig. 7.1).



(a) Current heuristics (b) With new term

Figure 7.1: Expected feet placement with and without the new heuristics term. Here the robot is just at the beginning of the stance phase for the front right and hind left feet, with $\dot{q}_{lin} = [0\ 0\ 0]^T$ and $\dot{q}_{ang} = [0\ 0\ \dot{q}_{ang,z}]^T$. The red rectangles represent the expected orientation of the body at the end of the stance phase. By shifting the feet with the new term in (b), shoulders would be more centered on the contacts on average, compared to what is obtained with the current heuristic (a).

7.1.2 Footstep locations in world frame

Now that footstep locations in horizontal frame at the start of the next incoming stance phase have been defined, we add another term to take into account the time remaining until it starts. With only the previous terms, whether a foot is just at the start of its swing

phase or almost at the end, the desired target location returned by the footstep planner is the same. If the quadruped is moving forwards at the reference velocity then during the whole duration of the swing phase the target position will be at ${}^{h}r_i^{\star}$ with respect to the shoulder projection in the horizontal frame. Except that, with the base moving in world frame, the target location is moving as well. As such, at the start of each swing phase, the associated foot will target a position $[x\ y\ 0]^T + {}^{h}r_i^{\star}$ in world frame but, by the end of the swing phase, this position becomes $[x\ y\ 0]^T + {}^{h}r_i^{\star} + T_{swing}\ {}^{h}\dot{\mathbf{q}}_{lin}$, due to the movement of the base. With the assumption that current and reference velocities do not change much over one period of gait, feet could directly aim for their final target location by taking into account the movement of the base during their swing phase.

With the assumption that the quadruped moves with constant linear and angular velocities during the remaining duration of the swing phase then the predicted movement is:

- if ${}^{h}\dot{\psi}^{\star} \neq 0$:

$$ {}^{h}x_{pred}(t_r) = \int_0^{t_r} \left({}^{h}\dot{x}\cos({}^{h}\dot{\psi}\ t) - {}^{h}\dot{y}\sin({}^{h}\dot{\psi}\ t) \right)\ dt \tag{7.6} $$

$$ {}^{h}y_{pred}(t_r) = \int_0^{t_r} \left({}^{h}\dot{x}\sin({}^{h}\dot{\psi}\ t) + {}^{h}\dot{y}\cos({}^{h}\dot{\psi}\ t) \right)\ dt \tag{7.7} $$

$$ {}^{h}x_{pred}(t_r) = \frac{{}^{h}\dot{x}\sin({}^{h}\dot{\psi}\ t_r) + {}^{h}\dot{y}\left(\cos({}^{h}\dot{\psi}\ t_r) - 1\right)}{{}^{h}\dot{\psi}} \tag{7.8} $$

$$ {}^{h}y_{pred}(t_r) = \frac{-{}^{h}\dot{x}\left(\cos({}^{h}\dot{\psi}\ t_r) - 1\right) + {}^{h}\dot{y}\sin({}^{h}\dot{\psi}\ t_r)}{{}^{h}\dot{\psi}} \tag{7.9} $$

- otherwise if ${}^{h}\dot{\psi}^{\star} = 0$:

$$ {}^{h}x_{pred}(t_r) = {}^{h}\dot{x}\ t_r \tag{7.10} $$

$$ {}^{h}y_{pred}(t_r) = {}^{h}\dot{y}\ t_r \tag{7.11} $$

The remaining duration t_r for the swing phase of a foot can be directly retrieved using information contained in the gait matrix $\mathcal{G}_c$ ($\Delta t \times$ number of remaining rows with 0). This prediction term is:

$$ {}^{h}r_{pred} = \begin{bmatrix} {}^{h}x_{pred} \\ {}^{h}y_{pred} \\ 0 \end{bmatrix} \tag{7.12} $$

The desired location of footsteps is the sum of all the terms that were introduced. Symmetry, feedback and centrifugal terms are the same for all feet contrary to the shoulder and prediction terms.

In the end, we get corresponding footstep positions for each step of the gait matrix $\mathcal{G}_c$: the current feet positions ${}^{h}r_i$ for feet that are currently in stance phase and the future desired footstep locations ${}^{h}r_i^{\star}$ for incoming stance phases. With:

$$ \mathcal{G}_c = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \tag{7.13} $$

we get:

$$r_{\mathcal{G}} = \begin{bmatrix} {}^{h}r_1 & 0 & 0 & {}^{h}r_4 \\ {}^{h}r_1 & 0 & 0 & {}^{h}r_4 \\ 0 & {}^{h}r_2^{\star} & {}^{h}r_3^{\star} & 0 \\ 0 & {}^{h}r_2^{\star} & {}^{h}r_3^{\star} & 0 \\ 0 & {}^{h}r_2^{\star} & {}^{h}r_3^{\star} & 0 \\ {}^{h}r_1^{\star} & 0 & 0 & {}^{h}r_4^{\star} \end{bmatrix} \tag{7.14}$$

These positions can be obtained in world frame as well with a direct change of frame:

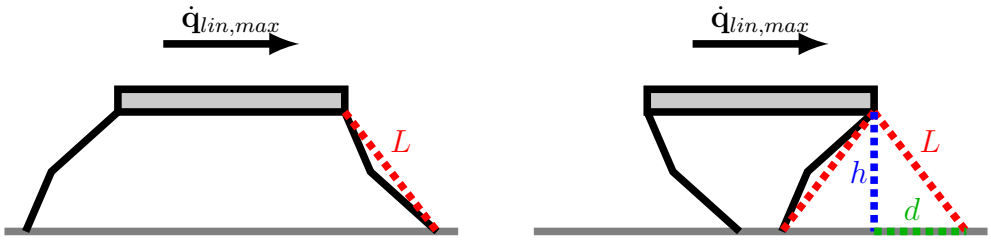
$$\forall i \in \{1,2,3,4\},\ {}^{o}r_i = {}^{o}\mathbf{R}_h\, {}^{h}r_i + {}^{o}\mathbf{T}_h \tag{7.15}$$

The body of the quadruped is moving w.r.t the world while the active contacts remain immobile, so they move w.r.t the horizontal frame in the opposite direction to what was shown in (5.52):

$$\forall i \in \{1,2,3,4\} \quad \text{s.t} \quad \mathcal{G}_c(0,i) = 1,\ {}^{h}r_{i,k} = \mathcal{R}_z(-\Delta t\, \dot{\psi}^{\star}) \left({}^{h}r_{i,k-1} - \Delta t \begin{bmatrix} {}^{h}\dot{x}^{\star} \\ {}^{h}\dot{y}^{\star} \\ 0 \end{bmatrix} \right) \tag{7.16}$$

7.1.3 Maximum reachable velocity

For a standard trotting gait with the two pairs of diagonally opposed feet in contact alternatively, there is a hard limit $\dot{\mathbf{q}}_{lin,max}$ on the velocity the robot can reach before the legs enter in singularity. This limit is linked to the duration of stance phases T_{stance}, the height h of the base and the maximum extension L of legs. In the limit case, the legs reach singularity just at touchdown when entering stance phase, and at the end of the stance phase when leaving the ground. This will be obtained using r_{sym} if the robot moves at constant velocity $\dot{\mathbf{q}}_{lin,max}$ with the reference velocity being $\dot{\mathbf{q}}_{lin,max}$, with a contact centered on the shoulder in average. In that case, the robot travels a distance $d = \dot{\mathbf{q}}_{lin,max}\, T_{stance}$ over a single stance. This distance is also equal to $d = \sqrt{L^2 - h^2}$ by geometric reconstruction (see Fig. 7.2). If $h = 22$ cm, $L = 28$ cm and $T_{stance} = 0.24$ s, we obtain $v_{max} \approx 1.44$ m/s in a perfect scenario with the base horizontal. If the base tilts a bit, for instance with a pitch angle of $5°$, hind shoulders are raised up to $h = 23.7$ cm, so the legs can extend less further and v_{max} falls down to 1.24 m/s. This maximum velocity is actually lowered by various parameters: the natural tilt of the base in roll and pitch during motion, a non-constant velocity during a single stance phase, and the r_{fb} and r_c terms of the heuristics which shift the contact so that it is not perfectly centered on the shoulder. The lower the base, the more efforts the knee actuators have to apply. So, further lowering the base to go faster is not a satisfying solution. Increasing the gait frequency can be a solution as long as the controller manages to handle it (feet tracking performances, base stabilization). Making the robot able to handle full flight phases would be another way to go beyond that limit.



(a) Start of contact (b) End of contact

Figure 7.2: Limit cases for which legs are in singularity, right at the beginning (a) and at the end of their stance phase (b). The robot moves at velocity $\dot{\mathbf{q}}_{lin,max}$ to the right (side view). Contacts have to stay inside the bounds of the red cone to avoid leg singularity. If for a given velocity this constraint is not respected, one can either lower the base to widen the cone or increase gait frequency to decrease the duration of stance phase so that feet land closer to the shoulder projection and get back in the cone.

7.2 Swing phase trajectory generator

During swing phases feet have to be guided from their current position to their next target position on the ground outputted by the footstep planner. To do so, we use polynomial interpolation, one for each foot, to generate a parabolic trajectory in the air, both in position, velocity and acceleration. Each foot is managed independently based on the contact sequence stored in the gait matrix $\mathcal{G}_c$. The trajectory generation is done in world frame (the ideal one in which the robot moves at reference velocity) so that the target position of the interpolation is constant in a stationary frame instead of having to deal with position, velocity and acceleration in a moving body frame.

The foot trajectory generator keeps feet in stance phase at the positions received as inputs. It receives $[{}^{o}r_{i,x}\ {}^{o}r_{i,y}\ 0]^T$ from the footstep planner and outputs as position, velocity and acceleration targets $[{}^{o}r_{i,x}\ {}^{o}r_{i,y}\ 0]^T$, $[0\ 0\ 0]$ and $[0\ 0\ 0]^T$ respectively. Feet in swing phase are led from their previous contact position to the next one using polynomials. The $[{}^{o}r^{\star}_{i,x}\ {}^{o}r^{\star}_{i,y}\ 0]$ location of the next contact is decided by the footstep planner as well. Note that the height of the target location is always zero following our assumption of moving on a flat ground.

7.2.1 Polynomial interpolation

For brevity purpose, we focus on a single foot to avoid having i indexes in all variables.

The polynomial interpolation is done from the current state of the associated foot $[x_f\ \dot{x}_f\ \ddot{x}_f\ y_f\ \dot{y}_f\ \ddot{y}_f\ z_f\ \dot{z}_f\ \ddot{z}_f]$ to reach the desired position on the ground $[x^{\star}_f\ y^{\star}_f\ 0]$. The control time step is denoted by Δt, the expected duration of the swing phase by t_1 and the time elapsed since the start of the swing phase by $t_0 \in [0, t_1]$. This information is processed to output a command $[x^{\star}_f\ \dot{x}^{\star}_f\ \ddot{x}^{\star}_f\ y^{\star}_f\ \dot{y}^{\star}_f\ \ddot{y}^{\star}_f\ z^{\star}_f\ \dot{z}^{\star}_f\ \ddot{z}^{\star}_f]$ for the foot.

For the x component, the generator tunes a 5-th order polynomial function $\mathcal{X}$ to have $\mathcal{X}(t_0) = x_f$, $\dot{\mathcal{X}}(t_0) = \dot{x}_f$ and $\ddot{\mathcal{X}}(t_0) = \ddot{x}_f$ while having $\mathcal{X}(t_1) = x^{\star}_f$, $\dot{\mathcal{X}}(t_1) = 0$ and $\ddot{\mathcal{X}}(t_1) = 0$. The generator can then output $[x^{\star}_f\ \dot{x}^{\star}_f\ \ddot{x}^{\star}_f]$ by computing $\mathcal{X}(t_0{+}\Delta t)$, $\dot{\mathcal{X}}(t_0{+}\Delta t)$ and $\ddot{\mathcal{X}}(t_0 + \Delta t)$. The same happens with a polynomial function $\mathcal{Y}$ for the y component to

output $[y_f^\star \ \dot{y}_f^\star \ \ddot{y}_f^\star]$. For instance, the coefficients of the $\mathcal{X}$ polynomial are as follows:

$$\mathcal{X}(t) = \sum_{i=0}^{5} a_i \, t^i \tag{7.17}$$

$$\begin{cases} a_5 &= -\frac{\ddot{x}_f \, T_s^2 + 6 \, \dot{x}_f \, t1 + 12 x_f - 12 x_f^\star}{2 \, T_s^5} \\ a_4 &= \frac{3 \, \ddot{x}_f \, T_s^2 + 16 \, \dot{x}_f \, t1 + 30 x_f - 30 x_f^\star}{2 \, T_s^4} \\ a_3 &= -\frac{3 \, \ddot{x}_f \, T_s^2 + 12 \, \dot{x}_f \, t1 + 20 x_f - 20 x_f^\star}{2 \, T_s^3} \\ a_2 &= \frac{\ddot{x}_f}{2} \\ a_1 &= \dot{x}_f \\ a_0 &= x_f \end{cases} \tag{7.18}$$

Command for the z component is deterministic and there is no feedback unlike the x and y components for which the current position, velocity and acceleration of the foot are taken into account. The z trajectory is made of two 5-th order polynomial that do not change and defined in such a way that $\mathcal{Z}(0) = \dot{\mathcal{Z}}(0) = \ddot{\mathcal{Z}}(0) = 0$ and $\mathcal{Z}(t_1) = \dot{\mathcal{Z}}(t_1) = \ddot{\mathcal{Z}}(t_1) = 0$ with $\mathcal{Z}(\frac{t_1}{2}) = h_{apex}$. h_{apex} is a constant value that sets the desired apex height of feet during their swing phases. The switch from one polynomial to the other happens at $t_1/2$. The shape of all trajectories are highlighted in Fig. 7.3.

Due to these characteristics, the trajectory generated can be described as a bell-shaped trajectory that goes from the initial position of the foot to its target trajectory while respecting non-slipping constraints during take-off and landing (no horizontal speed) and trying to land softly (0 final velocity and acceleration for z).

To keep this slipping-avoidance property, the target position on the ground $[{}^o r_{i,x}^\star \ {}^o r_{i,y}^\star \ 0]$ is locked t_{lock} seconds before landing. Basically $[{}^o r_{i,x}^\star \ {}^o r_{i,y}^\star \ 0]$ is not updated if $t_0 > (t_1 - t_{lock})$. Changing the desired position on the ground just before landing would create a non-negligible horizontal speed to correct the position of the foot in order to land at the new position. It is required because the target position is always changing since it is linked to the current velocity of the robot through the symmetry term of the footstep planner and this velocity is never exactly the same from one time step to another. This locking mechanism is illustrated in Fig. 7.4.

The tracking of the trajectory in swing phase is not perfect so at the next iteration the foot will not be in the commanded state. It could be possible to compute new coefficients for each iteration so that the polynomial starts at the new state of the foot and ends up at the desired location on the ground, as in Fig. 7.5. However, as we will see later, a foot tracking task in the whole-body controller is in charge of finding the adequate torques, joint positions and joint velocities to follow the commands of the trajectory generator. These are then tracked by the low-level impedance controller that runs on the embarked control board of the robot. So by taking into account the estimated state of the foot at each iteration, we would have two layers of feedback: one through the polynomial that guides the foot towards the next target position in the air, and another one through the impedance controller which drives the joints toward the reference joint positions and velocities that corresponds to the next target position in the air. This can result in stability issues with two feedbacks working toward the same goal, especially since they run at different frequencies: the trajectory generator runs at 1 kHz while the onboard impedance runs at 10 kHz. As a consequence, we do not include feedback in the foot trajectory generator. In other words, at the start of the swing phase the trajectory generator receives $[{}^o r_{i,x}^\star \ {}^o r_{i,y}^\star \ 0]$ to update the position of the foot and then the command of the generator

is supposed to be perfectly followed. As such, we just need to compute the coefficients of the polynomial once and then use $[\mathcal{X}(\Delta t)\ \dot{\mathcal{X}}(\Delta t)\ \ddot{\mathcal{X}}(\Delta t)]$, $[\mathcal{X}(2\Delta t)\ \dot{\mathcal{X}}(2\Delta t)\ \ddot{\mathcal{X}}(2\Delta t)]$ and so on. To work well, this assumes a good enough tracking by the low-level impedance controller so that joints are not too far from their reference state.

So far, the horizontal velocity command for the swinging feet only becomes 0 m/s right when the contact happens with $\dot{\mathcal{X}}(t_1) = 0$ and $\dot{\mathcal{Y}}(t_1) = 0$ as limit conditions for the interpolation. With imperfect tracking, a foot in swing phase could still have a small horizontal velocity at touchdown and drag on the ground. To decrease the risk of this happening at the start or end of the swing phase, we added a margin T_m during which swinging feet only move vertically so that the horizontal motion only occurs in the $[T_m, t_1 - T_m]$ interval, as shown in Fig. 7.6. The width of the margin must be reasonable since it decreases the time during which feet in swing phase move horizontally, hence increasing the necessary speed to go from one location to another one in the imparted time, as well as decreasing the maximum velocity the robot can reach before its legs reach singularity during motion, as shown in Fig. 7.7.

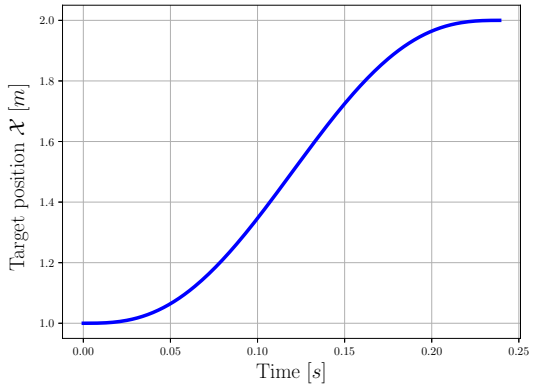
7.2.2 Extension with contact detection

The use of the online contact detection presented in Section 6.4 leads to the introduction of additional features in the footstep planner to be properly exploited. With it, contacts can be detected and thus enabled earlier or later than the expected end of the swing phase. If a swinging foot lands on a stair-like object at time t_c earlier than expected, then the contact is enabled at position $[\mathcal{X}(t_c)\ \mathcal{Y}(t_c)\ FG_z(\mathbf{q})]$ rather than the target position on the ground $[{}^o r^\star_{i,x}\ {}^o r^\star_{i,y}\ 0]$ that should have been reached at time t_1 (see Fig. 7.8). It relieves the assumption that contact heights are always 0, which could work on a flat ground but not on a rough terrain with bumps, holes or steps of various height. Note that we use forward geometry to assess the height of the contact instead of using $\mathcal{Z}(t_c)$.

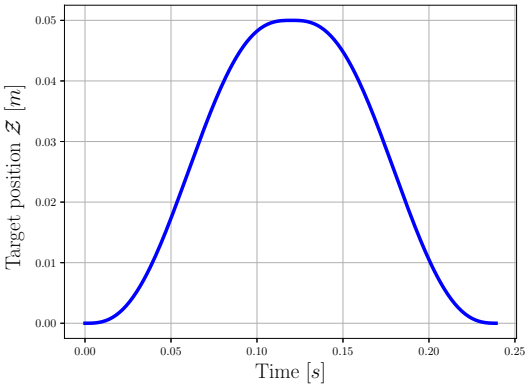
Without information about the environment, we cannot know if an early or late contact is a single occurrence or if we encountered a long-lasting change of the ground so we decided to keep to flat ground assumption by principle. So when a foot lands at a non-zero height ${}^o r_{i,z}$, the height of the next footstep location ${}^o r^\star_{i,z}$ will still be $\mathcal{Z}(t_1) = 0$. That way, if there is a single stair-like object separating two large flat areas, after the transition from one to the other, the robot will not try to put its feet back at the previous height.

If there is an unexpected depression at the target location, the foot will stop in the air a few centimeters above the ground. Without contact detection, that case would not be taken into account and the legs would just quickly extends as the contact gets enabled and the controller tries to apply forces on the ground. The trajectory generator is thus modified to extend the reference trajectory beyond the end of the swing phase at position ${}^o r^\star_i$. If the contact is late, the foot is slowly lowered at constant velocity until the ground is met. To avoid stopping the momentum of the foot at the end of the swing phase, and then moving once the controller realizes the contact is late, we modify the limit conditions of the interpolation so that the swing trajectory ends with a small vertical velocity $\dot{\mathcal{Z}}(t_1) \neq 0$, the same velocity that is used to lower the foot after the swing phase for late contacts.

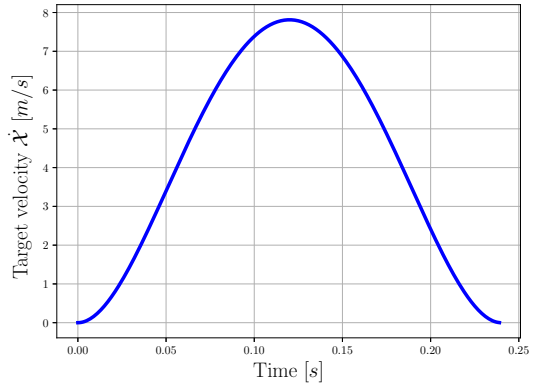
$$\dot{\mathcal{Z}}(t_1) = -\frac{h_{apex}}{T_s} \tag{7.19}$$



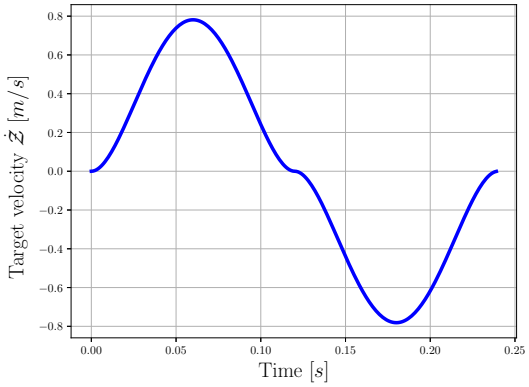
(a) Reference trajectory for $\mathcal{X}$



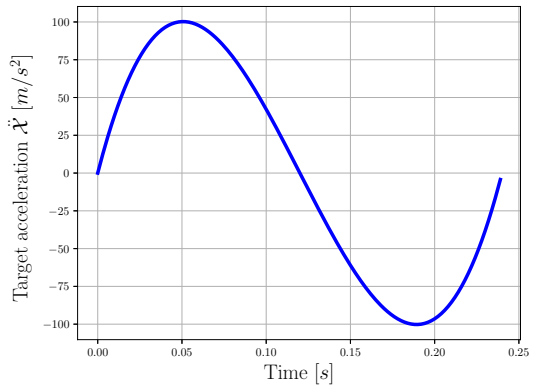
(b) Reference trajectory for $\mathcal{Z}$



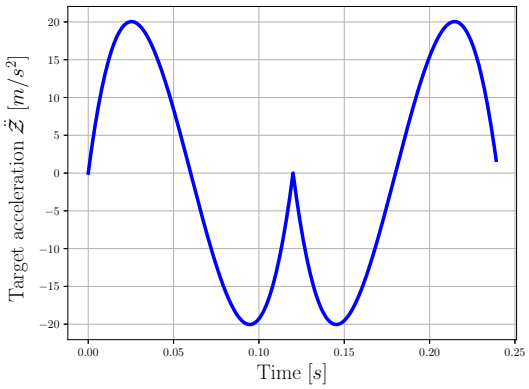
(c) Reference trajectory for $\dot{\mathcal{X}}$



(d) Reference trajectory for $\dot{\mathcal{Z}}$



(e) Reference trajectory for $\ddot{\mathcal{X}}$



(f) Reference trajectory for $\ddot{\mathcal{Z}}$

Figure 7.3: Shape of the reference trajectories in position, velocity and acceleration along the x and z axis generated through polynomial interpolation for a 0.24s long swing phase. The trajectories along y are similar to the ones along x and are thus not represented.

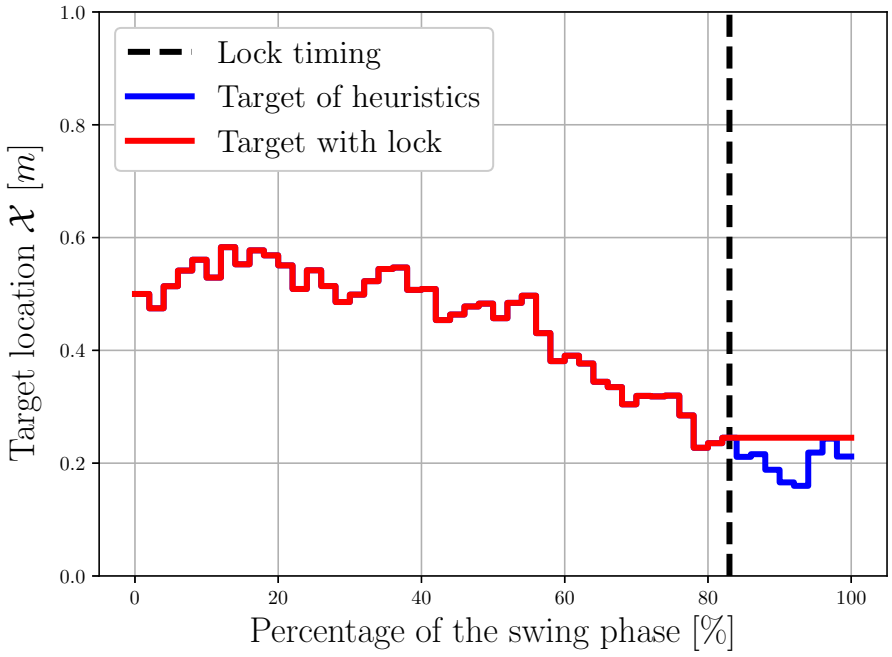


Figure 7.4: The target position on the ground is locked a few milliseconds before landing (here 4 ms for a 24 ms swing phase) to ensure the foot has no horizontal velocity at touchdown.

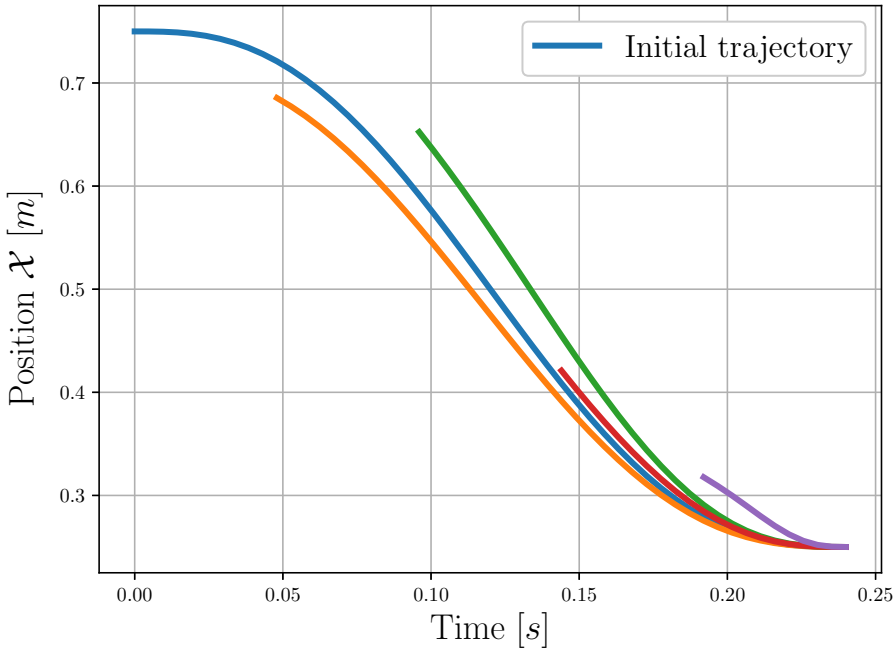
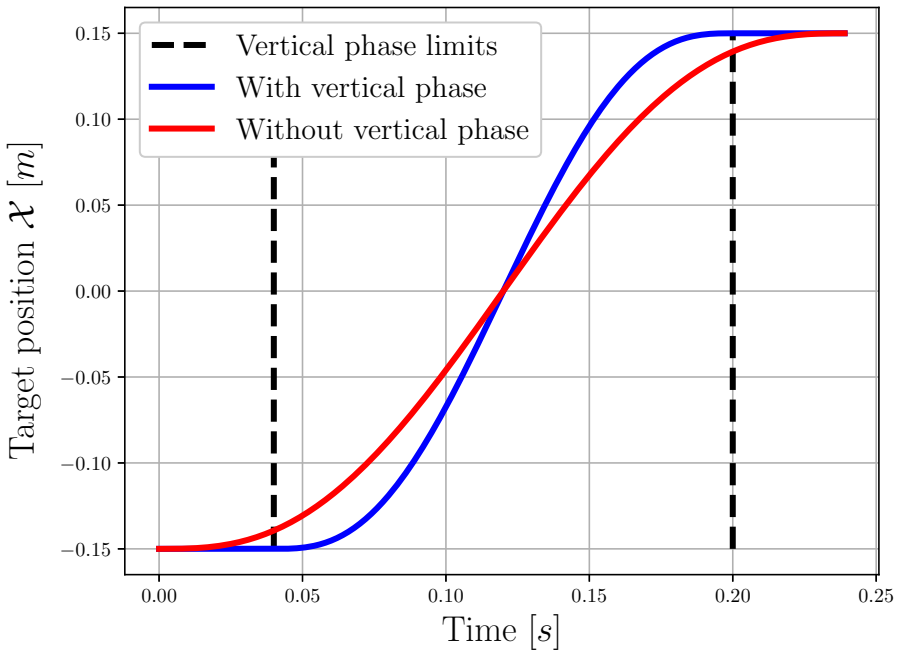
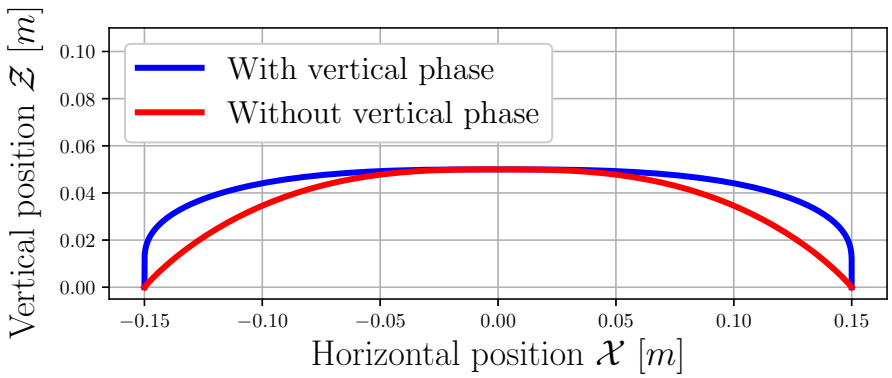


Figure 7.5: New trajectories can be computed online to bring a swinging foot from its current position to the target even if it deviates from the initial trajectory.



(a) Comparison of reference trajectories for $\mathcal{X}$ with and without vertical phase



(b) Trajectory in the air ($\mathcal{X} - \mathcal{Z}$ graph) with and without vertical phase. The scale of both axis is the same to better represent the shape.

Figure 7.6: Shape of the reference trajectories in position generated through polynomial interpolation for a 0.24s long swing phase with a vertical phase of 0.04 s at the start and end. With it, even with imperfect tracking the swinging foot has more time to stop moving horizontally before touchdown due to the vertical landing approach, yet it forces the foot to move faster in the air due to the shorter horizontal motion. The trajectory along y is similar to the one along x and is thus not represented.

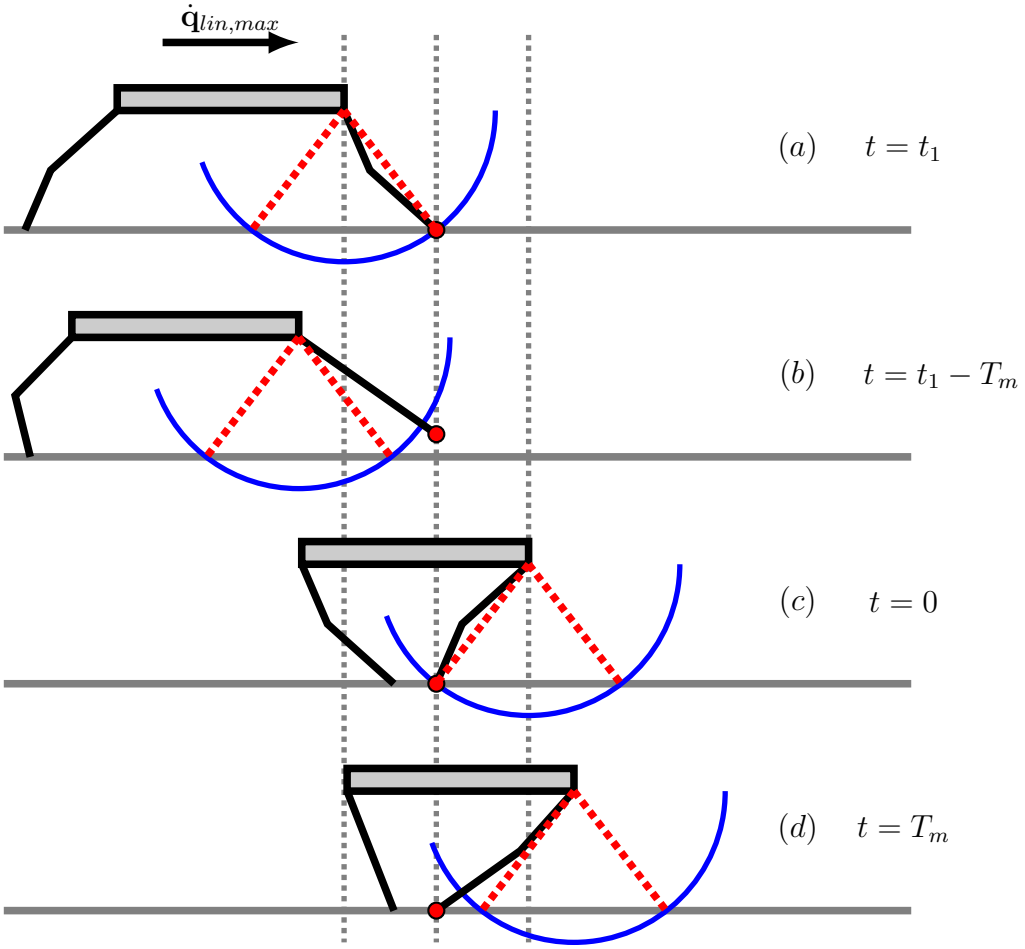
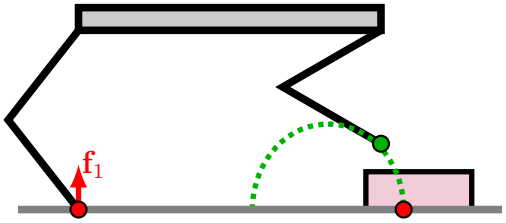
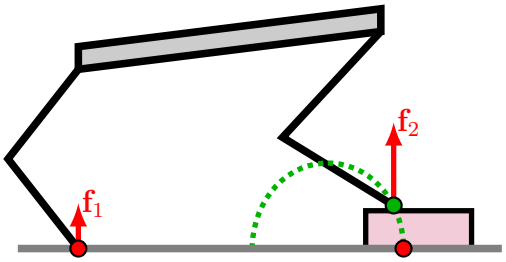


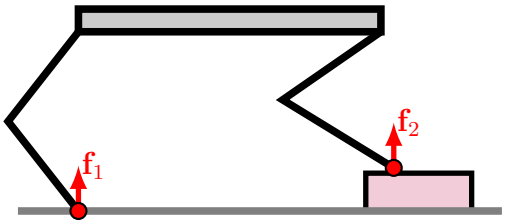
Figure 7.7: Limit cases similar to the ones of Fig. 7.2 for which legs are almost in singularity right at the beginning (a) and at the end (c) of their stance phase. The blue arc delimits the range of the leg. If this happens in a situation with no vertical phase for the swinging feet motion, then introducing a vertical phase will lead to singularity issues. Indeed, in (b), since the foot has to reach its final horizontal position sooner at the end of the swing phase due the phase with only vertical motion (see top right corner of Fig. 7.6a), the leg has to extend further forwards, which is not possible if the robot is already at its limit. Similarly, in (d), since the foot has to stay at the same horizontal position longer at the start of the swing phase while the foot is moving only vertically (see bottom left corner of Fig. 7.6a), the leg has to extend further backwards. As a consequence, to make this movement possible, we would either have to decrease the velocity of the robot or decrease the duration of the vertical phase.



(a) Due to the assumption of a flat ground and the lack of exteroception, the swinging foot aims for a target at ground level.



(b) Without online contact detection the swinging foot keeps trying to reach the target until the predefined end of the swing phase. As a consequence of an unexpected ground reaction force, the base is tilted.



(c) With online contact detection the early impact is detected and the foot enters stance phase at an updated location.

Figure 7.8: Online contact detection allows the controller to detect early or late impacts when walking on rough terrain. That way, the robot can react accordingly and avoid disturbing base stability by applying undesired forces on the ground (early contact) or trying to apply forces in vain with the foot still in the air (late contact).

7.3 Using information about the environment

Blind controllers, such as the one we are using, do not exploit from data or sensors that would allow them to get information about their environment. Yet some of them are still able to drive a robot through rough terrain thanks to their sheer robustness. So long as the environmental conditions are not overly harsh, they manage to react in real time to perturbations to maintain balance, such as when climbing stairs or navigating on uneven terrains with the assumption of a flat ground. However, blind controllers remains insufficient in circumstances where the environment contains deep holes or steps too high to climb: basically obstacles that are out of range of their designed capabilities. Those controllers can sometimes be augmented with exteroceptive sensors or priviledged information about the environement to simultaneously plan the motion and footstep locations several steps ahead. It can allow the robot to overcome complex situations while respecting its dynamics as long as hardware capabilities are sufficient to produce the desired motion.

Such an augmentation of the baseline architecture was developed in a collaborative effort led by Fanny Risbourg and Thomas Corbères to implement a 3-stage locomotion framework able to navigate on complex 3D terrains. The first stage consists of a contact planner formulated as a Mixed-Integer Program to choose on which surfaces to step on, based on a set of available surfaces in the environment (see Fig. 7.9). Then, a quadratic program optimizes the footsteps locations coming from the heuristics presented in Section 7.1 to respect the limits of the chosen surfaces. Finally, an improved trajectory generator using 3D Bézier curves drives the swinging feet from one location to another under collision-avoidance constraints. Thanks to the modularity of the architecture, this framework can seamlessly augment the control pipeline. The whole implementation is presented in details in [Ris+22].

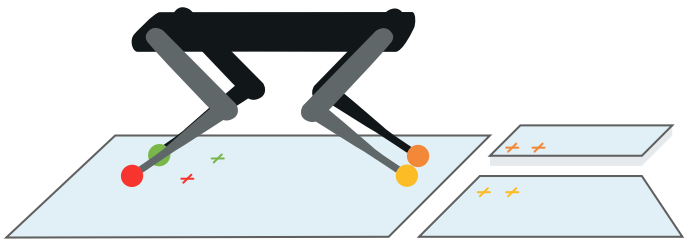


Figure 7.9: Contact planning with a mixed-integer program using a set of available convex surfaces, for each foot independently and using heuristics as a baseline. Figure extracted from [Ris+22].

In this framework, the environment is the union of m disjoint quasi-flat contact surfaces $\mathcal{S} = \bigcup_{j=1}^{m} \mathcal{S}^j$. A quasi-flat contact surface is a surface for which the opposite of the gravity vector is contained in the normal friction cone at each. These surfaces can be either directly given to the robot as ground truth or reconstructed from exteroceptive measurements using a camera or a LIDAR for instance. The contact plan consists of a list of contact surfaces for each foot and for each contact phase in the planning horizon.

A set of n variables $\mathbf{a}_i = [a_i^1, \cdots, a_i^m] \in \{0,1\}^m, 1 \leq i \leq n$ is defined such that $a_i^j = 0$ implies that the i-th footstep position r_i belongs to the j-th contact surface, $a_i^j = 1$ otherwise. The following mixed integer problem is then solved to find the contact surfaces to aim for:

$$
\begin{aligned}
\textbf{find} \quad & \mathbf{R} = [r_1, \cdots, r_n] \in \mathbb{R}^{3\times n} \\
& \mathbf{A} = [\mathbf{a}_1, \cdots, \mathbf{a}_n] \in \{0,1\}^{n\times m} \\
\textbf{min} \quad & \ell(\mathbf{R}) \\
\textbf{s.t.} \quad & \mathbf{R} \in \mathcal{I} \cap \mathcal{F} \\
& \forall i, 1 \leq i \leq n : \\
& \mathbf{card}(\mathbf{a}_i) = m - 1 \\
& \quad \forall j, 1 \leq j \leq m : \\
& \quad\quad \mathbf{S}_j r_i \leq \mathbf{s}_j + M a_i^j \mathbf{1}.
\end{aligned}
\tag{7.20}
$$

where ℓ is a quadratic objective function to minimise, $\mathcal{I}$ is a user-defined convex set of initial constraints, $\mathcal{F}$ is a convex set of feasibility constraints, $M \in \mathbb{R}^n$ is a sufficiently large number, and $\mathbf{S}^j \in \mathbb{R}^{h\times 3}$ and $\mathbf{s}^j \in \mathbb{R}^h$ are respectively a constant matrix and a vector defining the h half-spaces bounding the surface to check whether r_i lies within surface $\mathcal{S}^j$. The constraint $\mathbf{card}(\mathbf{a}_i) = m - 1$ guarantees that at each step the position planned lies exactly on one contact surface, with $\mathbf{card}(\mathbf{a}_i)$ is the number of non-zero coefficients in $\mathbf{a}_i$ (cardinality).

The cost function ℓ includes the distance between the 2D feet positions r_i optimized in $\mathbf{R}$ and the 2D target feet positions $r_i^\star$ given by the Raibert heuristics as well as the distance between r_i and its associated shoulder.

$$
\ell(\mathbf{R}) = \|r_i^\star - r_i\|^2 + 0.1\|r_{sh,i} - r_i\|^2 \tag{7.21}
$$

Then, given the current state of the robot and the target contact surfaces chosen by the mixed-integer problem, we compute the next contact locations of the swinging feet $r_i, 1 \leq i \leq n$ on said surfaces, this time using the whole footsteps heuristics as introduced in Section 7.1. Each position variable is defined with an offset $\epsilon_i = [\epsilon_{i,x}\ \epsilon_{i,y}\ \epsilon_{i,z}]^T \in \mathbb{R}^3$ with respect to the heuristic locations that depends on the reference velocity $\mathbf{q}_u^\star$:

$$
r_i(\mathbf{q}_u^\star) + \epsilon_i = \begin{bmatrix} r_{i,x}^\star(\mathbf{q}_u^\star) \\ r_{i,y}^\star(\mathbf{q}_u^\star) \\ 0 \end{bmatrix} + \begin{bmatrix} \epsilon_{i,x} \\ \epsilon_{i,y} \\ \epsilon_{i,z} \end{bmatrix} \tag{7.22}
$$

The reference velocity $\mathbf{q}_u^\star$ is updated with an offset $\xi = [\xi_{i,x}\ \xi_{i,y}]^T \in \mathbb{R}^2$, resulting in $\mathbf{q}_u^{\star,+}$:

$$
\mathbf{q}_u^{\star,+} = \begin{bmatrix} \xi_x + \mathbf{q}_{u,x}^\star \\ \xi_y + \mathbf{q}_{u,y}^\star \end{bmatrix} \tag{7.23}
$$

The QP problem is written as follows:

$$
\begin{aligned}
\min_{\epsilon,\xi} \quad & \frac{1}{2}\Big(\sum_{i=1}^{n} \|\epsilon_i\|^2 + 1000\,\|\xi\|^2\Big) \\
\text{s.t.} \quad & \forall 1 \leq i \leq n,\ \mathbf{S}_i(r_i(\mathbf{q}_u^{\star,+}) + \epsilon_i) \leq \mathbf{s}_i
\end{aligned}
\tag{7.24}
$$

where $\mathbf{S}_i$ and $\mathbf{s}_i$ are associated with surface $\mathcal{S}_i$ that has been selected for the i-th contact. The goal of this quadratic problem is to satisfy at best the surface constraints while minimising the violation of the heuristics in the least-square sense.

Once the target position on the surface has been determined, the swing trajectory is computed as a compromise between a reference obtained from the polynomial interpolation described in Section 7.2 and the adjustments required to avoid collisions with the

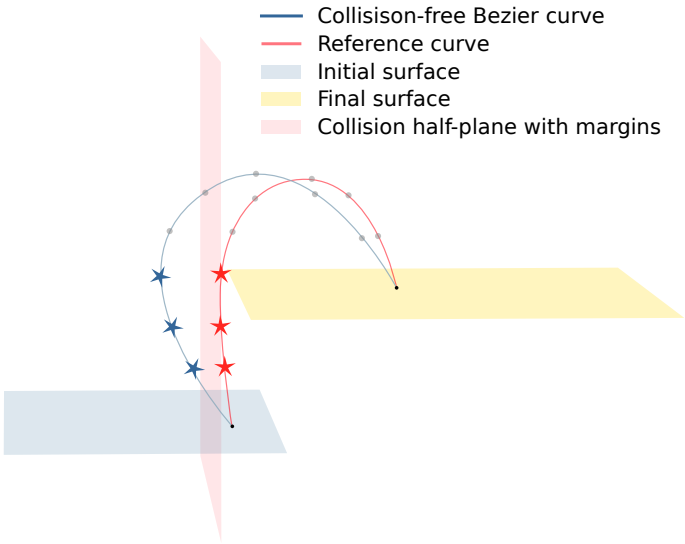


Figure 7.10: When climbing a stair, using a predefined bell-like trajectory would lead to a collision with the edge of the step. A security margin can instead be taken by using a Bezier curve whose control points are modified by a QP problem. Figure extracted from [Ris+22].

environment (see Fig. 7.10). To do so, the control points of a 3D Bézier curve are modified by a quadratic program. The resulting commands are then sent to the whole-body control as replacements of the blind polynomial interpolation.

This augmented pipeline was successfully deployed on the quadruped to handle environments more complex than a simple flat ground, such as stairs, stepping stones or small bridges. As the goal was to test the architecture before implementing a full online detection, the positions of surfaces in the environment were known and the robot was located thanks to a motion capture system (see Fig. 7.11).

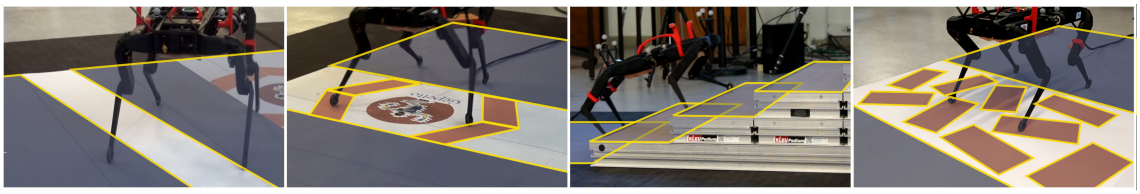


Figure 7.11: Only surfaces bounded in yellow were available for footstep locations to mimic stepping stones or bridges without the risk of having actual holes in case of a failure. Figure extracted from [Ris+22].

In this collaboration my contribution was focused on integrating this planning scheme in the control architecture and on helping to the realization of the experiments.

7.4 Conclusion

In this chapter we have first described how the footstep planner uses several heuristic terms to determine target locations on the ground with the main assumption of a flat ground. It is based on the Raibert heuristics so that contacts are centered on their associated shoulder projection in average. The planner works over the whole sequence contained in the gait matrix $\mathcal{G}_c$ so that the controller can directly aim for the final position of incoming contacts instead of reasoning in a moving body frame. Then, we explained how

polynomial interpolation is used to generate reference trajectories in position, velocity and acceleration to guide swinging feet from their current position to their next target on the ground. We showed how the online contact detection presented in Section 6.4 leads to the introduction of new features to exploit early or late contact detections. The most notable of them is the switch from purely 2D contacts due to the assumption of a flat ground to 3D contacts that can handle terrains with various heights. Finally, we presented how the control architecture can be augmented to leverage information about the environment in order to navigate complex terrains that would be out of range of our baseline blind controller.

The next chapter will focus on the different kinds of centroidal model predictive controllers that were implemented to obtain forces that should be applied at contact points to follow the reference base velocity. It uses information coming from previously described control elements to perform an optimization over a prediction horizon.

Chapter 8

Centroidal Model Predictive Control

Contents

As locomotion decisions must be taken by considering the future evolution of the system [KE08], a wide range of quadruped controllers leverages a model predictive controller (MPC) to generate the motion in real time by predicting the behavior of the robot over a prediction horizon. Then, a whole-body controller (WBC) converts those decisions into actuator commands to follow the movement. MPCs usually exploit a reduced model of the dynamics to limit the computational complexity. Since quadruped robots tend to have lightweight limbs, most of their mass is localized in their trunk and, as such, centroidal dynamics [OGL13] can provide an appropriate approximation of their whole-body dynamics. It describes the dynamics of the center of mass of the robot due to its interactions with the environment and corresponds to the under-actuated dynamics [CBM17]:

$$m\ddot{c} = \sum_{i=1}^{n_c} f_i + m\mathbf{g} \tag{8.1a}$$

$$\mathcal{I}\dot{\omega} + \omega \times (\mathcal{I}\omega) = \sum_{i=1}^{n_c} (r_i - c) \times \mathbf{f}_i \tag{8.1b}$$

with c the position of the CoM, ω the angular velocity of the body, m the total mass of the robot, $\mathcal{I}$ its inertia matrix, and $\mathbf{g}$ the gravity vector. n_c is the number of 3D forces $\mathbf{f}_i$ applied at the contact points r_i.

8.1 Reference state trajectory

8.1.1 Locomotion on the ground

The reference velocity ${}^h\dot{\mathbf{q}}_u^\star$ that is sent to the robot is expressed in its horizontal frame (see Fig. 5.1). It has 6 dimensions: 3 for the linear velocity and 3 for the angular one.

$${}^h\dot{\mathbf{q}}_u^\star = [{}^h\dot{x}^\star \;\; {}^h\dot{y}^\star \;\; {}^h\dot{z}^\star \;\; \dot{\phi}^\star \;\; \dot{\theta}^\star \;\; \dot{\psi}^\star]^T \tag{8.2}$$

The velocity vector of the robot is:

$${}^h\dot{\mathbf{q}}_u = [{}^h\dot{x} \;\; {}^h\dot{y} \;\; {}^h\dot{z} \;\; \dot{\phi} \;\; \dot{\theta} \;\; \dot{\psi}]^T \tag{8.3}$$

Strictly speaking, the angular velocities in roll, pitch and yaw ($\dot{\phi}$, $\dot{\theta}$, $\dot{\psi}$) should be replaced by angular velocities around the axes of the horizontal frame (ω_{h_x}, ω_{h_y}, ω_{h_z}). We decided to keep the former notation since it is more in line with the idea of controlling roll, pitch and yaw angles. Moreover, with the assumption of small angles with a base that remains almost horizontal, those quantities are close to each other.

At the start each of iteration of the MPC, the current position and orientation of the robot define a new frame in which the solver will work. This frame is similar to the horizontal frame h, with the x axis pointing forwards, the y axis pointing laterally to the left and the z axis upwards. However, it is at ground level rather than being centered on the robot body. As a result, instead of working in terms of position, rotation and velocity along the x, y and z axes of the world frame, the solver will work in this new local frame, noted l, as already presented in Fig. 5.1. The relations between l and the other frames are:

$${}^o\mathbf{T}_l = \begin{bmatrix} x & y & 0 \end{bmatrix}^T \tag{8.4}$$

$${}^o\mathbf{R}_l = \mathcal{R}(0, 0, \psi) = \mathcal{R}_z(\psi) \tag{8.5}$$

$${}^l\mathbf{T}_h = \begin{bmatrix} 0 & 0 & z \end{bmatrix}^T \tag{8.6}$$

$${}^l\mathbf{R}_h = \mathcal{R}(0, 0, 0) = \mathbf{I}_3 \tag{8.7}$$

with $\mathcal{R}_z(\psi)$ the 3 by 3 rotation matrix by an angle ψ about the vertical axis. The initial conditions of the solving process express as follows:

$${}^l\mathbf{q}_{u,0} = [{}^lc_x \;\; {}^lc_y \;\; {}^lc_z \;\; {}^l\phi \;\; {}^l\theta \;\; 0]^T \tag{8.8}$$

$${}^l\dot{\mathbf{q}}_{u,0} = [{}^l\dot{x} \;\; {}^l\dot{y} \;\; {}^l\dot{z} \;\; {}^l\dot{\phi} \;\; {}^l\dot{\theta} \;\; {}^l\dot{\psi}]^T \tag{8.9}$$

with $c = [{}^lc_x \;\; {}^lc_y \;\; {}^lc_z]$ the position of the CoM in local frame. In the rest of the thesis, the first three components of $\mathbf{q}$ contain the position of the base center, but here for the centroidal MPC, we are working with the position of the CoM.

When the robot is in its default initial position, we compute the offset Δc_0 between the center of the base $\mathbf{q}_{lin}$ and the center of mass c and keep it in memory ($c_0 = \mathbf{q}_{lin,0} + \Delta c_0$). We later assume that this offset is constant instead of recomputing it at each new configuration of the robot.

$$\forall t \;\; c_t = \mathbf{q}_{lin,t} + \Delta c_0 \tag{8.10}$$

In practice, with the knees bent in the same direction, this offset is slightly downwards and backwards with no lateral component due to the lateral symmetry of the robot. With the knees bent in opposite direction, this offset is only downwards due to the front-back symmetry. In (8.9) we assume that the linear velocity of the center of mass is the one of the body. The state vector of the robot and the reference state vector at step k are then:

$$\mathbf{X}_k = \begin{bmatrix} {}^l\mathbf{q}_{u,k} \\ {}^l\dot{\mathbf{q}}_{u,k} \end{bmatrix} \quad \mathbf{X}_k^\star = \begin{bmatrix} {}^l\mathbf{q}_{u,k}^\star \\ {}^l\dot{\mathbf{q}}_{u,k}^\star \end{bmatrix} \tag{8.11}$$

The reference velocity is supposed to be constant over the prediction horizon in the local frame of the robot, so it has to be properly rotated to be consistent with its future orientation. At time step k of the prediction horizon, the reference velocity vector is defined as follows:

$$\forall k \in [1, n_{steps}],\; {}^l\dot{\mathbf{q}}_{u,k}^\star = \begin{bmatrix} \mathcal{R}_z(k\;\Delta t\;{}^l\dot{\psi}^\star) \\ \mathcal{R}_z(k\;\Delta t\;{}^l\dot{\psi}^\star) \end{bmatrix}\; {}^l\dot{\mathbf{q}}_u^\star \tag{8.12}$$

with Δt the time step of the MPC. Contrary the estimator, planner and whole-body control which run at 1 kHz and thus use $\Delta t = 1$ ms, the MPC runs only at 50 Hz so in this chapter $\Delta t = 20$ ms. There is no rotation about the roll and pitch axes for ${}^l\dot{\mathbf{q}}_{u,k}^\star$ due to the assumption that the trunk is almost horizontal.

Then, an integration similar to the one that has been done for the prediction term of the footstep planner in Section 7.1 is performed to get the reference position vector for all time steps of the prediction horizon:

- if ${}^l\dot{\psi}^\star = 0$:

$$\forall k \in [1, n_{steps}],\; {}^l\mathbf{q}_{u,k}^\star = {}^l\mathbf{q}_{u,0} + k\;\Delta t\;{}^l\dot{\mathbf{q}}_u^\star \tag{8.13}$$

- otherwise if ${}^l\dot{\psi}^\star \neq 0$:

$${}^lx_k^\star = {}^lc_x + \frac{{}^l\dot{x}^\star \sin(k\;\Delta t\;{}^l\dot{\psi}^\star) + {}^l\dot{y}^\star\left(\cos(k\;\Delta t\;{}^l\dot{\psi}^\star) - 1\right)}{{}^l\dot{\psi}^\star} \tag{8.14}$$

$${}^ly_k^\star = {}^lc_y + \frac{-{}^l\dot{x}^\star\left(\cos(k\;\Delta t\;{}^l\dot{\psi}^\star) - 1\right) + {}^l\dot{y}^\star \sin(k\;\Delta t\;{}^l\dot{\psi}^\star)}{{}^l\dot{\psi}^\star} \tag{8.15}$$

$${}^lz_k^\star = {}^lc_z + k\;\Delta t\;{}^l\dot{z}^\star \tag{8.16}$$

$${}^l\phi_k^\star = {}^l\phi + \frac{\dot{\phi}^\star \sin(k\;\Delta t\;{}^l\dot{\psi}^\star) + {}^l\dot{\theta}^\star\left(\cos(k\;\Delta t\;{}^l\dot{\psi}^\star) - 1\right)}{{}^l\dot{\psi}^\star} \tag{8.17}$$

$${}^l\theta_k^\star = {}^l\theta + \frac{-{}^l\dot{\phi}^\star\left(\cos(k\;\Delta t\;{}^l\dot{\psi}^\star) - 1\right) + {}^l\dot{\theta}^\star \sin(k\;\Delta t\;{}^l\dot{\psi}^\star)}{{}^l\dot{\psi}^\star} \tag{8.18}$$

$${}^l\psi_k^\star = 0 + k\;\Delta t\;{}^l\dot{\psi}^\star \tag{8.19}$$

Previous equations could be used in a general case for which there is a velocity control for all linear and angular components. However, in our case, since we want the quadruped

to move around while keeping the trunk horizontal and at constant height, we want a velocity control in x, y and ψ and a position control in z, ϕ and θ to keep $\forall t,\ {}^{l}z(t) = h^{\star}$ and ${}^{l}\phi(t) = {}^{l}\theta(t) = 0$. In other words and as previously pointed out in Section 5.5, the reference body velocity given to the robot by a user or a higher-level controller is limited to the horizontal plane: only forward, lateral and yaw reference velocities are non-zero. Roll, pitch and vertical ones are always 0:

$$ {}^{l}\dot{z}_k^{\star} = 0 \text{ and } {}^{l}z_k^{\star} = h \tag{8.20} $$

$$ {}^{l}\dot{\phi}_k^{\star} = 0 \text{ and } {}^{l}\phi_k^{\star} = 0 \tag{8.21} $$

$$ {}^{l}\dot{\theta}_k^{\star} = 0 \text{ and } {}^{l}\theta_k^{\star} = 0 \tag{8.22} $$

To sum up:

$$ \forall k \in [1, n_{steps}],\ \mathbf{X}_k^{\star} = \begin{bmatrix} {}^{l}c_x + \frac{{}^{l}\dot{x}^{\star}\sin(k\ \Delta t\ {}^{l}\dot{\psi}^{\star}) + {}^{l}\dot{y}^{\star}\left(\cos(k\ \Delta t\ {}^{l}\dot{\psi}^{\star}) - 1\right)}{{}^{l}\dot{\psi}^{\star}} \\ {}^{l}c_y + \frac{-{}^{l}\dot{x}^{\star}\left(\cos(k\ \Delta t\ {}^{l}\dot{\psi}^{\star}) - 1\right) + {}^{l}\dot{y}^{\star}\sin(k\ \Delta t\ {}^{l}\dot{\psi}^{\star})}{{}^{l}\dot{\psi}^{\star}} \\ h \\ 0 \\ 0 \\ k\ \Delta t\ {}^{l}\dot{\psi}^{\star} \\ {}^{l}\dot{x}^{\star}\cos(k\ \Delta t\ {}^{l}\dot{\psi}^{\star}) - {}^{l}\dot{y}^{\star}\sin(k\ \Delta t\ {}^{l}\dot{\psi}^{\star}) \\ {}^{l}\dot{x}^{\star}\sin(k\ \Delta t\ {}^{l}\dot{\psi}^{\star}) + {}^{l}\dot{y}^{\star}\cos(k\ \Delta t\ {}^{l}\dot{\psi}^{\star}) \\ 0 \\ 0 \\ 0 \\ {}^{l}\dot{\psi}^{\star} \end{bmatrix} \tag{8.23} $$

The MPC solvers will optimize with contact forces so that the predicted trajectory of the center of mass stays close to the reference trajectory. We define the optimization state vector as:

$$ \mathcal{X}_k = \mathbf{X}_k - \mathbf{X}_k^{\star} \tag{8.24} $$

8.1.2 Jumping

Whereas we imposed a 0 vertical velocity with a fixed body height reference in the previous section for walking purpose, this choice should be modified to generate a jump trajectory for the center of mass. The implementation of a jump phase is seamless by inserting rows full of 0s the content of the gait matrix $\mathcal{G}_c$, corresponding to phases of the gait sequence for which all feet are in swing phase. The duration of the jump is noted T_J. This duration directly sets the apex height of the jump as well as the vertical velocity of the body that needs to be reached just before the four feet leave the ground. These values come from the trajectory of a mass in ballistic fall under the effect of the gravity.

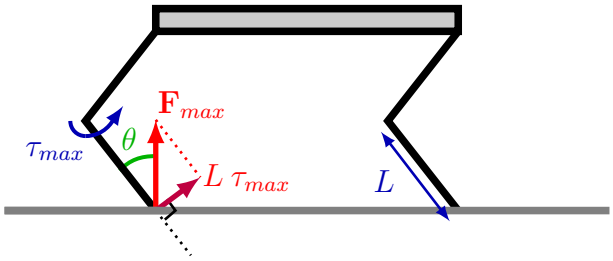


Figure 8.1: The angle that a leg makes with the ground for a given height of the base can be computed assuming that Hip Flexion-Extension and Knee joints have the same angle. The maximum force that can be applied by the leg along the vertical axis can then be obtained with a projection knowing leg length L and actuator limit τ_{max}.

Assuming the jump lasts from $t = 0$ to $t = T_J$, with $g = 9.81\ m/kg/s^2$:

$$ {}^{l}z^{\star}(t) = \frac{1}{2}\, g\, t\, (T_J - t) + h \tag{8.25} $$

$$ {}^{l}\dot{z}^{\star}(t) = g\, (\frac{T_J}{2} - t) \tag{8.26} $$

$$ {}^{l}z^{\star}(\frac{T_J}{2}) = g\frac{T_J^2}{8} + h \tag{8.27} $$

$$ {}^{l}\dot{z}^{\star}(0) = g\, \frac{T_J}{2} = {}^{l}\dot{z}_0^{\star} \tag{8.28} $$

These values of ${}^{l}z^{\star}$ and ${}^{l}\dot{z}^{\star}$ could directly be included in $\mathbf{X}_k^{\star}$. However, it might be difficult for the solver to find the correct motion to jump, that is lowering the base before the jump and suddenly raising it to reach the target vertical velocity. Moreover, the solver has no information about actuators capabilities which impact the maximum vertical acceleration of the base, so it cannot know how much the base should be lowered according to actuators limits. Besides, this pre-jump motion would go against the reference trajectory ${}^{l}z^{\star}(t) = h^{\star}$ and ${}^{l}\dot{z}^{\star}(t) = 0$ so it would lead to a high tracking error and the solver might not find a proper solution due to the shape of the cost function.

A way to help the solver find a proper solution could be to modify the pre-jump reference trajectory to include this lowering and raising of the base. How much the base needs to be lowered depends on the vertical acceleration applied during the raising phase, which is directly linked to the contact forces. The amount of force that can be applied on the ground is limited by actuator capabilities, depending on the angle $\theta_{leg} \in]0, \frac{\pi}{2}]$ made by the legs with the ground. This angle can be directly computed for a given height of the base assuming Hip-Flexion Extension (HFE) and Knee joints have the same values and opposite sign (see Fig. 8.1). The higher θ_{leg}, the less force can be applied:

$$ \theta_{leg} = \cos^{-1}(\frac{z_{low}}{2L}) \tag{8.29} $$

$$ F_{z,max} = \frac{\tau_{max}}{L\ \sin(\theta_{leg})} \tag{8.30} $$

$$ \ddot{z}_{max} = -g + \frac{4F_{z,max}}{m} \tag{8.31} $$

At the lowest position, the vertical velocity is null. At the end of the raising phase, it should be equal to ${}^{l}\dot{z}_0^{\star}$. So if we aim for a constant acceleration from the lowest position, we get:

$$ \dot{z}(t_{max}) = \ddot{z}_{max}\, t_{max} + 0 = {}^{l}\dot{z}_0^{\star} \implies t_{max} = \frac{{}^{l}\dot{z}_0^{\star}}{\ddot{z}_{max}} \tag{8.32} $$

Knowing the raising time, we can deduce how much the base will raise before reaching the target velocity that makes the jump happen:

$$z(t_{max}) - z_{low} = \ddot{z}_{max} \frac{t_{max}^2}{2} \tag{8.33}$$

If we want the jump to happen at the reference height $h^\star$ then we can replace $z(t_{max})$ by h in (8.33) and express t_{max} according to z_{low} using previous equations:

$$h^\star - z_{low} = \frac{{}^l\dot{z}_0^{\star,2}}{2\ddot{z}_{max}} \tag{8.34}$$

$$= \frac{{}^l\dot{z}_0^{\star,2}}{2(-g + \frac{4\tau_{max}}{mL\sin(\cos^{-1}(\frac{z_{low}}{2L}))})} \tag{8.35}$$

$$= \frac{{}^l\dot{z}_0^{\star,2}}{2(-g + \frac{4\tau_{max}}{mL\sqrt{1-\frac{z_{low}^2}{4L^2}}})} \tag{8.36}$$

This equation can be solved numerically to get $z_{low}(T_J)$ for a whole range of T_J using (8.28) to replace ${}^l\dot{z}_0^{\star,2}$. We can also get $\ddot{z}_{max}(T_J)$ and $t_{max}(T_J)$ from (8.34) and (8.32) respectively:

$$\ddot{z}_{max} = \frac{{}^l\dot{z}_0^{\star,2}}{2(h^\star - z_{low})} \tag{8.37}$$

$$t_{max} = \frac{2(h^\star - z_{low})}{{}^l\dot{z}_0^\star} \tag{8.38}$$

We can now generate the raising trajectory ($t \in [-t_{max}, 0]$) using polynomial interpolation and link it to the ballistic fall of (8.28).

$${}^l\ddot{z}^\star(t) = \ddot{z}_{max} \tag{8.39}$$

$${}^l\dot{z}^\star(t) = \ddot{z}_{max}(t - t_{max}) \tag{8.40}$$

$${}^lz^\star(t) = \frac{\ddot{z}_{max}}{2}(t - t_{max})^2 \tag{8.41}$$

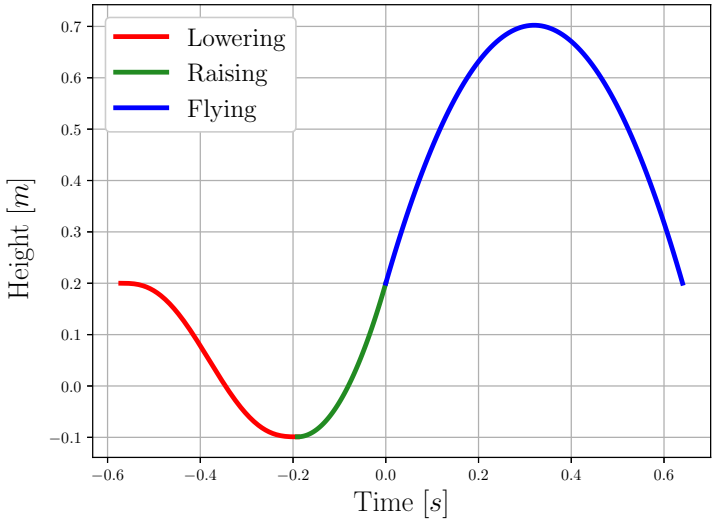
There are no particular force or acceleration constraints for the lowering phase ($t \in [-3\,t_{max}, -t_{max}]$). We chose to make it last $2t_{max}$ to slowly lower the base. Limit conditions for the polynomial interpolation are as follows:

$$\begin{array}{llllll} {}^lz^\star(-3t_{max}) & = h^\star & {}^l\dot{z}^\star(-3t_{max}) & = 0 & {}^l\ddot{z}^\star(-3t_{max}) & = 0 \\ {}^lz^\star(-t_{max}) & = z_{low} & {}^l\dot{z}^\star(-t_{max}) & = 0 & {}^l\ddot{z}^\star(-t_{max}) & = \ddot{z}_{max} \end{array} \tag{8.42}$$

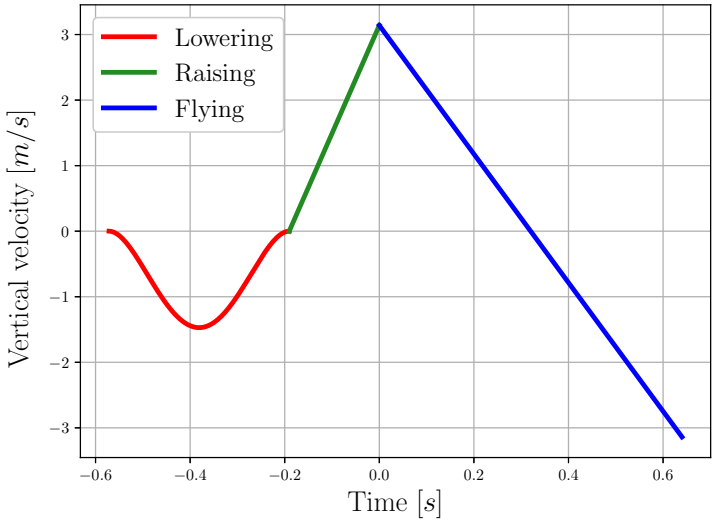
The coefficients of the polynomial corresponding to the lowering section are as follows:

$$\begin{cases} a_5 &= \frac{-3\,T_J^5\,g^5}{16384\,(h^\star - z_{low})^4} \\ a_4 &= \frac{15\,T_J^4\,g^4}{4096\,(h^\star - z_{low})^3} \\ a_3 &= \frac{-5\,T_J^3\,g^3}{256\,(h^\star - z_{low})^2} \\ a_2 &= 0 \\ a_1 &= 0 \\ a_0 &= h^\star \end{cases} \tag{8.43}$$

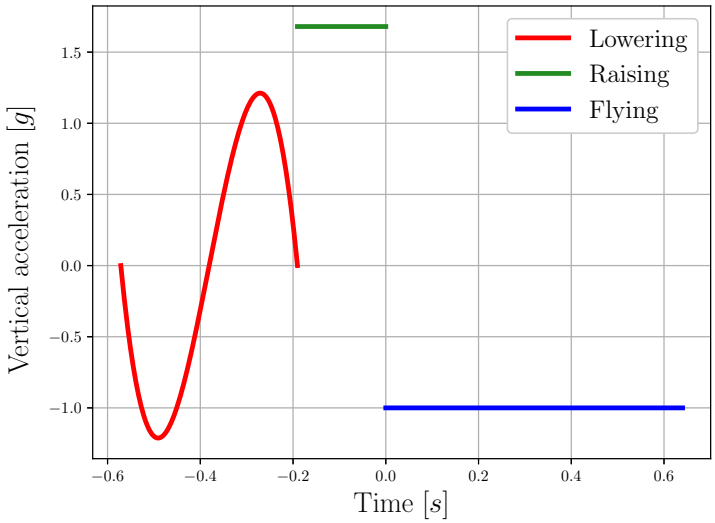
In the end, the final pre-jump position, velocity and acceleration trajectories along the vertical axis follow the profiles described in Fig. 8.2.



(a) Reference trajectory for the base height



(b) Reference trajectory for the base vertical velocity



(c) Reference trajectory for the base vertical acceleration

Figure 8.2: Reference trajectories in position, velocity and acceleration along the vertical axis to perform a 0.64s-long jump, with $\tau_{max} = 2.5$ Nm. The jumping sequence is divided into lowering, raising and flying phases.

8.2 Linear convex Quadratic Programming

The lumped-mass model presented in (8.1a) and (8.1b) can be used to perform model predictive control over a prediction horizon. Using this model, the solver can predict how the system will react to ground reaction forces and will adjust them so that the predicted state trajectory follows the reference trajectory given as input. This is illustrated in Fig. 8.3. As we said earlier, the MPC works in the local frame l, which is a projection at ground level of the horizontal frame h at the start of each optimization. The gait matrix $\mathcal{G}_c$ is used to known which feet are in contact at each step of the prediction horizon, with the footstep planner providing the contact locations r. We define the set of feet in contact as:

$$\mathcal{C} = \Big\{ i \in \{1,2,3,4\} \text{ s.t. } \mathcal{G}_c(0,i) = 1 \Big\} \tag{8.44}$$

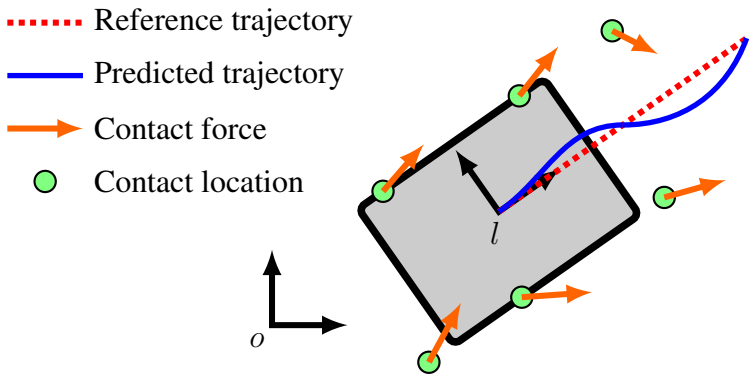


Figure 8.3: The MPC optimizes contact forces at footstep locations so that the state trajectory predicted using the lumped-mass model follows the reference one.

8.2.1 Assumptions, dynamics and constraints

Recall that the lumped-mass model (centroidal dynamics) used by the MPC can be written in world frame as follows:

$$m\, {}^l\ddot{c} = \sum_{i \in \mathcal{C}} {}^l\mathbf{f}_i - \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} \tag{8.45}$$

$$\frac{d}{dt}({}^l\mathcal{I}\,{}^l\omega) = \sum_{i \in \mathcal{C}} ({}^l r_i^\star - {}^l c) \times {}^l\mathbf{f}_i \tag{8.46}$$

with ${}^l\mathbf{f}_i$ the ground reaction forces, ${}^l r_i^\star$ the location of contact points given by the footstep planner, ${}^l\ddot{c} = {}^l\ddot{\mathbf{q}}_{lin}$ the acceleration of the center of mass, ${}^l\mathcal{I}$ the rotational inertia tensor and ${}^l w = {}^l\dot{\mathbf{q}}_{ang}$ the angular velocity of the body. As written, this model is nonlinear. We can however approximate it by a linear one with a few assumptions, the first one being that roll and pitch angles are small. It follows that:

$${}^l\omega \approx \begin{bmatrix} {}^l\dot{\phi} & {}^l\dot{\theta} & {}^l\dot{\psi} \end{bmatrix}^T \tag{8.47}$$

$${}^l\mathcal{I} \approx {}^b\mathcal{I} \tag{8.48}$$

The second assumption is that states are close to the desired trajectory, so in (8.46) the position of the center of mass ${}^{l}c$ can be replaced by the desired position for the center of mass ${}^{l}c^{\star}$. The last assumption is that pitch and roll velocities are small. In the end:

$$\frac{d}{dt}({}^{l}\mathcal{I}{}^{l}\omega) = {}^{l}\mathcal{I}{}^{l}\dot{\omega} + {}^{l}\omega \times ({}^{l}\mathcal{I}{}^{l}\omega) \approx {}^{l}\mathcal{I}{}^{l}\dot{\omega} \tag{8.49}$$

With these assumptions, (8.46) is simplified into:

$${}^{l}\mathcal{I} \begin{bmatrix} {}^{l}\dot{\phi} \\ {}^{l}\dot{\theta} \\ {}^{l}\dot{\psi} \end{bmatrix} = \sum_{i \in \mathcal{C}} ({}^{l}r_i^{\star} - {}^{l}c^{\star}) \times {}^{l}\mathbf{f}_i \tag{8.50}$$

After discretization of (8.45) and (8.50), the evolution of state variables for $k \in [0, n_{steps} - 1]$ becomes:

$$\mathcal{C}_k = \Big\{ i \in \{1,2,3,4\} \text{ s.t. } \mathcal{G}_c(k,i) = 1 \Big\} \tag{8.51}$$

$$\begin{bmatrix} {}^{l}x_{k+1} \\ {}^{l}y_{k+1} \\ {}^{l}z_{k+1} \end{bmatrix} = \begin{bmatrix} {}^{l}x_k \\ {}^{l}y_k \\ {}^{l}z_k \end{bmatrix} + \Delta t \begin{bmatrix} {}^{l}\dot{x}_k \\ {}^{l}\dot{y}_k \\ {}^{l}\dot{z}_k \end{bmatrix} \tag{8.52}$$

$$\begin{bmatrix} {}^{l}\phi_{k+1} \\ {}^{l}\theta_{k+1} \\ {}^{l}\psi_{k+1} \end{bmatrix} = \begin{bmatrix} {}^{l}\phi_k \\ {}^{l}\theta_k \\ {}^{l}\psi_k \end{bmatrix} + \Delta t \begin{bmatrix} {}^{l}\dot{\phi} \\ {}^{l}\dot{\theta} \\ {}^{l}\dot{\psi} \end{bmatrix} \tag{8.53}$$

$$\begin{bmatrix} {}^{l}\dot{x}_{k+1} \\ {}^{l}\dot{y}_{k+1} \\ {}^{l}\dot{z}_{k+1} \end{bmatrix} = \begin{bmatrix} {}^{l}\dot{x}_k \\ {}^{l}\dot{y}_k \\ {}^{l}\dot{z}_k \end{bmatrix} + \Delta t \left(\sum_{i \in \mathcal{C}_k} \frac{{}^{l}\mathbf{f}_{i,k}}{m} - \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} \right) \tag{8.54}$$

$$\begin{bmatrix} {}^{l}\dot{\phi}_{k+1} \\ {}^{l}\dot{\theta}_{k+1} \\ {}^{l}\dot{\psi}_{k+1} \end{bmatrix} = \begin{bmatrix} {}^{l}\dot{\phi}_k \\ {}^{l}\dot{\theta}_k \\ {}^{l}\dot{\psi}_k \end{bmatrix} + \Delta t \left({}^{l}\mathcal{I}^{-1} \sum_{i \in \mathcal{C}_k} [{}^{l}r_{i,k}^{\star} - {}^{l}c_k^{\star}] \times {}^{l}\mathbf{f}_{i,k} \right) \tag{8.55}$$

An alternative integration scheme can directly include the effect of ground reaction forces in (8.52) and (8.53) so that position and orientation are directly modified instead of waiting 1 time step for the velocities to be changed.

$$\begin{bmatrix} {}^{l}x_{k+1} \\ {}^{l}y_{k+1} \\ {}^{l}z_{k+1} \end{bmatrix} = \begin{bmatrix} {}^{l}x_k \\ {}^{l}y_k \\ {}^{l}z_k \end{bmatrix} + \Delta t \begin{bmatrix} {}^{l}\dot{x}_k \\ {}^{l}\dot{y}_k \\ {}^{l}\dot{z}_k \end{bmatrix} + \frac{\Delta t^2}{2} \left(\sum_{i \in \mathcal{C}_k} \frac{{}^{l}\mathbf{f}_{i,k}}{m} - \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} \right) \tag{8.56}$$

$$\begin{bmatrix} {}^{l}\phi_{k+1} \\ {}^{l}\theta_{k+1} \\ {}^{l}\psi_{k+1} \end{bmatrix} = \begin{bmatrix} {}^{l}\phi_k \\ {}^{l}\theta_k \\ {}^{l}\psi_k \end{bmatrix} + \Delta t \begin{bmatrix} {}^{l}\dot{\phi} \\ {}^{l}\dot{\theta} \\ {}^{l}\dot{\psi} \end{bmatrix} + \frac{\Delta t^2}{2} \left({}^{l}\mathcal{I}^{-1} \sum_{i \in \mathcal{C}_k} [{}^{l}r_{i,k}^{\star} - {}^{l}c_k^{\star}] \times {}^{l}\mathbf{f}_{i,k} \right) \tag{8.57}$$

For simplicity, we will use the first integration scheme in the following.

In terms of constraints, friction cone conditions to avoid slipping are linearized to the first order, as illustrated in Fig. 8.4:

$$\forall k \in [0, n_{steps} - 1], \forall i \in \mathcal{C}_k, \ |\mathbf{f}_{i,k}^x| \leq \mu \, \mathbf{f}_{i,k}^z \text{ and } |\mathbf{f}_{i,k}^y| \leq \mu \, \mathbf{f}_{i,k}^z \tag{8.58}$$

with μ the Coulomb friction coefficient. An upper limit has to be set for contact forces to respect hardware limits (maximum torque of actuators). This limit is only applied to

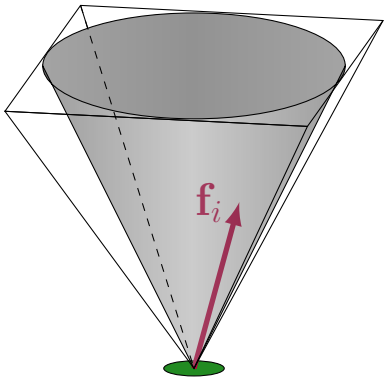


Figure 8.4: The friction cone is linearized to the first order along the x and y axes to enforce it in matrix form as a constraint in a linear optimization problem.

the z component since it will also limit the force along x and y due to the friction cone contraints.

$$\forall k \in [0, n_{steps} - 1], \forall i \in \mathcal{C}_k,\ \mathbf{f}_{i,k}^z \leq f_{max} \tag{8.59}$$

The quadruped cannot pull on the ground, it can only push, so the normal component of the contact forces has to be positive:

$$\forall k \in [0, n_{steps} - 1], \forall i \in \mathcal{C}_k, \mathbf{f}_{i,k}^z \geq 0\ N \tag{8.60}$$

To be sure that there is no slipping, we could impose a minimal non-zero vertical component of the contact forces because if it is close to 0 N the friction cone is small so on the real robot slipping could happen.

8.2.2 Quadratic Programming formulation

Applying our assumptions to the lumped-mass model, the system can be expressed as a linear dynamics with equality and inequality constraints. The evolution of the state vector of the robot over time can be described as follows:

$$\forall k \in [0, n_{steps} - 1], \mathbf{X}_{k+1} = \mathbf{A}_k\, \mathbf{X}_k + \mathbf{B}_k\, \mathbf{F}_k + \mathbf{g} \tag{8.61}$$

with $\mathbf{X}_k$ defined as in (8.11) by stacking position, orientation, linear velocity and angular velocity and $\mathbf{g} = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \ -9.81\ \Delta t\ 0\ 0\ 0]^T$ as the gravity vector. The contact forces vector $\mathbf{F}_k$ always includes the forces applied at the four feet even if some of them are not touching the ground. In that case we will set the problem in such a way that forces for such feet are not considered in the solving process since they are nonexistent. With $\mathbf{f}_{i,k}^x$, $\mathbf{f}_{i,k}^y$ and $\mathbf{f}_{i,k}^z$ the components along the x, y and z axes of the local frame l for the i-th foothold at time step k, the force vector expressed:

$$\forall k \in [0, n_{steps} - 1],\ \mathbf{F}_k = \begin{bmatrix} \mathbf{f}_{1,k} \\ \mathbf{f}_{2,k} \\ \mathbf{f}_{3,k} \\ \mathbf{f}_{4,k} \end{bmatrix} \tag{8.62}$$

$$\forall i \in \{1, 2, 3, 4\},\ \mathbf{f}_{i,k} = \begin{bmatrix} \mathbf{f}_{i,k}^x \\ \mathbf{f}_{i,k}^y \\ \mathbf{f}_{i,k}^z \end{bmatrix} \tag{8.63}$$

A quadratic programming solver such as OSQP solves convex quadratic programs of the form:

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\mathbb{X}^T\mathbf{P}\mathbb{X} + \mathbf{Q}^T\mathbb{X} \\ \text{subject to} & \mathbf{l} \leq \mathbb{A}\mathbb{X} \leq \mathbf{u} \end{array} \tag{8.64}$$

where $\mathbb{X} \in \mathbb{R}^n$ is the optimization variable, the objective function is defined by a positive semidefinite matrix $\mathbf{P} \in \mathbf{S}_+^n$ and vector $\mathbf{Q} \in \mathbb{R}^n$, the linear constraints are defined by matrix $\mathbb{A} \in \mathbb{R}^{m \times n}$ and vectors $\mathbf{l}$ and $\mathbf{u}$ so that $\mathbf{l}_i \in \mathbb{R} \cup \{-\infty\}$ and $\mathbf{u}_i \in \mathbb{R} \cup \{+\infty\}$, $\forall i \in \{1, \ldots, m\}$.

The goal of the MPC is to find contacts forces $\mathbf{F}_k$ that should be applied to have the state vector $\mathbf{X}_k$ of the robot as close as possible to $\mathbf{X}_k^\star$. The solver will output at the end of the optimization process the vector $\mathbb{X}$ that minimizes the cost function locally (globally in the best case). Using a convex quadratic program, we have the guarantee to convergence at the global minimum. The QP problem can be written in a simple way by putting both $\mathbf{F}_k$ (the output of the MPC) and $\mathcal{X}_k = \mathbf{X}_k - \mathbf{X}_k^\star$ (quantity that should be minimized) in the optimization vector. For brevity, let's consider a case with only 3 time steps in the prediction horizon:

$$\mathbb{X} = \begin{bmatrix} \mathcal{X}_1 \\ \mathcal{X}_2 \\ \mathcal{X}_3 \\ \mathbf{F}_0 \\ \mathbf{F}_1 \\ \mathbf{F}_2 \end{bmatrix} \tag{8.65}$$

In order to construct the matrix $\mathbb{A}$ that appears in (8.64), let us express the state evolution and both equality and inequality constraints as $\mathbb{M}\mathbb{X} = \mathbb{N}$ and $\mathbb{K}_{low} \leq \mathbb{L}\mathbb{X} \leq \mathbb{K}_{up}$, where matrix $\mathbb{M}$ is defined as:

$$\mathbb{M} = \begin{bmatrix} -\mathbf{I}_{12} & 0_{12} & 0_{12} & \mathbf{B}_0 & 0_{12} & 0_{12} \\ \mathbf{A}_1 & -\mathbf{I}_{12} & 0_{12} & 0_{12} & \mathbf{B}_1 & 0_{12} \\ 0_{12} & \mathbf{A}_2 & -\mathbf{I}_{12} & 0_{12} & 0_{12} & \mathbf{B}_2 \\ 0_{12} & 0_{12} & 0_{12} & \mathbf{E}_0 & 0_{12} & 0_{12} \\ 0_{12} & 0_{12} & 0_{12} & 0_{12} & \mathbf{E}_1 & 0_{12} \\ 0_{12} & 0_{12} & 0_{12} & 0_{12} & 0_{12} & \mathbf{E}_2 \end{bmatrix} \tag{8.66}$$

and matrix $\mathbb{N}$ is defined as:

$$\mathbb{N} = \begin{bmatrix} -\mathbf{g} \\ -\mathbf{g} \\ -\mathbf{g} \\ 0_{12\times 1} \\ 0_{12\times 1} \\ 0_{12\times 1} \end{bmatrix} + \begin{bmatrix} -\mathbf{A}_0\mathbf{X}_0 \\ 0_{12\times 1} \\ 0_{12\times 1} \\ 0_{12\times 1} \\ 0_{12\times 1} \\ 0_{12\times 1} \end{bmatrix} + \begin{bmatrix} \mathbf{I}_{12} & 0_{12} & 0_{12} & 0_{12} & 0_{12} & 0_{12} \\ -\mathbf{A}_1 & \mathbf{I}_{12} & 0_{12} & 0_{12} & 0_{12} & 0_{12} \\ 0_{12} & -\mathbf{A}_2 & \mathbf{I}_{12} & 0_{12} & 0_{12} & 0_{12} \end{bmatrix} \begin{bmatrix} \mathbf{X}_1^\star \\ \mathbf{X}_2^\star \\ \mathbf{X}_3^\star \\ 0_{12\times 1} \\ 0_{12\times 1} \\ 0_{12\times 1} \end{bmatrix} \tag{8.67}$$

with matrix $\mathbf{A}_k$ at time step k defined as follows:

$$\mathbf{A}_k = \begin{bmatrix} \mathbf{I}_3 & 0_3 & \Delta t\, \mathbf{I}_3 & 0_3 \\ 0_3 & \mathbf{I}_3 & 0_3 & \Delta t\, \mathbf{I}_3 \\ 0_3 & 0_3 & \mathbf{I}_3 & 0_3 \\ 0_3 & 0_3 & 0_3 & \mathbf{I}_3 \end{bmatrix} \tag{8.68}$$

Making the assumption that roll and pitch angles are small, the inertia matrix of the robot in local frame l at time step k is:

$$^{l}\mathcal{I}_k = \mathcal{R}_z(\Delta t\, k\, {}^{l}\dot{\psi}^{\star})\, {}^{b}\mathcal{I} \tag{8.69}$$

Matrix $\mathbf{B}_k$ at time step k is defined as follows:

$$\mathbf{B}_k = \Delta t \begin{bmatrix} 0_3 & 0_3 & 0_3 & 0_3 \\ 0_3 & 0_3 & 0_3 & 0_3 \\ \mathbf{I}_3/m & \mathbf{I}_3/m & \mathbf{I}_3/m & \mathbf{I}_3/m \\ {}^{l}\mathcal{I}_k^{-1}\,[{}^{l}r^{\star}_{1,k} - {}^{l}c^{\star}_{k}]_{\times} & {}^{l}\mathcal{I}_k^{-1}\,[{}^{l}r^{\star}_{2,k} - {}^{l}c^{\star}_{k}]_{\times} & {}^{l}\mathcal{I}_k^{-1}\,[{}^{l}r^{\star}_{3,k} - {}^{l}c^{\star}_{k}]_{\times} & {}^{l}\mathcal{I}_k^{-1}\,[{}^{l}r^{\star}_{4,k} - {}^{l}c^{\star}_{k}]_{\times} \end{bmatrix} \tag{8.70}$$

with $({}^{l}r_{i,k} - {}^{l}c^{\star}_{k})$ the vector in local frame going from the desired position of the center of mass at time step k to the position of the i-th foothold, and $[r_{k,i} - {}^{l}c^{\star}_{k}]_{\times}$ the associated skew-symmetric matrix. Then, matrix $\mathbf{E}_k$ for time step k is defined as follows:

$$\mathbf{E}_k = \begin{bmatrix} \mathbf{e}_{1,k} & 0_3 & 0_3 & 0_3 \\ 0_3 & \mathbf{e}_{2,k} & 0_3 & 0_3 \\ 0_3 & 0_3 & \mathbf{e}_{3,k} & 0_3 \\ 0_3 & 0_3 & 0_3 & \mathbf{e}_{4,k} \end{bmatrix} \tag{8.71}$$

with $\mathbf{e}_{i,k} = 0_3$ if the i-th foot is touching the ground during time step k, $\mathbf{e}_{i,k} = \mathbf{I}_3$ otherwise. In fact, if $\mathbf{e}_{i,k} = \mathbf{I}_3$ then with $\mathbb{M}\,\mathbb{X} = \mathbb{N}$ we are setting the constraint that $\mathbf{f}_{i,k} = [0\ \ 0\ \ 0]^T$ (no reaction force since the foot is not touching the ground).

Now that $\mathbb{M}$ and $\mathbb{N}$ have been defined, let's define $\mathbb{L}$ and $\mathbb{K}$ in $\mathbb{L}\,\mathbb{X} \leq \mathbb{K}$ to express the friction cone inequality constraint on the force. First, matrix $\mathbb{L}$ is defined as:

$$\mathbb{L} = \begin{bmatrix} 0_{20\times12} & 0_{20\times12} & 0_{20\times12} & \mathbf{F}_{\mu} & 0_{20\times12} & 0_{20\times12} \\ 0_{20\times12} & 0_{20\times12} & 0_{20\times12} & 0_{20\times12} & \mathbf{F}_{\mu} & 0_{20\times12} \\ 0_{20\times12} & 0_{20\times12} & 0_{20\times12} & 0_{20\times12} & 0_{20\times12} & \mathbf{F}_{\mu} \end{bmatrix} \tag{8.72}$$

with:

$$\mathbf{F}_{\mu} = \begin{bmatrix} \mathbf{G} & 0_{5\times3} & 0_{5\times3} & 0_{5\times3} \\ 0_{5\times3} & \mathbf{G} & 0_{5\times3} & 0_{5\times3} \\ 0_{5\times3} & 0_{5\times3} & \mathbf{G} & 0_{5\times3} \\ 0_{5\times3} & 0_{5\times3} & 0_{5\times3} & \mathbf{G} \end{bmatrix} \text{ and } \mathbf{G} = \begin{bmatrix} 1 & 0 & -\mu \\ -1 & 0 & -\mu \\ 0 & 1 & -\mu \\ 0 & -1 & -\mu \\ 0 & 0 & -1 \end{bmatrix} \tag{8.73}$$

To enforce the maximum normal reaction force f_{max} of (8.59), we have:

$$\mathbb{K}_{low} = \begin{bmatrix} \mathbf{K_F} \\ \mathbf{K_F} \\ \mathbf{K_F} \end{bmatrix} \text{ with } \mathbf{K_F} = \begin{bmatrix} \mathbf{K_f} \\ \mathbf{K_f} \\ \mathbf{K_f} \\ \mathbf{K_f} \end{bmatrix} \text{ and } \mathbf{K_f} = \begin{bmatrix} -\infty \\ -\infty \\ -\infty \\ -\infty \\ -f_{max} \end{bmatrix} \tag{8.74}$$

The matrix $\mathbb{K}_{up}$ is defined as $0_{60\times1}$. Now that all matrices have been defined, we can formulate the QP problem as in (8.64).

$$\mathbf{l} = \begin{bmatrix} \mathbb{N} \\ \mathbb{K}_{low} \end{bmatrix} \quad \leq \quad \mathbb{A} = \begin{bmatrix} \mathbb{M} \\ \mathbb{L} \end{bmatrix} \quad \mathbb{X} \quad \leq \quad \mathbf{u} = \begin{bmatrix} \mathbb{N} \\ \mathbb{K}_{up} \end{bmatrix} \tag{8.75}$$

8.2.3 Cost function

The QP solver tries to find a vector $\mathbb{X}$ that minimizes the cost function $\frac{1}{2}\mathbb{X}^T\mathbf{P}\mathbb{X}+\mathbf{Q}^T\mathbb{X}$ under constraints $\mathbf{l} \leq \mathbb{A}\mathbb{X} \leq \mathbf{u}$. Matrices $\mathbf{P}$ and $\mathbf{Q}$ define the shape of the cost function. The goal of the MPC is to find which contact forces should be applied at contact points so that the predicted trajectory of the center of mass is as close as possible to the reference trajectory. With previous notation, it amounts to minimize $|\mathbf{X} - \mathbf{X}^\star|$. This quantity is not directly available as a matrix product of the form $\frac{1}{2}\mathbb{X}^T\ \mathbf{P}\ \mathbb{X} + \mathbf{Q}^T\mathbb{X}$. However, we can minimize $(\mathbf{X} - \mathbf{X}^\star)^2$ instead by using $\mathbb{X}^T\ \mathbf{P}\ \mathbb{X}$. Remember that the placement of $\mathbf{X} - \mathbf{X}^\star$ in $\mathbb{X}$ is:

$$\mathbb{X} = \begin{bmatrix} \mathcal{X}_1 \\ \vdots \\ \mathcal{X}_{n_{steps}} \\ \mathbf{F}_0 \\ \vdots \\ \mathbf{F}_{n_{steps}-1} \end{bmatrix} = \begin{bmatrix} \mathbf{X}_1 - \mathbf{X}_1^\star \\ \vdots \\ \mathbf{X}_{n_{steps}} - \mathbf{X}_{n_{steps}}^\star \\ \mathbf{F}_0 \\ \vdots \\ \mathbf{F}_{n_{steps}-1} \end{bmatrix} \tag{8.76}$$

so the upper left block of $\mathbf{P}$ can be diagonal:

$$\mathbf{P} = \begin{bmatrix} \mathbf{W}_{\mathcal{X},1} & & 0 & * \\ & \ddots & & * \\ 0 & & \mathbf{W}_{\mathcal{X},n_{steps}} & * \\ * & * & * & * \end{bmatrix} \tag{8.77}$$

with $\forall k \in [1, n_{steps}],\ \mathbf{W}_{\mathcal{X},k}$ being a 12 by 12 diagonal matrice with coefficients ≥ 0 so that the deviation from the reference trajectory increases the value of the cost function and thus pushes the solver into minimizing the error $(\mathbf{X}_k - \mathbf{X}_k^\star)^2$. For safety reason, for energy consumption and to limit actuator heating, it is better to keep the contact forces low if possible. That is why a small regularization term is added to slightly penalize the norm of contact forces. Since the square root function is not directly available in the matrix product of the cost function, we regularize the square of the norm instead ($\|\mathbf{f}_{k,i}\|^2$).

$$\mathbf{P} = \begin{bmatrix} \mathbf{W}_{\mathcal{X},1} & & 0 & * & * & * \\ & \ddots & & * & * & * \\ 0 & & \mathbf{W}_{\mathcal{X},n_{steps}} & * & * & * \\ * & * & * & \mathbf{W}_{\mathbf{F},0} & & 0 \\ * & * & * & & \ddots & \\ * & * & * & 0 & & \mathbf{W}_{\mathbf{F},n_{steps}-1} \end{bmatrix} \tag{8.78}$$

With $\forall k \in [0, n_{steps} - 1],\ \mathbf{W}_{\mathbf{F},k}$ 12 by 12 diagonal matrices with coefficients ≥ 0. There is no cross-coupling between $\mathcal{X}$ and force components so both the upper-right and lower-left corners of $\mathbf{P}$ are zeros.

As the optimization is supposed to be uniform over the whole prediction horizon, all $\mathbf{W}_{\mathcal{X},k}$ are equal. It is the same for all $\mathbf{W}_{\mathbf{F},k}$. Coefficient at position $(i, i),\ \forall i \in [1, \ldots, 12]$, in $\mathbf{W}_{\mathcal{X},k}$ weights the deviation of the i-th component of the state vector from the reference trajectory. Remember that components of the state vector are in this order: $[{}^l x\ {}^l y\ {}^l z\ {}^l\phi\ {}^l\theta\ {}^l\psi\ {}^l\dot{x}\ {}^l\dot{y}\ {}^l\dot{z}\ {}^l\dot{\phi}\ {}^l\dot{\theta}\ {}^l\dot{\psi}]$. Coefficient at position $(i, i),\ \forall i \in [1, 12]$, in $\mathbf{W}_{\mathbf{F},k}$ weights the i-th component of the force vector for regularization purpose. Remember that components of the force vector are in this order: $[{}^l\mathbf{f}_0^x\ {}^l\mathbf{f}_0^y\ {}^l\mathbf{f}_0^z\ {}^l\mathbf{f}_1^x\ {}^l\mathbf{f}_1^y\ {}^l\mathbf{f}_1^z\ {}^l\mathbf{f}_2^x\ {}^l\mathbf{f}_2^y\ {}^l\mathbf{f}_2^z\ {}^l\mathbf{f}_3^x\ {}^l\mathbf{f}_3^y\ {}^l\mathbf{f}_3^z]$.

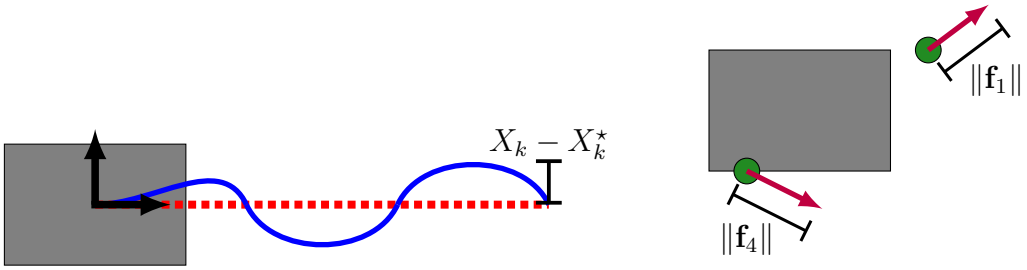
To properly regularize the norm of contact forces $\|\mathbf{f}_{i,k}\|^2 = (\mathbf{f}_{i,k}^x)^2 + (\mathbf{f}_{i,k}^y)^2 + (\mathbf{f}_{i,k}^z)^2$, the coefficients for the x, y and z components have to be equal. If no leg is privileged (to mimic a wounded leg we could try to apply less force with it) then all coefficients on the diagonal of $\mathbf{W}_{\mathbf{F},k}$ are equal and $\forall k \in [0, n_{steps} - 1],\ \mathbf{W}_{\mathcal{X},k} = w_f \mathbf{I}_{12}$ with $w_f \in \mathbb{R}^+$

The matrix $\mathbf{Q}$ in $\mathbf{Q}^T \mathbb{X}$ only contains zeroes since there is no reason to push $\mathbf{X}_k - \mathbf{X}_k^\star$ or $\mathbf{F}_k$ into being as negative/positive as possible. For instance if a coefficient of $\mathbf{Q}$ was positive then the solver would try to have the associated variable as negative as possible to have a high negative product between the coefficient and the variable since that minimizes the cost. As a consequence, $\mathbf{Q} = 0_{24\, n_{steps} \times 1}$.

The cost function during the optimization process is then:

$$cost(\mathbb{X}) = \sum_{k=1}^{n_{steps}} \left(\sum_{i=0}^{11} \left[w_{\mathcal{X}}^i (\mathbf{X}_k^i - \mathbf{X}_k^{i,\star})^2 \right] + w_f \sum_{i=0}^{3} \left[(\mathbf{f}_{i,k}^x)^2 + (\mathbf{f}_{i,k}^y)^2 + (\mathbf{f}_{i,k}^z)^2 \right] \right) \tag{8.79}$$

These costs are illustrated in Fig. 8.5.



(a) Cost on state error

(b) Cost on contact forces

Figure 8.5: One cost (a) penalizes the error between the reference and predicted state trajectories while the other one (b) acts as a regularization to minimize contact forces when possible.

In the end, the desired reaction forces that need to be applied at contact locations over the prediction horizon are stored in $\mathbf{f}_0$. These forces are expressed in the local frame l yet they can easily be brought back to the world frame since ${}^o\mathbf{R}_l$ is known. The same applies for the next desired position of the robot $\mathbf{X}_1$ that is stored in $\mathcal{X}_1$ and can be retrieved by adding $\mathbf{X}_1^\star$ to $\mathcal{X}_1$. It is expressed in the local frame as well.

8.3 Differential Dynamic Programming

Differential dynamic programming (DDP) can solve nonlinear parametric optimal control problems (OCP) by using locally-quadratic models of the dynamics and cost functions. DDP can be exploited to solve the same linear problem than Section 8.2 by applying the assumptions that allowed us to linearize the system, but also in a more general way to take into account nonlinearities by lifting those assumptions.

8.3.1 Optimal control problem

The optimal control problem can be written as follows:

$$\begin{aligned} \min_{\{\mathbf{X}\},\{\mathbf{F}\}} & \sum_{t=0}^{T} \ell_t(\mathbf{X}_t, \mathbf{F}_t | r_t^\star) + \ell_T(x_T) \\ \text{s.t.} \quad & \mathbf{X}_{t+1} = \mathcal{H}(\mathbf{X}_t, \mathbf{F}_t | r_t^\star) \\ & \mathbf{X}_t \in \mathcal{F} \\ & \mathbf{F}_t \in \mathcal{K} \end{aligned} \tag{8.80}$$

where ℓ_t and ℓ_T are respectively the running and the terminal costs. $\{\mathbf{X}\}$ and $\{\mathbf{F}\}$ are the state and contact forces decision variables, discretized at the optimization nodes indexed by t. As in Section 8.2, the state vector $\{\mathbf{X}\}$ includes the position, orientation, linear and angular velocities of the base. $\{\mathbf{X}\}$ has to remain in the feasibility manifold $\mathcal{F}$ to ensure that a valid whole-body movement that can achieve $\mathbf{X}$ exists. The control vector $\mathbf{F}$ contains the 3D forces at each contact point, constrained by the friction cone $\mathcal{K}$. $r^\star$ stores the position of footsteps at which contact forces are applied.

With the same assumptions than in Section 8.2 and the use of the footstep locations $r^\star$ given by the footstep planner, the optimal control problem ends up with a similar linear discretized centroidal dynamics:

$$\mathbf{X}_{t+1} = \mathcal{H}(\mathbf{X}_t, \mathbf{F}_t | r_t) = \mathbf{A}\,\mathbf{X}_t + \mathbf{B}(\mathbf{X}_t^\star, r_t^\star)\,\mathbf{F}_t \tag{8.81a}$$

$$\mathbf{A} = \begin{bmatrix} \mathbf{I}_6 & \Delta t \mathbf{I}_6 \\ 0_6 & \mathbf{I}_6 \end{bmatrix} \tag{8.81b}$$

$$\mathbf{X}_t = \begin{bmatrix} {}^l c_t & {}^l\Theta_t & {}^l\dot{c}_t & {}^l\dot{\Theta}_t \end{bmatrix}^T \tag{8.81c}$$

$$\mathbf{B}(\mathbf{X}_t^\star, r_t^\star) = \Delta t \begin{bmatrix} \dots & 0 & \dots \\ \dots & 0 & \dots \\ \dots & \mathbf{I}_3/m & \dots \\ \dots & {}^l\mathcal{I}^{-1}[{}^l r_{i,t}^\star - {}^l c_t^\star]_\times & \dots \end{bmatrix} \tag{8.81d}$$

$$\mathbf{F}_t = \begin{bmatrix} \mathbf{f}_{1,t} \\ \mathbf{f}_{2,t} \\ \mathbf{f}_{3,t} \\ \mathbf{f}_{4,t} \end{bmatrix} \tag{8.81e}$$

where Δt is the integration time between the nodes, m and $\mathcal{I}$ are mass and inertia of the body, $\mathbf{I}_6$ is the identity matrix of size 6, ${}^l c^\star$ is the position of the CoM in local frame and ${}^l\Theta = (\phi, \theta, \psi)$ is the orientation of the body (Euler angles). 3D forces $\mathbf{f}_i$ are applied at the contact points $r_i^\star$. The i-th column of B is disabled when the i-th foot is not in contact ($i \notin \mathcal{C}$). $[\dots]_\times$ is used to denote a 3x3 skew-symmetric matrix in order to express the cross product as matrix multiplication.

For the DDP solver, each time step of the prediction horizon is associated with a node which contains the state evolution model. The organization of these nodes is highlighted in Fig. 8.6.

To solve the OCP formulation described above, we use the library Crocoddyl [Mas+20] which provides a multiple shooting DDP solver. As written in (8.80), there is no inequality constraint in the formulation because the solver does not support them. Inequalities

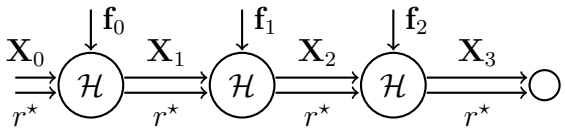


Figure 8.6: Factor graph displaying the correlations between the decision variables of the linear MPC. Example with a prediction horizon of 3 time steps.

such as the friction cone constraint will instead be enforced through the cost function: if respected, their cost will be equal to 0, otherwise it will raise up very quickly. This cannot guarantee that inequalities constraints will be respected, but it can be expected that the solver will not violate them *too much* because of the high cost that appears in that case.

8.3.2 Running and terminal costs

Four running costs are included in the cost function for all time steps:

- quadratic cost $\ell_{t,\mathcal{X}}$ on the error between predicted and desired state vectors to track the desired state trajectory

$$\ell_{t,\mathcal{X}} = \|\mathbf{X}_t - \mathbf{X}_t^\star\|^2 \tag{8.82}$$

- quadratic cost $\ell_{t,\mathbf{F}}$ on the norm of ground reaction forces to be minimized if possible (regularization) as in (8.79). This cost only applies to the feet that are in stance phase at time t ($i \in \mathcal{C}_t$). We can also regulate the vertical force around a constant so that the weight of the robot tends to be supported evenly by all legs in stance phase:

$$\ell_{t,f} = \sum_{i \in \mathcal{C}_t} \|\mathbf{f}_{i,t} - \mathbf{f}_{i,t}^\star\|^2 = \sum_{i \in \mathcal{C}_t} (\mathbf{f}_{i,t}^x)^2 + (\mathbf{f}_{i,t}^y)^2 + \left(\mathbf{f}_{i,t}^z - \frac{mg}{n(\mathcal{C}_t)}\right)^2 \tag{8.83}$$

- barrier cost $\ell_{t,\mathcal{K}}$ to avoid slipping by enforcing friction cone constraints. It only applies to the feet that are in stance phase at time t ($i \in \mathcal{C}_t$).

$$\ell_{t,\mathcal{K}} = \sum_{i \in \mathcal{C}_t} \| \begin{bmatrix} \mathbf{f}_{i,t}^x - \mu \mathbf{f}_{i,t}^z \\ -\mathbf{f}_{i,t}^x - \mu \mathbf{f}_{i,t}^z \\ \mathbf{f}_{i,t}^y - \mu \mathbf{f}_{i,t}^z \\ -\mathbf{f}_{i,t}^y - \mu \mathbf{f}_{i,t}^z \\ -\mathbf{f}_{i,t}^z \\ \mathbf{f}_{i,t}^z - f_{max}^z \end{bmatrix}^+ \|^2 \tag{8.84}$$

- barrier cost $\ell_{t,kin}$ to enforce kinematic limits on the distance between shoulders and their associated foot. That way, contact forces do not lead to an unfeasible motion for the whole-body control ($\forall t, \mathbf{X}_t \in \mathcal{F}$). It only applies to the feet that are in stance phase at time t ($i \in \mathcal{C}_t$) and is illustrated in Fig. 8.7

$$\ell_{t,kin} = \sum_{i \in \mathcal{C}_t} \| \left(\|r_{sh,i,t} - r_{i,t}\|^2 - d_{lim}^2 \right)^+ \|^2 \tag{8.85}$$

with $\{\cdot\}^+ = max(\{\cdot\}, 0)$, μ the Coulomb friction coefficient, $r_{sh,i,t}$ the position of the i-th shoulder at time t, d_{lim} a limit distance (80% of the leg limit). Since constraints are enforced through a quadratic penalization using $\{\cdot\}^+$, there is no guarantee that they will

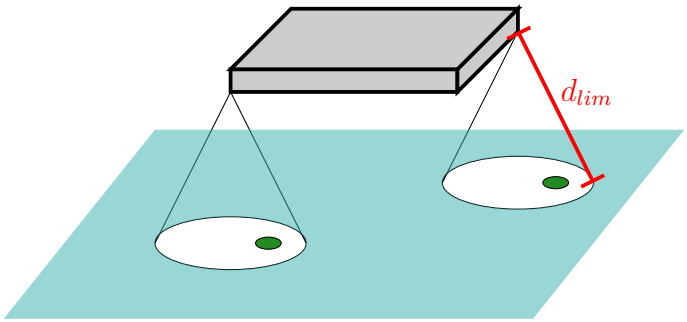


Figure 8.7: The barrier cost is enabled as soon as the footstep locations get too far from their associated shoulder to avoid breaking kinematics limit and produce a movement that the whole-body control will not be able to follow.

be respected. In practice, with a small margin for μ, this approximation works well and no slipping occurs. Just like (7.2), positions of shoulders in the base frame are as follows:

$$r_{sh,0} = \begin{bmatrix} 0.195 & 0.195 & -0.195 & -0.195 \\ 0.150 & -0.150 & 0.150 & -0.150 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{8.86}$$

Then, with the assumption of small angles and $\mathbf{q}_{lin,t}$ the position of the center of the body at time t, we get:

$$\forall i \in \{1,2,3,4\},\ r_{sh,i,t} = \mathbf{q}_{lin,t} + \begin{bmatrix} 1 & -\psi_t & 0 \\ -\psi_t & 1 & 0 \\ -\phi_t & \theta_t & 0 \end{bmatrix} r_{sh,0,i} \tag{8.87}$$

In the end, the weighted cost function is:

$$\ell_t(\mathbf{X}_t, \mathbf{F}_t | r_t) = w_{\mathcal{X}}^2 \ell_{t,\mathcal{X}}(\mathbf{X}_t) + w_{\mathbf{f}}^2 \ell_{t,\mathbf{f}}(\mathbf{F}_t) + w_{\mathcal{K}} \ell_{t,\mathcal{K}}(\mathbf{F}_t) + w_{kin} \ell_{t,kin}(\mathbf{X}_t | r_t) \tag{8.88}$$

with $w_{\mathcal{X}}$, $w_{\mathbf{f}}$, $w_{\mathcal{K}}$ and w_{kin} weight vectors for the various components.

8.3.3 Cost derivatives and Hessian matrices

The derivatives of the cost function and the Hessians can then be obtained by derivating (8.88) with respect to the state $\mathbf{X}$ and control $\mathbf{F}$. They are used by the DDP solver to get locally-quadratic models of the dynamics. First, the cost derivative with respect to $\mathbf{X}$ is:

$$L_{\mathbf{X}} = \frac{\partial \ell_t}{\partial \mathbf{X}} = w_{\mathcal{X}}^2 (\mathbf{X}_t - \mathbf{X}_t^\star)_{12\times 1} \tag{8.89}$$

For each $i \in \mathcal{C}_t$ such that $(\|r_{sh,i,t} - r_{i,t}\|^2 - d_{lim}^2)^+ > 0$, terms are added to $L_{\mathbf{X}}$:

$$L_{\mathbf{X}} \mathrel{+}= w_{kin} \begin{bmatrix} r_{sh,i,t}^x \\ r_{sh,i,t}^y \\ r_{sh,i,t}^z \\ r_{sh,i,0}^y\, r_{sh,i,t}^z \\ -r_{sh,i,0}^x\, r_{sh,i,t}^z \\ \begin{bmatrix} -\sin(\psi_t)\, r_{sh,i,t}^x & -\cos(\psi_t)\, r_{sh,i,t}^x \\ \cos(\psi_t)\, r_{sh,i,t}^y & -\sin(\psi_t)\, r_{sh,i,t}^y \end{bmatrix} \begin{bmatrix} r_{sh,i,0}^x \\ r_{sh,i,0}^y \end{bmatrix} \\ 0_{6\times 1} \end{bmatrix}_{12\times 1} \tag{8.90}$$

with the operator $+=$ used to denote that the term on the right is added to $L_{\mathbf{X}}$. Then, the cost derivative with respect to $\mathbf{F}$ is:

$$\forall i \in \mathcal{C}, L_{\mathbf{F},i} = \frac{\partial \ell_{i,t}}{\partial \mathbf{F}} = w_{\mathcal{K}} \begin{bmatrix} (\mathbf{f}^x_{i,t} - \mu \mathbf{f}^z_{i,t})^+ - (-\mathbf{f}^x_{i,t} - \mu \mathbf{f}^z_{i,t})^+ \\ (\mathbf{f}^y_{i,t} - \mu \mathbf{f}^z_{i,t})^+ - (-\mathbf{f}^y_{i,t} - \mu \mathbf{f}^z_{i,t})^+ \\ -\mu \Big((\mathbf{f}^x_{i,t} - \mu \mathbf{f}^z_{i,t})^+ + (-\mathbf{f}^x_{i,t} - \mu \mathbf{f}^z_{i,t})^+ + (\mathbf{f}^y_{i,t} - \mu \mathbf{f}^z_{i,t})^+ \\ +(-\mathbf{f}^y_{i,t} - \mu \mathbf{f}^z_{i,t})^+ + (-\mathbf{f}^z_{i,t})^+ + (\mathbf{f}^z_{i,t} - \mathbf{f}_{z,max})^+ \Big) \end{bmatrix}_{3\times 1} + w_{\mathbf{f}}^2 (\mathbf{f}_{i,t} - \mathbf{f}^\star_{i,t}) \tag{8.91}$$

The Hessian matrix $L_{\mathbf{XX}}$ is:

$$L_{\mathbf{XX}} = \frac{\partial^2 \ell_t}{\partial \mathbf{X}^2} = diag(w_{\mathcal{X}}^2)_{12\times 12} \tag{8.92}$$

For each $i \in \mathcal{C}_t$ such that $(\|r_{sh,i,t} - r_{i,t}\|^2 - d^2_{lim})^+ > 0$, terms are added to $L_{\mathbf{XX}}$:

$$L_{\mathbf{XX}} \mathrel{+}= w_{kin} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \gamma_1 & 0_{1\times 6} \\ 0 & 1 & 0 & 0 & 0 & \gamma_2 & 0_{1\times 6} \\ 0 & 0 & 1 & r^y_{sh,i,0} & -r^x_{sh,i,0} & 0 & 0_{1\times 6} \\ 0 & 0 & r^y_{sh,i,0} & r^{x,2}_{sh,i,0} + r^{y,2}_{sh,i,0} & -r^x_{sh,i,0}\, r^y_{sh,i,0} & 0 & 0_{1\times 6} \\ 0 & 0 & -r^x_{sh,i,0} & -r^x_{sh,i,0}\, r^y_{sh,i,0} & 0 & 0 & 0_{1\times 6} \\ \gamma_1 & \gamma_2 & 0 & 0 & 0 & r^{x,2}_{sh,i,0} + r^{y,2}_{sh,i,0} & 0_{1\times 6} \\ 0_{6\times 1} & 0_{6\times 1} & 0_{6\times 1} & 0_{6\times 1} & 0_{6\times 1} & 0_{6\times 1} & 0_{6\times 6} \end{bmatrix}_{12\times 12} \tag{8.93}$$

with $\begin{bmatrix} \gamma_1 \\ \gamma_2 \end{bmatrix} = \begin{bmatrix} -\sin(\psi) & -\cos(\psi) \\ \cos(\psi) & -\sin(\psi) \end{bmatrix} \begin{bmatrix} r^x_{sh,i,0} \\ r^y_{sh,i,0} \end{bmatrix}$

The Hessian matrix $L_{\mathbf{FF}}$ is:

$$\forall i \in \mathcal{C}, \mathcal{F}_i = int(\begin{bmatrix} \mathbf{f}^x_{i,t} - \mu \mathbf{f}^z_{i,t} \\ -\mathbf{f}^x_{i,t} - \mu \mathbf{f}^z_{i,t} \\ \mathbf{f}^y_{i,t} - \mu \mathbf{f}^z_{i,t} \\ -\mathbf{f}^y_{i,t} - \mu \mathbf{f}^z_{i,t} \\ -\mathbf{f}^z_{i,t} \\ \mathbf{f}^z_{i,t} - f_{z,max} \end{bmatrix}^+ > 0) = \begin{bmatrix} \mathcal{F}_{i,0} \\ \mathcal{F}_{i,1} \\ \mathcal{F}_{i,2} \\ \mathcal{F}_{i,3} \\ \mathcal{F}_{i,4} \\ \mathcal{F}_{i,5} \end{bmatrix} \text{ with } int(true) = 1, int(false) = 0 \tag{8.94}$$

$$\forall i \in \mathcal{C}, L_{\mathbf{FF},i} = \frac{\partial^2 \ell_{i,t}}{\partial \mathbf{F}^2} = w_{\mathcal{K}} \begin{bmatrix} \mathcal{F}_{i,0} + \mathcal{F}_{i,1} & 0 & \mu(\mathcal{F}_{i,1} - \mathcal{F}_{i,0}) \\ 0 & \mathcal{F}_{i,2} + \mathcal{F}_{i,3} & \mu(\mathcal{F}_{i,3} - \mathcal{F}_{i,2}) \\ \mu(\mathcal{F}_{i,1} - \mathcal{F}_{i,0}) & \mu(\mathcal{F}_{i,3} - \mathcal{F}_{i,2}) & \mu^2 \sum_{j=0}^{5} \mathcal{F}_{i,j} \end{bmatrix} + w_{\mathbf{f}}^2 I_3 \tag{8.95}$$

The derivatives of the dynamics are simply $\frac{\partial \mathcal{H}}{\partial \mathbf{X}} = \mathbf{A}$ and $\frac{\partial \mathcal{H}}{\partial \mathbf{F}} = \mathbf{B}$.

8.3.4 Implementation

The MPCs are implemented using Crocoddyl [Mas+20] which provides several solvers based on Differential Dynamic Programming (DDP). Computational efficiency is ensured with multi-threading, C++ template programming and sparsity exploitation. We limit the

number of iterations of the multiple-shooting DDP solver to 10 since, from our tests, it appears that the residual value starts to stagnate after a few iterations.

The MPC runs in a parallel process distinct from the main loop of the controller. That way, it can run at a lower frequency (50 Hz) without hampering the higher frequency of the other control blocks (1 kHz). Every 20 ms, data about the problem that should be solved (robot state, reference trajectory, footsteps positions) is sent to the parallel process, which retrieves them in a shared memory and starts solving. Then, every main loop (every 1 ms) we check in the shared memory if a new result is available. If there is one, then the main loop retrieves the contact forces that should be applied at contact points, otherwise it uses a temporary result (for instance the result of the previous solving if feet contact statuses have not changed). As a whole, we also tried to avoid allocations of memory at runtime by creating all "big" matrices at initialization and only updating their content.

8.4 Relieving linearity assumption with DDP

In Section 8.3, we kept the same assumptions than in Section 8.2 to get a similar linear problem and check that the results given by a convex QP solver and a multi-shooting DDP solver were the same. One of the key assumptions was that predicted and reference states were close to each other so that we could replace $({}^l r_i^\star - {}^l c) \times {}^l \mathbf{f}_i$ by $({}^l r_i^\star - {}^l c^\star) \times {}^l \mathbf{f}_i$ in (8.46). This removed the nonlinearity of the cross-product since both ${}^l c$ and ${}^l \mathbf{f}_i$ are optimization variable. It also simplified the problem with constant lever arms for the ground reaction forces (both ${}^l r_i^\star$ and ${}^l c^\star$ are given as inputs) instead of having them modified as the optimization goes by.

Most things remain exactly the same than in Section 8.3 for the formulation of the optimal control problem, with only (8.81d) replaced by:

$$\mathbf{B}(\mathbf{X}_t, r_t^\star) = \Delta t \begin{bmatrix} \dots & 0 & \dots \\ \dots & 0 & \dots \\ \dots & \mathbf{I}_3/m & \dots \\ \dots & {}^l\mathcal{I}^{-1}[{}^l r_{i,t}^\star - {}^l c_t]_\times & \dots \end{bmatrix} \tag{8.96}$$

The derivatives of the dynamics according to the state is no more simply $\frac{\partial \mathcal{H}}{\partial \mathbf{X}} = \mathbf{A}$ but:

$$\frac{\partial \mathcal{H}}{\partial \mathbf{X}} = \mathbf{A} + \Delta t \begin{bmatrix} 0_3 & 0_3 & 0_3 & 0_3 \\ 0_3 & 0_3 & 0_3 & 0_3 \\ 0_3 & 0_3 & 0_3 & 0_3 \\ -{}^l\mathcal{I}^{-1} \sum_{i \in \mathcal{C}_t} \mathbf{I}_3 \times \mathbf{f}_{i,t} & 0_3 & 0_3 & 0_3 \end{bmatrix} \tag{8.97}$$

since with $b(\mathbf{X}) = {}^l\mathcal{I}^{-1} \sum_{i \in \mathcal{C}} ({}^l r_{i,t}^\star - {}^l c_t) \times {}^l \mathbf{f}_i$:

$$\frac{\partial \mathbf{B}}{\partial \mathbf{X}} = \Delta t \begin{bmatrix} 0_3 & 0_3 & 0_3 & 0_3 \\ 0_3 & 0_3 & 0_3 & 0_3 \\ 0_3 & 0_3 & 0_3 & 0_3 \\ \frac{\partial b}{\partial c_x} & \frac{\partial b}{\partial c_y} & \frac{\partial z}{\partial c_x} & 0_3 & 0_3 & 0_3 \end{bmatrix} \tag{8.98}$$

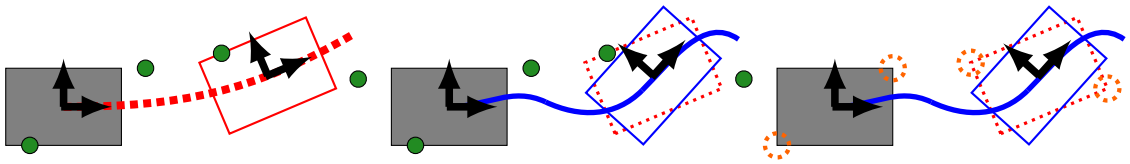
And for example with the c_x variable:

$$\frac{\partial b}{\partial c_x} = -{}^l\mathcal{I}^{-1} \sum_{i \in \mathcal{C}_t} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \times \begin{bmatrix} \mathbf{f}_{i,t}^x \\ \mathbf{f}_{i,t}^y \\ \mathbf{f}_{i,t}^z \end{bmatrix} \tag{8.99}$$

because $(u \times v)' = u' \times v + u \times v'$ and $\frac{\partial c}{\partial c_x} = \begin{bmatrix} \frac{\partial c_x}{\partial c_x} \\ \frac{\partial c_y}{\partial c_x} \\ \frac{\partial c_z}{\partial c_x} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$

8.5 Optimizing footsteps locations with DDP

We can go even further in the nonlinearity to exploit the possibilities offered by the DDP approach compared to the convex QP problem that has to be linear. We can for instance consider footstep locations as part of the optimization problem instead of using those determined by the heuristics of the footstep planner. This all-in-one optimization would allow a completely heuristic-free choice of footstep location without the need to formulate the heuristics described in Chapter 7. The three MPC variants introduced so far are illustrated in Fig. 8.8.



(a) Linear (b) Nonlinear (c) Footsteps optimization

Figure 8.8: Summary of the differences between MPC variants. The front-left and hind-right feet are first in contact, with a switch to the front-right and hind-left feet later in the prediction horizon. Both linear (a) and nonlinear (b) variants use footsteps locations defined by heuristics (green dots) while the one optimizing footsteps locations (c) works around the reference projections of shoulders on the ground (orange dotted circles). (a) relies on the reference trajectory of the CoM (red dotted line) while the two others use instead the predicted one (blue line) that results from the application of (8.1a).

The OCP is thus changed from (8.80) to:

$$\begin{aligned} \min_{\{\mathbf{X}\},\{\mathbf{F}\},\{r\}} & \sum_{t=0}^{T} \ell_t(\mathbf{X}_t, \mathbf{F}_t | r_t) + \ell_T(x_T) \\ \text{s.t.} \quad & \mathbf{X}_{t+1} = \mathcal{H}(\mathbf{X}_t, \mathbf{F}_t | r_t) \\ & \mathbf{X}_t \in \mathcal{F} \\ & \mathbf{F}_t \in \mathcal{K} \end{aligned} \tag{8.100}$$

Footstep locations r are now a decision variable. To sum up, we now have three variants of MPC, for which the nature of the lever arms in (8.81d) varies, as recapped in Table 8.1.

To include footstep locations in the optimization process, plain state variables whose values can only be changed at impact time are added. This is implemented as a specific dynamic function inserted in the time line at the beginning of each contact phase:

$$r_{t+1} = \mathcal{G}(\Delta r_t | \mathbf{X}_t, \mathbf{F}_t) = r_t + \Delta r_t \tag{8.101}$$

where Δr_t is the step length taken by the corresponding foot during the previous flying phase. The size of Δr_t depends on the number of contacts that are modified.

For the DDP solver, each time step of the prediction horizon is associated with a node which contains the state evolution model. Contact switches are instantaneous and are done

MPC Variant	Footsteps Locations	CoM Traj.	Lever arm
Linear	Heuristics	Reference	$[r_i^\star - c^\star]_\times$
Nonlinear	Heuristics	Predicted	$[r_i^\star - c]_\times$
Footsteps	Optimized	Predicted	$[r_i - c]_\times$

Table 8.1: Differences between MPC variants

independently of the state evolution, they are thus modeled by another set of nodes $\mathcal{G}$ that are inserted between model nodes $\mathcal{H}$ when contact switches occur. The organization of these nodes is highlighted in Fig. 8.9.

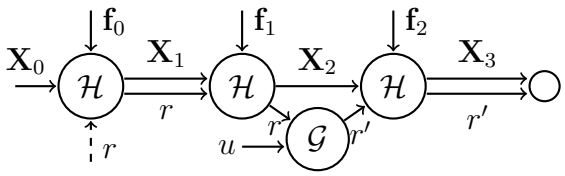


Figure 8.9: Factor graph displaying the correlations between the decision variables of the nonlinear MPC which optimizes footstep locations. Example with a prediction horizon of 3 time steps and a contact switch occurring at the 2nd time step.

The positions of footsteps is included even in the nodes for model evolution $\mathcal{H}$ so that the information is transmitted for the lever arms, cost, derivatives and Hessians computation. The ground is assumed flat so the height of contacts is always 0 in local frame. Only 8 variables are thus needed to process the 4 contacts. Compared to (8.81c), the new state vector is:

$$\mathbf{X}_t = \begin{bmatrix} {}^l c_t \\ {}^l\Theta_t \\ {}^l\dot{c}_t \\ {}^l\dot{\Theta}_t \\ {}^l\mathbf{r}_t \end{bmatrix} \text{ with } \forall t,\ \mathbf{r}_t = \begin{bmatrix} r^x_{1,t} \\ r^y_{1,t} \\ r^x_{2,t} \\ r^y_{2,t} \\ r^x_{3,t} \\ r^y_{3,t} \\ r^x_{4,t} \\ r^y_{4,t} \end{bmatrix} \tag{8.102}$$

Only $\mathcal{G}$ nodes are responsible for changes of footsteps positions, so compared to (8.81b) the new state evolution matrix is:

$$\mathbf{A} = \begin{bmatrix} \mathbf{I}_6 & \Delta t \mathbf{I}_6 & 0_{6\times 8} \\ 0_6 & \mathbf{I}_6 & 0_{6\times 8} \\ 0_{8\times 6} & 0_{8\times 6} & \mathbf{I}_8 \end{bmatrix} \tag{8.103}$$

Cost derivatives and Hessian matrices are again slightly different than for the previous MPCs, due to the addition of the new footstep variables in the state vector.

For each $i \in \mathcal{C}_t$ such that $(\|r_{sh,i,t} - r_{i,t}\|^2 - d_{lim}^2)^+ > 0$ (in that case, $\delta_i = 1$, 0 other-

wise), terms are added to $L_{\mathbf{X}}$ that is now of size 18×1 due to the new footsteps variables:

$$L_{\mathbf{X}} \mathrel{+}= w_{kin} \begin{bmatrix} 0_{12\times1} \\ r^x_{sh,1,t}\ \delta_1 \\ r^y_{sh,1,t}\ \delta_1 \\ r^x_{sh,2,t}\ \delta_2 \\ r^y_{sh,2,t}\ \delta_2 \\ r^x_{sh,3,t}\ \delta_3 \\ r^y_{sh,3,t}\ \delta_3 \\ r^x_{sh,4,t}\ \delta_4 \\ r^y_{sh,4,t}\ \delta_4 \end{bmatrix}_{18\times1} \tag{8.104}$$

Terms are also added to $L_{\mathbf{XX}}$ which is now of size 18×18, with γ_1 and γ_2 defined as in (8.93):

$$L_{\mathbf{XX}} \mathrel{+}= w_{kin} \begin{bmatrix} 0_{12\times12} & L_{\mathbf{XX},fsteps} \\ (L_{\mathbf{XX},fsteps})^T & 0_{8\times8} \end{bmatrix}_{18\times18} \tag{8.105}$$

$$L_{\mathbf{XX},fsteps} = \begin{bmatrix} \text{-}\delta_1 & 0 & \text{-}\delta_2 & 0 & \text{-}\delta_3 & 0 & \text{-}\delta_4 & 0 \\ 0 & \text{-}\delta_1 & 0 & \text{-}\delta_2 & 0 & \text{-}\delta_3 & 0 & \text{-}\delta_4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \text{-}\gamma_1\delta_1 & 0 & \text{-}\gamma_1\delta_2 & 0 & \text{-}\gamma_1\delta_3 & 0 & \text{-}\gamma_1\delta_4 & 0 \\ 0 & \text{-}\gamma_2\delta_1 & 0 & \text{-}\gamma_2\delta_2 & 0 & \text{-}\gamma_2\delta_3 & 0 & \text{-}\gamma_2\delta_4 \end{bmatrix} \tag{8.106}$$

New coefficients also appear in the derivative of the state dynamics compared to (8.97):

$$\frac{\partial \mathcal{H}}{\partial \mathbf{X}} = \mathbf{A} + \Delta t \begin{bmatrix} 0_3 & 0_3 & 0_3 & 0_3 & 0_{3\times2} & 0_{3\times2} & 0_{3\times2} & 0_{3\times2} \\ 0_3 & 0_3 & 0_3 & 0_3 & 0_{3\times2} & 0_{3\times2} & 0_{3\times2} & 0_{3\times2} \\ 0_3 & 0_3 & 0_3 & 0_3 & 0_{3\times2} & 0_{3\times2} & 0_{3\times2} & 0_{3\times2} \\ \text{-}{}^l\mathcal{I}^{-1} \sum_{i\in\mathcal{C}_t} \mathbf{I}_3 \times \mathbf{f}_{i,t} & 0_3 & 0_3 & 0_3 & \Gamma_4\delta_4 & \Gamma_4\delta_4 & \Gamma_4\delta_4 & \Gamma_4\delta_4 \end{bmatrix} \tag{8.107}$$

$$\forall i \in \{1,2,3,4\},\ \Gamma_i = \text{-}{}^l\mathcal{I}^{-1} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \times \mathbf{f}_{i,t} \tag{8.108}$$

8.6 Optimising footsteps timings with DDP

For the sake of completeness, let's briefly focus on a fourth MPC variant that has yet to be tested. Its OCP formulation includes contact timings as a decision variable on top of previous ones. The solver can now optimize not only the footstep locations r but their timings T_s as well. The idea behind that is that the solver might prefer to delay or hasten a contact switch to better track the reference trajectory of the center of mass in some situations.

$$\begin{aligned} \min_{\{\mathbf{X}\},\{\mathbf{F}\},\{r\},\{T_s\}} & \sum_{t=0}^{T} \ell_t(\mathbf{X}_t, \mathbf{F}_t | r_t) + \ell_T(x_T) \\ \text{s.t.} \quad \forall t \quad & \mathbf{X}_{t+1} = \mathcal{H}(\mathbf{X}_t, \mathbf{F}_t | r_t) \\ \forall t \quad & \mathbf{X}_t \in \mathcal{F} \\ \forall t \quad & \mathbf{F}_t \in \mathcal{K} \end{aligned} \tag{8.109}$$

From a stability point of view, the shorter the duration until the next contact switch, the less time the base has to deviate from the reference due to the inherent instability of having only two punctual contacts. As pointed out in Section 6.3, a higher gait frequency limits the natural tilting of the base since the robot cannot apply momentum along the axis that goes through its two only punctual contacts. In fact, it has a double beneficial effect with the base tilting during a shorter duration and the robot being able to correct that tilt more often as the contact pair switches. However, shorter swing phases can imply higher feet velocities in the air, especially if the solver decides to take long steps. This effect can be controlled by adding an additional term ℓ_{hvel} in the cost function to penalize the horizontal velocity of swinging feet. The maximum velocity can be simply estimated through interpolation, knowing the start and end locations of the current or incoming swing phases. The challenge remains to find the proper equilibrium between base stability and feet timings to avoid having the solver overly focused on one of them, which would be detrimental to the general behavior of the robot. In term of implementation, this optimization can be done by setting the integration time between DDP nodes as an optimization variable so that the number of nodes is always the same and it is only the duration between them that changes to shorten or lengthen a contact phase. The discretization fineness can however deteriorate if the time step between nodes is increased too much. Another way to do it is to keep the duration between nodes the same but to add or remove some nodes on the fly to tune the duration of contact phases. This avoids the fineness issue but means the changes of duration are incremental and not continuous anymore since the time step between nodes becomes the minimum time quantum.

8.7 Conclusion

In this chapter we have first described how the reference state trajectory of the base is generated from the reference base horizontal velocity (forward, lateral, rotation around vertical axis) given by a user or a higher level controller. This trajectory is given along with footstep locations as inputs to a centroidal model predictive controller whose goal is to output desired contact forces that should be applied to track the reference. This reference trajectory can be augmented along the vertical axis both in position and velocity to perform jumping motions. We presented several ways of increasing complexity to solve this optimal control problem. It started from an initial convex quadratic program which forced us to formulate several assumptions to get a linear problem, notably that reference and predicted states are close from each other. Then, the same problem was solved, this time using differential dynamic programming to prove that similar results could be obtained with this approach. From there on, we were able to relieve one of the linearity assumptions thanks to the capability of DDP to handle nonlinear problems. We went even further by including footsteps locations as part of the optimization problem instead of considering them as given by the footstep planner of Chapter 7 to get a all-in-one problem that avoids the need to formulate heuristics. Finally, we hinted at a way to include footstep timings as part of the optimization as well to allow a more complete control of the gait by the solver.

The next chapter focuses on the different kinds of whole-body controllers that were implemented to combine both desired contact forces coming for the MPC and the feet commands from the swing phase trajectory generator. The goal is to convert these quantities into joint torques, positions and velocities for the low-level impedance controller with still the overall goal to follow the reference horizontal base velocity.

Chapter 9

Whole-Body Control

Contents

Model predictive control allows to take locomotion decisions considering the future evolution of a system. By reasoning over a prediction horizon, future events such as contact switches can be taken into account before they even happen to obtain an improved behavior compared to instantaneous controllers that only act according to the current state. Yet due to real-time computational requirements, these predictive controllers often have to use simplified models to ease computations. In our case, the centroidal model of the MPC forgets all notion of joints to consider the robot as a single lumped mass. The evolution of the system is then only guided by the inertia, gravity and the ground reaction forces at contact points. However in practice the robot is a poly-articulated system with multiple joints that have to be controlled. Thus, the role of the whole-body controller is to convert the desired contact forces provided by the MPC and the reference feet position, velocity and acceleration given by the trajectory generators into torque, position and velocity commands that are sent to the low-level impedance controller.

9.1 Task-Space Inverse Dynamics

Task Space Inverse Dynamics allows to perform task-orientated optimization-based inverse-dynamics control. To this end we used the rigid multi-body dynamics library

Pinocchio. Unlike the MPC it does not consider a centroidal model of the robot over a prediction horizon but instead its whole poly-articulated model with each link having its own mass and inertia instantaneously. It reasons over a weighted set of tasks that should be performed while respecting constraints and enforcing the whole-body dynamics. In our case, five kinds of tasks are being used to track the desired contact forces of the MPC and the feet commands of the swing trajectory generator, with the ultimate goal to follow the reference horizontal base velocity:

- Contact tasks for feet in contact with the ground to inform the solver that these feet should not move and that they can be used to apply forces on the ground.
- Force tasks to have the contact forces close to the desired contact forces outputted by the MPC. These tasks are associated with the contact tasks.
- Tracking tasks for feet in swing phase to follow the 3D trajectory in position, velocity and acceleration generated by the foot trajectory generator, in order to land at the positions desired by the footstep planner.
- Tracking task for the trunk to follow the reference horizontal velocity given by the user or higher level controller
- Posture task for all legs to get back to a default position if some degrees of freedom are not used.

One instance of the first three task is initially created and assigned to each foot. Then during the gait these tasks are enabled or disabled depending on the state of the feet. In swing phase only the tracking task is active while in stance phase only contact and force tasks are enabled. There exists a single posture task for the body and the legs which is always active and affect the whole body.

A weight is assigned to each task to make it more or less important compared to the other ones. "Contact + Force" and 3D tracking are never active at the same time so they do not compete with each other. As the posture task is just intended to be use as a form of regularization, it should not interfere with the other tasks. Its weight is kept at least 10^{-2} times less than the others to mimic a hierarchical solver: the relative weight is so small that it does not impact the other tasks even if in practice all tasks are considered together during the solving process.

If the i-th foot is in stance phase then the force reference of its force task is updated with the desired contact force $\mathbf{f}_i$ outputted by the MPC. If the i-th foot is in swing phase then its tracking task is updated with the desired position $[\mathcal{X}^\star\ \mathcal{Y}^\star\ \mathcal{Z}^\star]$, velocity $[\dot{\mathcal{X}}^\star\ \dot{\mathcal{Y}}^\star\ \dot{\mathcal{Z}}^\star]$ and acceleration $[\ddot{\mathcal{X}}^\star\ \ddot{\mathcal{Y}}^\star\ \ddot{\mathcal{Z}}^\star]$ outputted by the foot trajectory generator associated with this foot, as described in Section 7.2.

The inverse dynamics solver is first updated with the current state of the quadruped (position, orientation and velocity of the base as well as positions and velocities of the joints). It then tries to find the accelerations for both the underactuated dynamics (base) and the actuated one (joints) using the contact forces to minimize the cost function (weighted sum of task errors) while respecting the constraints (contacts, dynamics equations, torque limits). Joint torques can be retrieved at the end of the optimization with the accelerations and the contact forces. These torques are then sent to the low-level impedance controller along with the desired joint positions and velocities.

The decision variables $\mathbf{y}$ of the Task-Space Inverse Dynamics are the base and joint accelerations $\ddot{\mathbf{q}}$, the ground reaction forces $\mathbf{F}$ and the joint torques τ.

$$\mathbf{y} = \begin{bmatrix} \ddot{\mathbf{q}} \\ \mathbf{F} \\ \tau \end{bmatrix} \tag{9.1}$$

All tasks are associated with an error $\|\mathbf{A}\mathbf{y} - \mathbf{a}\|^2$ to minimize that depends on the decision variables $\mathbf{y}$. For the contact force task:

$$\|\mathbf{A}_{\mathbf{F}}\mathbf{y} - \mathbf{a}_{\mathbf{F}}\|^2 = \| \begin{bmatrix} 0 & \mathbf{I} & 0 \end{bmatrix} \mathbf{y} - \mathbf{F}^\star\|^2 = \|\mathbf{F} - \mathbf{F}^\star\|^2 \tag{9.2}$$

For the feet tracking task:

$$\|\mathbf{A}_{feet}\mathbf{y} - \mathbf{a}_{feet}\|^2 = \| \begin{bmatrix} \mathbf{J} & 0 & 0 \end{bmatrix} \mathbf{y} - \ddot{\mathbf{x}}^\star_{feet}\|^2 = \|\mathbf{J}\ddot{\mathbf{q}} - \ddot{\mathbf{x}}^\star_{feet}\|^2 \tag{9.3}$$

For the posture task of the base:

$$\|\mathbf{A}_{base}\mathbf{y} - \mathbf{a}_{base}\|^2 = \| \underbrace{\begin{bmatrix} [\mathbf{I}_u \; 0_a] & 0 & 0 \end{bmatrix}}_{\text{only underactuated}} \mathbf{y} - \ddot{\mathbf{q}}^\star_{base}\|^2 = \|\ddot{\mathbf{q}}_u - \ddot{\mathbf{q}}^\star_u\|^2 \tag{9.4}$$

For the posture task of the legs:

$$\|\mathbf{A}_{post}\mathbf{y} - \mathbf{a}_{post}\|^2 = \| \underbrace{\begin{bmatrix} [0_u \; \mathbf{I}_a] & 0 & 0 \end{bmatrix}}_{\text{only actuated}} \mathbf{y} - \ddot{\mathbf{q}}^\star_{post}\|^2 = \|\ddot{\mathbf{q}}_a - \ddot{\mathbf{q}}^\star_a\|^2 \tag{9.5}$$

Considering the error of a task function e, we can impose a second order linear dynamic for the evolution of the error by applying an acceleration $\ddot{e} = K_d\,\dot{e} + K_p\,e$. So, for the feet tracking and posture tasks, using estimation from forward kinematics, forward geometry and encoder measurements, the task reference will be:

$$\ddot{\mathbf{x}}^\star_{feet} = \begin{bmatrix} \ddot{\mathcal{X}}^\star \\ \ddot{\mathcal{Y}}^\star \\ \ddot{\mathcal{Z}}^\star \end{bmatrix} + K_{d,feet} \begin{bmatrix} \dot{\mathcal{X}}^\star - \dot{\mathcal{X}}^{FK} \\ \dot{\mathcal{Y}}^\star - \dot{\mathcal{Y}}^{FK} \\ \dot{\mathcal{Z}}^\star - \dot{\mathcal{Z}}^{FK} \end{bmatrix} + K_{p,feet} \begin{bmatrix} \mathcal{X}^\star - \mathcal{X}^{FG} \\ \mathcal{Y}^\star - \mathcal{Y}^{FG} \\ \mathcal{Z}^\star - \mathcal{Z}^{FG} \end{bmatrix} \tag{9.6}$$

$$\ddot{\mathbf{q}}^\star_u = 0 + K_{d,base}(\dot{\mathbf{q}}^\star_u - \dot{\mathbf{q}}_u) + K_{p,base}(\mathbf{q}^\star_u - \mathbf{q}_u) \tag{9.7}$$

$$\ddot{\mathbf{q}}^\star_a = 0 + K_{d,post}(0 - \dot{\mathbf{q}}_a) + K_{p,post}(\mathbf{q}^\star_a - \mathbf{q}_a) \tag{9.8}$$

The goal of the solver is to minimize the following cost function:

$$\min_{\mathbf{y}} \sum_i w_i \|\mathbf{A}_i\mathbf{y} - \mathbf{a}_i\|^2 \tag{9.9}$$

$$\text{such that} \begin{bmatrix} \mathbf{J} & 0 & 0 \\ \mathbf{M}_u & -\mathbf{J}^{\mathbf{T}}_{\mathbf{u}} & 0 \\ \mathbf{M}_a & -\mathbf{J}^{\mathbf{T}}_{\mathbf{a}} & -\mathbf{I} \end{bmatrix} \mathbf{y} = \begin{bmatrix} -\dot{\mathbf{J}}\dot{\mathbf{q}} \\ -\mathbf{h}_u \\ -\mathbf{h}_a \end{bmatrix} \tag{9.10}$$

$$\underbrace{\begin{bmatrix} 0 & \mathbf{F}_\mu & 0 \\ 0 & 0 & \mathbf{I} \end{bmatrix}}_{\mathbf{B}} \mathbf{y} \leq \underbrace{\begin{bmatrix} 0 \\ \tau^{max} \end{bmatrix}}_{\mathbf{b}} \tag{9.11}$$

where subscripts $_u$ and $_a$ refer to the underactuated and actuated parts respectively. $\mathbf{M}$ is the generalized mass matrix, $\mathbf{h}$ the vector of the gravitational and nonlinear terms and $\mathbf{J}$ the contact Jacobian.

The first line in (9.10) comes from the assumption of non-moving rigid contacts, differentiated twice. If contact points do not move, then contact point velocities are null ($\mathbf{J}\dot{\mathbf{q}} = 0$), so contact point accelerations are null ($\mathbf{J}\ddot{\mathbf{q}} + \dot{\mathbf{J}}\dot{\mathbf{q}} = 0$). The second and third lines in (9.10) are the equation of the dynamics $\mathbf{M}\ddot{\mathbf{q}} + \mathbf{h} = \mathbf{S}^T\tau + \mathbf{J}^T\mathbf{F}$ with the underactuated and actuated parts separated. For the inequalities, (9.11) enforces friction cone constraints for contacts and ensures that torques remain under the maximum values that the actuators can produce. Matrix $\mathbf{F}_\mu$ is similar to (8.73) used for the friction cone constraint in the MPC.

$$\mathbf{F}_\mu = \begin{bmatrix} \mathbf{G} & 0_{5\times3} & 0_{5\times3} & 0_{5\times3} \\ 0_{5\times3} & \mathbf{G} & 0_{5\times3} & 0_{5\times3} \\ 0_{5\times3} & 0_{5\times3} & \mathbf{G} & 0_{5\times3} \\ 0_{5\times3} & 0_{5\times3} & 0_{5\times3} & \mathbf{G} \end{bmatrix} \text{ and } \mathbf{G} = \begin{bmatrix} 1 & 0 & -\mu \\ -1 & 0 & -\mu \\ 0 & 1 & -\mu \\ 0 & -1 & -\mu \\ 0 & 0 & -1 \end{bmatrix} \tag{9.12}$$

For computational efficiency, the size of the problem can be reduced by expressing τ according to $\ddot{\mathbf{q}}$ and $\mathbf{F}$ in (9.10).

$$\underbrace{\begin{bmatrix} \ddot{\mathbf{q}} \\ \mathbf{F} \\ \tau \end{bmatrix}}_{\mathbf{y}} = \underbrace{\begin{bmatrix} \mathbf{I} & 0 \\ 0 & \mathbf{I} \\ \mathbf{M}_a & -\mathbf{J}_\mathbf{a}^\mathbf{T} \end{bmatrix}}_{\mathbf{D}} \underbrace{\begin{bmatrix} \ddot{\mathbf{q}} \\ \mathbf{F} \end{bmatrix}}_{\bar{\mathbf{y}}} + \underbrace{\begin{bmatrix} 0 \\ 0 \\ \mathbf{h}_a \end{bmatrix}}_{\mathbf{d}} \tag{9.13}$$

The reformulated problem is then:

$$\min_{\bar{\mathbf{y}}} \quad \sum_i w_i \|\mathbf{A}_i\mathbf{D}\bar{\mathbf{y}} + \mathbf{A}_i\mathbf{d} - \mathbf{a}_i\|^2 \tag{9.14}$$

$$\text{such that } \begin{bmatrix} \mathbf{J} & 0 \\ \mathbf{M}_u & -\mathbf{J}_\mathbf{u}^\mathbf{T} \end{bmatrix} \bar{\mathbf{y}} = \begin{bmatrix} -\dot{\mathbf{J}}\dot{\mathbf{q}} \\ -\mathbf{h}_u \end{bmatrix} \tag{9.15}$$

$$\mathbf{B}\mathbf{D}\,\bar{\mathbf{y}} \leq \mathbf{b} - \mathbf{B}\mathbf{d} \tag{9.16}$$

Once the optimization is over, the joint torques τ can be directly retrieved from the optimization vector $\mathbf{y}$. Target joint positions and velocities are obtained by integrating twice the acceleration $\ddot{\mathbf{q}}$ over the duration of one time step from the joint positions and velocities used as input for the whole-body control.

On that point, two feedback schemes can be considered. First, we can choose to use a direct feedback from the robot by providing the WBC with the joint positions $\mathbf{q}_a^{mes}$ and velocities $\dot{\mathbf{q}}_a^{mes}$ measured by the encoders. In that case, the state of the robot given to the whole-body model is similar to the real-world one. The targets given to the impedance controller will thus be the current joint state with a small increment from the integration:

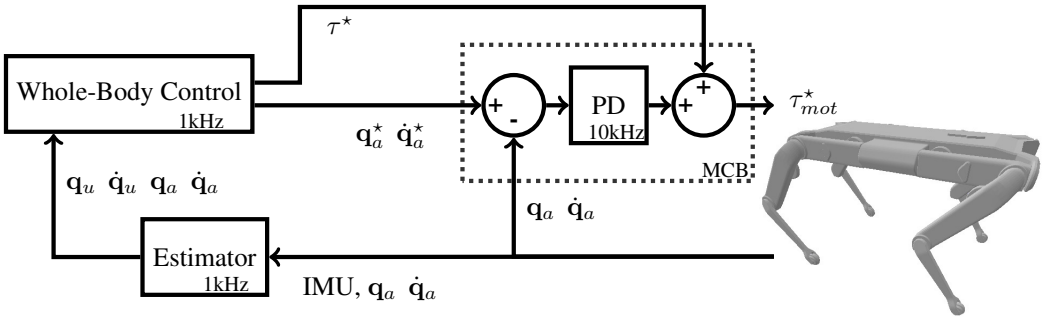
$$\dot{\mathbf{q}}_{a,k}^\star = \dot{\mathbf{q}}_a^{mes} + \Delta t\, \ddot{\mathbf{q}}_{a,k}^\star \tag{9.17}$$

$$\mathbf{q}_{a,k}^\star = \mathbf{q}_a^{mes} + \Delta t\, \dot{\mathbf{q}}_{a,k}^\star \tag{9.18}$$

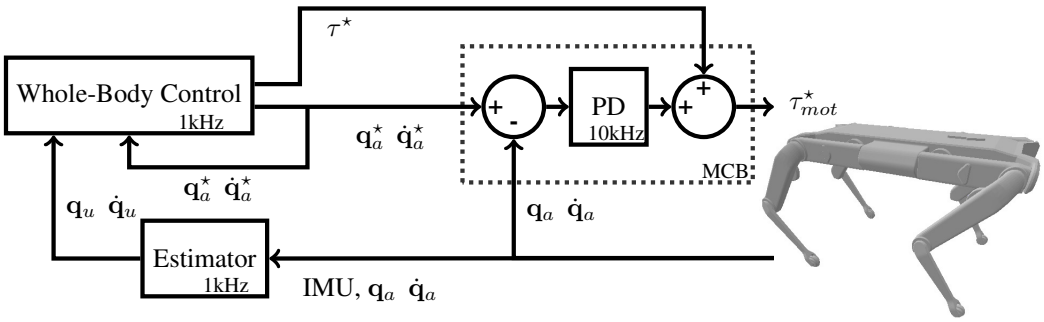
Otherwise, we can choose to use a hybrid control with feedback only on the underactuated part of the state. For the actuated part, the WBC loops on itself with joint positions and velocities that are simply the targets of the previous iteration. In this case, only the impedance controller performs feedback on the measured positions and velocities.

$$\dot{\mathbf{q}}_{a,k}^\star = \dot{\mathbf{q}}_{a,k-1}^\star + \Delta t\, \ddot{\mathbf{q}}_{a,k}^\star \tag{9.19}$$

$$\mathbf{q}_{a,k}^\star = \mathbf{q}_{a,k-1}^\star + \Delta t\, \dot{\mathbf{q}}_{a,k}^\star \tag{9.20}$$



(a) Direct feedback from the robot using encoders measurements



(b) Hybrid control using references as the state for the next iteration

Figure 9.1: Comparison of the two feedback schemes that can be use for the actuated part of the state. The PD controller is directly performed by the motor control board (MCB) of the robot.

However, even if we managed to get a trotting gait working at low speed ($\leq 0.5m/s$) in simulation with such Task Space Inverse Dynamics schemes, stability issues were encountered when deploying the architecture on the real robot with either of the two feedbacks. It seemed that the whole-body control was not able to generate proper commands for the impedance controller to track the feet motion.

We cannot state for sure the causes of those instabilities since we did not study them extensively. Yet our hypothesis is that they are linked to how central the robot dynamics is in the scheme when adopting an inverse dynamics approach. Indeed, though the approach can work well in simulation, because the dynamic model used to compute the commands is exactly the same as the one used by the simulator to compute the response of the system, this is no longer true when dealing with the real robot because of model errors and simplifications. Hence the struggle to regulate a system that does not evolve as expected. For this reason, we decided to implement the well tested IK + QP strategy instead of a sole inverse dynamics scheme, as in [Kim+19]. Inverse kinematics, which relies purely on geometric quantities that are well-known, is less sensitive to those errors in the dynamics model. Furthermore, another point that likely aggravates this issue is that Solo-12 is a lightweight quadrupedal robot with almost direct-drive actuators (reduction ratio of 9). As pointed out and studied in [SHV+06], coupling among the links is more significant with direct-drive robots and achieving high performances with such actuators is particularly challenging. So as we did not manage to run the inverse dynamics faster than 500 Hz, it might have been insufficient to avoid instability due to the fast dynamics of the actuators.

9.2 Transition to Inverse Kinematics with a QP problem

The whole-body control relies on two successive blocks: inverse kinematics (IK) and a torque computation (Fig. 9.2). First, the IK computes the joint accelerations to perform a set of position and velocity tasks, based on which a QP problem is solved to find a compromise between tracking these joint accelerations and taking into account the MPC decisions and the equation of dynamics. This QP problem has to balance decisions that might be conflicting since the instantaneous decisions of the IK are done without knowing what has been decided by the MPC that works on a prediction horizon. In that sense, this approach is less "all-in-one" than the task-space inverse dynamics which performs both steps at once.

9.2.1 Computing desired accelerations

The first step is to compute command accelerations $\ddot{\mathbf{q}}_{IK}$ by IK of the full model of the quadruped. The IK scheme is defined by 3 tasks:

- Keep the base at constant height and follow the reference horizontal velocity (3 DoF for base position)
- Keep the base orientation horizontal and follow the reference yaw angular velocity (3 DoF for base orientation)
- Follow the reference trajectory of the swing feet while maintaining feet in stance phase immobile (3 DoF per leg, 12 DoF in total)

For a quadruped robot with 18 DoF, these tasks fully constrain the system but are compatible as the number of DoF is sufficient to satisfy each of them independently. As all tasks are compatible, we prefer to discard the hierarchical inverse kinematic scheme introduced in [Kim+19], which relied on a series of projections in the null space of tasks of higher priority. In our case, these projections are not required because tasks do not overlap. We implement instead the following simpler resolution. By stacking all the task functions in a global vector according to the above description order, a global task Jacobian of size 18 by 18 can be defined as:

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{\mathbf{x}}_{lin} \\ \dot{\mathbf{x}}_{ang} \\ \dot{\mathbf{x}}_1 \\ \dot{\mathbf{x}}_2 \\ \dot{\mathbf{x}}_3 \\ \dot{\mathbf{x}}_4 \end{bmatrix} = \mathbf{J}\dot{\mathbf{q}} = \begin{bmatrix} {}^o\mathbf{R}_b & 0 & 0 & \dots & 0 \\ 0 & {}^o\mathbf{R}_b & 0 & & \\ {}^o\mathbf{R}_b & {}^b\mathbf{T}_1 \times {}^o\mathbf{R}_b & \mathbf{J}_1 & \ddots & \vdots \\ {}^o\mathbf{R}_b & {}^b\mathbf{T}_2 \times {}^o\mathbf{R}_b & 0 & \mathbf{J}_2 & \ddots \\ {}^o\mathbf{R}_b & {}^b\mathbf{T}_3 \times {}^o\mathbf{R}_b & \vdots & \ddots & \mathbf{J}_3 \quad 0 \\ {}^o\mathbf{R}_b & {}^b\mathbf{T}_4 \times {}^o\mathbf{R}_b & 0 & \dots & 0 \quad \mathbf{J}_4 \end{bmatrix} \begin{bmatrix} \dot{\mathbf{q}}_{lin} \\ \dot{\mathbf{q}}_{ang} \\ \dot{\mathbf{q}}_1 \\ \dot{\mathbf{q}}_2 \\ \dot{\mathbf{q}}_3 \\ \dot{\mathbf{q}}_4 \end{bmatrix} \tag{9.21}$$

with ${}^o\mathbf{R}_b$ the rotation matrix from base to world frame, ${}^b\mathbf{T}_i$ the position of the i-th foot in base frame ($\forall i \in \{1, 2, 3, 4\}$), $\mathbf{J}_i$ the Jacobian of the i-th foot. $\dot{\mathbf{q}}$ is the time derivative of the configuration vector $\mathbf{q}$ (6D base + 12 joints), while $\dot{\mathbf{x}}$ is the time derivative of the state vector in task space. As such, $\dot{\mathbf{x}}_{lin}$, $\dot{\mathbf{x}}_{ang}$ are the base linear and angular velocities in world frame, $\dot{\mathbf{x}}_i$ the velocity of the i-th foot in world frame, $\dot{\mathbf{q}}_{lin}$, $\dot{\mathbf{q}}_{ang}$ the base linear and angular velocities in base frame and $\dot{\mathbf{q}}_i$ the joint velocities of the i-th leg. Rather than directly inverting the whole matrix $\mathbf{J}$, it is sufficient to invert analytically only the $\mathbf{J}_i^{-1}$

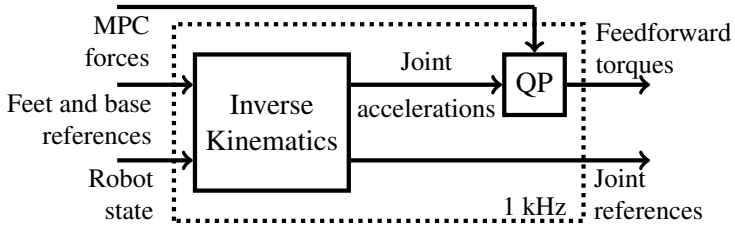


Figure 9.2: Whole-body control scheme with its two successive steps: inverse kinematics and solving of a QP problem

matrices as other quantities are known, which is more computationally efficient:

$$\mathbf{J}^{-1} = \begin{bmatrix} {}^b\mathbf{R}_h & 0 & 0 & 0 & 0 & 0 \\ 0 & {}^b\mathbf{R}_h & 0 & 0 & 0 & 0 \\ -\mathbf{J}_1^{-1} & \mathbf{J}_1^{-1}[{}^b\mathbf{T}_1]_\times & \mathbf{J}_1^{-1} & 0 & 0 & 0 \\ -\mathbf{J}_2^{-1} & \mathbf{J}_2^{-1}[{}^b\mathbf{T}_2]_\times & 0 & \mathbf{J}_2^{-1} & 0 & 0 \\ -\mathbf{J}_3^{-1} & \mathbf{J}_3^{-1}[{}^b\mathbf{T}_3]_\times & 0 & 0 & \mathbf{J}_3^{-1} & 0 \\ -\mathbf{J}_4^{-1} & \mathbf{J}_4^{-1}[{}^b\mathbf{T}_4]_\times & 0 & 0 & 0 & \mathbf{J}_4^{-1} \end{bmatrix} \tag{9.22}$$

$\mathbf{J}_i$ are 3 by 3 invertible matrices except when the i-th leg is in singularity, that is when the knee joint is aligned with the upper leg. In practice a damped pseudo-inverse is used to avoid near-zero singular values that would lead to numerical instabilities when legs are near singularities [Bus04].

With $\mathbf{x}^\star$, $\dot{\mathbf{x}}^\star$, $\ddot{\mathbf{x}}^\star$ the stacked desired positions, velocities and accelerations of all tasks and K_p, K_d their position and velocity feedback gains, the command acceleration in task space is computed as (9.6)-(9.8) to impose a second-order linear dynamics for the reduction of the task errors:

$$\ddot{\mathbf{x}}^{cmd} = K_p(\mathbf{x}^\star - \mathbf{x}) + K_d(\dot{\mathbf{x}}^\star - \dot{\mathbf{x}}) + \ddot{\mathbf{x}}^\star \tag{9.23}$$

The command joint accelerations can then be obtained:

$$\ddot{\mathbf{q}}_{IK} = \mathbf{J}^{-1}(\ddot{\mathbf{x}}^{cmd} - \dot{\mathbf{J}}\dot{\mathbf{q}}) \tag{9.24}$$

These accelerations are sent to the second step of the whole-body control to compute feedforward torque commands.

9.2.2 Computing reference positions and velocities

As Solo-12 motors are current controlled without torque feedback, the low level controller consists of a joint-level impedance controller, as shown in Fig. 4.1. The joint positions and velocities commands $\mathbf{q}_a^\star$ and $\dot{\mathbf{q}}_a^\star$ can be retrieved from the actuated part of $\mathbf{q}^{cmd}$ and $\dot{\mathbf{q}}^{cmd}$, defined as:

$$\mathbf{q}_k^{cmd} = \mathbf{q}_{k-1}^{cmd} + \mathbf{J}^{-1}\,(\mathbf{x}^\star - \mathbf{x}) \tag{9.25}$$

$$\dot{\mathbf{q}}^{cmd} = \mathbf{J}^{-1}\,\dot{\mathbf{x}}^\star \tag{9.26}$$

9.2.3 Feedforward torques computation

For the computation of feedforward torques, a quadratic programming solver which relies on relaxation variables $\delta_{\ddot{\mathbf{q}}}$ and $\delta_{\mathbf{F}}$ is used to find contact forces $\mathbf{F} = \mathbf{F}_{MPC} + \delta_{\mathbf{F}}$ and

accelerations $\ddot{\mathbf{q}} = \ddot{\mathbf{q}}_{IK} + \delta_{\ddot{\mathbf{q}}}$ that are as close as possible to the force references provided by the MPC and the command accelerations computed by the IK, while taking into account the underactuated part of the dynamics.

$$\min_{\delta_{\ddot{\mathbf{q}}}, \delta_{\mathbf{F}}} \delta_{\ddot{\mathbf{q}}}^T Q_1 \delta_{\ddot{\mathbf{q}}} + \delta_{\mathbf{F}}^T Q_2 \delta_{\mathbf{F}} \tag{9.27}$$

$$\text{s.t. } \mathbf{F}_{MPC} + \delta_{\mathbf{F}} \in \mathcal{K} \tag{9.28}$$

$$S(\mathbf{M}(\begin{bmatrix} \ddot{\mathbf{q}}_{IK,u} \\ \ddot{\mathbf{q}}_{IK,a} \end{bmatrix} + \begin{bmatrix} \delta_{\ddot{\mathbf{q}}} \\ 0 \end{bmatrix}) + \mathbf{g} + \mathbf{C}) = S\mathbf{J}_c^T(\mathbf{F}_{MPC} + \delta_{\mathbf{F}}) \tag{9.29}$$

with subscripts $_u$ and $_a$ refering to the underactuated and actuated parts respectively, M the generalized mass matrix, $\mathbf{g}$ the gravitational force, $\mathbf{C}$ the nonlinear forces, S the matrix selecting the underactuated dynamics, $\mathbf{J}_c$ the augmented contact Jacobian and $\mathcal{K}$ the friction cone linearized to the first order. We can separate the variables between the unactuated part (base) and the actuated joints:

$$\mathbf{M} = \begin{bmatrix} \mathbf{N} & \mathbf{M}_u \\ \mathbf{M}_u^T & \mathbf{M}_a \end{bmatrix}_{18\times18} \quad \mathbf{J}_c^T = \begin{bmatrix} \mathbf{J}_u \\ \mathbf{J}_a \end{bmatrix}_{18\times12} \tag{9.30}$$

Similarly to the IK, we can use a more computationally efficient way to compute feedforward torques from the reaction forces $\mathbf{F}_{MPC}$ outputted by the MPC and the accelerations $\ddot{\mathbf{q}}_{IK}$. It is possible to transform the quadratic programming problem into an equivalent problem faster to solve as it amounts to a few matrix inversions. In a way, this is similar to how one of the constraints was integrated into the cost function in Section 9.1 through a change of optimization variable. Using the underactuated part of (9.29) $\delta_{\ddot{\mathbf{q}}}$ can be expressed as an affine function of $\delta_{\mathbf{F}}$:

$$\delta_{\ddot{\mathbf{q}}} = \Lambda\, \delta_{\mathbf{F}} + \gamma \tag{9.31}$$

$$\Lambda = \mathbf{N}^{-1}\mathbf{J}_u \tag{9.32}$$

$$\gamma = \mathbf{N}^{-1}\left(\mathbf{J}_u\mathbf{F}_{MPC} - (\mathbf{N}\ddot{\mathbf{q}}_{IK,u} + \mathbf{M}_u\ddot{\mathbf{q}}_{IK,a} + \mathbf{g}_u + \mathbf{C}_u)\right) \tag{9.33}$$

$\mathbf{N}\ddot{\mathbf{q}}_{IK,u} + \mathbf{M}_u\ddot{\mathbf{q}}_{IK,a} + \mathbf{g}_u + \mathbf{C}_u$ can be computed by a cheap Recursive Newton-Euler Algorithm (RNEA) evaluation [CVM+19]. $\delta_{\ddot{\mathbf{q}}}$ is then replaced in (9.27) using (9.31):

$$\min_{\delta_{\mathbf{F}}} \delta_{\mathbf{F}}^T \mathbf{U} \delta_{\mathbf{F}} + 2\delta_{\mathbf{F}}^T \mathbf{v} \tag{9.34}$$

$$\text{s.t. } \mathbf{F}_{MPC} + \delta_{\mathbf{F}} \in \mathcal{K} \tag{9.35}$$

with:

$$\mathbf{U} = \Lambda^T Q_1 \Lambda + Q_2 \tag{9.36}$$

$$\mathbf{v} = \Lambda^T Q_1 \gamma \tag{9.37}$$

$\mathbf{F}_{MPC} + \delta_{\mathbf{F}} \in \mathcal{K}$ is equivalent to $\mathbf{F}_{MPC} + \delta_{\mathbf{F}} = \begin{bmatrix} (G_1\lambda_1)^T & (G_2\lambda_2)^T & (G_3\lambda_3)^T & (G_4\lambda_4)^T \end{bmatrix}^T$ where $\forall k \in \{1..4\}, \lambda_k \in \mathbb{R}^{4,+}$ and G_k are the edges of the linearized friction cone of the

k-th foot with friction coefficient μ. The final QP problem is thus of the form:

$$\min_{\lambda} \frac{1}{2}\lambda^T G^T \mathbf{U}\, G\lambda + (G^T\mathbf{v} - G^T\mathbf{U}\, \mathbf{F}_{MPC})^T\lambda \tag{9.38}$$

$$\text{s.t. } \forall k \in \{1..4\},\ \forall i \in \{1..4\},\ \lambda_{k,i} \geq 0 \tag{9.39}$$

$$G_k\lambda_k = \begin{bmatrix} \mu & \mu & -\mu & -\mu \\ \mu & -\mu & \mu & -\mu \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \lambda_{k,1} \\ \lambda_{k,2} \\ \lambda_{k,3} \\ \lambda_{k,4} \end{bmatrix} \tag{9.40}$$

$$\lambda = \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \lambda_4 \end{bmatrix} \quad G = \begin{bmatrix} G_1 & 0 & 0 & 0 \\ 0 & G_2 & 0 & 0 \\ 0 & 0 & G_3 & 0 \\ 0 & 0 & 0 & G_4 \end{bmatrix} \tag{9.41}$$

This last problem is a box-QP [Dos97; Ye97]. Box constraints are easier to handle than generic linear constraints, and lead to simpler and more efficient implementations. In particular, box-QP algorithms are straightforward to implement, do not imply computing the Lagrange multipliers, and have a much better worst-case performance than regular QP (linear versus exponential) [NW06]. Once $\delta_{\mathbf{F}}$ has been determined, $\delta_{\ddot{\mathbf{q}}}$ can be deduced from (9.31). The multi-body dynamics can be written as:

$$\begin{bmatrix} \tau_u \\ \tau_a \end{bmatrix} = \mathbf{M}(\ddot{\mathbf{q}}_{IK} + \begin{bmatrix} \delta_{\ddot{\mathbf{q}}} \\ 0 \end{bmatrix}) + \mathbf{g} + \mathbf{C} - \mathbf{J}_c^T(\mathbf{F}_{MPC} + \delta_{\mathbf{F}}) \tag{9.42}$$

Since we only want joint torques, only τ_a has to be computed so (9.42) is reduced to:

$$\tau_a = \mathbf{M}_u^T(\ddot{\mathbf{q}}_{IK,u} + \delta_{\ddot{\mathbf{q}}}) + \mathbf{M}_a\ddot{\mathbf{q}}_{IK,a} + \mathbf{g}_a + \mathbf{C}_a - \mathbf{J}_a^T(\mathbf{F}_{MPC} + \delta_{\mathbf{F}}) \tag{9.43}$$

These joint torques τ_a are then sent to the low-level impedance controller. Compared to the previous approach based on inverse dynamics, this one was successfully deployed on the robot. It allowed the robot to trot around at low speed but failed to reach velocities higher than 0.5 m/s.

9.3 Improving the IK + QP architecture

9.3.1 Removing position feedback

Both in Section 9.1 and Section 9.2, the tasks of the whole-body control included the estimated position of the base in world frame, as part of the feedback scheme through the tracking task of the base with a position error multiplied by a gain in (9.23), and to a lesser extent to the feet tracking whose position is related to the base one. The footstep planner and swinging feet trajectory generator of Chapter 7 also involved this estimated position since future footstep locations depends on the position of the base at the time of the decision.

Without any exteroceptive sensors, the body position in world frame is not an observable quantity. It can still be estimated through a mix of velocity integration and forward geometry to avoid drift coming from pure integration, yet it will still drift slowly due to imperfect sensor measurements and forward geometry: the immobile contact point assumption does not take into account slipping and rolling around the foot radius. As a

result, this position estimation will have poor accuracy compared to other more accessible quantities like the velocity estimation. Because of that, leaving this position in the control scheme may deteriorate the quality of the behavior due to all the undesired estimation noise it is bringing with it when, at second glance, it should not even be necessary since the overall goal of the architecture is to track a desired base velocity. That is why the "ideal" world frame was introduced in Section 5.5 with a robot moving in the world at its reference velocity. It allows to keep using a world frame for convenience for the footstep planner and the foot trajectory generator, without introducing undesired estimation noise. Base position is removed as well from the tasks of the whole-body control, to leave only the base velocity component in the base tracking task, as illustrated in Fig. 9.3.

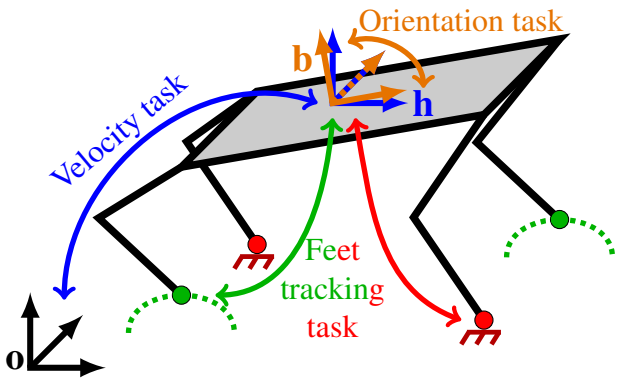


Figure 9.3: Tasks and frames of the WBC without base position feedback.

9.3.2 Removing joint feedback

To further continue this redesign of the feedback mechanisms in the control architecture, we can now focus on low-level quantities: joint positions and velocities. As pointed out at the end of Section 9.1, for inverse dynamics, we could either use a direct feedback from the robot by providing the WBC with the joint positions and velocities measured by the encoders, or adopt a hybrid approach with feedback only on the underactuated part of the state. For the actuated part, the WBC would then loops on itself by integrating twice the reference joint accelerations to get joint positions and velocities for impedance control and for obtaining the robot state at its next iteration. The inverse kinematics + QP scheme can benefit from a similar approach by discarding the encoders measurements as well, relying only on command quantities for joint positions and velocities. To this end, there is no need to integrate twice the reference joint accelerations as the IK directly provides target positions (9.26) and velocities (9.25) that can be used for the next iteration.

Relying on this kind of ideal feedback by reusing the command as the state of the robot at the next iteration has the advantage of removing a source of noise in the scheme. The noisy joint velocities that are estimated through finite difference of the increments of joint encoders could degrade the feet velocity estimation that is used for the feet tracking task of the WBC. A small noise at the hip and knee level is amplified when computing the velocity at the tip of the leg. Furthermore, this approach avoids the risk of instability brought by a double feedback loop that tries to track the same reference with two controllers at two different frequencies. Typically, the WBC runs at 1 kHz while the impedance controller runs at 10 kHz. If not properly controlled, the effect of this double feedback could be more prominent due to the fast dynamics of Solo's almost direct-drive actuators. A thorough

frequency study would be needed to better ascertain the limits of this phenomenon in the case of Solo-12.

A drawback that has to be noted is that by removing this feedback, the impedance controller is now the only one ensuring the tracking of joint state since the torques outputted by the WBC are no longer computed with the measured position of actuators. So these torques will not necessarily act to reduce the tracking error even if it gets bigger.

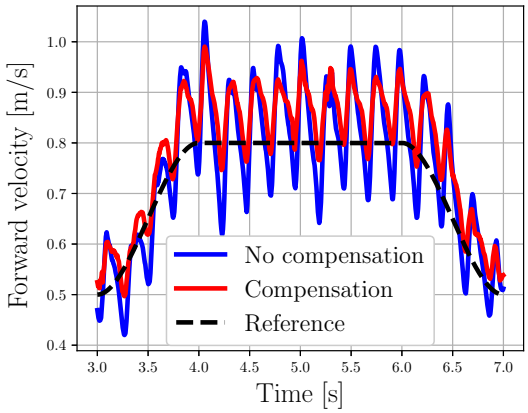
9.3.3 Compensation term for contact forces

As the MPC works with a centroidal model of the robot, it does not take into account the inertia effects that result from leg movements nor the nonlinear effects. Hence the contact forces computed by the MPC will not compensate or benefit from the forces related to these effects to stabilize the base and follow the reference velocity. As a result, while the left side of (9.29) includes the inertia of the base, the inertia of the joints, the nonlinear effects and the gravitational force, the MPC forces on the right side only take into account the inertia of the base and the gravitational force. If the inertia of the joints and the nonlinear effects are non-negligible, as it seems to be the case when the upper-leg joints are in motion at high speed, the QP will not work around an equilibrium point because the left and right sides of (9.29) may be widely different (requiring substantial $\delta_{\ddot{\mathbf{q}}}$ or $\delta_{\mathbf{F}}$ to respect the constraint).

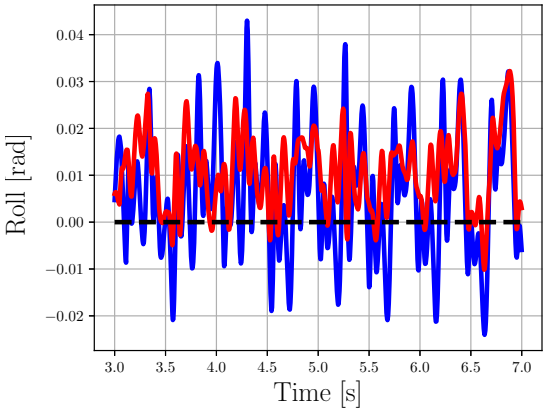
To limit this effect, we introduced a compensating term $\mathbf{F}_{comp}$ that is added to the contact forces of the MPC to diminish the offset between both side of the equation, so that the QP starts working closer to the equilibrium and $(\delta_{\ddot{\mathbf{q}}},\ \delta_{\mathbf{F}})$ are lower. Instead of considering $\mathbf{F}_{MPC} + \delta_{\mathbf{F}}$, we consider $\mathbf{F}_{MPC} + \mathbf{F}_{comp} + \delta_{\mathbf{F}}$ with:

$$\mathbf{F}_{comp} = (\mathbf{J}_u^T)^\dagger \left(\mathbf{C}_u + \mathbf{M}_u \ddot{\mathbf{q}}_{IK,a}\right) \tag{9.44}$$

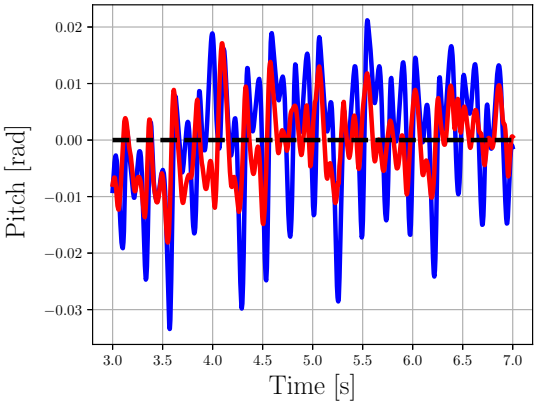
where $\mathbf{M}_u \ddot{\mathbf{q}}_{IK,a}$ accounts for the effect of joints inertia on the base dynamics. $\{\cdot\}^\dagger$ denotes the pseudo-inverse operator. As seen in Fig. 9.4, adding this compensation term resulted in a reduction of the oscillation of the linear velocity by a factor of roughly 2.



(a) Forward velocity oscillations



(b) Roll oscillations



(c) Pitch oscillations

Figure 9.4: The compensation of contact forces reduces oscillations in forward velocity, roll and pitch.

9.4 Low-level impedance controller

The low-level impedance controller is the last interface between our control architecture and the motor boards that drive the joints with a high-frequency current loop. It accepts target torques, joint positions and velocities as inputs and returns a final desired torques for the actuators.

$$\tau^\star_{mot} = \tau^\star + K_p(\mathbf{q}^\star_a - \mathbf{q}_a) + K_d(\dot{\mathbf{q}}^\star_a - \dot{\mathbf{q}}_a) \tag{9.45}$$

These torques are then converted into current references using the motor torque constant $K_\tau = 0.025$ Nm/A and the reduction ratio $R = 9$:

$$I^\star_{mot} = \frac{\tau^\star_{mot}}{K_\tau \, R} \tag{9.46}$$

Using only target positions and velocities in (9.45) would force to have high K_p and K_d gains to move properly, as the torque commands would only be generated through tracking error. This would be akin to position control, with a very stiff behavior of the joints. It is for instance the case in [Kum+21] with the A1 quadruped for which a K_p value up to 50 is used, as their neural networks only provides position targets. On the opposite, torque control with $\tau^\star$ alone would quickly diverge as these torques are computed with

a model that does not perfectly fit the reality, so the robot would not move as expected due to torque constants that are not exactly the same for all motors, the elasticy of the mechanical structure, unmodeled phenomena, and so on. In the end, combining torque commands with target positions and velocities is beneficial to the locomotion for two reasons. Compared to a pure torque command, taking into account position and velocity errors helps correcting model inaccuracies and unexpected events. It also enables the use of lower gains because $\tau^\star$ provides most of the torque amount even without any tracking error. As a result, we can achieve a more compliant behavior of the joints to dampen impacts with the ground. In practice, we use $K_p = 3$ Nm/rad and $K_d = 0.3$ Nm/(rad/s) for all joints.

9.5 Conclusion

In this chapter, we have described how the whole-body controller has evolved over the duration of this thesis. Our initial implementation consisted of a Task Space Inverse Dynamics that allowed to perform task-oriented optimization-based inverse dynamics. However, instability issues during real-world deployment led us to switch to a combination of inverse kinematics with a quadratic program. We then presented several improvements to this architecture, such as removing base position and joint feedbacks, and adding a compensation term to take into account inertia effects due to leg movements. Those changes resulted in a reduction of base oscillations and thus increased the velocity the robot could reach before falling. We also briefly evoked the low-level impedance controller which computes the final torque commands sent to the motors, based on the difference between the desired and current joint positions and velocities, and the feedforward torques of the whole-body controller.

The next part of the thesis presents various implementation results in which the control architecture described so far was deployed.

Part II

Implementation Results

Chapter 10

Validation of the baseline architecture

Contents

10.1 Introduction

This chapter presents a validation of the Inverse Kinematics + Quadratic Programming scheme that was presented in Section 9.2. It replaces the Inverse Dynamics that we initially tried to deploy on the quadruped, but did not succeed, as explained at the end of Section 9.1. At that point, no controller had yet been successfully ran to move around with a real Solo quadruped since our Inverse Dynamics failed when we went from simulation to reality. As a secondary goal, if the deployment is successful, we want to assess the effectiveness of the IK + QP scheme in terms of reachable velocities and robustness to disturbances.

We will first present preliminary results obtained in simulation using a full model of the quadruped provided by the Open Dynamic Robot Initiative [ODR] based on the expected dimensions and weights of the various components of the Solo-12 quadruped. This model is shown in Fig. 10.1. Then, we will describe the experimental results obtained by deploying the controller the Solo-12 robot built at LAAS while choosing its reference velocity with a joystick and disturbing the robot by pushing it with a stick.

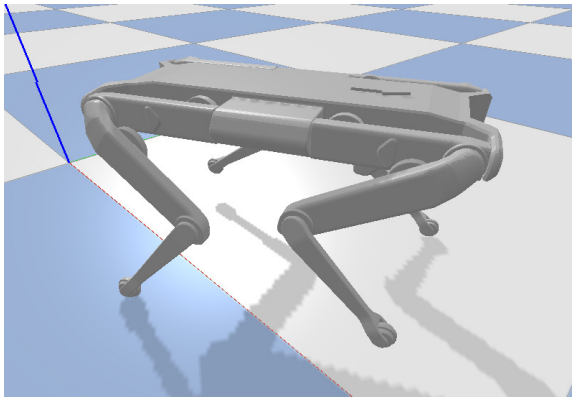


Figure 10.1: Solo-12 walking in simulation on a flat ground. Each link has its own mass and inertia, computed beforehand based on the CAD model of the assembly. The visual mesh is also used for collision checks by default. It would potentially be possible to fasten the simulation by using collision capsules instead if it ever proved to be a problem.

10.2 Simulation setup

The control framework was implemented in Python for ease of use and prototyping. Achieving real time performance was made possible by exploiting NumPy vectorial capabilities, using libraries that provides Python bindings for their C implementation and compiling computation intensive parts (coded in C++ with Python bindings).

The main control loop (footstep planner, foot trajectory generators, whole-body control and state estimator) runs at 500 Hz on a i7-7700 CPU (3.60 GHz) while the MPC runs at 50 Hz in a parallel process and communicates with the main loop through a shared memory. Low-level kinematics and dynamics computation were performed using the Pinocchio library that provides standard rigid body operations and algorithms for poly-articulated systems [CVM+19], [Car+19]. The MPC variant that is used here is the linear convex quadratic programming one that was presented in Section 8.2. It exploits the sparcity of its constraint matrices using the OSQP solver [Ste+20]. The simulation environment was set up using PyBullet which offers contacts simulation and a Python API to send torques and retrieve relevant data [CB20].

Before testing the proposed control scheme on the real hardware we assessed its stabilization capabilities in a simulated environment for a walking trot gait with period set to $0.32s$. The first scenario consists of a straight walk followed by a turn on a flat ground with an external perturbation force of $+5$ N along X at $t = 9s$ and another one of $+5$ N along Y at $t = 11s$. The perturbations are directly applied at the center of the base in the simulator. The second scenario places the robot on a rough terrain: the ground is full of small bumps whose height is random (uniform distribution between 0 and 5 cm), as shown in Fig. 10.2. In both scenarios the reference velocity is initially zero. During the first phase of the motion, where the robot is moving forwards, its velocity is slowly increased, up to 1.5 m/s in the first scenario, and up to 1 m/s in the second one. Then, in the second phase, the angular velocity is increased up to 0.4 rad/s. These reference velocity profiles are represented by dotted orange lines in Fig. 10.4b and Fig. 10.5b.

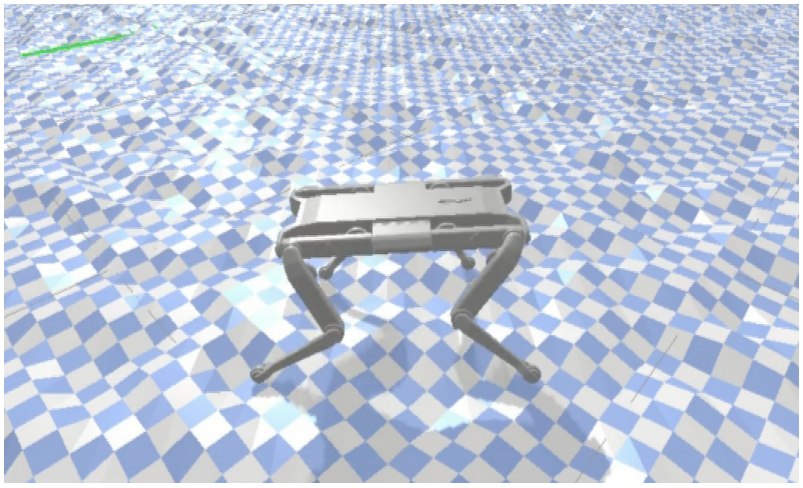


Figure 10.2: Solo-12 quadruped walking in simulation on a rough terrain.

10.3 Simulation results

Simulation results for both scenarios are shown in Fig. 10.4 and Fig. 10.5 respectively. In scenario 1 the robot reaches its reference forward velocity of 1.5 m/s and returns to its nominal behaviour after both external perturbations. As the robot moves faster it gets increasingly tilted in pitch despite a reference angle at $0°$. This may be due to a compromise with other quantities in the cost function of the MPC which leads to a minima with a non-zero pitch angle in average. Scenario 2 highlights a limit of the proposed control scheme: the ground is supposed to be flat so feet can slip during stance phase when landing on an unexpected tilted surface such as the sides of bumps. Since the controller expects to work in nominal conditions (flat ground), the contact forces it wants to apply can be out of the friction cone of the actual tilted surface. Without knowledge of the environment a possible solution would be to continuously check for slipping during stance phases and react accordingly if such an event is detected, as done in [Blo+13].

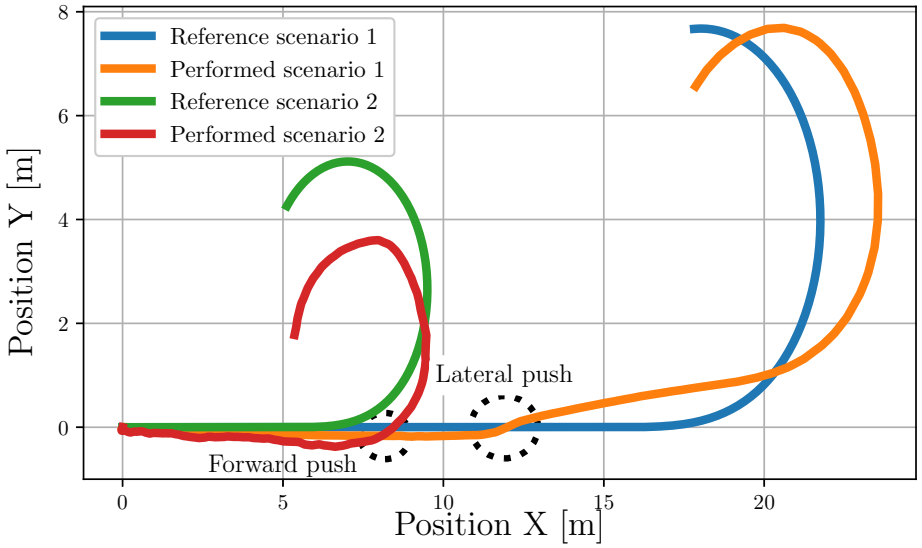
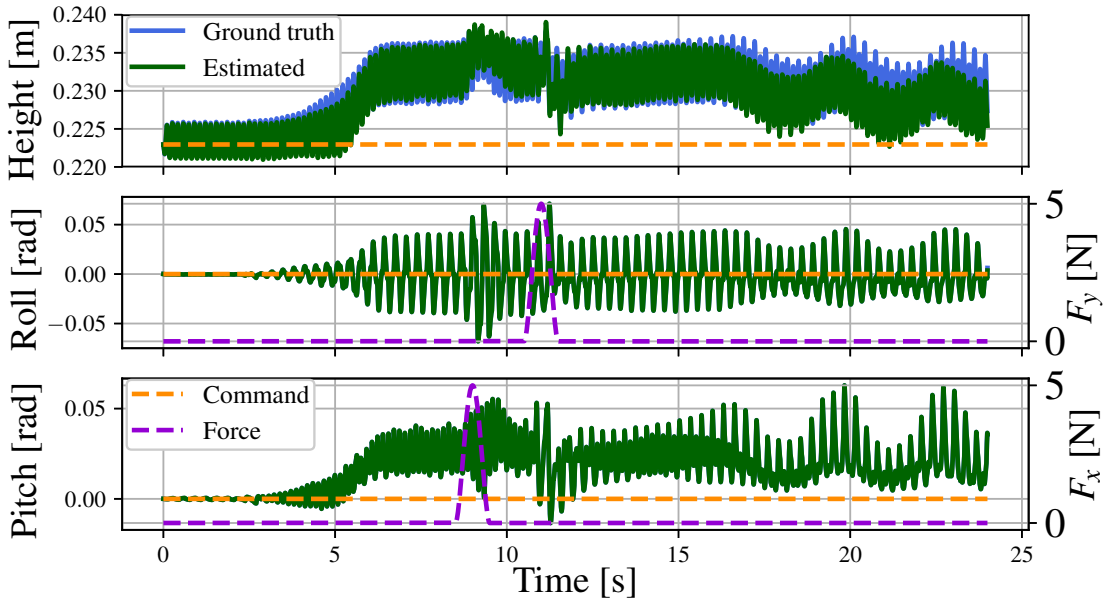
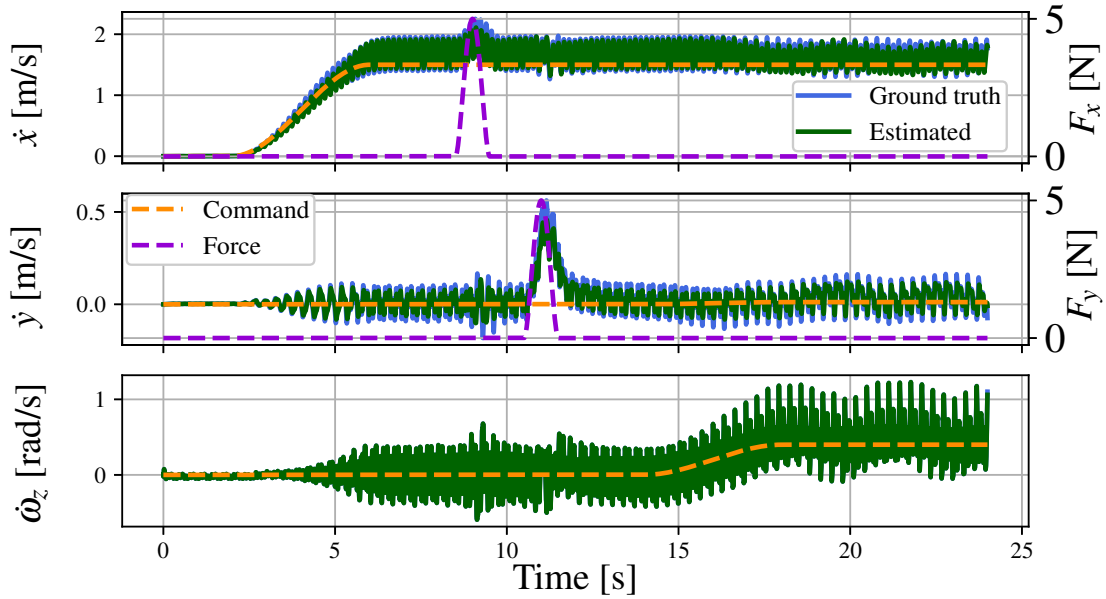


Figure 10.3: Reference trajectories that the robot would have done if it had perfectly followed the reference velocity at all time in both scenaros, compared to its actual trajectories. There is no feedback in position in the world so a drift is expected due to imperfect tracking and butterfly effect as times goes by.

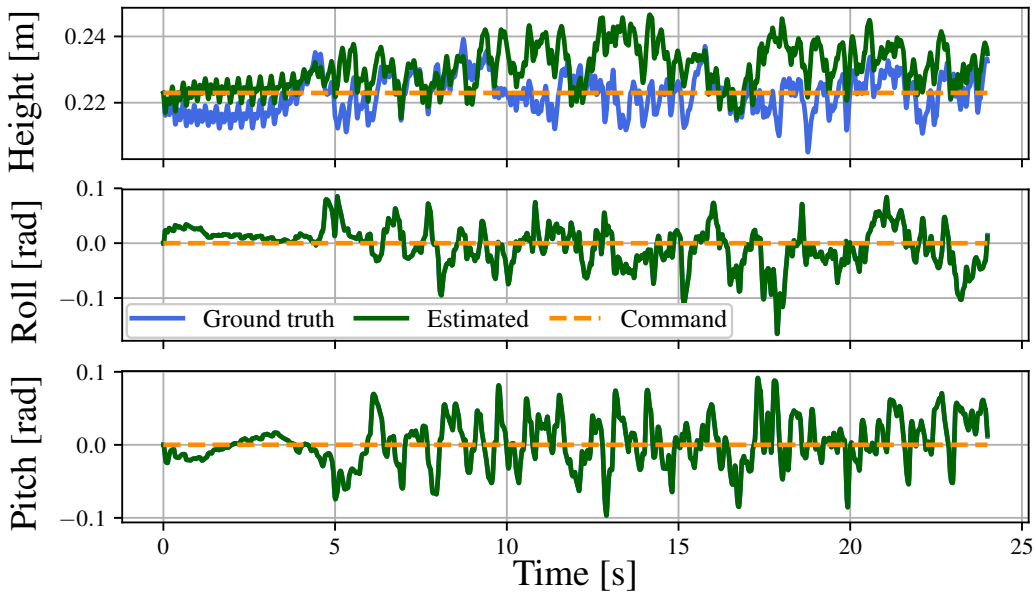


(a) Trunk height and orientation in scenario 1

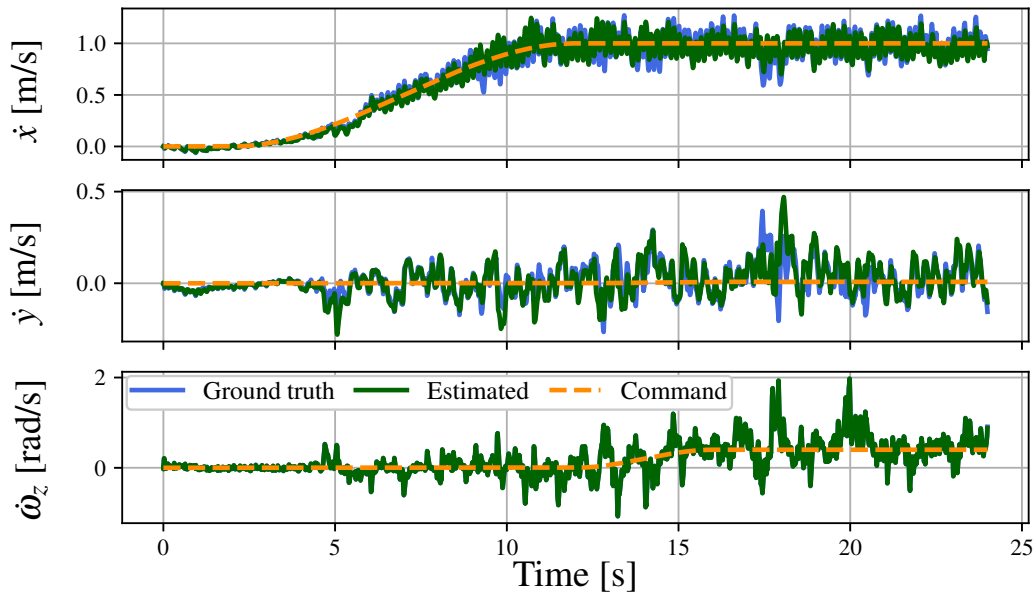


(b) Trunk velocity in scenario 1

Figure 10.4: In scenario 1, the quadruped recovers from both perturbations occurring at 9s and 11s and reaches a maximum lateral velocity of +0.56 m/s at t =11.15s. It returns to its nominal behaviour in around 0.5s.



(a) Trunk height and orientation in scenario 2



(b) Trunk velocity in scenario 2

Figure 10.5: In scenario 2, the quality of the estimation is worse than in scenario 1 as feet sometimes slip on bumps sides which breaks our immobile contact assumption.

10.4 Experimental setup

Experiments were performed indoors on a flat ground. Small rubber bands were glued on the robot feet to improve friction with the plastic flooring. To be conservative, we used a friction coefficient of 0.9 with the actual coefficient assessed around 1.0. Ground truth was retrieved thanks to a motion capture system consisting of a set of 20 infrared cameras spread around the workspace that track at 200 Hz 9 reflective markers installed on top of the robot base. Cut frequencies f_c^v and f_c^p of the velocity and position complementary filters were set to 3 Hz and 0.4 Hz respectively. The MPC weights chosen for position, orientation, linear velocity and angular velocity errors are respectively $[2.0, 2.0, 20.0, 0.25, 0.25, 10.0, 0.2, 0.2, 0.2, 0.0, 0.0, 0.3]$. They are the same as the ones used for Mini Cheetah [MIT] and worked out of the box for us. The weights for contact force regularization were set to 1×10^{-5} for all components. To perform inverse kinematics we used $K_p = 100$ and $K_d = 2\sqrt{K_p} = 20$ for all tasks. In the initial formulation of the QP problem (9.27) we used 0.1 and 1.0 for the weights of the acceleration and contact force relaxation variables (Q_1 and Q_2 respectively). For the on-board impedance controller, all joints shared the same proportional feedback control gains of 6 Nm/rad and 0.2 Nm/(rad/s) respectively. The performed gait was a trot with a period of $0.32s$. During the experiments the robot was powered via an external power supply. Communications with the robot (sensors data retrieval and command sending) were done using an Ethernet link to the control desktop computer.

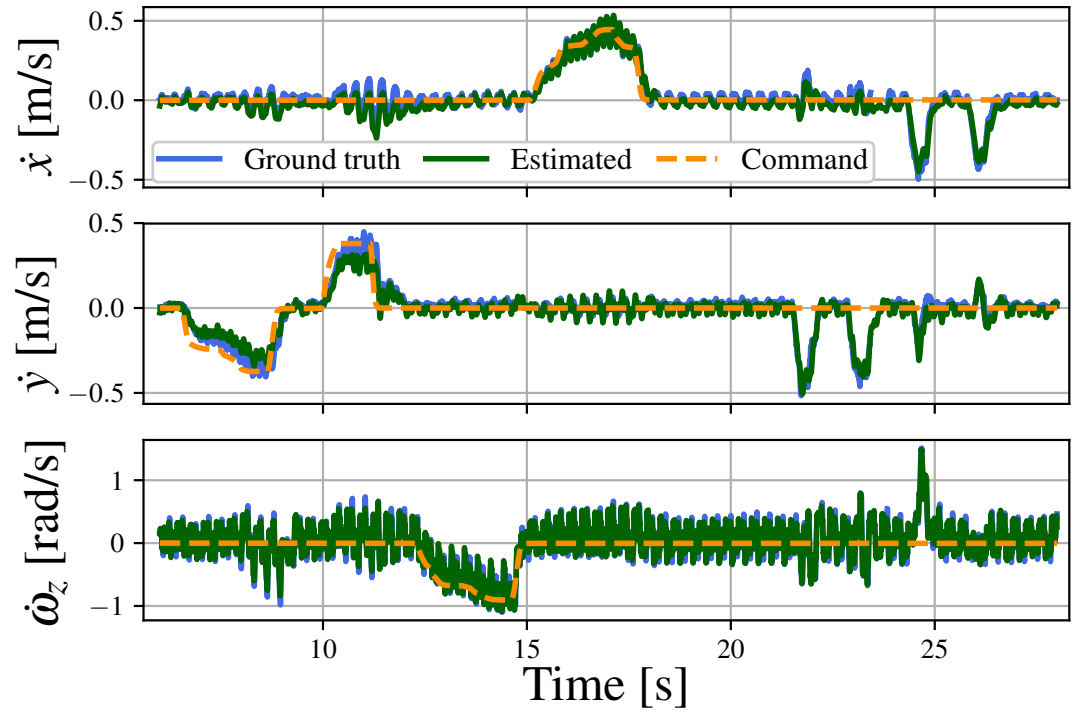
10.5 Experimental results

Fig. 10.6 presents the results of an experiment during which the Solo-12 quadruped is controlled by a user with a joystick (until $t = 18s$). The robot performs first a lateral walk to the right then to the left, a clockwise rotation along the vertical axis and finally a short walk forwards. The robot is then ordered to stay immobile (zero velocity command) while it is being pushed sideways by the user (after $t = 18s$).

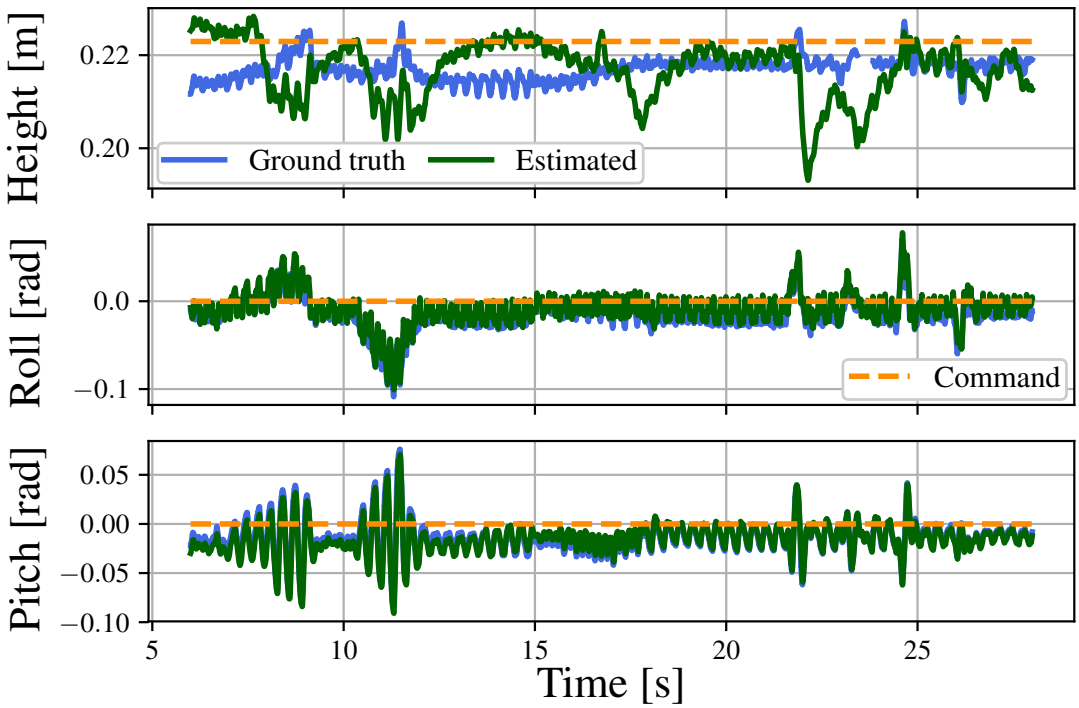
Thanks to the motion capture ground truth we can assess both the quality of estimation and the reference tracking. The quality of the height estimation seems greatly influenced by the variation of others quantities both in orientation and velocity. This can be explained by slight slippings of feet which produce an undesired swaying motion in pitch. Estimation of other quantities seems robust to perturbations except during lateral walks at $t = 8s$ and $t = 11s$. The velocity reference given by the user is correctly followed when the quadruped does not have to face external perturbations.

In the second half of the experiment the robot manages to recover from the four sideways perturbations that it receives at $t = 22s$, $t = 23s$, $t = 24.5s$ and $t = 26.5s$. The third push also transmits a rotating motion to the robot. In all cases it counters the undesired velocity and returns to a nominal behavior in less than half a second. A video of locomotion and push recovery is available online[1].

[1] https://gepettoweb.laas.fr/articles/leziart2021.html



(a) Reference, estimated and motion-captured velocities of the base.



(b) Reference, estimated and motion-captured height and orientation of the trunk.

Figure 10.6: State trajectories when controlling the reference velocity sent to the robot with a joystick, then pushing it as it tries to remain in place.

10.6 Conclusion

This chapter reports our first experimental results, validating the various control blocks that have been implemented so far on a real robot after prior confirmation in simulation. The set of heuristics used in the footstep planner reacts accordingly to changes of the reference and measured base velocities. The rest of the implementation reproduces what is now a well-known control architecture for quadruped robots, with a combination of centroidal predictive control and instantaneous whole-body control, as previously described. These experiments also confirm that performing inverse kinematics by stacking compatible WBC tasks in a single Jacobian that is then inverted is a valid way to avoid a hierarchical scheme with successive projections in null spaces of tasks with higher priorities. We have been able to obtain a robust trotting gait at low velocity on a flat ground. With this controller, the robot can walk on small obstacles on the ground without falling despite the level ground assumption, as well as handle reasonable pushes with a stick.

However, the maximum velocity the robot can reach with this controller is still far from the state of the art for quadrupeds of that size, which can go above 3 m/s such as in [Kim+07] with a similar control architecture or [Ji+22] with neural networks, both deployed on the Mini-Cheetah platform. For what we have now, with base velocities lower than 0.5 m/s, the full capabilities of Solo-12 are far from being reached yet. For comparison, a standard human walk is around 1.4 m/s. These experiments were primarily done to confirm the soundness of the approach before building further upon it. As such, what has been shown in this chapter will act as a baseline for future developments. In the next chapter, we will show how the improvements presented in Section 9.3, that were not implemented in Chapter 10, allow both to reach higher velocities with the linear MPC and to deploy the other MPC variants of Section 8.4 and Section 8.5.

Chapter 11

Comparison of Model Predictive Controllers

Contents

11.1 Introduction

In this chapter, we compare the performances of the three MPC variants that were presented in Chapter 8, whose real world deployment relies on our improved control architecture. The MPC first variant presented in Section 8.2 is based on the assumptions of small pitch and roll angles and angular velocities. The trajectory of the center of mass is also assumed to follow its reference perfectly. Foot placements are not part of the optimization problem. They are obtained beforehand, based on the heuristics described in Chapter 7. A first assumption that can be lifted is to consider the predicted trajectory of the center of mass, instead of the reference one, to compute the lever arms with the contact point on the ground, as described in Section 8.4. The problem is then no more linear, due to the cross product of the lever arms involving an optimization variable (predicted trajectory of the CoM). Furthermore, another assumption can be lifted by considering the location of footsteps as optimization variables of the optimal control problem, instead of taking them as granted using the heuristics of the footstep planner, as described in Section 8.5. The linear centroidal MPC, tested in the previous chapter and solved with convex linear quadratic programming, has been rewritten in Section 8.3 to be solved with differential dynamic programming so that all variants use the same solver.

While the general control architecture is the same as in the previous chapter, efforts have been made to improve the nominal behavior in terms of stability and robustness

by using the developments presented in Section 9.3. Formerly, base oscillations during motion hampered real-world deployments by impacting the consistency of desired contact forces and footsteps locations over a gait period. These oscillations limited the maximum velocity the robot could reach to less than 0.5 m/s and made unstable the variant that optimizes footstep locations. Reducing their amplitude was especially helpful for the stability of results.

To sum up, the overall objective is to evaluate whether the nonlinear centroidal MPC scheme optimizing footholds performs better than a more modular scheme that uses a heuristic for foot placement and relies on a simplified model. Depending on how well the heuristic-free MPC can perform on the real robot compared to variants that rely on hand-defined heuristics, it could be a way to relieve the need for expert knowledge at the cost of losing modularity by centralizing decisions in a single control module.

11.2 Experimental setup

Experiments were first performed indoors on a flat carpet-like material. Ground truth was retrieved thanks to a motion capture system comprising 20 infrared cameras spread around the workspace that track 13 reflective markers, installed on top of the robot base, at 200 Hz. During the experiments the robot was powered via an external power supply. Communications with the robot (sensors data retrieval and command sending) were done using an Ethernet link to the control desktop computer. Out of the prototyping phase, all control blocks were converted from Python to C++ for computational efficiency, except for the main loop which calls them, which allowed it to go from 500 Hz to 1 kHz. MPCs were implemented using Crocoddyl [Mas+20] as in [Cor+21]. For real-time purpose, they all run in a parallel process called at 50 Hz. Desired contact forces are retrieved after a delay due to the solving time ($\approx$ 2 ms for the linear and nonlinear MPCs, $\approx$ 6 ms for the footsteps MPC). The QP problem in the WBC is solved with OSQP [Ste+20]. The MPC weights chosen for position, orientation, linear velocity and angular velocity errors are respectively $[2.0, 2.0, 10, 0.25, 0.25, 10, 0.2, 0.2, 0.2, 0, 0, 0.3]$. The weights for contact force regularization were set to 5×10^{-5} for all components. To perform inverse kinematics we used $K_p = 10$, $K_d = 2\sqrt{K_p} = 6.3$ and a weight of 1 for all tasks. For the QP problem (9.27) we used $Q_1 = 0.1I_6$ and $Q_2 = 10I_{12}$ for the weights of the acceleration and contact force relaxation variables. For the on-board impedance controller, all joints shared the same proportional and derivative feedback control gains of 3 Nm/rad and 0.3 Nm/(rad/s) respectively. The performed gait was a trot with a period of $0.48s$ as it proved to be a good trade-off for evaluating the MPC performances. A faster gait would be naturally more stable due to the faster switching between diagonally opposed pairs of contacts, thus making the MPC role less crucial. A slower gait proved to be harder to stabilize because the base can tilt too much during a single swing phase, which can harldy be corrected (with two contact points we can only act along an axis). Deployment of the last MPC variant on the robot was made possible thanks to the reduction of velocity oscillations that resulted from the compensating contact forces described in Section 9.3. They improved the consistency of the footsteps optimization which was previously diverging. As the estimated velocity of the base influences footsteps positions over the prediction horizon, the smaller the oscillations, the less these positions are modified over the span of a gait period.

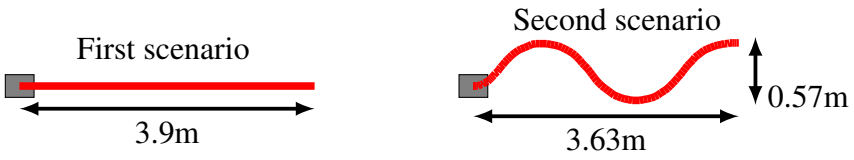


Figure 11.1: Top view of the robot trajectory in the two considered scenarios obtained by integration of the reference velocity.

11.3 Experimental results

11.3.1 Indoor tests

Performances are compared for the two scenarios shown in Fig. 11.1. During the first scenario the quadruped goes straight forwards. The velocity command is slowly increased during 4 seconds, stays at 0.8 m/s during 2 seconds, then goes back down to 0 m/s in 4 seconds. During the second scenario the quadruped performs several turns in a row. The velocity command goes up to 0.5 m/s forwards with ± 0.8 rad/s along the vertical axis to produce a S-shaped trajectory. Polynomial interpolation generates command profiles that are continuous both in velocity and acceleration. Motion capture data is reported in Fig. 11.3 and Fig. 11.4. Linear and non-linear variants lead to very close behaviors over the whole movement in both scenarios. Differences with the variant that optimizes footsteps location are noticeable but the values and amplitudes of errors and oscillations with respect to the references are roughly the same. The oscillations of the forward velocity around its reference have a maximum amplitude of around 0.15 m/s during the high-velocity phase of the first scenario. Lateral velocity tracking seems stable during both tests with an amplitude of roughly 0.1 m/s, even during the turn in scenario 2, and with a shift of the average value toward the outside of the turns. For the considered angular velocities (up to ± 0.8 rad/s), turning does not impact forward velocity tracking in a noticeable way. Joint torques estimated through current measurements peak at 2.1 Nm during the high-velocity phase. Actuators can deliver up to 2.5 Nm at 12 A, so hardware capabilities are not fully exploited yet [Gri+20]. There is still way to improve the control architecture and reach higher velocities.

11.3.2 Outdoor test

In complement to indoor locomotion, the different controllers were tested on wet grass, as shown in Fig. 11.2. The quadruped was powered by 2 on-board batteries, one at the front, one at the back, that were not included in the model. Their weight of 100 g acted as an additional perturbation. The robot managed to follow the velocity profile up to 0.8 m/s without falling with each MPC variant. The lack of contact detection on this wet and bumpy surface resulted in numerous foot slipping when the robot tried to apply forces with a foot that had not properly landed. As seen on Fig. 11.5, the most notable slipping occurs at 6.6 s for the linear MPC, with peak lateral and angular velocities of -0.5 m/s and 1.3 rad/s respectively. The terrain also leads to larger oscillations around the reference values compared to indoors, especially during the high-velocity phase. Currently, neither the footsteps heuristics of the first two variants nor the footsteps optimization of the last one are design for handling a non-flat ground (slope, stairs, bumps). They could be extended either by using privileged information about the environment or by implementing

some sort of online slope detection. Friction cone constraints would have to be refactored as well to match non-horizontal surfaces, hence why performances were only compared on a flat ground. Moreover, applying a repeatable disturbance to a walking quadruped is not trivial in practice, contrary to simulation. Slight differences in direction, strength or duration of the push can lead to widely different behaviors, especially depending on the gait status. If it happens at the beginning of a swing phase, the controller can directly react and adjust footsteps positions accordingly. However, if it occurs near touchdown, then it is too late to widely modify contact location. For these reasons, robustness to disturbances was not compared during our experiments. Videos of both indoor and outdoor tests is available online[1].

Figure 11.2: Trotting gait on wet grass. Connection with the robot was done by Wifi with the same computer than we used indoors. Previous experiments showed that packet loss did not hamper the behavior for distance under 25 m.

11.4 Discussion

As explained in Chapter 9, our first attempt in designing the whole-body control block of this architecture was to implement an inverse dynamics (ID) that could directly handle both position and velocity for the base and the feet, and ground reaction forces, as in simulation [Cor+21]. However, using this approach we did not achieve a stable behavior on the real robot. This could be due to the sensitivity of ID to mismatches between model and hardware. Moreover, Solo-12 is a lightweight quadrupedal robot with almost direct-drive actuators. As studied in [SHV+06], coupling among links is more significant with the fast dynamics of such actuators. So, as we did not manage to run the ID faster than 500 Hz, it might have been insufficient to avoid instability. For this reason, we decided to use instead the approach proposed in [Kim+19], that includes an IK to compute the joint accelerations to perform the position and velocity tasks, described in Section 9.2, based on which a QP problem is solved to find a compromise between tracking these joint accelerations and taking into account MPC decisions and the equation of dynamics (9.27)-(9.29). This QP problem has to balance decisions that might be conflicting since

[1] https://gepettoweb.laas.fr/articles/leziart2022.html

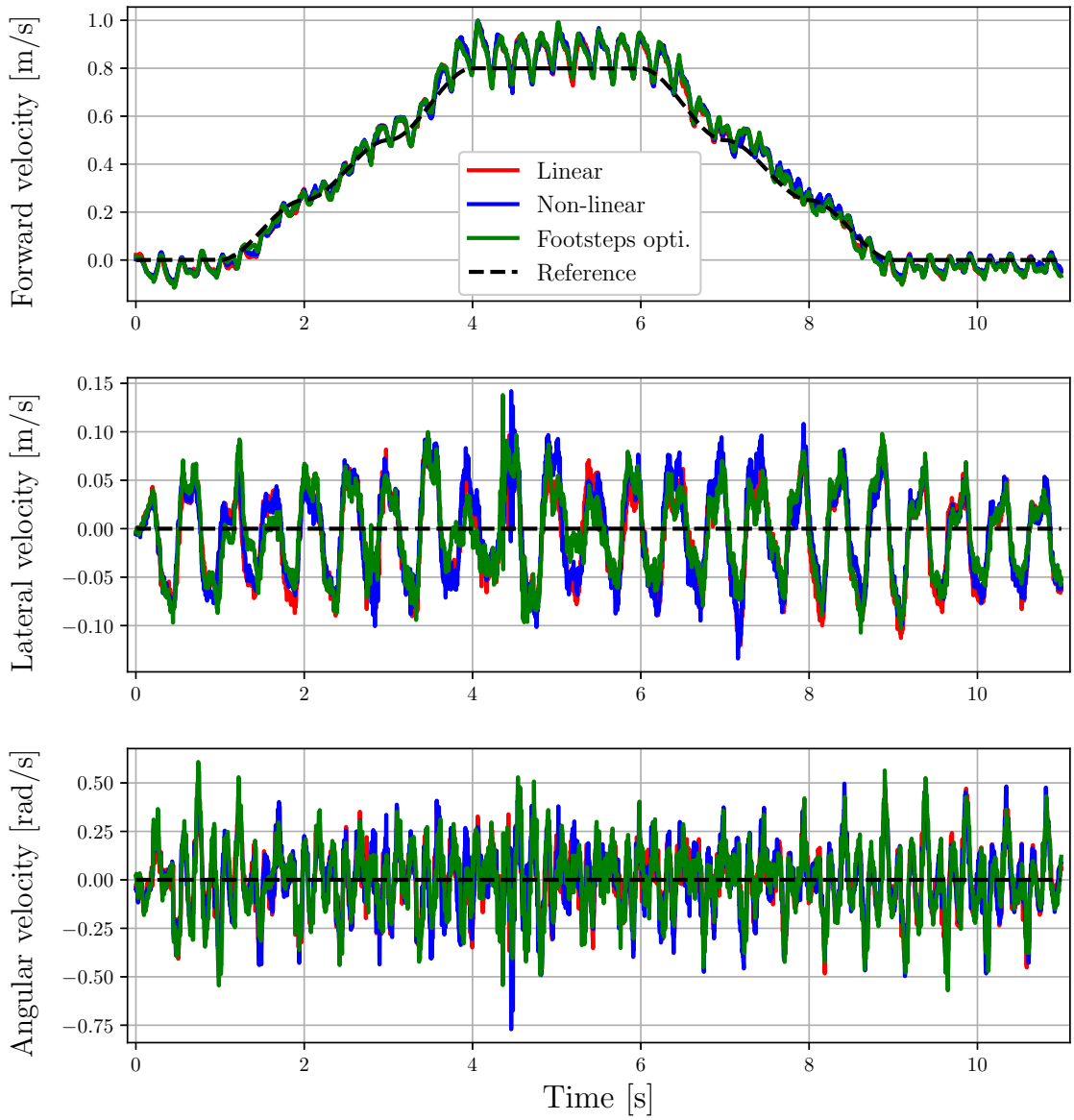


Figure 11.3: Forward, lateral and angular velocity profiles for the first scenario of Fig. 11.1.

the instantaneous decisions of the IK are done without knowing what has been decided by the MPC that works on a prediction horizon. Finding the right balance is not trivial, as mainly relying on IK ($Q_1 \gg Q_2$) would remove the predictive aspect of the architecture, whereas mainly relying on the MPC ($Q_2 \gg Q_1$) would instead hamper the feet tracking tasks. Both simulations and experimental tests show that the implementation of the three MPC variants leads to quite similar behaviors, contrary to what we could have expected *a priori*. Now that the whole architecture has been refined and the ID replaced by an IK+QP scheme, the margin for behavioral improvement has been reduced, so the gains of using nonlinearity and footsteps optimization might have been somehow smoothed out, at least for the considered velocities. The quadruped falls at higher velocities due to the legs reaching singularity during motion. Our controller does not handle flight phases yet (no feet in contact) so the velocity the robot can reach is limited by the gait frequency and the dimensions of the legs. A key result is to have shown that a centroidal MPC which optimizes footsteps location online can be successfully deployed on our quadruped

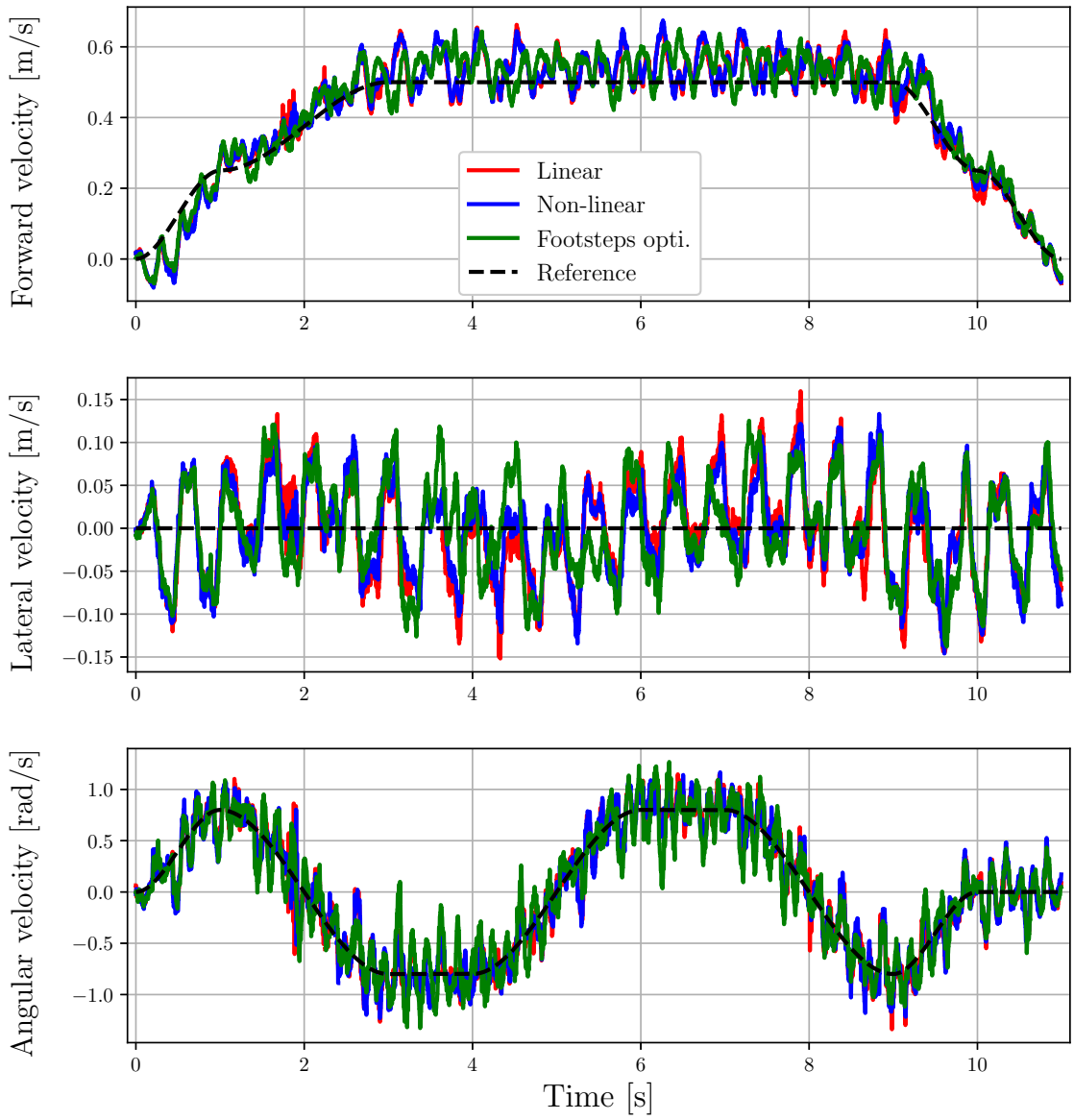


Figure 11.4: Forward, lateral and angular velocity profiles for the second scenario of Fig. 11.1.

robot, which was an important objective following [Cor+21]. The question of optimizing all the variables in the same MPC or treating them in separate blocks arises. On the one hand, modularity can be preferred with footsteps decisions that are independent from the computation of contact forces by the MPC. With the first and second variants, the planner with simple heuristics could be replaced by another control system to explore new paradigms. It could go from a neural network that implicitly learns adaptive heuristics during its training [Mag+19], to more path-planning oriented approaches that leverage information about the environment to place the feet at the best locations [Son+21]. Gait type and period could be tuned as well since the MPC does not have any notion of gait per se. On the other hand, all-in-one optimization with the third variant avoids the need to formulate heuristics and let the possibility to tackle more extensive and difficult challenges, such as an optimization of footsteps timings as well [Cor+21].

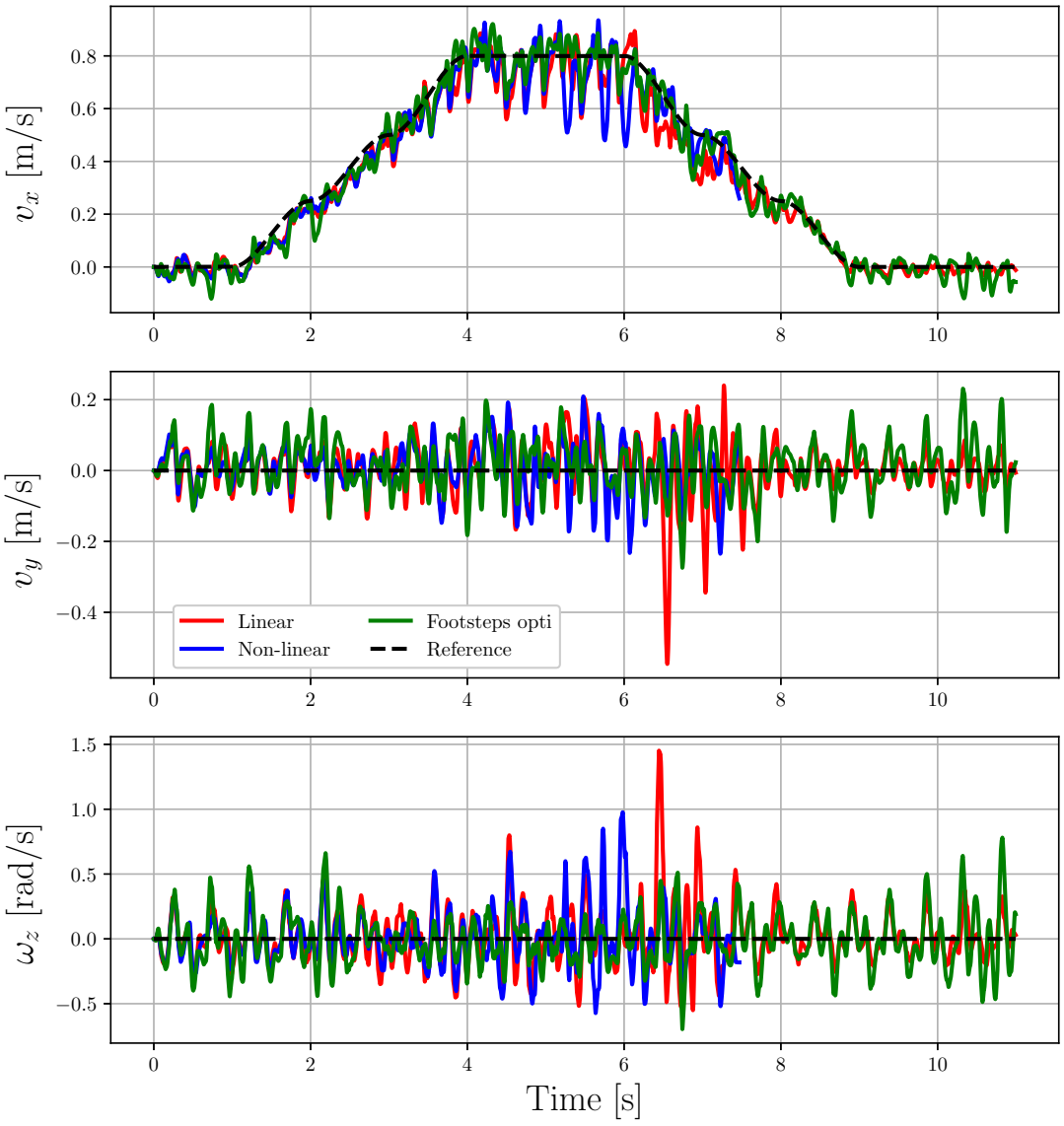


Figure 11.5: Forward, lateral and yaw velocities during the outdoor tests. Due to a lack of motion capture system outdoors, represented quantities are the estimated ones.

11.5 Conclusion

This chapter reports experimental results that validate the improvements done to the nominal control architecture of Solo-12 to reduce base oscillations. Three variants of a centroidal MPC previously tested in simulation only were deployed on the real robot. The overall improvement of the architecture also increased the maximum velocity the quadruped could reach, from 0.5 m/s in Chapter 10 up to 1 m/s.

The central result is to have shown that comparable performances can be obtained both in simulation and on real hardware with all MPC variants, either with the nonlinear centroidal MPC scheme optimizing footholds or the modular scheme that relies on heuristics for foot placement and on a simplified model. In view of the modifications that were made to implement these controllers on the real robot, the reasons for this difference have been thoroughly discussed. We also discussed the interest of using a modular architecture versus a more global optimization scheme.

Chapter 12

Footsteps planning using information about the environment

Contents

12.1 Introduction

The results presented in this chapter are assessing the implementation of a planning scheme that augments the proposed control architecture by using privileged information about the environment to simultaneously plan the motion and footstep locations several steps ahead. It provides elements for the robot to overcome situations that would be too complex for the baseline heuristics, while respecting its kinematic constraints, as long as hardware capabilities are sufficient to produce the desired motion. Such situations appears when the assumption of a flat ground and the sheer robustness of the controller are not enough anymore, such as terrains with deep holes or steps too high to climb with the nominal feet swinging motion.

Several stages of the scheme presented in Section 7.3 have to be validated. The first one chooses on which surfaces to step on based on a set of available surfaces in the environment. Then, a quadratic program optimizes the footsteps locations to respect the limits of chosen surfaces. Finally, an improved trajectory generator using 3D Bézier curves drives swinging feet from one location to another under collision-avoidance constraints.

Preliminary tests were done in simulation using the physics engine PyBullet with the full model of the robot, as in Chapter 10, to confirm that the pipeline was working properly before deploying it on the robot.

12.2 Experimental setup

Experiments were first performed indoors with a motion capture system comprising 20 infrared cameras spread around the workspace. They are used to locate the robot by tracking 13 reflective markers installed on top of the robot base, at 200 Hz. Solo-12 has no exteroceptive sensors yet so there is no other way to get an accurate position of the base with respect to the surfaces in the environment so that there is no misalignment between the actual robot position and the position used by the planner.

Similarly, the challenge of detecting and reconstructing available surfaces for the first stage of the planner is also circumvented with the motion capture. The exact location of surfaces in the world is defined before the experiment, we then ensure the physical obstacles are correctly placed to fit these locations with the help of infrared markers. The next 6 steps of the robot are planned every 160 ms, while the robot whole-body control is computed at 1 kHz as during the previous experiments of Chapter 11. Several environments were built to perform qualitative evaluations of the framework:

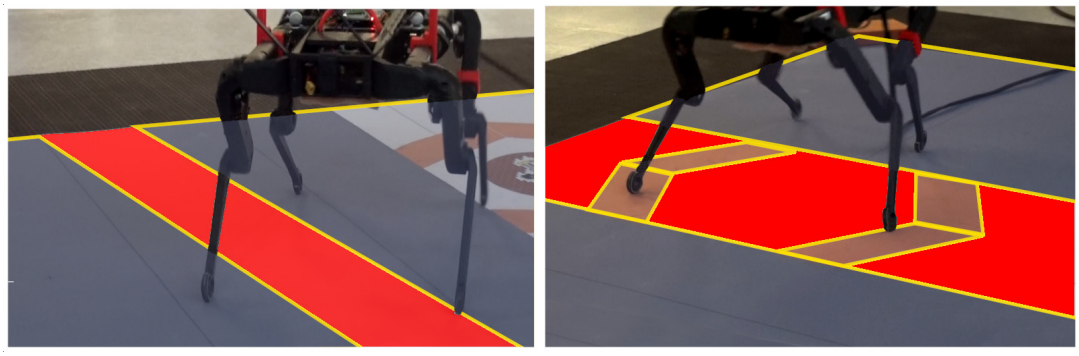
- Flat terrain to assess the overall robustness of the controller to perturbations.
- Straight hole on the ground, up to 15cm wide, to show the ability to avoid a forbidden area between two contact surfaces while maintaining balance for any angle of approach.
- Bridge-like structures with two non-parallel 10cm-wide walkways that make foot positions more challenging to find by forcing contacts to move away from the vertical of the shoulders to land in allowed areas.
- Straight stairs to show the usefulness of the collision avoidance module that modifies the swing trajectories to avoid hitting the edges of steps.
- Stepping stones to illustrate the necessity of the contact plan and check if the footstep plan is adapted fast enough to handle quick changes of the reference velocity. The stepping stone were rectangles of 22 cm by 11 cm, either flat or with a height of 6 cm.

12.3 Experimental results

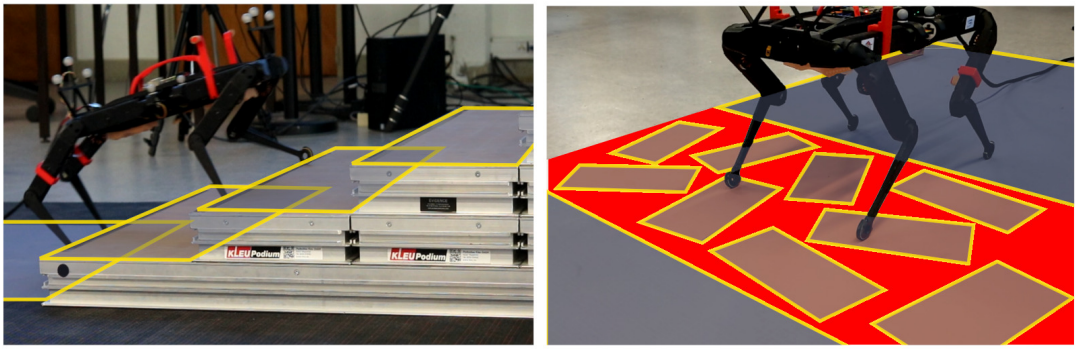
During experiments, the robot managed to recover from strong perturbations on flat ground and could cross a hole materialized by a line drawn on the ground, go back-and-forth and turn over it (Fig. 12.1a). It successfully walked on a drawn bridge which required to move its left feet away from his right one (Fig. 12.1b). The stairs were climbed up an down with varying angles of approach and the robot could turn in the middle of the stairs without falling (Fig. 12.1c). On stepping stones the robot managed to place correctly its feet and maintain its balance even with quick variations in the reference velocity, for example when deciding to stop or to turn in the middle of the challenging area (Fig. 12.1d).

A video that highlights some of these experiments as well as simulation results is available online[1].

[1] https://gepettoweb.laas.fr/articles/risbourg_corberes_2022.html

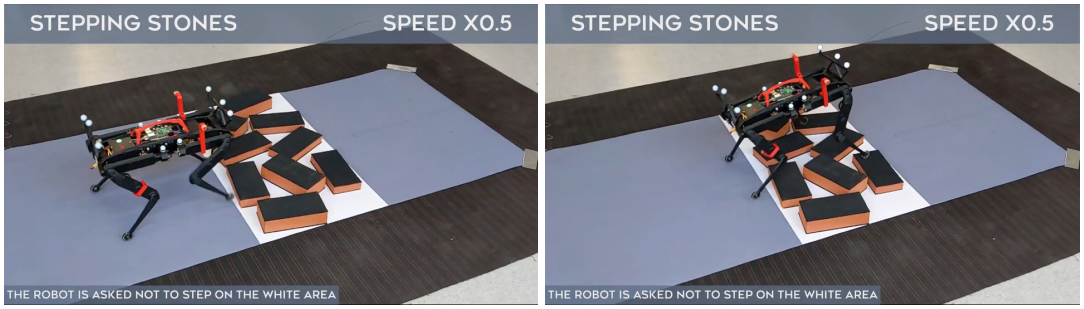


(a) Straight hole (b) Bridges

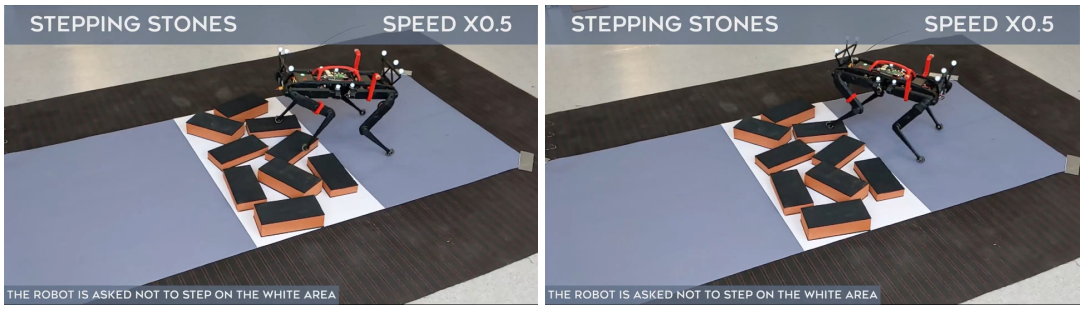


(c) Straight stairs (d) Stepping stones

Figure 12.1: Situations that were considered to evaluate the planning framework. The planner is given the yellow areas as available surfaces. Feet are not supposed to land in red areas since they are not part of the set of available surfaces to choose from. Pictures extracted from [Ris+22].

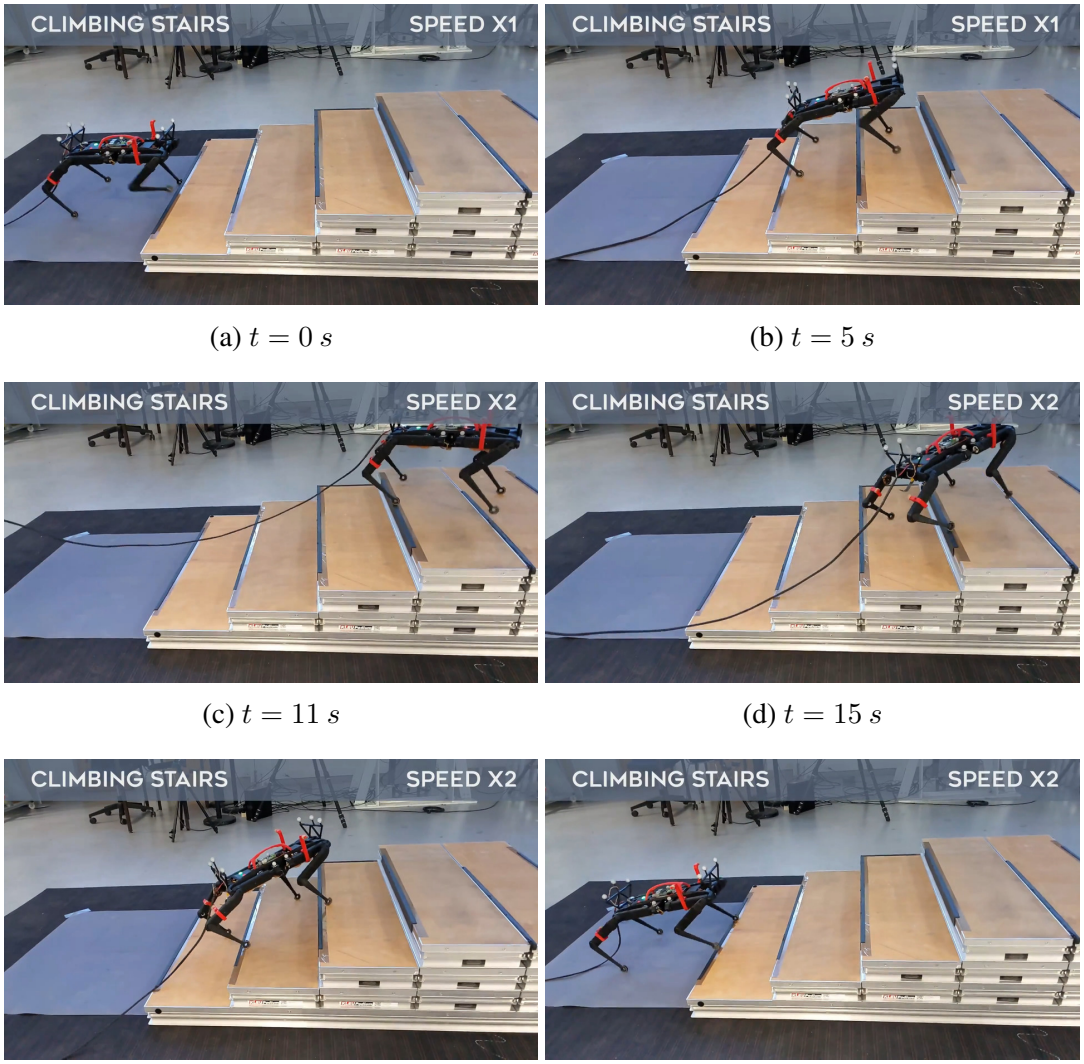


(a) $t = 0\ s$ (b) $t = 2\ s$



(c) $t = 4\ s$ (d) $t = 6\ s$

Figure 12.2: The quadruped successfully crosses the area with stepping stones. Knowing the location of available surfaces, it avoids landing its feet in the holes, even when turning on the spot. Snapshots extracted from [RIS].



(a) $t = 0\ s$ (b) $t = 5\ s$

(c) $t = 11\ s$ (d) $t = 15\ s$

(e) $t = 23\ s$ (f) $t = 29\ s$

Figure 12.3: The quadruped first climbs the stairs while being oriented perpendicular to the steps. At the top, it turns right then left and goes down while being oriented diagonally. It successfully goes back to the initial position without falling or hitting the edges of the steps. Snapshots extracted from [RIS].

12.4 Conclusion

This chapter reported experimental results that validate a planning scheme added on top of the baseline control architecture. Using exteroceptive information about the location of potential contact surfaces in the environment, it allows the robot to overcome situations that would be to complex to handle with a blind heuristic walk. It serves as a proof of concept for what could be a future improvement of the autonomous navigation capabilities of Solo-12, once exteroceptive sensors combined with a surface detection pipeline are installed and deployed.

Chapter 13

Contact detection for rough terrains and jumping motion

In this chapter we present early results of the online contact detection described in Section 6.4. We compare performances during locomotion on rough terrain, in simulation, with and without this detection. Without this detection, contacts are enabled or disabled based on predefined contact timings that are considered as trustful. This works well for trotting indoors on a flat ground without obstacles since the swinging feet tracking is good enough. On uneven grounds, contact mismatches have more chance to be significant enough to be detrimental to the quality of the locomotion. Because the controller runs with the assumption of a flat ground, if the target position of a swinging feet is in a depression, the foot will stop in the air a few centimeters above the ground, then it will hit the floor as the leg suddenly extends due to the activation of the contact. On the contrary, if it hits an obstacle, like a stair, earlier than expected, the controller will try to reach the target position at floor level, thus will apply unexpected forces on the obstacle. Online contact detection would allow to detect if a contact happens too early or too late instead of relying on the predefined contact sequence. So, in such cases, we can expect an improvement of the robot behavior. These tests in simulation are a way to partially validate the approach for future real-world deployments.

This contact detection has another application in the case of jumping motion. Due to control inaccuracies, in practice the flight duration for each foot only roughly corresponds to the expected one. It can be caused either by a slightly wrong vertical velocity at takeoff, or by an unexpected evolution of the base orientation in the air which make feet touch the ground earlier or later than expected. A safe way to avoid contact mismatches in such cases is to simply not activate contacts and to only rely on the impedance controller to dampen the impact. With online detection, we could actively act to absorb the impact and stabilize the base with individual feet once they have been detected as in contact. This will also partially validate the method presented in Section 8.1.2 to produce jumping motion.

13.1 Simulation setup for the height field

The control architecture is deployed in simulation in the physics engine PyBullet. It is overall similar to the one that was used for the comparison on model prediction controllers in Chapter 11, except with an additional contact detector which checks at each iteration whether a new contact has been detected or not, based on the method and metrics

presented in Section 6.4. To test the relevance of contact detection, we do not use a flat ground has we did so far but a height-field instead: the ground is divided into cells and each cell is given a random height between 0 cm and 5 cm, using an uniform distribution. The robot is placed on a flat area at the start of the simulations for easier initialization. An example of height field is displayed in Fig. 13.1.

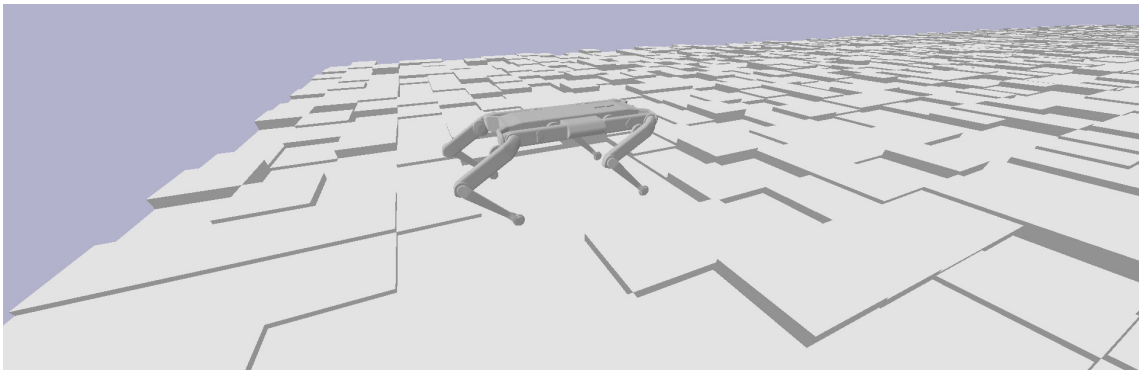


Figure 13.1: Random height-field loaded in simulation. The 3D model of the ground is generated at the start of the simulation. Different distributions of steps can be obtained by changing the value of the random seed if variability is desired.

The apex height of the swing trajectories is set to 10 cm to have enough margin to avoid mid-flight collision when going from a 0 cm step to a 5 cm step. The reference velocity profile slowly climbs up to 1 m/s forwards in 8 s and then maintains this value.

13.2 Simulation results for the height field

Fig. 13.2 highlights trajectory tracking performances of the quadruped on random height field with and without online contact detection. In both cases the quadruped reaches the end of the scenario without falling. State oscillations are lower for all components when using the online contact detection, which points out the interest of the method. This can be explained because the whole behavior is impacted when a contact is early or late compared to what the baseline controller was expecting. For instance, if the contact is actually late because the foot stopped above a hole in the ground, without contact detection the contact will be enabled anyway and the controller will plan and act as if it could apply a force on the ground with this foot. Yet, until the leg extends and touches the floor, the base will not be stabilized as expected, causing a tilt in roll or pitch, and the quadruped will push less than expected on the floor, thus decreasing the velocity tracking performances. The online contact detection avoids such situations as the contact is not enabled and the rest of the controller knows this foot cannot be used. On the contrary, for early contacts, the controller can immediately react and do not apply unexpected forces on the ground while trying to reach a target position that is inside the ground. The displayed height of the base is the ground truth in world frame. It appears higher than the reference value since steps have a height between 0 and +5 cm and the controller tries to keep the base at the reference height (here ≈ 0.19 cm) with respect to the contact points.

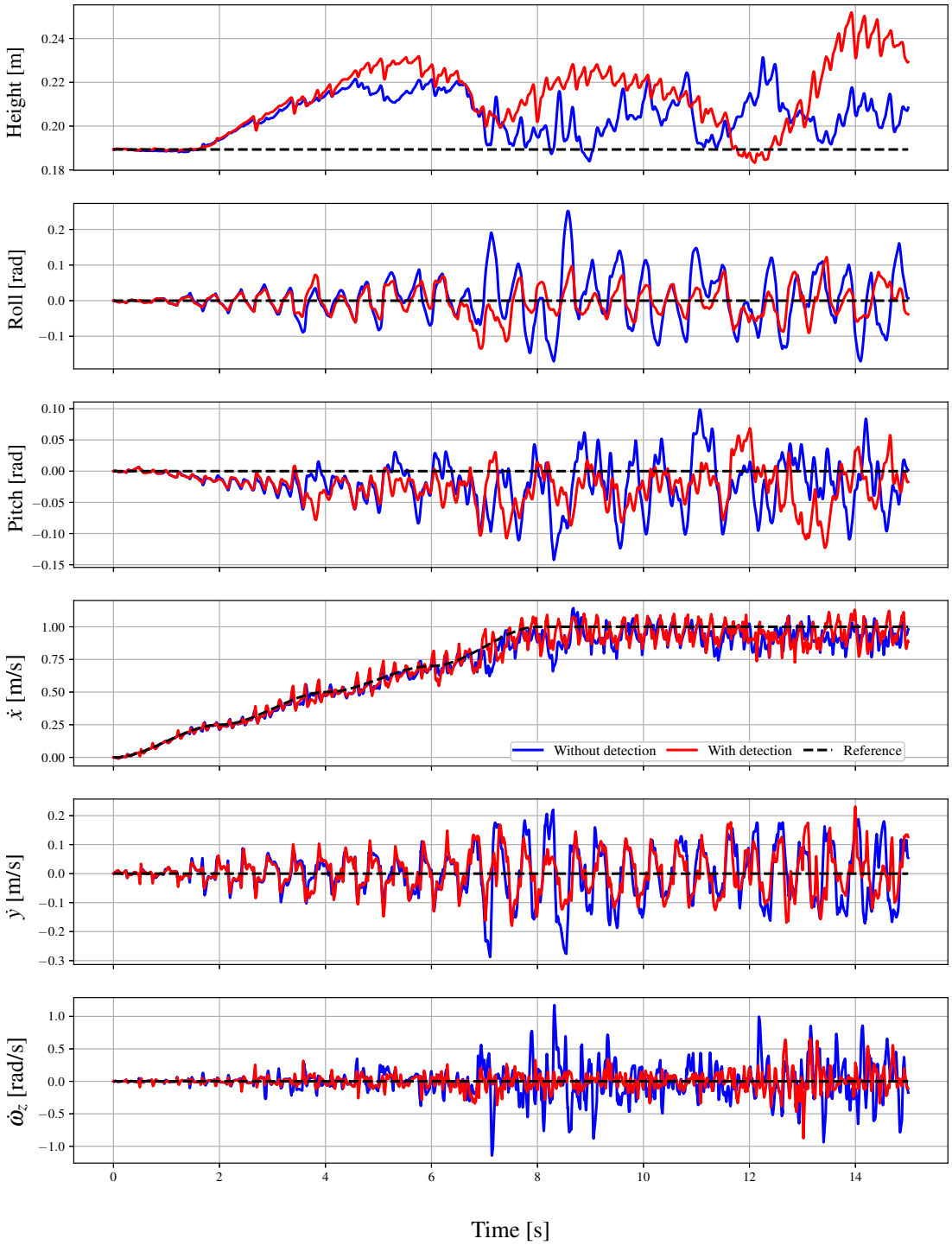


Figure 13.2: Comparison of tracking performances with (red) and without (blue) online contact detection compared to the reference trajectories (dotted black).

13.3 Simulation setup for jumping

For the jumping tests, the robot relies on the approach that was described in Section 8.1.2. It is placed on a flat ground and given a gait sequence that includes a flight phase, as follows:

$$\mathcal{G} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{matrix} \left.\vphantom{\begin{matrix} 1 \\ \vdots \\ 1 \end{matrix}}\right\} \text{Pre-jump phase} \\ \left.\vphantom{\begin{matrix} 0 \\ \vdots \\ 0 \end{matrix}}\right\} \text{Flying phase} \end{matrix} \tag{13.1}$$

The corresponding reference state trajectory is given to the MPC for solving. The contact forces profile over the whole horizon is then retrieved and kept in memory to be played without replanning each 20 ms as we usually do during walking. This is done to ensure consistency of the force profile during the critical pre-jump phase. Otherwise, as times goes by and the ballistic phase gets closer, lines of 1s appear at the end of the gait matrix and the MPC starts to consider the post-touchdown stabilization at the expense of the jump phase. This changes a bit the force profile as the tracking error is spread equally over the whole horizon. Ideally we could tune down or disable the post-touchdown error so that the MPC entirely focus on reaching the proper velocity at the beginning of the ballistic phase. The reference vertical velocity is also given to the whole-body control for its base velocity task.

Two scenarios are investigated. First, the robot jumps on the spot to reach a height of 0.6 m, corresponding to a flight phase of 0.56 s. Then, it also tries to jump up to 0.6 m, but this time while moving forwards with a target velocity of 0.8 m/s during the flight phase, which would correspond to a total displacement of around 0.6 m forwards.

13.4 Simulation results for jumping

As shown in Fig. 13.3, the quadruped follows the expected jumping sequence, with an initial lowering phase followed by a sudden rising to reach a sufficient vertical velocity to leave the ground in a jumping motion. Fig. 13.4 reports the tracking performances of the robot during this scenario. As we can see, in its lowest vertical position, the base almost touches the ground. How much the base is lowered depends on the vertical velocity to reach at the beginning of the flying phase, which itself depends on the desired duration of the flight phase. This means a flight phase of 0.56 s is a limit for the current rising rate of the base. To perform longer flight phase we would have to make the base accelerate faster so that the base does not have to lower as much. Tracking quality for all quantities is good during the whole pre-jump phase, until $t = 0.96$ s, with roll and pitch close to 0 and height and vertical velocity following what has been outputted by the trajectory planner. During the flight phase, orientation slowly drifts from the reference. This is expected as the robot does not have any contact point anymore to apply forces and stabilize the base, and we do not try to stabilize base orientation through limb motion. Yet the most notable tracking error is in height and vertical velocity. The base only goes up to 40 cm at the apex of its ballistic trajectory at $t = 1.2$ s. The vertical motion of the base in the air is decided by the vertical velocity at takeoff. Although velocity tracking is excellent during the rising phase, up to the supposed start of the flight phase, there is a sudden loss of velocity at takeoff, which might be due to a poor transition between standing and flying

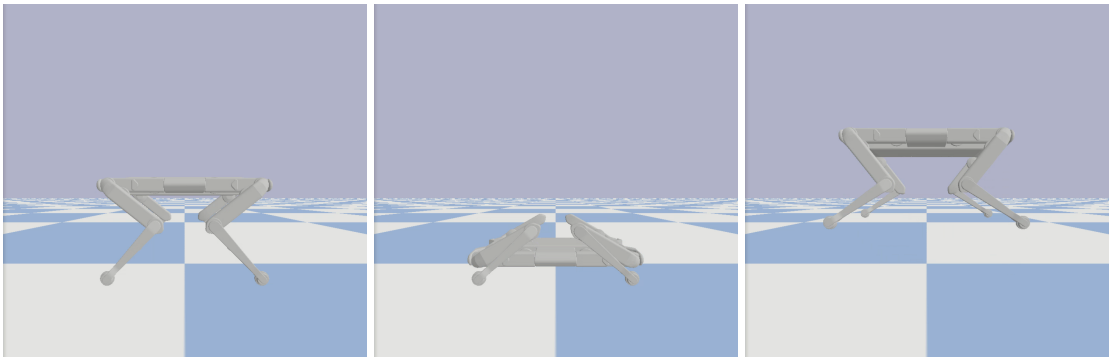
phases. This is a point that should be investigated in the future since it impacts the whole following flight phase. Another interesting aspect to note in this scenario is the relevance of online contact detection. Because the vertical velocity was lower than expected, the base went back to its initial height far earlier than expected at $t \approx 1.4$ s. The moment at which contact is detected clearly appears in height graph, when the reference ballistic trajectory at 0.45 m suddenly goes down to the standing base height at around 0.2 m. This detection enables the controller to immediately apply contact forces on the ground to nullify the negative vertical velocity, as it can be seen in Fig. 13.5. Without online contact detection, this would have happened only after the expected end of the flight phase, that is $t = 1.52$ s.

In the second scenario, the quadruped successfully jumps forwards, as shown in Fig. 13.6. Again, Fig. 13.7 reveals that tracking performances are good until the very start of the flight phase, where there is a sudden loss of velocity both for the forward and vertical components. As a result, the quadruped does not jump as high nor as far as it should: 0.45 m upwards and 0.35 forwards instead of 0.6 m and 0.6 m respectively. The flight phase also ends earlier than expect, around $t = 1.35$ s, yet the online detection allows an immediate reaction to stabilize the base by applying forces on the ground (see Fig. 13.8).

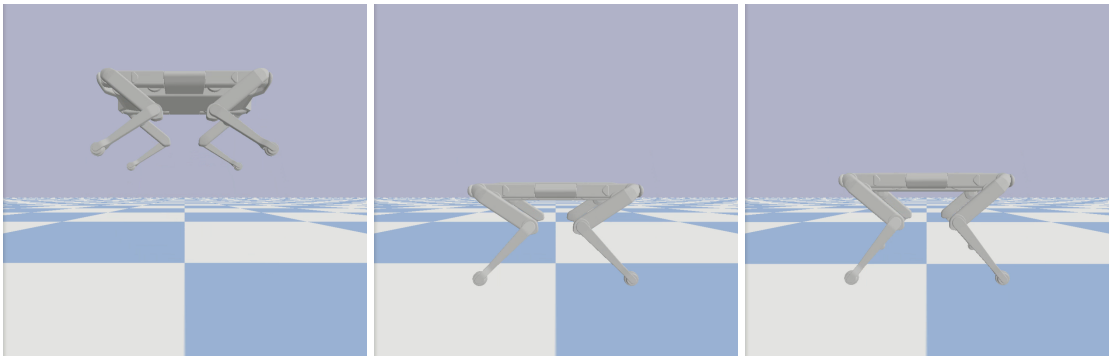
13.5 Conclusion

This chapter reports simulation results that validate the relevance of online contact detection for locomotion on rough terrain or for jumping motion. On rough terrain, tracking performances are improved because online contact detection allows to ascertain if a contact happens too early or too late instead of relying on the predefined contact sequence. In such cases the controller receives the information and can react accordingly. This online detection also allows to detect early or late touchdown after a flight phase that are caused by a non-perfect velocity tracking during the pre-jump phase.

This chapter also explores early results for jumping motions that happen when providing the MPC with a ballistic trajectory for the vertical position and velocity components. The control architecture successfully performed on-the-spot and forward jump although not as high or as far as expected due to a velocity loss at takeoff that will have to be investigated.



(a) $t = 0\ s$ (b) $t = 0.75\ s$ (c) $t = 1\ s$



(d) $t = 1.25\ s$ (e) $t \approx 1.4\ s$ (f) $t = 1.75\ s$

Figure 13.3: First scenario: jumping on the spot in simulation. From an initial position (a), the base is first lowered to a minimum height (b), before suddenly rising up to reach the desired velocity at the start of the flight phase (c). In the air, the base follows a ballistic trajectory (d). Touchdown (e) is detected by the online contact detection, which allows the controller to react as soon as possible to absorb the impact and stabilize the base (f).

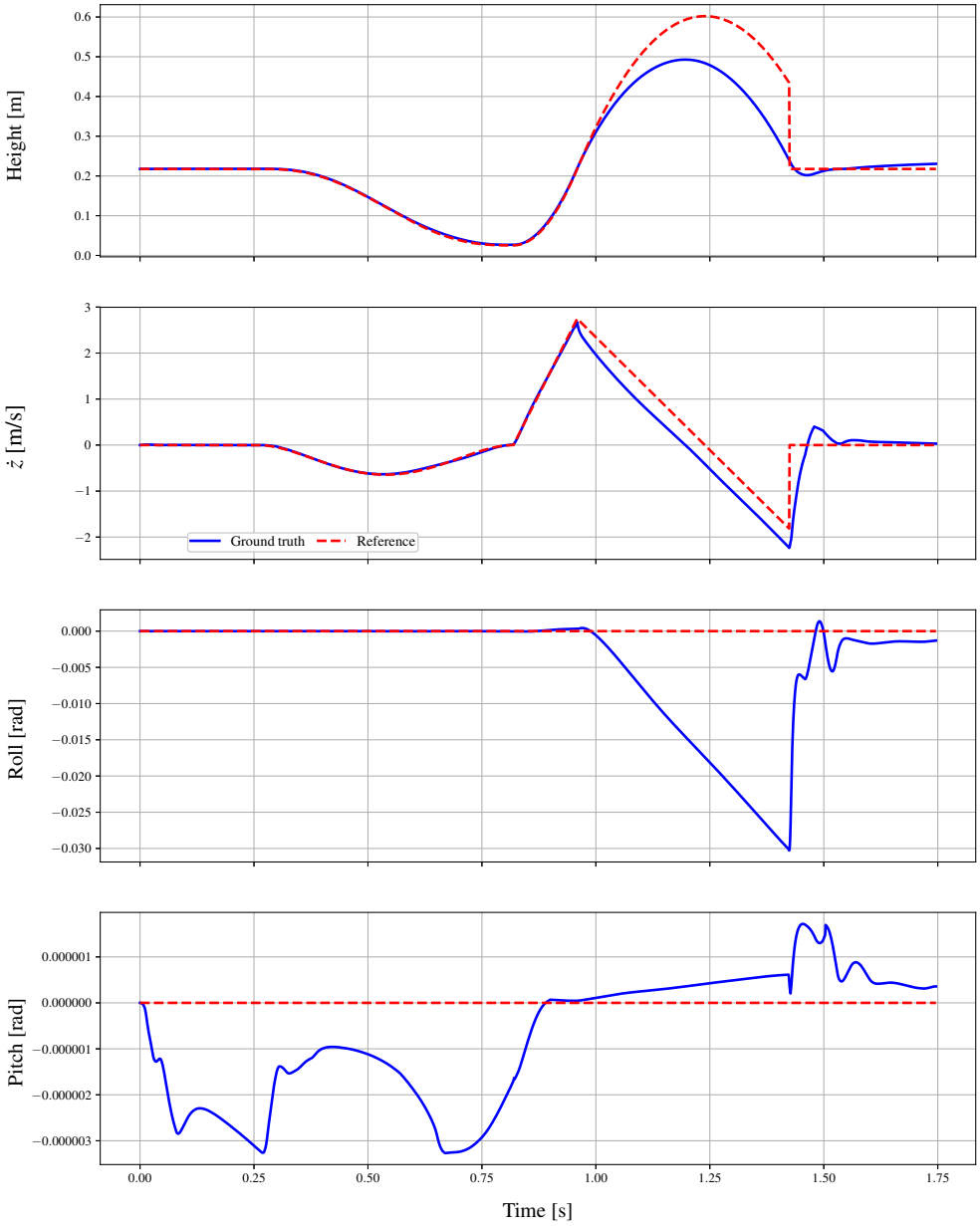


Figure 13.4: Height, vertical velocity, roll and pitch angles of the base during the flight phase.

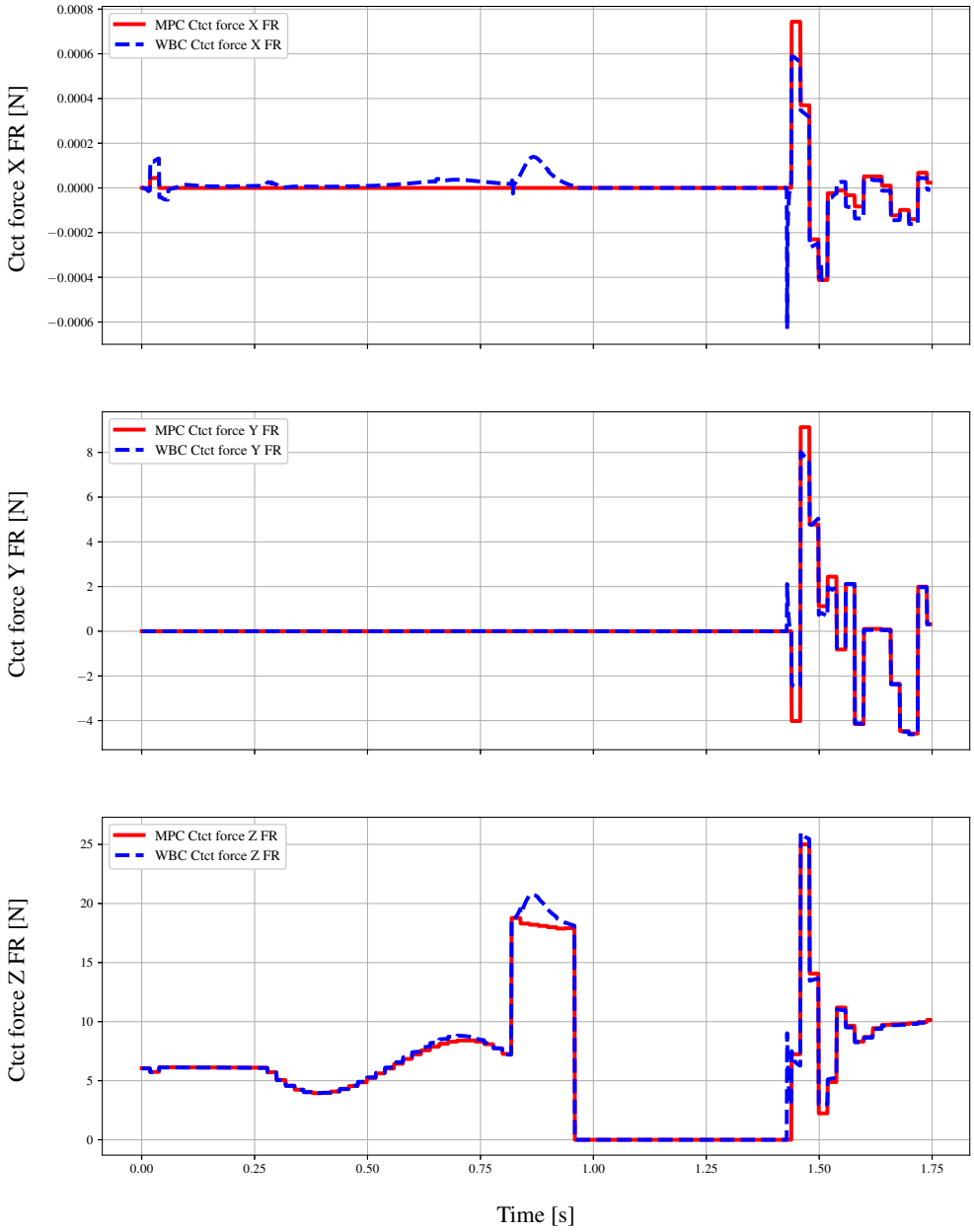
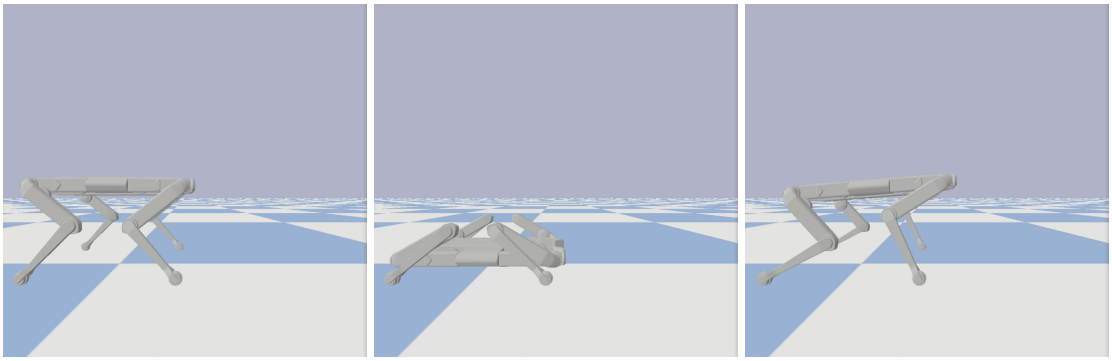
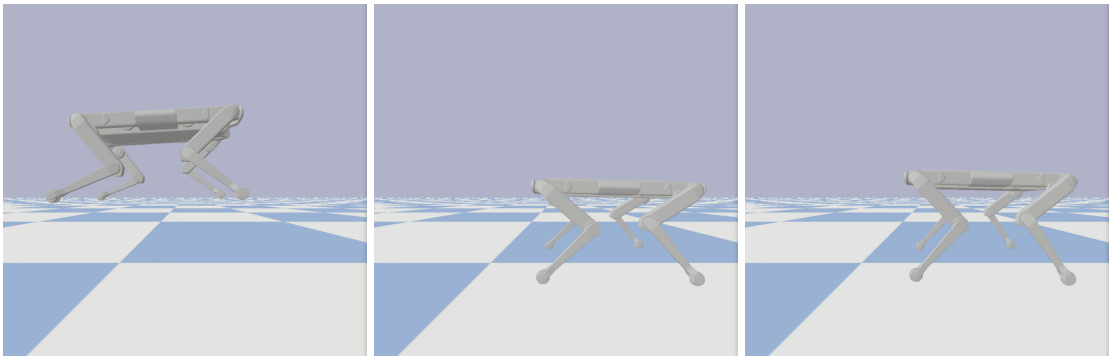


Figure 13.5: Contact forces components specified by the MPC and the WBC after trade-off with the joint accelerations of the IK. Only the three components of the contact forces for the front right (FR) foot are represented for better visualization since, due to the symmetric nature of the problem, the force profiles of the four feet are similar.



(a) $t = 0\ s$ (b) $t = 0.75\ s$ (c) $t = 1\ s$



(d) $t = 1.25\ s$ (e) $t \approx 1.4\ s$ (f) $t = 1.75\ s$

Figure 13.6: Second scenario: forward jump in simulation. From an initial position (a), the base is first lowered to a minimum height (b), before suddenly rising up to reach the desired velocity at the start of the flight phase (c). In the air, the base follows a ballistic trajectory (d). Touchdown (e) is detected by the online contact detection, which allows the controller to react as soon as possible to absorb the impact and stabilize the base (f).

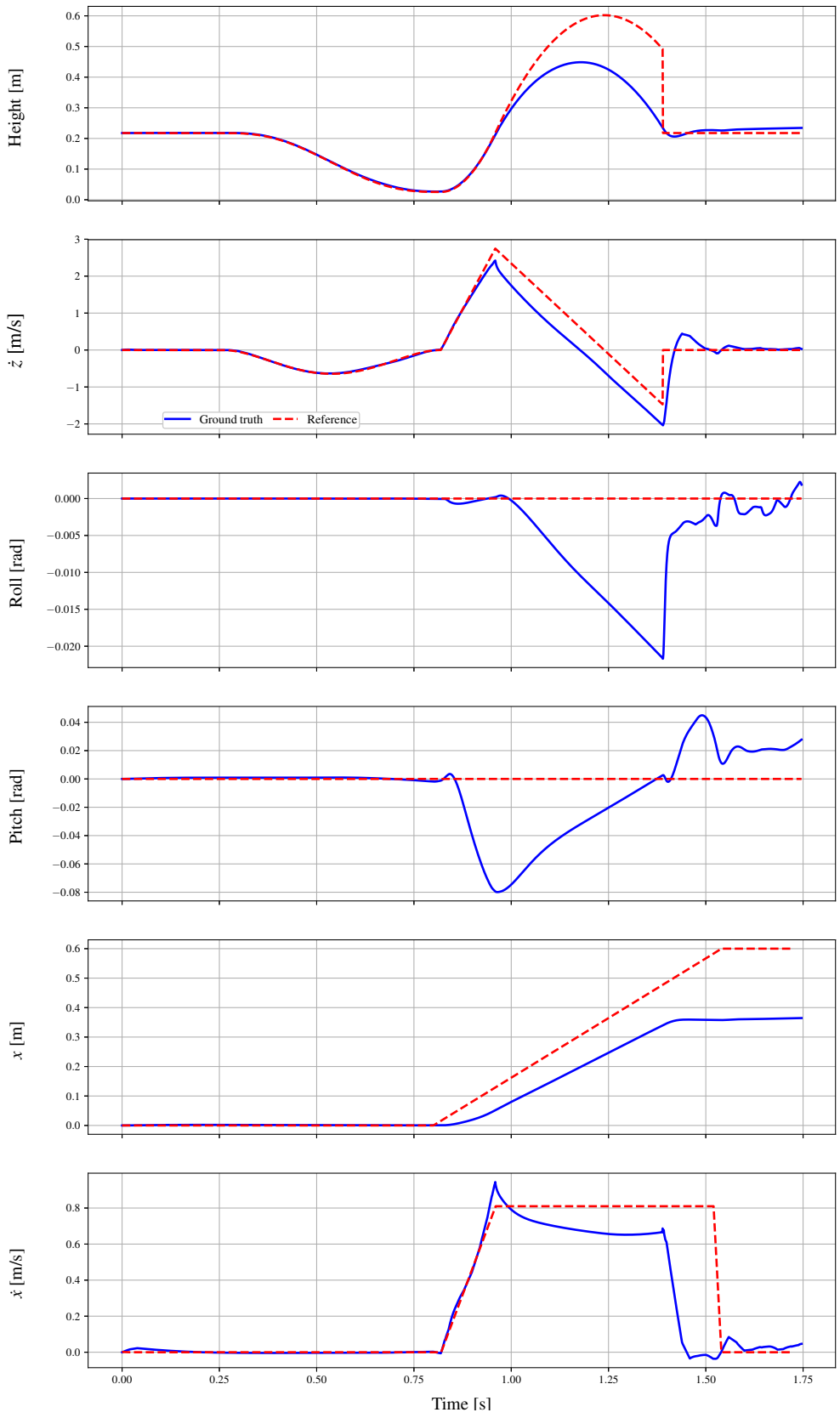


Figure 13.7: Height, vertical velocity, roll and pitch angles of the base during the flight phase of the second scenario.

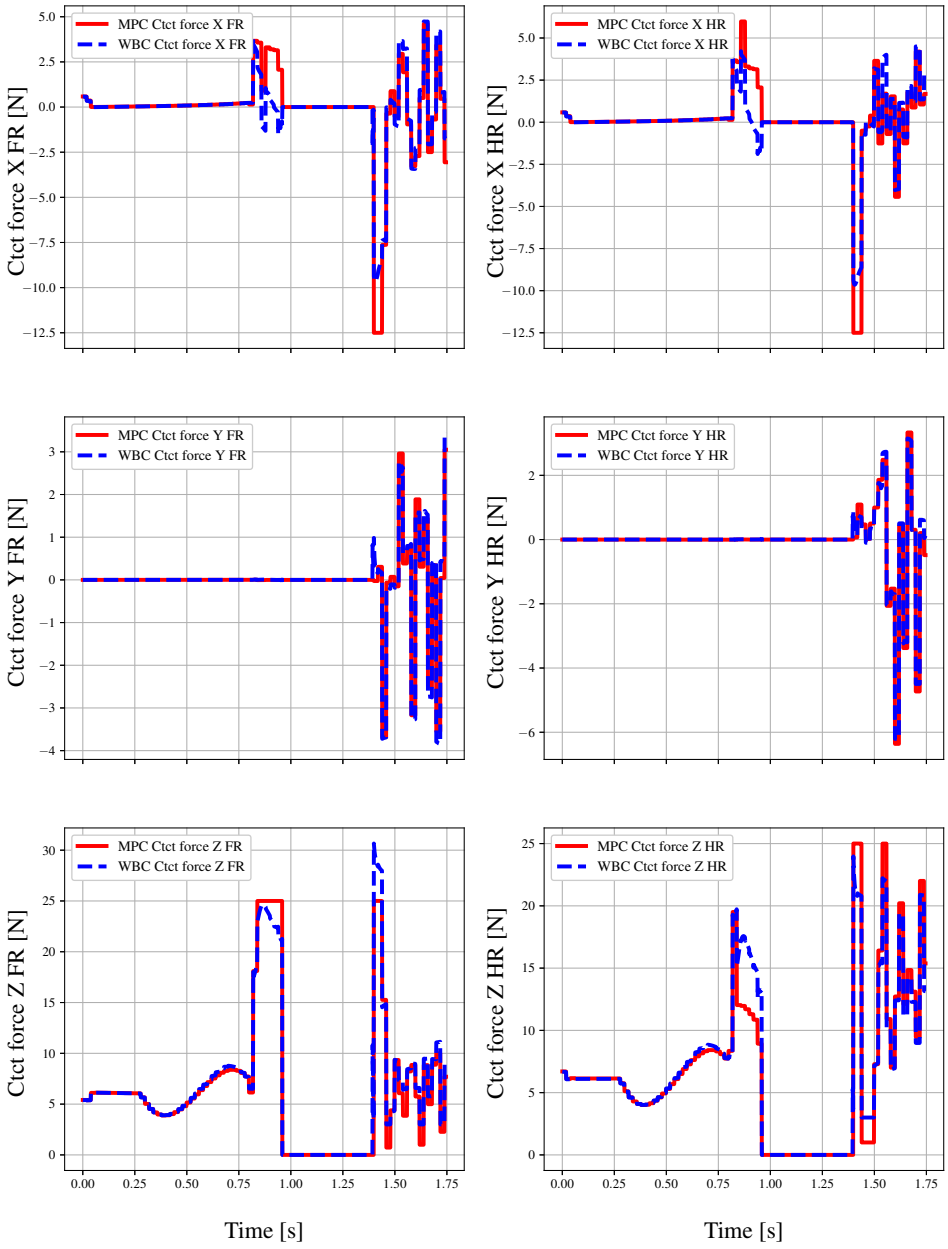


Figure 13.8: Contact forces components specified by the MPC and the WBC after trade-off with the joint accelerations of the IK. Only the three components of the contact forces for the front right (FR) and hind right (HR) feet are represented for better visualization since, due to the symmetric nature of the problem, the force profiles on the left side of the body are similar.

Chapter 14

Locomotion with end-to-end deep reinforcement learning

Contents

14.1 Introduction

In this chapter, we present the results of a work that was done withing the framework of the PhD thesis of Michel Aractingi, in collaboration between the Gepetto team and Naver Labs Europe [NAV]. It explores how reinforcement learning could be used either to augment our model-based architecture or to deploy an independent data-based control architecture independent that would be able to make the Solo quadruped walk around. A possible augmentation, as already been hinted at in Section 6.3.2, could be to modify the gait characteristics online with a neural network to go beyond the use of a predefined gait sequence. Another data-based architecture was later developed to achieve locomotion in various situations (flat and sloped grounds with bumps, stairs) and track a reference velocity. This time, the goal was to train an end-to-end architecture that would control the quadruped on its own instead of being grafted to the model-based architecture.

To do so, a two-stage hierarchical deep reinforcement learning architecture was trained to perform tasks. A low-level policy learns desired joint angle targets based on an elaborate reward function with multiple cost terms related to the desired behavior of the robot. A high-level policity then learns to determine the parameters of these cost terms to dynamically adapt the low-level policy.

Preliminary training and tests were done in simulation using the physics engine Raisim [HLH18] with the full model of the robot. The transfer capabilities were first checked in PyBullet [CB21] to ensure the trained policies were robust enough to handle the difference

in robot behavior linked to a change of simulator. Then, the architecture was deployed on the real robot.

14.2 Overview of the data-based architecture

As a brief overview of the deep reinforcement learning architecture, let us outline the state and action spaces of both low-level and high-level policies.

First, the state space considered by the low-level policy consists of the orientation of the base, the linear and angular velocities of the base, the joint angles and velocities. Alongside these quantities, a history of the joint angle errors and joint velocities is used so that the network can better figure out the situation the robot is in, based on what happened in the past time steps, and ultimately take better decisions. The history of joint angle errors is more informative than mere joint angles as it can be used to infer the contact state of the feet. Finally, the network actions for the two last time steps are considered as well, again for a better understanding of the temporal behavior of the robot. These actions are the joint angle targets that are fed to the low-level impedance controller of the robot, with the joint velocity targets remaining at zero. To make learning easier for the network, the policy does not directly learn the absolute value of joint angle targets but rather the offset from nominal joint angles that correspond to the default configuration of the quadruped.

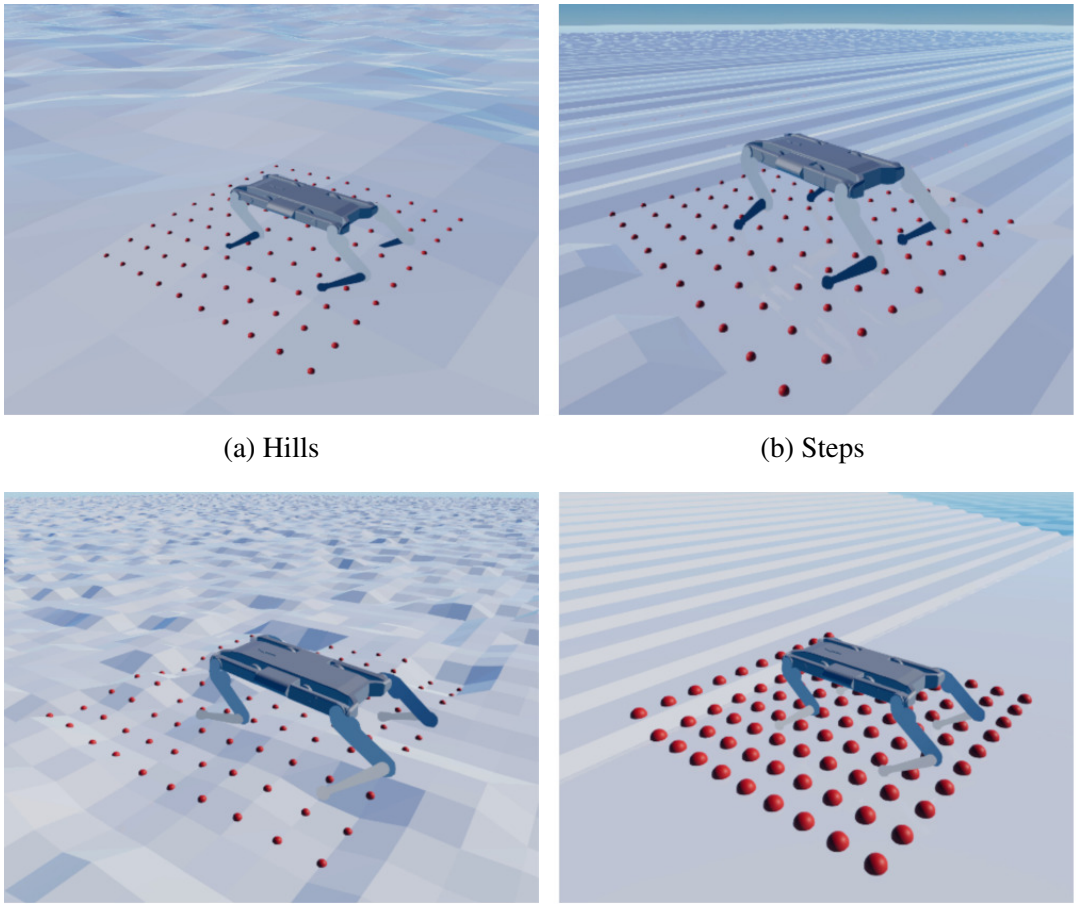
The high-level policy uses the same state space. However its action space is different. It modifies the locomotion parameters to modulate the behavior of the low-level policy, with the idea that the behavior of the robot can be tuned online while still producing stable locomotion. If not commanded by a user or a higher-level controller, the reference base velocity can be included in the action space to perform autonomous navigation to a target location. The apex height of feet swinging trajectories is included in the action space so that the policy can lift the feet higher on complex terrains while keeping them low on flat terrains to save energy. The penalization of the deviation from the nominal joint angles is also part of the action space to affect the stride length as needed. Finally, the high-level policy controls the proportional and derivative gains of the impedance controllers to make the control stiffer or more compliant depending on the variations of the environment, as a way to increase robustness.

14.3 Experimental setup

Both the low-level and the high-level policies are run with at same frequency of 100 Hz. They are trained over 300 million samples with the Proximal Policy Optimization (PPO) approach [Sch+17]. Among all terms of the cost function, the weights of penalties are progressively increased so that the robot first learns to walk and run before trying to optimize its energy consumption. If they would be fully enabled from the start the robot would not properly learn, preferring to not move at all to keep the energy consumption penalty at a minimum. For better transfer to the real robot and to avoid overtraining, random uniform noise is added to the robot dynamics and state observations. This noise is progressive as well, starting with noiseless simulations and increasing in magnitude as the training progresses. The goal is to have the network produce a robust behavior even if the model does not perfectly fit the real system, as it is inevitably the case (all motors

have slightly different characteristics and they vary as coils get warmer, the model does not include joint friction, inertia matrices are not perfectly accurate, ...).

The performance was first assessed in simulation. The Solo-12 quadruped was able to adapt its step length and gait sequence online to move up to 1.5 m/s on a flat ground, with in that case a gait period of 0.26 s and a flying phase of 0.04 s that naturally emerges from the movements of the legs. The architecture was tested as well on more complex terrains displayed in Fig. 14.1, such as stairs, hills and bumpy ground. In all those cases, the high-level policy adapted the locomotion parameters to tweak the behavior of the low-level policy and ultimately generated a successful motion.



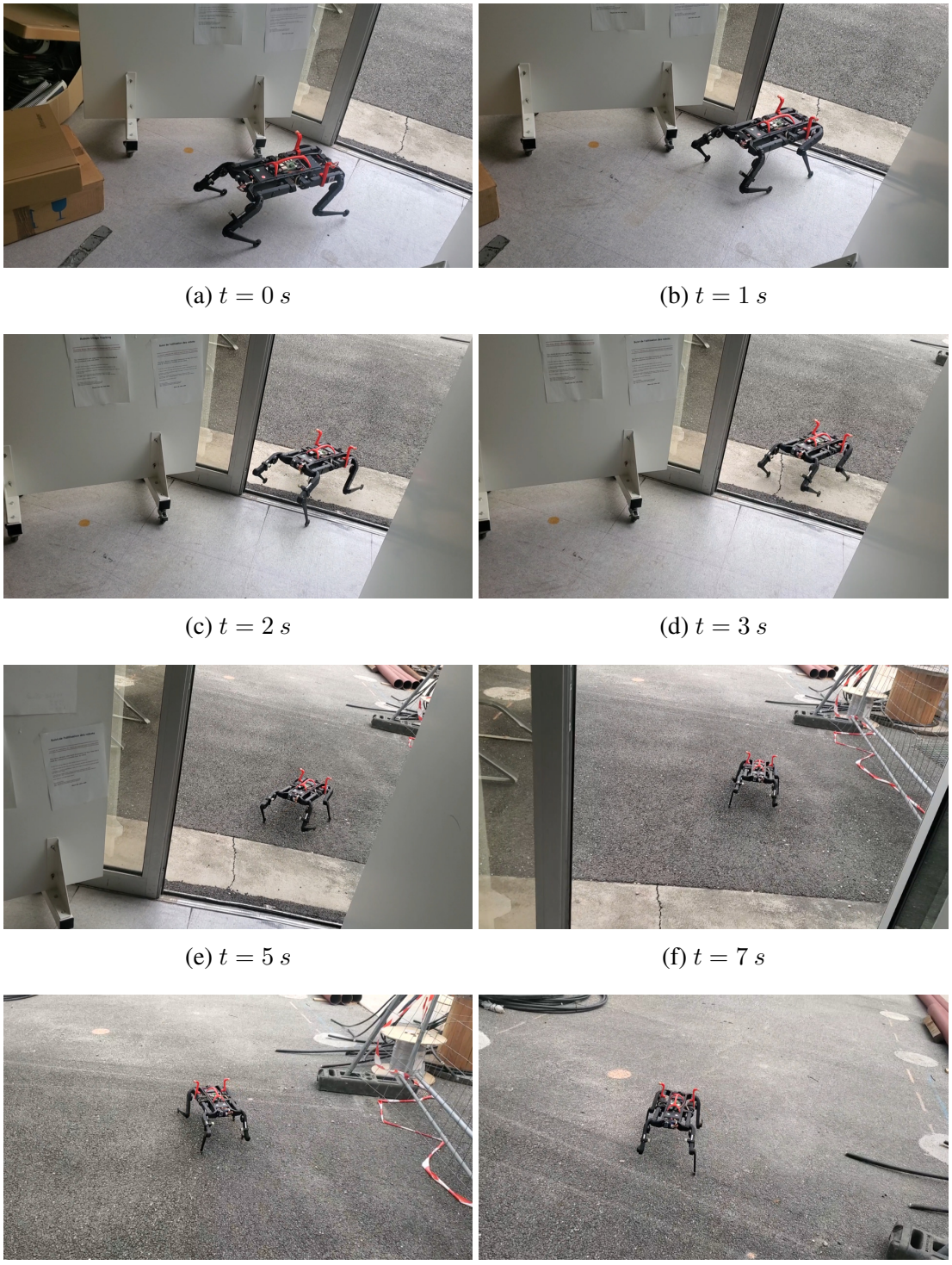
(a) Hills (b) Steps

(c) Height-field (d) Stairs

Figure 14.1: Examples of complex terrains the policies are trained on. The red dots represent the discretization of the surroundings that is given to the high-level policicty to find suitable locomotion parameters. Graphical credits to Michel Aractingi.

14.4 Experimental results

The low-level policy was successfully transferred on the real Solo-12, for now without the high-level policy that modifies the locomotion parameters online. The transfer led to relevant movements on the first try. This seems to indicate that the randomization during training combined with the fast dynamics of Solo-12 (low inertia actuators with fast bandwidth) are suitable for direct sim-to-real transfer. Fig. 14.2 highlights some snapshots of the policy in action.



(a) $t = 0\ s$ (b) $t = 1\ s$

(c) $t = 2\ s$ (d) $t = 3\ s$

(e) $t = 5\ s$ (f) $t = 7\ s$

(g) $t = 9\ s$ (h) $t = 29\ s$

Figure 14.2: Solo-12 walking indoors and outdoors with the presented RL controller.

14.5 Conclusion

This chapter reports simulation and experimental results that validate an end-to-end deep reinforcement learning architecture trained for the Solo-12 quadruped. It is though-provoking to compare the behavior obtained with such a data-based architecture compared to the model-based one developed throughout this thesis. It manages to reach similar forward velocities (≈ 1 m/s) than what was obtained in Section 11. Overall, the motion seems smoother with this architecture in the sense that legs and base movements are less sharp. The way the feet hit the floor is softer, which might be partially linked to the joint velocity targets being always zero. The base wobbles more, yet in a more natural or animal way compared to the data-based behavior which keeps the base close to the horizontal during the whole motion (reference for roll and pitch angles are kept at zero in the whole-body control). Even if it is still work in progress, the clear difference in obtained behaviors is already particularly interesting.

This work resulted in a paper that was submitted to Autonomous Robots, including a more thorough simulation study. While the network design, implementation and training were done by Michel Aractingi, my contribution consisted in regular discussions about control choices (state and action quantities, costs) exploiting my robotics knowledge and his expertise in machine learning, as well as in providing him with the low-level code and interface to run the neural network for real-world deployment.

Chapter 15

Conclusion

The work done in the framework of this thesis contributed to the locomotion of legged robots by developing a control architecture that is able to exploit the dynamical capabilities of a lightweight quadruped robot. We mainly explored the use of a model-based predictive control approach to reason over a prediction horizon. This type of approach relies on the knowledge of incoming events (contact locations and timings, disturbances, environment, ...) and on a model of the robot to predict the evolution of its state in the near future and ultimately take the best control decisions for a given set of criteria. In our case, this prediction was done using a centroidal model of the robot to find out which forces should be applied at contact locations to follow a reference state trajectory and handle disturbances. An instantaneous whole-body controller would then convert these contact forces and swinging feet trajectories into low-level commands. The accent was put on the possibilities offered by this architecture, both in terms of performances and versatility, through simulations and deployments on a real quadruped robot. We also emphasized how various aspects of the scheme could be augmented with or replaced by other approaches.

15.1 Contributions

As part of the Open Dynamic Robot Initiative project, we implemented a model-based control architecture to exploit the dynamical capabilities of the lightweight open-source quadruped robot Solo-12.

- We implemented an estimator that leverages sensor fusion to reliably estimate the state of the robot by combining information coming from the encoders and the inertia measurement unit. The low-accuracy non-drifting assessments coming from forward kinematics have been merged with highly-accurate IMU acceleration measurements that would drift when integrated alone, hence resulting in good quality non-drifting estimates of base position and linear velocity. To do so, we used either a cascade of complementary filters or a Kalman filter. That way, we showed that a reduced set of sensors and straightforward estimators can still provide state estimates that are good enough to obtain state-of-the-art quadruped locomotion.

- Then, we described how our use of binary matrices allowed us to obtain cyclic patterns of footsteps during locomotion with seamless transitions between different kinds of gaits. While the gait follows a predefined pattern by default, we presented

a way to adapt its characteristics online with a neural network trained by reinforcement learning. We also explained how a probabilistic approach was implemented to go beyond predefined contact timings and instead perform an online contact detection to detect and react to contact timing mismatches that can happen when moving on rough terrain.

- Based on contact information, we implemented a small set of heuristics to determine target footstep locations on a flat ground. We then presented how polynomial interpolation can be used to generate reference trajectories in position, velocity and acceleration to guide swinging feet from their current position to their next target on the ground, and how it could be linked with online contact detection. Moreover, we described an augmentation of this baseline scheme with mixed-integer programming to choose contact locations in 3D space from a set of known surfaces in the environment. This allowed to navigate complex terrains that would be out of range of the previous blind controller.

- Next, we outlined how to obtain reference state trajectories from the reference horizontal base velocity given by a user or a higher level controller, including a possible augmentation along the vertical axis to generate jumping motions. This trajectory is given along with footstep locations as inputs to a centroidal model predictive controller whose goal is to output desired contact forces that should be applied to track the reference. We presented and compared several approaches of increasing complexity to solve this optimal control problem. Starting from an initial convex quadratic program, we relieved some linearity assumptions and even included footstep locations as part of the optimization problem by using differential dynamic programming. Finally, we hinted at a way to include footstep timings as part of the optimization as well to allow a more complete control of the gait by the solver.

- Lastly, we described our successive implementations of whole-body controllers, from Task Space Inverse Dynamics that allows to perform task-oriented optimization-based inverse dynamics to a combination of inverse kinematics with a quadratic program. We presented several improvements to the initial architecture, such as removing base position and joint feedbacks, and a compensation term to take into account inertia effects due to leg movements. Those resulted in a diminution of base oscillations and thus increased the velocity the robot could reach.

These developments led to a number of implementation results:

- The initial baseline control architecture was validated with a deployment on a real Solo-12 quadruped after prior confirmation in simulation. We were able to obtain a robust trotting gait at low velocity (up to 0.5 m/s) on a flat ground. Despite the assumption of a flat ground, the robot was able to walk on small obstacles spread on the ground while handling reasonable pushes. The work done up to this point was presented at the IEEE ICRA conference 2021 [Léz+21].

- We validated the improvements done to the nominal control architecture to reduce base oscillations, which allowed to compare three variants of centroidal model predictive controllers with live experiments. Alongside an increase of the maximum velocity up to 1 m/s, we discussed the comparable performances that were obtained

with all MPC variants. The preliminary implementation of these MPCs in simulation by Thomas Corbères was presented at the IEEE ICRA conference 2021 [Cor+21]. The following improvements done to the control architecture to be able to deploy them on the real robot and their experimental comparison will be presented at the IEEE/RSJ IROS conference 2022 and have been published in the IEEE Robotics and Automation Letters [Léz+22].

- We reported how the planning scheme based on mixed-integer programming performed when combined with the baseline control architecture. With it, Solo-12 was able to overcome situations it would have failed in, such as stairs, stepping stones or bridge-like structures. In this work led by Fanny Risbourg and Thomas Corbères that will be presented at the IEEE/RSJ IROS conference 2022 [Ris+22], my contribution was in the integration in the control architecture and to the realization of the experiments.

- We showed in simulation that the robot benefits from online contact detection when walking on rough terrain, with better velocity tracking performances and reduced roll and pitch oscillations compared to the use of predefined timings for contact switches. This online detection was also exploited to validate jumping motions in simulation to detect touchdown of individual feet at the end of the ballistic phase.

Two other notable collaborations happened during this thesis:

- While exploring how reinforcement learning could be used to augment the model-based architecture, like what was presented to modify the gait characteristics online, an independent data-based architecture was trained and deployed on Solo-12. In this work led by Michel Aractingi, my contribution consisted in regular discussions about control choices to combine our respective expertise as well as in providing the low-level code to run the neural network into for simulation in PyBullet and real-world deployment. This work has been submitted to Autonomous Robots.

- The work on state estimation was further pursued by Médéric Fourmy with a more complex estimator based on factor graph with IMU and contact forces pre-integration, which is more computationally demanding but leads to better performances, notably for base position reconstruction over time. My contribution in this endeavor is mostly experimental to log relevant quantities and retrieve datasets by running the control architecture in specific scenarios. This work was presented at the IEEE ICRA conference 2021 [Fou+21].

15.2 Perspectives

The work we have done so far has laid the foundations of a model-based control architecture for quadruped robots. However, a lot of alleys have yet to be explored either to improve current performances (maximum velocity, robustness, ...) or to extend the scheme with new features. Here we present a few of the next projects that we wish to undertake.

15.2.1 Short term

We presented ongoing work in simulation showing that the tracking performances of the robot benefited from online contact detection when walking on rough terrain. A short-term goal would be to confirm this assessment on a real Solo-12 walking in an environment cluttered with steps of different height. Along this contact detection, early jumping motions have been shown by extending the reference state trajectory generation with a ballistic profile in position and velocity along the vertical axis. We also plan to deploy these motions on Solo-12 to exploit its dynamical capabilities. To do so, work has to be done on the pre-jump phase to ensure that the linear and angular velocities of the body at the start of the ballistic phase correspond to the planned ones. We also have to work on landing strategies to get an adaptive leg posture in flight phase that allows to absorb the impact at touchdown even if the base is unexpectedly tilted in roll or pitch. Open-loop replays of jumping trajectories have already shown that Solo-8 could jump on the spot up to 106 cm high [JMP]. Although Solo-12 is heavier due to 4 additional motors, batteries and electronics for power management, we can still expect impressive movements if its actuation is pushed to its limits.

15.2.2 Mid term

In a later stage, we would like to better explore how flight phases can be integrated into the gait for locomotion purpose. This would be done for instance to switch from a trot to a flying trot at higher velocity to go past the kinematic limit of the legs that caps the maximum velocity the robot can reach for a given gait period before reaching leg singulary. We could also try to find the right trade-off between duration of flight phases and change of the gait frequency. Having flight phases as part of the gait would also be a way to try out gallop, which is the gait pattern horses switch to for gait efficiency purpose (cost of transport) at high velocity [HT81].

Further investigating our initial implementation of inverse dynamics for whole-body control would also be a mid term goal. As we did not achieve stable locomotion when deploying this scheme on the robot, we decided to switch to an inverse kinematics strategy, combined with quadratic programming to ensure the equation of the dynamics is respected. Currently, this QP problem has to balance decisions that might be conflicting since the instantaneous decisions of the IK are done without knowing what has been decided by the MPC that works on a prediction horizon. This also adds a layer of complexity to the scheme with an approach that is less “all-in-one” than the task-space inverse dynamics which performs both steps at once. On top of that, the current scheme introduces additional parameters that have to be properly tuned so that the trade-off is not unbalanced toward either the MPC or the IK. We believe this topic would be worth digging into now that we have a better understanding of the whole control architecture.

15.2.3 Longer term

In the long term, broader aspects could be explored. As we have pointed out in Chapter 5 about state estimation, the onboard sensors are currently limited to joint encoders and an inertia measurement unit. With them, we can assess the base height, base orientation in roll and pitch and base linear and angular velocities, yet the absolute position of the base in the world and its yaw orientation are not observable. They can be reconstructed through forward geometry and velocity integration, but they will inevitably drift

as time goes by. Information about obstacles in the environment is inaccessible as well. Adding new exteroceptive sensors on Solo-12 would open new possibilities from a control perspective. Cameras and LIDARs installed on top of the base could provide rich information about the robot surroundings, either for localization or obstacle detection. Instead of relying on privileged knowledge, the contact surfaces that were used to handle complex situations in Chapter 7 could instead be detected on the fly. It would also open the way for simultaneous localization and mapping, and a whole range of navigation tasks by acting on the reference base velocity with a higher level position controller. Without going that far, other simpler exteroceptive sensors could be integrated. For instance, a low-resolution camera attached under the body and pointing downwards could contribute to the base velocity estimation through a measure of optical flux, as long as the texture of the ground is not overly homogenous.

On another aspect, the rise of data-based methods in recent years and the increasingly impressive results obtained with them are thought-provoking. Even if end-to-end machine learning approaches now achieve robust and efficient locomotion, one can still wonder how the principles and techniques developed in this field could be retrieved to contribute to existing model-based architectures. Their capacity to extract relevant information from datasets could potentially allow to capture effects that would be hard to model or which would be untractable for some model-based architectures. Some approaches already try to bridge the gap between more traditional control viewpoints and those novel machine learning perspectives. To go beyond simplified models and kinodynamic constraints that are often non-differentiable, Mitchell [Mit+20] captures a statistical representation of feasible joint configurations to form a structured latent space. Through the use of semantic indicators and learned classifiers, constraints are made differentiable and performing motion optimization amounts to finding a trajectory in this latent configuration space. Fawcett [Faw+22] investigates the use of data-enabled predictive control to capture nonlinear information about the classic lumped-mass model while avoiding linearization. It does so through a data collection phase which leads to the construction of Hankel matrices that contain implicit information about the dynamics of the system. In a way, such methods circumvent some of the interrogations evoked in this thesis about choosing the right model with the relevant assumptions since there they are directly encoded in the network or matrices. Beyond replacing previously hand-formulated models in our centroidal MPC or WBC, data-based methods bring adaptivity to other elements. We have briefly presented a way to implement an adaptive gait pattern with reinforcement learning. It could be applied to other points of the architecture to augment our heuristics in the estimator or footsteps planner, to provide adaptive weights in the MPC and WBC solvers or to tune joint impedance gains on the fly.

Finally, seeing the complexity brought by a two-stage architecture with a centroidal MPC combined with a WBC, especially to ensure their respective decisions are well taken into account, it would be worth investigating all-in-one whole-body predictive control. It has already been achieved on other platforms such as ANYmal [Neu+18], although computational efficiency is paramount to reach real-time performances due to the size of the optimization problem. Some data-based methods that we talked about earlier might be a way to fasten the solving process, either by encapsulating part of the model or by providing a warm-start trajectory to start the optimization close from the optimal solution [Man+18].

Bibliography

[Ace+17] Bernardo Aceituno-Cabezas et al. “Simultaneous contact, gait, and motion planning for robust multilegged locomotion via mixed-integer convex optimization”. In: *IEEE Robotics and Automation Letters* 3.3 (2017), pp. 2531–2538 (cit. on p. 14).

[Aja+13] Mostafa Ajallooeian, Soha Pouya, Alexander Sproewitz, and Auke J Ijspeert. “Central pattern generators augmented with virtual model control for quadruped rough terrain locomotion”. In: *2013 IEEE international conference on robotics and automation*. IEEE. 2013, pp. 3321–3328 (cit. on p. 12).

[Alt+01] Richard Altendorfer et al. “Rhex: A biologically inspired hexapod runner”. In: *Autonomous Robots* 11.3 (2001), pp. 207–213 (cit. on p. 5).

[Ara+21] Michel Aractingi, Pierre-Alexandre Leziart, Thomas Flayols, Julien Perez, Tomi Silander, and Philippe Souères. “Learning to Adapt the Trotting Gait of the Solo Quadruped”. working paper or preprint. Oct. 2021. URL: https://hal.laas.fr/hal-03409682 (cit. on pp. 48–50).

[Ara+22] Michel Aractingi, Pierre-Alexandre Léziart, Thomas Flayols, Julien Perez, Tomi Silander, and Philippe Souères. “Controlling the Solo12 Quadruped Robot with Deep Reinforcement Learning”. Submitted to Autonomous Robots. 2022 (cit. on p. 9).

[AH96] Keisuke Arikawa and Shigeo Hirose. “Development of quadruped walking robot TITAN-VIII”. In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems. IROS’96*. Vol. 1. IEEE. 1996, pp. 208–214 (cit. on p. 3).

[Bai+01] Sean A Bailey, Jorge G Cham, Mark R Cutkosky, and Robert J Full. “Comparing the locomotion dynamics of the cockroach and a shape deposition manufactured biomimetic hexapod”. In: *Experimental Robotics VII*. Springer, 2001, pp. 239–248 (cit. on p. 7).

[Bar+13] Victor Barasuol, Jonas Buchli, Claudio Semini, Marco Frigerio, Edson R De Pieri, and Darwin G Caldwell. “A reactive controller framework for quadrupedal locomotion on challenging terrain”. In: *2013 IEEE International Conference on Robotics and Automation*. IEEE. 2013, pp. 2554–2561 (cit. on p. 12).

[Bar87] Stephen J Bartholet. “The evolution of Odetics walking machine technology”. In: *Mobile Robots I*. Vol. 727. SPIE. 1987, pp. 25–31 (cit. on p. 2).

[Bel+17] C Dario Bellicoso, Fabian Jenelten, Péter Fankhauser, Christian Gehring, Jemin Hwangbo, and Marco Hutter. "Dynamic locomotion and whole-body control for quadrupedal robots". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 3359–3365 (cit. on p. 15).

[Bel+18] C Dario Bellicoso, Fabian Jenelten, Christian Gehring, and Marco Hutter. "Dynamic locomotion through online nonlinear motion optimization for quadrupedal robots". In: *IEEE Robotics and Automation Letters* 3.3 (2018), pp. 2261–2268 (cit. on p. 15).

[Bel+19] C Dario Bellicoso et al. "Alma-articulated locomotion and manipulation for a torque-controllable robot". In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 8477–8483 (cit. on p. 15).

[Bha+19] Shounak Bhattacharya et al. "Learning active spine behaviors for dynamic and efficient locomotion in quadruped robots". In: *2019 28th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*. IEEE. 2019, pp. 1–6 (cit. on p. 18).

[BWK17] Gerardo Bledt, Patrick M Wensing, and Sangbae Kim. "Policy-regularized model predictive control to stabilize diverse quadrupedal gaits for the MIT cheetah". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 4102–4109 (cit. on p. 15).

[Blo+13] Michael Bloesch, Christian Gehring, Péter Fankhauser, Marco Hutter, Mark A Hoepflinger, and Roland Siegwart. "State estimation for legged robots on unstable and slippery terrain". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2013, pp. 6058–6064 (cit. on p. 115).

[BFH19] Martim Brandao, Maurice Fallon, and Ioannis Havoutis. "Multi-controller multi-objective locomotion planning for legged robots". In: *2019 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE. 2019, pp. 4714–4721 (cit. on p. 15).

[Bro89] Rodney A Brooks. "A robot that walks; emergent behaviors from a carefully evolved network". In: *Neural computation* 1.2 (1989), pp. 253–262 (cit. on p. 7).

[Buc+09] Jonas Buchli, Mrinal Kalakrishnan, Michael Mistry, Peter Pastor, and Stefan Schaal. "Compliant quadruped locomotion over rough terrain". In: *2009 IEEE/RSJ international conference on Intelligent robots and systems*. IEEE. 2009, pp. 814–820 (cit. on p. 15).

[Bus04] Samuel R Buss. "Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods". In: *IEEE Journal of Robotics and Automation* 17 (2004) (cit. on p. 103).

[CBM17] Justin Carpentier, Rohan Budhiraja, and Nicolas Mansard. "Learning feasibility constraints for multi-contact locomotion of legged robots". In: *Robotics: Science and Systems*. 2017 (cit. on p. 73).

[Car+16] Justin Carpentier, Steve Tonneau, Maximilien Naveau, Olivier Stasse, and Nicolas Mansard. "A versatile and efficient pattern generator for generalized legged locomotion". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2016, pp. 3555–3561 (cit. on p. 4).

[CVM+19] Justin Carpentier, Florian Valenza, Nicolas Mansard, et al. *Pinocchio: fast forward and inverse dynamics for poly-articulated systems*. https://stack-of-tasks.github.io/pinocchio. 2015–2019 (cit. on pp. 104, 114).

[Car+19] Justin Carpentier et al. "The Pinocchio C++ library: A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives". In: *2019 IEEE/SICE International Symposium on System Integration (SII)*. IEEE. 2019, pp. 614–619 (cit. on p. 114).

[Che+03] Christine Chevallereau, Gabriel Abba, Franck Plestan, Eric Westervelt, Carlos Canudas de Wit, Jessy Grizzle, et al. "Rabbit: A testbed for advanced control theory". In: *IEEE Control Systems Magazine* 23.5 (2003), pp. 57–79 (cit. on p. 13).

[Cor+21] Thomas Corbères et al. "Comparison of predictive controllers for locomotion and balance recovery of quadruped robots". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2021, pp. 5021–5027 (cit. on pp. 9, 122, 124, 126, 153).

[CB20] Erwin Coumans and Yunfei Bai. *PyBullet, a Python module for physics simulation for games, robotics and machine learning*. http://pybullet.org. 2016–2020 (cit. on p. 114).

[CB21] Erwin Coumans and Yunfei Bai. *PyBullet, a Python module for physics simulation for games, robotics and machine learning*. http://pybullet.org. 2016–2021 (cit. on p. 145).

[Cox70] W Cox. "Big muskie". In: *The Ohio State Engineer* (1970), pp. 25–52 (cit. on p. 2).

[Cru+91] Holk Cruse, Jeffrey Dean, U Muller, and Josef Schmitz. "The stick insect as a walking robot". In: *Fifth International Conference on Advanced Robotics' Robots in Unstructured Environments*. IEEE. 1991, pp. 936–940 (cit. on p. 7).

[Di +18] Jared Di Carlo, Patrick M Wensing, Benjamin Katz, Gerardo Bledt, and Sangbae Kim. "Dynamic locomotion in the mit cheetah 3 through convex model-predictive control". In: *2018 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE. 2018, pp. 1–9 (cit. on p. 15).

[DLG] DLG. *Dynamic Locomotion Group*. Accessed: 2022-07-06. URL: https://dlg.is.mpg.de/ (visited on 07/06/2022) (cit. on p. 19).

[Dos97] Zdenek Dostál. "Box constrained quadratic programming with proportioning and projections". In: *SIAM Journal on Optimization* 7.3 (1997), pp. 871–887 (cit. on p. 105).

[Dyna] Boston Dynamics. *Spot Autonomous Navigation*. Accessed: 2021-03-08. URL: https://www.youtube.com/watch?v=Ve9kWX_KXus (visited on 03/08/2021) (cit. on p. 6).

[Dynb] Boston Dynamics. *Spot's On It*. Accessed: 2022-06-29. URL: https://www.youtube.com/watch?v=7atZfX85nd4 (visited on 06/29/2022) (cit. on p. 6).

[EGK08] Markus Eich, Felix Grimminger, and Frank Kirchner. "A versatile stair-climbing robot for search and rescue applications". In: *2008 IEEE international workshop on safety, security and rescue robotics*. IEEE. 2008, pp. 35–40 (cit. on p. 5).

[Fah+19] Shamel Fahmi, Carlos Mastalli, Michele Focchi, and Claudio Semini. "Passive whole-body control for quadruped robots: Experimental validation over challenging terrain". In: *IEEE Robotics and Automation Letters* 4.3 (2019), pp. 2553–2560 (cit. on p. 16).

[Fan+18] Péter Fankhauser, Marko Bjelonic, C Dario Bellicoso, Takahiro Miki, and Marco Hutter. "Robust rough-terrain locomotion with a quadrupedal robot". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 5761–5768 (cit. on p. 14).

[Far+17] Farbod Farshidian, Michael Neunert, Alexander W Winkler, Gonzalo Rey, and Jonas Buchli. "An efficient optimal planning and control framework for quadrupedal locomotion". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 93–100 (cit. on p. 14).

[Faw+22] Randall T Fawcett, Kereshmeh Afsari, Aaron D Ames, and Kaveh Akbari Hamed. "Toward a Data-Driven Template Model for Quadrupedal Locomotion". In: *IEEE Robotics and Automation Letters* (2022) (cit. on pp. 16, 155).

[Fer+17] Pierre Fernbach, Steve Tonneau, Andrea Del Prete, and Michel Taix. "A kinodynamic steering-method for legged multi-contact locomotion". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 3701–3707 (cit. on p. 14).

[Foc+13] Michele Focchi, Victor Barasuol, Ioannis Havoutis, Jonas Buchli, Claudio Semini, and Darwin G Caldwell. "Local reflex generation for obstacle negotiation in quadrupedal locomotion". In: *Nature-Inspired Mobile Robotics*. World Scientific, 2013, pp. 443–450 (cit. on p. 12).

[Fou+21] Médéric Fourmy, Thomas Flayols, Pierre-Alexandre Léziart, Nicolas Mansard, and Joan Solà. "Contact Forces Preintegration for Estimation in Legged Robotics using Factor Graphs". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. 2021, pp. 1372–1378 (cit. on pp. 9, 153).

[Fuk+19] Takahiro Fukui, Hisamu Fujisawa, Kotaro Otaka, and Yasuhiro Fukuoka. "Autonomous gait transition and galloping over unperceived obstacles of a quadruped robot with CPG modulated by vestibular feedback". In: *Robotics and Autonomous Systems* 111 (2019), pp. 1–19 (cit. on p. 12).

[Geh+16] Christian Gehring et al. "Practice makes perfect: An optimization-based approach to controlling agile motions for a quadruped robot". In: *IEEE Robotics & Automation Magazine* 23.1 (2016), pp. 34–43 (cit. on p. 17).

[Geh+13] Christian Gehring, Stelian Coros, Marco Hutter, Michael Bloesch, Markus A Hoepflinger, and Roland Siegwart. "Control of dynamic gaits for a quadrupedal robot". In: *2013 IEEE international conference on Robotics and automation*. IEEE. 2013, pp. 3287–3292 (cit. on p. 12).

[Geh+21] Christian Gehring et al. "ANYmal in the field: Solving industrial inspection of an offshore HVDC platform with a quadrupedal robot". In: *Field and Service Robotics*. Springer. 2021, pp. 247–260 (cit. on p. 6).

[Gos99] Ambarish Goswami. "Postural stability of biped robots and the foot-rotation indicator (FRI) point". In: *The International Journal of Robotics Research* 18.6 (1999), pp. 523–533 (cit. on p. 13).

[Gri+20] F. Grimminger et al. "An Open Torque-Controlled Modular Robot Architecture for Legged Locomotion Research". In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 3650–3657. DOI: 10.1109/LRA.2020.2976639 (cit. on pp. 19, 20, 123).

[GT] GT. *Gepetto Team*. Accessed: 2022-07-06. URL: https://gepettoweb.laas.fr/ (visited on 07/06/2022) (cit. on p. 19).

[HKP20] Kaveh Akbari Hamed, Jeeseop Kim, and Abhishek Pandala. "Quadrupedal locomotion via event-based predictive control and QP-based virtual constraints". In: *IEEE Robotics and Automation Letters* 5.3 (2020), pp. 4463–4470 (cit. on p. 16).

[HWM20] Nils Hareng, Bruno Watier, and Franck Multon. "Prediction of plausible locomotion using nonlinear kinematic optimization". In: *Computer Methods in Biomechanics and Biomedical Engineering* 23.sup1 (2020), S136–S138 (cit. on p. 7).

[Hav+13] Ioannis Havoutis, Jesus Ortiz, Stephane Bazeille, Victor Barasuol, Claudio Semini, and Darwin G Caldwell. "Onboard perception-based trotting and crawling with the hydraulic quadruped robot (HyQ)". In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2013, pp. 6052–6057 (cit. on p. 14).

[Hei+17] Elco Heijmink, Andreea Radulescu, Brahayam Ponton, Victor Barasuol, Darwin G Caldwell, and Claudio Semini. "Learning optimal gait parameters and impedance profiles for legged locomotion". In: *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*. IEEE. 2017, pp. 339–346 (cit. on p. 17).

[HT81] Donald F Hoyt and C Richard Taylor. "Gait and the energetics of locomotion in horses". In: *Nature* 292.5820 (1981), pp. 239–240 (cit. on pp. 7, 44, 154).

[HLH18] Jemin Hwangbo, Joonho Lee, and Marco Hutter. "Per-contact iteration method for solving contact dynamics". In: *IEEE Robotics and Automation Letters* 3.2 (2018), pp. 895–902. URL: www.raisim.com (cit. on p. 145).

[JIC19] Deepali Jain, Atil Iscen, and Ken Caluwaerts. "Hierarchical reinforcement learning for quadruped locomotion". In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2019, pp. 7551–7557 (cit. on p. 18).

[JIC20] Deepali Jain, Atil Iscen, and Ken Caluwaerts. "From pixels to legs: Hierarchical learning of quadruped locomotion". In: *arXiv preprint arXiv:2011.11722* (2020) (cit. on p. 18).

[Ji+22] Gwanghyeon Ji, Juhyeok Mun, Hyeongjun Kim, and Jemin Hwangbo. "Concurrent Training of a Control Policy and a State Estimator for Dynamic and Robust Legged Locomotion". In: *IEEE Robotics and Automation Letters* 7.2 (2022), pp. 4630–4637 (cit. on pp. 18, 120).

[JMP] JMP. *Executing high jumps for testing the impact resistance of the Solo8 hardware*. Accessed: 2022-07-08. URL: https://www.youtube.com/watch?v=javHeKRKbAc (visited on 07/08/2022) (cit. on pp. 24, 154).

[KE08] Shuuji Kajita and Bernard Espiau. *Legged Robot*. Springer Handbook, 2008 (cit. on p. 73).

[Kal+11] Mrinal Kalakrishnan, Jonas Buchli, Peter Pastor, Michael Mistry, and Stefan Schaal. "Learning, planning, and control for quadruped locomotion over challenging terrain". In: *The International Journal of Robotics Research* 30.2 (2011), pp. 236–258 (cit. on p. 13).

[Kan+04] Fumio Kanehiro et al. "Locomotion planning of humanoid robots to pass through narrow spaces". In: *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*. Vol. 1. IEEE. 2004, pp. 604–609 (cit. on p. 4).

[KV02] Dusko Katic and Miomir Vukobratovic. "Intelligent soft-computing paradigms for humanoid robots". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. Vol. 3. IEEE. 2002, pp. 2533–2538 (cit. on p. 4).

[Kem98] AL Kemurdjian. "Planet rover as an object of the engineering design work". In: *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No. 98CH36146)*. Vol. 1. IEEE. 1998, pp. 140–145 (cit. on p. 3).

[Ker+04] T Kerscher, J Albiez, JM Zoellner, and R Dillmann. "Airinsect-a new innovative biological inspired six-legged walking machine driven by fluidic muscles". In: *Proceedings of IAS 8, The 8th Conference on Intelligent Autonomous Systems, Amsterdam, The Netherlands*. 2004 (cit. on p. 7).

[Kim+19] Donghyun Kim, Jared Di Carlo, Benjamin Katz, Gerardo Bledt, and Sangbae Kim. "Highly dynamic quadruped locomotion via whole-body impulse control and model predictive control". In: *arXiv preprint arXiv:1909.06586* (2019) (cit. on pp. 16, 55, 57, 101, 102, 124).

[Kim+07] Sangbae Kim, Matthew Spenko, Salomon Trujillo, Barrett Heyneman, Virgilio Mattoli, and Mark R Cutkosky. "Whole body adhesion: hierarchical, directional and distributed control of adhesive forces for a climbing robot". In: *Proceedings 2007 IEEE International Conference on Robotics and Automation*. IEEE. 2007, pp. 1268–1273 (cit. on pp. 7, 120).

[KB19] Tobias Klamt and Sven Behnke. "Towards learning abstract representations for locomotion planning in high-dimensional state spaces". In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 922–928 (cit. on p. 17).

[KRN08] J Zico Kolter, Mike P Rodgers, and Andrew Y Ng. "A control architecture for quadruped locomotion over rough terrain". In: *2008 IEEE International Conference on Robotics and Automation*. IEEE. 2008, pp. 811–818 (cit. on p. 14).

[Kum+21] Ashish Kumar, Zipeng Fu, Deepak Pathak, and Jitendra Malik. "Rma: Rapid motor adaptation for legged robots". In: *arXiv preprint arXiv:2107.04034* (2021) (cit. on p. 108).

[LAA] LAAS. *Laboratory of Architecture and Analysis of Systems LAAS-CNRS*. Accessed: 2022-07-06. URL: https://www.laas.fr/public/ (visited on 07/06/2022) (cit. on p. 19).

[LP19] Jeong Hoon Lee and Jong Hyeon Park. "Time-dependent genetic algorithm and its application to quadruped's locomotion". In: *Robotics and Autonomous Systems* 112 (2019), pp. 60–71 (cit. on p. 17).

[Léz+22] Pierre-Alexandre Léziart, Thomas Corbères, Thomas Flayols, Steve Tonneau, Nicolas Mansard, and Philippe Souères. "Improved Control Scheme for the Solo Quadruped and Experimental Comparison of Model Predictive Controllers". In: *IEEE Robotics and Automation Letters (RA-L)* (2022) (cit. on pp. 9, 153).

[Léz+21] Pierre-Alexandre Léziart, Thomas Flayols, Felix Grimminger, Nicolas Mansard, and Philippe Souères. "Implementation of a Reactive Walking Controller for the New Open-Hardware Quadruped Solo-12". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2021, pp. 5007–5013 (cit. on pp. 9, 152).

[LT07] Hun-ok Lim and Atsuo Takanishi. "Biped walking robots created at Waseda University: WL and WABIAN family". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 365.1850 (2007), pp. 49–64 (cit. on p. 3).

[LSP15] Xin Liu, Claudio Semini, and Ioannis Poulakakis. "Active compliance hybrid zero dynamics control of bounding on HyQ". In: *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE. 2015, pp. 1047–1052 (cit. on p. 13).

[Mag+19] Octavio Antonio Villarreal Magana et al. "Fast and continuous foothold adaptation for dynamic locomotion through cnns". In: *IEEE Robotics and Automation Letters* 4.2 (2019), pp. 2140–2147 (cit. on pp. 17, 126).

[Man+18] Nicolas Mansard, Andrea DelPrete, Mathieu Geisert, Steve Tonneau, and Olivier Stasse. "Using a memory of motion to efficiently warm-start a nonlinear predictive controller". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 2986–2993 (cit. on p. 155).

[MSW20] Isabelle Maroger, Olivier Stasse, and Bruno Watier. "Walking human trajectory models and their application to humanoid robot locomotion". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 3465–3472 (cit. on p. 6).

[Mas+19] Elisa Massi et al. "Combining evolutionary and adaptive control strategies for quadruped robotic locomotion". In: *Frontiers in Neurorobotics* 13 (2019), p. 71 (cit. on p. 12).

[Mas+15] Carlos Mastalli, Ioannis Havoutis, Alexander W Winkler, Darwin G Caldwell, and Claudio Semini. "On-line and on-board planning and perception for quadrupedal locomotion". In: *2015 IEEE International Conference on Technologies for Practical Robot Applications (TePRA)*. IEEE. 2015, pp. 1–7 (cit. on p. 14).

[Mas+20] Carlos Mastalli et al. “Crocoddyl: An efficient and versatile framework for multi-contact optimal control”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 2536–2542 (cit. on pp. 87, 90, 122).

[MKK02] Osamu Matsumoto, Shuuji Kajita, and Kiyoshi Komoriya. “Flexible locomotion control of a self-contained biped leg-wheeled system”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. Vol. 3. IEEE. 2002, pp. 2599–2604 (cit. on p. 5).

[McG+90] Tad McGeer et al. “Passive dynamic walking”. In: *Int. J. Robotics Res.* 9.2 (1990), pp. 62–82 (cit. on p. 3).

[McG+85] RB McGhee, DE Orin, DR Pugh, and MR Patterson. “A hierarchically structured system for computer control of a hexapod walking machine”. In: *Theory and Practice of Robots and Manipulators*. Springer, 1985, pp. 375–381 (cit. on p. 2).

[MGC] MGCG. *Motion Generation and Control Group*. Accessed: 2022-07-06. URL: https://mg.is.tuebingen.mpg.de/ (visited on 07/06/2022) (cit. on p. 19).

[MIT] MIT. *Cheetah-Software repository*. Accessed: 2020-03-02. MIT Biomimetics. URL: https://github.com/mit-biomimetics/Cheetah-Software (visited on 03/02/2021) (cit. on p. 118).

[Mit+20] Alexander L Mitchell et al. “First steps: Latent-space control with semantic constraints for quadruped locomotion”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 5343–5350 (cit. on pp. 16, 155).

[MML] MML. *Machines in Motion Laboratory*. Accessed: 2022-07-06. URL: https://wp.nyu.edu/machinesinmotion (visited on 07/06/2022) (cit. on p. 19).

[Mos68] R Mosher. “Test and evaluation of a versatile walking truck”. In: *Proceedings of Off-Road Mobility Research Symposium, Washington DC, 1968*. 1968, pp. 359–379 (cit. on p. 2).

[MPI] MPI. *Max-Planck Institute for Intelligent System*. Accessed: 2022-07-06. URL: https://is.tuebingen.mpg.de/ (visited on 07/06/2022) (cit. on p. 19).

[Muy87] Eadweard Muybridge. *Animal locomotion*. Vol. 534. Da Capo Press, 1887 (cit. on p. 7).

[NAV] NAV. *Naver Labs Europe*. Accessed: 2022-07-19. URL: https://europe.naverlabs.com/ (visited on 07/19/2022) (cit. on p. 145).

[Net+09] Armando Alves Neto, Douglas Guimarães Macharet, Víctor Costa da Silva Campos, and Mario Fernando Montenegro Campos. “Adaptive complementary filtering algorithm for mobile robot localization”. In: *Journal of the Brazilian Computer Society* 15.3 (2009), pp. 19–31 (cit. on p. 36).

[Neu+18] Michael Neunert et al. “Whole-body nonlinear model predictive control through contacts for quadrupeds”. In: *IEEE Robotics and Automation Letters* 3.3 (2018), pp. 1458–1465 (cit. on pp. 15, 155).

[NW06] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006 (cit. on p. 105).

[Nus+85] Martha C Nussbaum et al. *Aristotle's De motu animalium: text with translation, commentary, and interpretive essays*. Princeton University Press, 1985 (cit. on p. 7).

[NYU] NYU. *New York University's Tandon School of Engineering*. Accessed: 2022-07-06. URL: https://engineering.nyu.edu/ (visited on 07/06/2022) (cit. on p. 19).

[ODR] ODRI. *Open Dynamic Robot Initiative*. Accessed: 2022-07-06. URL: https://open-dynamic-robot-initiative.github.io/ (visited on 07/06/2022) (cit. on pp. 19, 21–23, 113).

[OGL13] David E. Orin, Ambarish Goswami, and Sung-Hee Lee. "Centroidal dynamics of a humanoid robot". In: *Autonomous Robots* 35.2-3 (Oct. 2013), pp. 161–176. ISSN: 0929-5593. DOI: 10.1007/s10514-013-9341-4. URL: http://link.springer.com/10.1007/s10514-013-9341-4 (cit. on p. 73).

[Ori+79] David E Orin, RB McGhee, M Vukobratović, and G Hartoch. "Kinematic and kinetic analysis of open-chain linkages utilizing Newton-Euler methods". In: *Mathematical Biosciences* 43.1-2 (1979), pp. 107–130 (cit. on p. 2).

[PFB14] Brahayam Pontón, Farbod Farshidian, and Jonas Buchli. "Learning compliant locomotion on a quadruped robot". In: *IROS Workshop (Ed.), Compliant manipulation: Challenges in learning and control*. Citeseer. 2014 (cit. on p. 17).

[PGH05] Marko B Popovic, Ambarish Goswami, and Hugh Herr. "Ground reference points in legged locomotion: Definitions, biological trajectories and control implications". In: *The international journal of robotics research* 24.12 (2005), pp. 1013–1032 (cit. on p. 13).

[PDP97] Jerry Pratt, Peter Dilworth, and Gill Pratt. "Virtual model control of a bipedal walking robot". In: *Proceedings of International Conference on Robotics and Automation*. Vol. 1. IEEE. 1997, pp. 193–198 (cit. on p. 12).

[Rai86] Marc H Raibert. "Legged robots". In: *Communications of the ACM* 29.6 (1986), pp. 499–514 (cit. on pp. 1, 3).

[Rai+08] Marc Raibert, Kevin Blankespoor, Gabriel Nelson, and Rob Playter. "Bigdog, the rough-terrain quadruped robot". In: *IFAC Proceedings Volumes* 41.2 (2008), pp. 10822–10825 (cit. on p. 6).

[RCB86] Marc Raibert, Michael Chepponis, and HBJR Brown. "Running on four legs as though they were one". In: *IEEE Journal on Robotics and Automation* 2.2 (1986), pp. 70–82 (cit. on p. 12).

[RCS] RCSF. *Robotics Central Scientific Facility*. Accessed: 2022-07-06. URL: https://is.tuebingen.mpg.de/en/robotics (visited on 07/06/2022) (cit. on p. 19).

[RI08] Ludovic Righetti and Auke Jan Ijspeert. "Pattern generators with sensory feedback for the control of quadruped locomotion". In: *2008 IEEE International Conference on Robotics and Automation*. IEEE. 2008, pp. 819–824 (cit. on p. 11).

[RIS] RIS. *risbourg_corberes_2022 project page*. Accessed: 2022-07-13. URL: https://gepettoweb.laas.fr/articles/risbourg_corberes_2022.html (visited on 07/13/2022) (cit. on pp. 131, 132).

[Ris+22] Fanny Risbourg, Thomas Corbères, Pierre-Alexandre Léziart, Thomas Flayols, Nicolas Mansard, and Steve Tonneau. "Real time footstep planning and control of the Solo quadruped robot in 3D environments". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2022 (cit. on pp. 9, 68, 70, 131, 153).

[Rob] Unitree Robotics. *200 pieces: The world's largest quadruped robot cluster performance*. Accessed: 2022-06-29. URL: https://www.youtube.com/watch?v=ON1WKGAnbGk (visited on 06/29/2022) (cit. on p. 6).

[Ros94] Mark E Rosheim. *Robot evolution: the development of anthrobotics*. John Wiley & Sons, 1994 (cit. on p. 1).

[Sak+02] Yoshiaki Sakagami, Ryujin Watanabe, Chiaki Aoyama, Shinichi Matsunaga, Nobuo Higaki, and Kikuo Fujimura. "The intelligent ASIMO: System overview and integration". In: *IEEE/RSJ international conference on intelligent robots and systems*. Vol. 3. IEEE. 2002, pp. 2478–2483 (cit. on p. 4).

[SB04] Philippe Sardain and Guy Bessonnet. "Forces acting on a biped robot. Center of pressure-zero moment point". In: *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 34.5 (2004), pp. 630–637 (cit. on p. 13).

[Sch+17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017) (cit. on p. 146).

[ST07] Manuel F Silva and JA Tenreiro Machado. "A historical perspective of legged robots". In: *Journal of Vibration and Control* 13.9-10 (2007), pp. 1447–1486 (cit. on p. 1).

[Sin+19] Abhik Singla et al. "Realizing learned quadruped locomotion behaviors through kinematic motion primitives". In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 7434–7440 (cit. on p. 17).

[Son+21] Daeun Song et al. "Solving Footstep Planning as a Feasibility Problem using L1-norm Minimization". In: *IEEE RA-L* (2021) (cit. on p. 126).

[SW89] Shin-Min Song and Kenneth J Waldron. *Machines that walk: the adaptive suspension vehicle*. MIT press, 1989 (cit. on pp. 1, 2).

[SHV+06] Mark W Spong, Seth Hutchinson, Mathukumalli Vidyasagar, et al. *Robot modeling and control*. Vol. 3. Wiley New York, 2006 (cit. on pp. 101, 124).

[Ste+20] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd. "OSQP: an operator splitting solver for quadratic programs". In: *Mathematical Programming Computation* (2020). DOI: 10.1007/s12532-020-00179-2. URL: https://doi.org/10.1007/s12532-020-00179-2 (cit. on pp. 114, 122).

[Str18] Steven H Strogatz. *Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering*. CRC press, 2018 (cit. on p. 13).

[Sun+18] Tao Sun, Donghao Shao, Zhendong Dai, and Poramate Manoonpong. "Adaptive neural control for self-organized locomotion and obstacle negotiation of quadruped robots". In: *2018 27th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*. IEEE. 2018, pp. 1081–1086 (cit. on p. 17).

[Suz+21] Shura Suzuki, Takeshi Kano, Auke J Ijspeert, and Akio Ishiguro. "Sprawling quadruped robot driven by decentralized control with cross-coupled sensory feedback between legs and trunk". In: *Frontiers in Neurorobotics* (2021), p. 116 (cit. on p. 12).

[Tag94] Gentaro Taga. "Emergence of bipedal locomotion through entrainment among the neuro-musculo-skeletal system and the environment". In: *Physica D: Nonlinear Phenomena* 75.1-3 (1994), pp. 190–208 (cit. on p. 11).

[Tag95] Gentaro Taga. "A model of the neuro-musculo-skeletal system for human locomotion". In: *Biological cybernetics* 73.2 (1995), pp. 97–111 (cit. on p. 11).

[Tak+85] Atsuo Takanishi, Masami Ishida, Yoshiaki Yamazaki, and Ichiro Kato. "The realization of dynamic walking by the biped walking robot wl-10 rd". In: *Journal of the Robotics Society of Japan* 3.4 (1985), pp. 325–336 (cit. on p. 3).

[Tan+18] Jie Tan et al. "Sim-to-real: Learning agile locomotion for quadruped robots". In: *arXiv preprint arXiv:1804.10332* (2018) (cit. on p. 17).

[Thr85] Meredith Wooldridge Thring. "Robots and telechirs". In: (1985) (cit. on pp. 1, 2, 6).

[TVM19] Carlo Tiseo, Sethu Vijayakumar, and Michael Mistry. "Analytic model for quadruped locomotion task-space planning". In: *2019 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. IEEE. 2019, pp. 5301–5304 (cit. on p. 13).

[Tra+22] Marco Tranzatto et al. "CERBERUS in the DARPA Subterranean Challenge". In: *Science Robotics* 7.66 (2022), eabp9742 (cit. on p. 6).

[TTO01] Katsuyoshi Tsujita, Kazuo Tsuchiya, and Ahmet Onat. "Adaptive gait pattern control of a quadruped locomotion robot". In: *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No. 01CH37180)*. Vol. 4. IEEE. 2001, pp. 2318–2325 (cit. on p. 11).

[TXA] TXAS. *Texas Instruments Evaluation Board*. Accessed: 2022-07-06. URL: https://github.com/open-dynamic-robot-initiative/open%5C_robot%5C_actuator%5C_hardware/blob/master/electronics/ti%5C_electronics/README.md%5C#texas-instruments-evaluation-board-electronics (visited on 07/06/2022) (cit. on p. 19).

[Ugu+13] Barkan Ugurlu, Ioannis Havoutis, Claudio Semini, and Darwin G Caldwell. "Dynamic trot-walking with the hydraulic quadruped robot—HyQ: Analytical trajectory generation and active compliance control". In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2013, pp. 6044–6051 (cit. on p. 13).

[Vel+98] Andrija Velimirovic, Maja Velimirovic, Vincent Hugel, Amine Iles, and Pierre Blazevic. "A new architecture of robot with" wheels-with-legs"(WWL)". In: *AMC'98-Coimbra. 1998 5th International Workshop on Advanced Motion Control. Proceedings (Cat. No. 98TH8354)*. IEEE. 1998, pp. 434–439 (cit. on p. 5).

[Vil+20] Octavio Villarreal, Victor Barasuol, Patrick M Wensing, Darwin G Caldwell, and Claudio Semini. "MPC-based controller with terrain insight for dynamic legged locomotion". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 2436–2442 (cit. on p. 16).

[Vir+19] Yvain de Viragh, Marko Bjelonic, C Dario Bellicoso, Fabian Jenelten, and Marco Hutter. "Trajectory optimization for wheeled-legged quadrupedal robots using linearized zmp constraints". In: *IEEE Robotics and Automation Letters* 4.2 (2019), pp. 1633–1640 (cit. on p. 13).

[VB04] Miomir Vukobratović and Branislav Borovac. "Zero-moment point—thirty five years of its life". In: *International journal of humanoid robotics* 1.01 (2004), pp. 157–173 (cit. on p. 13).

[Win+15] Alexander W Winkler, Carlos Mastalli, Ioannis Havoutis, Michele Focchi, Darwin G Caldwell, and Claudio Semini. "Planning and execution of dynamic whole-body locomotion for a hydraulic quadruped on challenging terrain". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2015, pp. 5148–5154 (cit. on pp. 12, 14).

[Wüt+20] Manuel Wüthrich et al. "Trifinger: An open-source robot for learning dexterity". In: *arXiv preprint arXiv:2008.03596* (2020) (cit. on p. 19).

[Yam+99] Jin'ichi Yamaguchi, Eiji Soga, Sadatoshi Inoue, and Atsuo Takanishi. "Development of a bipedal humanoid robot-control method of whole body cooperative dynamic biped walking". In: *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C)*. Vol. 1. IEEE. 1999, pp. 368–374 (cit. on p. 3).

[Ye97] Yinyu Ye. "Approximating quadratic programming with bound constraints". In: *Mathematical programming* 84 (1997) (cit. on p. 105).

[ZP03] Andreas Zagler and Friedrich Pfeiffer. "" MORITZ" a pipe crawler for tube junctions". In: *2003 IEEE international conference on robotics and automation (Cat. No. 03CH37422)*. Vol. 3. IEEE. 2003, pp. 2954–2959 (cit. on p. 5).

[ZH18] Wuming Zhang and Kris Hauser. "Single-image footstep prediction for versatile legged locomotion". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 4407–4413 (cit. on p. 14).

[Zur] Robotic Systems Lab ETH Zurich. *Team CERBERUS Wins the DARPA Subterranean Challenge*. Accessed: 2022-06-29. URL: https://www.youtube.com/watch?v=fCHOU-fw2c0 (visited on 06/29/2022) (cit. on p. 6).

Abstract

Since Antiquity, humans have tried to understand the functioning of legged locomotion by adopting multidisciplinary approaches. In recent years, this question has become central in robotics, where the development of legged robots has boomed thanks to advances in mechatronics and increased computer power. However, due to the complexity of the problem, obtaining a robust autonomous locomotion in a wide range of situations is still an open problem, even if increasingly impressive performances are obtained. Several key characteristics of this kind of locomotion require carefully designed mechanical and control architectures to be able to perform tasks in a meaningful way. Legged robots have to move around without being attached to the ground. Their dynamics is an underactuated problem and, as such, they have to be controlled indirectly through the motion of their actuated appendages. This has to be done for task purpose, like going to a target location or orienting the body in a particular way, but also to keep balance. Most quadrupeds and bipeds are only dynamically stable when walking in non-conservative ways so they require careful control to stay upright. To do so, the controller has to take into account the dynamics of the robot while coordinating multiple degrees of freedom. This is a challenge with legged robots that are by nature highly nonlinear systems.

Over the last decades a wide range of methods have been developed to perform dynamic locomotion with legged robots. Some of them are bio-inspired and rather intuitive, like central pattern generators, while others have a strong theoretical aspect to achieve formal proofs of stability, like hybrid zero dynamics. Recently, the rise of machine learning also opens a whole new paradigm where, instead of being formulated by hand, the locomotion model is learned from data either online or during a training phase. It potentially allows to capture effects that would be hard to model or which would be untractable for some model-based architectures. In this thesis, we rather explore a predictive model-based approach which reasons over a prediction horizon to take the best control decisions for a given set of criterion. This kind of approach has already led to efficient locomotion with a wide range of quadruped robots.

This thesis contributes to the locomotion of legged robots by developing a control architecture able to exploit the dynamical capabilities of a lightweight quadruped robot. The use of complementary filters allows a straightforward sensors fusion for the estimation of the robot state. Binary matrices make it possible to handle contact sequences in a generic way to modify the gait pattern on the fly. This information can then be used to determine footstep locations online using a small set of heuristics. By reasoning on a prediction horizon a centroidal model predictive control can then find out which forces should be applied at contact locations to follow a reference state trajectory and handle disturbances. Next, a whole-body controller translates desired contact forces and swinging feet trajectories into joint trajectories and feedforward torques. Finally, an impedance controller provides feedback torques based on the difference between the desired and current joint positions and velocities to obtain the commands sent to the robot. The modularity of the architecture allows to easily augment some aspects of the scheme or to replace them to test out other methods, as it will be shown several times in this thesis.

This control scheme is implemented in a real-time framework and successfully deployed on the Solo-12 quadruped. The possibilities offered by this architecture, both in terms of performances and versatility, are validated through simulations and experiments. Several applications allowed us to quantify the interest and relevance of the presented scheme for the locomotion control of lightweight quadruped robots.

Keywords

Locomotion, Command, Robot, Quadruped

Résumé

Depuis l'Antiquité, l'humanité tente de comprendre le fonctionnement de la locomotion à pattes en adoptant des approches multidisciplinaires. Ces dernières années, cette question est devenue centrale en robotique, où le développement des robots à pattes a connu un essor considérable grâce aux progrès de la mécatronique et à l'augmentation de la puissance des ordinateurs. Cependant, en raison de la complexité du problème, l'obtention d'une locomotion autonome robuste dans un large éventail de situations reste un problème ouvert, même si des performances de plus en plus impressionnantes sont obtenues. Les robots à pattes doivent se déplacer sans être attachés au sol, ils doivent donc être contrôlés indirectement via le mouvement de leurs membres actionnés. Cela doit être fait pour accomplir une tâche, comme se rendre à un endroit cible ou orienter le corps d'une certaine manière, mais aussi pour garder l'équilibre. La plupart des quadrupèdes et des bipèdes ne sont que dynamiquement stables lorsqu'ils marchent de manière non conservative, ce qui demande un contrôle minutieux pour rester debout. Pour ce faire, le contrôleur doit prendre en compte la dynamique du robot tout en coordonnant plusieurs degrés de liberté. C'est un défi pour les robots à pattes qui sont par nature des systèmes hautement non linéaires.

Au cours des dernières décennies, un large panel de méthodes a été développé pour obtenir une locomotion dynamique avec de tels robots. Certaines d'entre elles sont bio-inspirées et plutôt intuitives, comme les réseaux locomoteurs spinaux, tandis que d'autres ont un fort aspect théorique afin d'obtenir des preuves formelles de stabilité, comme la dynamique hybride zéro. Récemment, l'essor de l'apprentissage automatique offre également un nouveau paradigme où, au lieu d'être formulé à la main, le modèle de locomotion est appris à partir de données, soit en ligne, soit pendant une phase d'entraînement. Dans cette thèse, nous explorons plutôt une approche prédictive basée modèle qui raisonne sur un horizon de prédiction pour prendre les meilleures décisions de contrôle selon certains critères. Ce type d'approche a déjà abouti à une locomotion efficace pour une large gamme de robots quadrupèdes.

Cette thèse contribue à la locomotion des robots à pattes en développant une architecture de contrôle capable d'exploiter les capacités dynamiques d'un robot quadrupède léger. L'utilisation de filtres complémentaires permet une fusion simple des capteurs pour l'estimation de l'état du robot. Des matrices binaires permettent de traiter les séquences de contact de manière générique afin de modifier la démarche à la volée. Cette information peut ensuite être utilisée pour déterminer la position des pas en ligne en utilisant un ensemble réduit d'heuristiques. En raisonnant sur un horizon de prédiction, une commande prédictive centroïdale peut alors déterminer les forces qui doivent être appliquées aux points de contact pour suivre une trajectoire de référence et gérer les perturbations. Ensuite, un contrôleur corps complet traduit les forces de contact souhaitées et les trajectoires de balancement des pieds en trajectoires articulaires et en couples moteurs. Enfin, un contrôleur d'impédance fournit des couples de rétroaction basés sur la différence entre les positions et les vitesses articulaires souhaitées et actuelles. La modularité de l'architecture permet d'étendre facilement certains de ses aspects ou de les remplacer pour tester d'autres méthodes, comme cela sera montré à plusieurs reprises dans cette thèse.

Ce schéma de contrôle est implémenté en temps réel et déployé avec succès sur le quadrupède Solo-12. Les possibilités offertes par cette architecture, tant en termes de performances que de polyvalence, sont validées par des simulations et des expériences. Plusieurs applications nous ont permis de quantifier l'intérêt et la pertinence du schéma présenté pour le contrôle de la locomotion de robots quadrupèdes légers.

Mots clefs

Locomotion, Commande, Robot, Quadrupède