Rheinische
Friedrich-Wilhelms-
Universität Bonn

Institute for Computer Science
Department VI
Autonomous Intelligent Systems

RHEINISCHE
FRIEDRICH-WILHELMS-UNIVERSITÄT BONN

MASTER THESIS

Dynamic Hybrid Locomotion for Driving-Stepping Quadrupeds

Author:
Mojtaba HOSSEINI

First Examiner:
Prof. Dr. Sven BEHNKE

Second Examiner:
Dr. Marcell MISSURA

Advisor:
Dr. Diego RODRIGUEZ

Date: September 20, 2021

Declaration

I hereby declare that I am the sole author of this thesis and that none other than the specified sources and aids have been used. Passages and figures quoted from other works have been marked with appropriate mention of the source.

Bonn, 20.10.2021

Place, Date

[illegible]

Mojtaba Hosseini

Signature

Abstract

Fast and versatile locomotion can be achieved with wheeled quadruped robots that provide agile and long-range navigation over challenging terrain. By integrating dynamic jumps into hybrid motion control and planning, obstacles can be overcome without stopping to step over them or changing paths to avoid them. In this thesis, we present a control optimization framework for quadruped robots that incorporates non-steerable wheels into the control problem and provides hybrid driving-stepping locomotion capabilities to quadruped robots with wheels. We formulated a Kalman Filter (KF) for state estimation that integrates the wheels into the filter equations. We present a Model Predictive Controller (MPC) that uses a time-varying Rigid Body Dynamics (RBD) model of the robot, including legs and wheels, which helps in tracking dynamic motions such as jumping. We formulated an approach to generate reference trajectories and commands for hybrid locomotion and dynamic driving-jumping, which are optimized by MPC to generate reaction forces, and are followed by a Whole-Body Controller (WBC) to generate joint commands. To the best of our knowledge, this is the first wheeled quadruped robot that can jump while driving. We introduce a method for driving with minimal leg swings to reduce the robot's energy consumption due to the effort required to move the heavy wheels at the end of legs. We tested our approach on the wheeled Mini Cheetah robot in simulation and in the real world.

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my thesis advisor, Dr. Diego Rodriguez, for his insightful discussions, fruitful comments, and valuable feedback. His brilliant ideas, encouragement and constant support have played a key role in helping me complete this poject.

I owe a debt of gratitude to the members of the Autonomous Intelligent Systems Lab who helped me pursue this project and give my best in a cooperative and cordial atmosphere. I offer my profound appreciation for the learning opportunities provided by their robotics team. I would especially like to thank Prof. Sven Behnke for giving me the opportunity to work in this lab. This project was made possible under the auspices of his team. I would like to thank Michael Schreiber, who creatively designed and fabricated the necessary hardware for the robot used in this work.

I cannot even put into words my gratitude to my old friend Hafez Farazi, who has inspired and encouraged me throughout the years of our friendship to move forward, especially in the most difficult moments. Without his encouragement, care, kindness and useful advice, I would not have reached this position. I am incredibly lucky to have a friend like him.

Last but not least, I am grateful to my parents and siblings for their immense support during my study abroad. I extend my gratitude to my friends for their advice and encouragement to complete this thesis.

Contents

1 Introduction

Robots that can be safely used in close proximity to humans are the subject of ongoing research. Flying systems, such as drones, are highly maneuverable but have a short flight time, are very noisy, and are not considered safe around humans. Ground robots, on the other hand, can navigate quickly and can be used safely near humans, but generally do not have high speed or versatility in their locomotion system.

Robots with legs are well suited for locomotion on irregular terrain, as the use of limbs in such terrain provides a much greater flexibility compared with pure wheeled robots [17]. The ability to place feet in discrete positions allows legged systems to climb stairs or maneuver in unstructured terrain like humans, and are therefore versatile platforms for such terrain [19].

Four-legged systems offer many advantages in mobility and versatility over bipeds, and are considered reasonable candidates for robust and safe locomotion over difficult terrain in close proximity to humans. Bipeds can only walk and run and have limited balancing abilities. A quadruped, on the other hand, can use its four legs to maintain balance while performing various gaits (e.g., trotting, bounding, pacing, etc.) and can move easily over uneven terrain or stairs. Various systems demonstrated dynamic motions [12, 18, 61], overcame slippery surfaces [8], or performed optimized jumps and backflips [11, 16].

Although walking robots are great at overcoming obstacles such as stairs, they are slow and consume a lot of energy. In contrast, wheeled robots can move quickly and efficiently on flat terrain and have very low transportation costs compared to their legged counterparts. However, they are usually not able to handle rough terrain, especially if the obstacles are larger than the radius of their wheels. Figure 1.1 highlights the benefits of wheeled, legged, and hybrid locomotion and demonstrates the motivation to choose the hybrid locomotion over the other.

The terrain around humans is mostly flat with height differences and obstacles. This motivates us to combine the capabilities of wheeled and legged robots, using fast and efficient wheels in flat terrain and locomotion on foot to bypass obstacles. The resulting hybrid locomotion solves the trade-off between efficiency and speed in systems with wheels and legs by reducing the number of leg swings, since driving is usually preferred to walking, while stepping is applied either at the request of

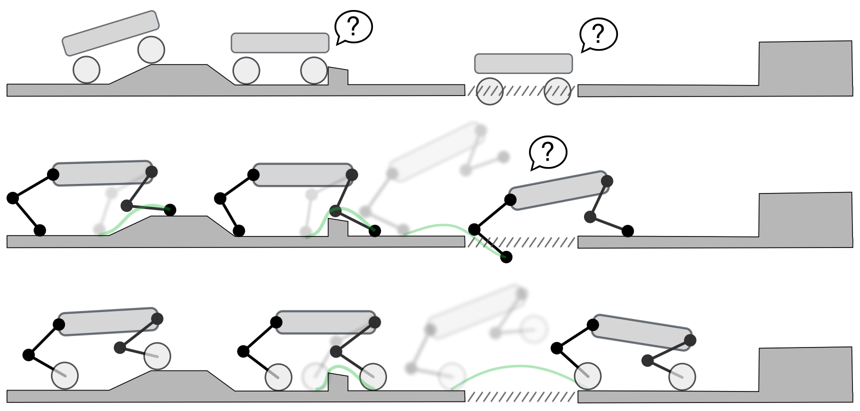


Figure 1.1: **Locomotion comparison.** Top: Wheeled systems are very fast on regular terrain, but fail if they encounter an obstacle larger than the radius of their wheels, or if there is a gap in the terrain. Middle: Four-legged systems can avoid obstacles by stepping through them, but are usually slow. They can jump over gaps in the terrain, but the length of the jump is limited by the power of the actuators. Below: The hybrid four-legged robot takes advantage of a wheeled system to travel quickly on regular terrain to overcome obstacles. The jumping motion while driving allows the robot to utilize leg joints only for vertical acceleration, which increases the jump height. During the jump, the rolling wheels continue to move the robot forward, increasing the jump length.

the user or due to irregularities in the terrain. Various systems have demonstrated hybrid locomotion [5, 14] on dynamic quadruped robots [2, 1, 7], and on bipedal systems with wheels [15]. Due to the above capabilities, wheeled legged systems are beneficial to society since the need for robotic caregivers increases as people live longer [2]. For example, people with poor eyesight can use such an energy-efficient robot that can overcome height differences, to navigate them.

The environment may have obstacles or elevation changes that require strategies for the system to slow down and to move around the obstacle or to spend a lot of time walking across them. The high forward speed of the wheels would make it expensive to stop when an obstacle appears in front of the robot. The better option is to perform a driving jump motion and to use the robot's dynamics to overcome the obstacle without slowing down and switching to walking mode (Figure 1.1).

While robots with legs have already found their way into real-world applications, legged robots with wheels are mostly found only in research labs [7, 25] and their locomotion capabilities are not well known since there are no natural counterparts to them [1]. The approaches [1, 7, 12, 16, 21, 30, 57] contain simplifications that capture the dynamics of hybrid locomotion by mostly assuming massless legs in the system.

Our is to develop a robot that can be used among humans, combining the versatility of legs to overcome difficult terrain with the efficiency of wheels to go fast. In this way, we increase the locomotion speed and energetic efficiency of the robot without compromising its versatility.

With the above motivation, this thesis presents algorithms that enable dynamic hybrid driving-stepping locomotion for a wheeled quadruped robot capable of jumping, using online MPC , taking into account the additional wheels in terms of their shape, mass and inertia tensor.

To this end, we have improved the open platform hardware of the Mini Cheetah by adding wheels to the ends of each limb. The Mini Cheetah robot is a torque-controlled quadruped robot developed in the MIT Biomimetic Robotics Lab using low-cost, highly reliable, and powerful hardware [11]. It is equipped with an Inertial Measurement Unit (IMU), a computer and a battery so that it can be operated without an external wired connection. The open source software, low cost, high performance and reliability of the Mini Cheetah platform make it an ideal choice for researchers. In the AIS Lab[1], we transferred the above open software into the ROS[2] echo system and used our robot control architecture, which allows us to separate the motion controls from the hardware interfaces so that we can exchange the hardware with similar quadrupeds or visualize and simulate them using different tools such as RViz[3], MuJoCo[4] and Gazebo[5].

We transformed the Mini Cheetah's hardware into a hybrid quadruped by designing and fabricating relatively small wheels and enhanced shank links. These wheels can produce a maximum torque of 2.1 Nm and a speed of 2150 rpm, theoretically allowing the robot to reach up to 40 km/h. We use ODrive[6] BLDC[7] drivers with custom firmware to control the speed and torque of the wheels with a low-resolution encoder (Hall sensor with 84 pulses per revolution). The replaced components on each leg increased the weight of each leg by 0.39 kg (with each wheel weighing about 0.32 kg). Together with the additional components, the total weight of the modified robot increased to 12.5 kg (from its original weight of 10 kg). Figure 1.2 shows the resulted hardware. We employed MuJoCo as a multibody simulator that can accurately simulate the contact dynamics and rolling friction of the additional wheels, as well as the joints and corresponding rotor dynamics. We intend to use the resulting system to study hybrid locomotion and

[1]https://www.ais.uni-bonn.de
[2]https://www.ros.org
[3]http://wiki.ros.org/rviz
[4]http://mujoco.org
[5]http://gazebosim.org
[6]https://odriverobotics.com
[7]Brushless Direct Current motor

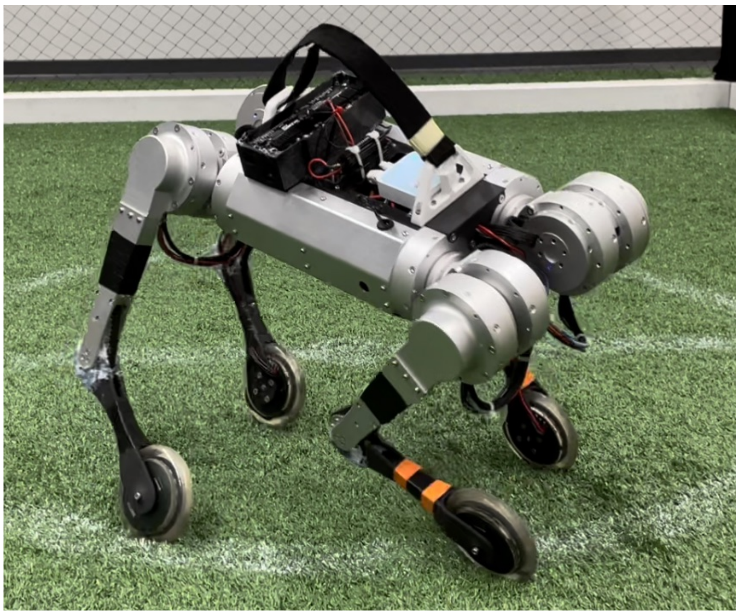


Figure 1.2: **The utilized hardware.** This figure shows the upgraded hardware of the MIT Mini Cheetah open platform. The additional wheels and shank links were designed and fabricated in the AIS lab at the university of Bonn, and the electronic equipment was upgraded to meet the wheels' requirements.

associated control strategies. It will also serve as a research platform for other desired areas of hybrid mobile robotics, including navigation, path planning in complex terrain, and the application of learning-based control algorithms.

1.1 Contributions and Outline

In this thesis, we work on the use of predictive and whole-body controllers for hybrid driving-stepping and jumping locomotion in quadrupedal wheeled robots. For this goal, which requires research in several areas, we have identified the following main categories:

- State Estimation
- Control
- Planning
- Driving Locomotion

The above list provides a convenient way to identify our contributions to the overall problem.

While the design of the robotic systems is a major part of our research, the discussion of hardware developments for wheels, the software framework that embodies all the requirements to make the actual robot work, and the simulation of the robot out of the scope of this document. Below we explain our contributions to each category defined above.

State Estimation. In terms of state estimation, our main contributions lie in incorporating the geometry of the end effector into the robot kinematics and the rolling wheels into the Kalman Filter (KF) equations .

We have replaced the rubber balls at the end of each limb with wheels, magnifying the physical geometry of the end effector, which can no longer be ignored. We incorporate the above enlarged geometry into the robot's kinematic model, and instead of simply considering the center of the end effector as the ground contact point, we define the exact touchdown position of the end effector, the corresponding Jacobian matrix, and the effective radius of the wheel (Section 4.2). We calculate the accurate velocity for the touchdown point above by converting the measured angular velocity of the wheel into a linear velocity using the effective radius, considering the contribution of the angular velocity of the body and the velocities of the leg joints to the contact point velocity, and adding the velocity measured by the kinematic model (Section 4.3.2).

We include the touchdown velocities above into KF state and measurement space and update the filter equations to simultaneously correct for the position of the foot contacts, the pose of the main body, and the contribution of driving with wheels, resulting the state estimation for the hybrid driving-stepping quadruped (Section 4.3.3).

Control. Our major contribution to the control method is the time-varying dynamic model of the robot used for the MPC, which accounts for the legs in the model, and allows for tracking various motions such as jumping. We explain the whole-body kinematic model of the robot in Section 5.4.1, which computes the effective mass, composite inertia tensor, and Center of Mass (CoM) of the robot according to the known future body orientation and foot positions. In this way, we obtain a set of expected single rigid bodies for the prediction horizon of MPC, which enables the implementation of a time-varying dynamics model in Section 5.4.2.

We consider the whole 3D orientation of the body, rather than restricting ourselves to the yaw angles, and keep the MPC formulation convex by linearizing the orientation dynamics around the known body orientations of the prediction horizon (Section 5.4.2). In this way, the inertia tensor in the world is more accurately described.

We include the weight and inertia of the wheels in the dynamic model of the robot in Section 4.1 to incorporate the additional wheels in the control accuracy of the WBC (Section 5.5).

We exploit the control over wheels' velocity and torque to achieve robust locomotion (Section 5.6). Rotating the wheel with respect to the speed of the swinging foot in the world frame allows for lower swing heights and enhanced swings along the rolling direction. This is because the continuous rotation of the wheels during the swing reduces the disturbance caused by the foot hitting an obstacle or landing earlier than expected. By feed-forwarding the torque commands for wheels according to the reaction forces received from the WBC and the commanded driving accelerations, the feet are kept at the desired contact points, resulting in improved control performance. We use the effective radius explained earlier to convert linear velocity and force into angular velocity and torque.

Planning. The control method described above requires expected trajectories for the body state, foot positions, and contact states to generate the time-varying model for the MPC. In addition, body and foot commands are needed to perform the tasks of the WBC. These trajectories and commands should be generated differently for the hybrid driving-jumping and driving-stepping behaviors. We show that realistic trajectories improve control performance. Therefore, we introduce the Trajectory and Contact (Section 5.1), and Footstep (Section 5.2) planner modules to describe the above trajectories and commands separately for each behavior.

To generate the expected state trajectory, we integrated driving and stepping velocity and acceleration commands for driving-stepping behavior and proposed a formulation based on the commanded jump height for jumping behavior (Section 5.1.2).

We add a dynamic variable to each gait, which changes the ratio of contact and swing phase duration, allowing for a wide range of dynamic and static gaits (Figure 5.5). We use the resulting gaits to create the expected contact trajectory for driving-stepping behavior (Section 5.1.3).

The target landing foot locations are manipulated by the Foot Step Planner using the symmetry offset (Equation 5.24) according to the walking speed of the robot. However, for a hybrid robot, the measured velocity is only partially related to the walking speed. Therefore, we define the expected gait velocity based on the current speed of the robot and the commanded driving velocity. We also compute the linear velocities induced by the yaw angular velocity of the robot at each shoulder location as additional terms in the symmetry equation to account for the rotational velocity of the robot or the yaw torque applied externally. With the above contribution, even a driving only robot (without stepping commands) responds to the external pushes or yaw torques by performing corrective swings.

We add yaw offset (to the direction of travel) to the body pose at runtime which allows the robot to walk or drive forward while looking at a different direction 4 Figure 4.2

Driving Locomotion. Our goal is to use the wheels to drive with lower energy consumption and to increase the efficiency of the robot by executing steps only when requested by the user or when instabilities occur due to irregular terrain. For this purpose, we can use driving as the main tool for locomotion and the stepping as an additional tool to correct the robot's posture or to bypass challenging terrain.

To achieve a natural driving locomotion, we aim to minimize the number of leg swings. To this end, we propose the driving assistant in Section 5.3 which assigns a utility value to each leg and dynamically determines the swing duration and height as well as the gait frequency to achieve a dynamic gait, and adds corrective velocities to the stance legs along the rolling direction.

In contrast to [1] which uses leg utilities to switch between pure driving mode, static walking, and trotting to recover legs that have reached their kinematic limits, this approach is not limited to specific gaits and always executes the dynamic gait cycle to simultaneously perform user-requested walking commands and corrective leg swings to correct the robot's kinematics all the time.

A lower leg utility results in a higher swing height and duration and a lower gait frequency. When a foot encounters an obstacle, the leg utility decreases significantly and the robot recovers the stuck leg with a larger swing. Finally, the stance leg correction applies additional control inputs to the wheels to reduce the error in the desired contact leg positions along the rolling direction (Section 5.3).

When the robot is near its nominal kinematic configuration, performing the dynamic gait described above allows the reaction forces on each leg to periodically approach 0 without the need to lift the legs, allowing the controller to slide the legs along the ground with very little friction. The further the robot moves away from the nominal configuration, the larger the steps it can take due to the decreased gait frequency and increased swing heights and duration. Therefore, the robot always keeps its kinematics close to the nominal configuration instead of discretely switching between pure driving and leg swinging, as in [1].

This introduction is followed in Chapter 2 by the presentation of the related works. Chapter 3 introduces the methods necessary for the dynamic control of quadruped robots. It derives the formulation of MPC from optimum control and Linear Quadratic Regulators (LQR), and then explains the WBC, in particular the

Whole-Body Impulse Controller (WBIC) and its formulations. Chapter 4 explains our approach in a larger context and discusses the inclusion of wheels in terms of mass, inertia tensor, and shape in the state estimation, as well as the kinematic and dynamic model of the robot. Chapter 5 describes the hybrid driving-stepping control framework in detail. It explains the reference trajectories and control commands generation for the driving-jumping and driving-stepping behaviors in Sections 5.1 and 5.2, the role of the driving assistant module in the locomotion of the hybrid system in Section 5.3, the whole-body kinematic model of the robot in Section 5.4.1 from which the time-varying dynamics model for the MPC is derived in Section 5.4.2, and the Quadratic Programming (QP) formulation of the MPC in Section 5.4.3, the considered prioritized tasks and the configuration parameters for the WBIC in Section 5.5, and the wheel controller module executing the WBC output commands in Section 5.6. Chapter 6 explains our evaluation and presents quantitative and qualitative results related to our contributions to simulated and real robots. Finally, Chapter 7 concludes this thesis by summarizing the main contributions of this thesis and by providing an outlook on future work.

2 Related Work

In this thesis, we present a multibody system that interacts with the environment through multiple contact points to perform hybrid dynamic locomotion. The robot must be aware of its state in the environment to perform dynamic motions on unknown and difficult terrain. In the following, we briefly explain existing approaches related to our work in three categories: Legged locomotion, hybrid locomotion, and state estimation.

Legged Locomotion

Motion plans for robots with legs are successfully generated in a number of works on real hardware using the Linear Inverted Pendulum Model (LIPM) as a robot model [4, 72], which is fast but comes at the price of inaccurate modeling of the real robot.

However, the LIPM is not suitable for generating motions with flight phases. A common approach that can generate such dynamic motions is based on the Trajectory Optimization (TO), which deals with systems subject to a set of physical constraints, with the aim of reducing the influence of the user by using numerical optimization techniques. TO requires a physical model of the system that defines its dynamics in order to generate motions for the given task and to control the legged locomotion on a more general level [41, 43]. Solving such TO problems is addressed in [31] by three methods, dynamic programming over the entire state space, indirect methods, and direct methods that minimize an objective function.

The physical model can be approximated with a Single Rigid Body Dynamics (SRBD) model, which assumes that the momentum of the joint accelerations is negligible and that the inertia of the system is constant. The work in [26] replaces the LIPM model with the 6D SRBD model using 3D contact forces of each end effector with friction constraints and optimizes the gait with a Nonlinear Programming (NLP) solver that takes into account forces and support-area constraints in TO. However, the results are only shown in simulation as the approach could not be performed online due to the time required for the NLP solver. The work in [33] considers the mass and inertia of all limbs to obtain a complete RBD model that accounts for the changing CoM positions and inertial properties based on the

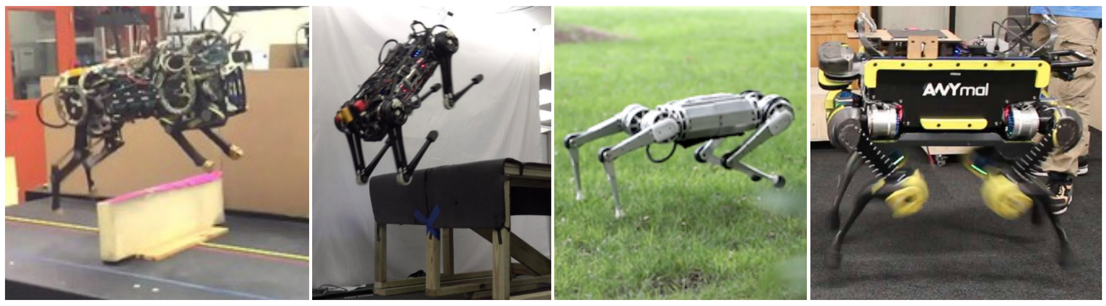


Figure 2.1: **Legged Robots.** From left: Cheetah 2 (from [45]), Cheetah 3 (from [16]), Mini Cheetah (from [12]), and ANYmal (from [27])

foot positions and the Coriolis forces generated by the leg motions. However, this model is very computationally expensive and cannot be used online.

Recent results have shown that quadrupeds can walk, trot, and climb stairs by using whole-body control and motion optimization algorithms. The approach presented in [35] for controlling the walking of quadrupeds is based on whole-body control combined with hierarchical optimization by solving a prioritized task using QP solvers, but is not flexible in responding to perturbations. The dynamic control method in [38] uses the principle of learning through practice to automatically fine-tune the parameters of the state-feedback controller on a parameterized whole-body model by repeatedly executing slight variations of the same motion in the simulation. However, the process would need to be rerun when changes are made to the hardware or when tasks involving new motions are executed. Climbing stairs, demonstrated in [36] is a similar motion plan to [39] that is tracked by a whole-body feedback controller to ensure accurate motion execution, but is only performed offline based on the predefined motions and the tasks should be redefined according to changes in the environment.

The TO methods discussed so far share a common problem: they are computationally expensive, which limits their use for online applications. A common alternative to using TO methods for online applications is to execute them in the form of MPC. MPC methods (Section 3.1) follow the dynamic model of the robot during the prediction horizon and solve an optimization problem that minimizes the objective function, which is usually to minimize the control effort and/or the deviation from the expected reference trajectory during the control horizon. The resulting optimization problem, which is computationally expensive to solve (and becomes more expensive as the length of the horizon increases), must be solved at each control iteration, making the approach unsuitable for online control. However, a highly accurate model of the robot dynamics during the prediction horizon may not be as important as an accurate model of the instantaneous dynamics.

Therefore, there are several approaches that simplify the model of the robot to achieve a faster update rate for optimization, or use a different controller (e.g., WBC) to account for the disturbances in real time, or a combination of the above approaches. The following related works use MPC in their respective approaches.

The method in [27] uses a simplified and linearized Zero Moment Point (ZMP)-based model to optimize a motion planner that generates planar CoM motions through a sequence of arbitrary support polygons in an MPC-like manner at low update rates, and uses a hierarchical whole-body controller to track the reference motion at higher update rates. This results in a controller that executes trot and step gaits while responding to perturbations and dealing with model inaccuracies, which is tested on the ANYmal quadruped (Figure 2.1). However, it is unable to control a gait with flight phases and cannot properly handle gait transitions or strong perturbations.

The work in [18] extends the above method by optimizing for the full position of the CoM and solving a separate optimization of footholds to achieve a wider range of gaits with full flight phases (trot, dynamic pace, and pronk). The nonlinear ZMP constraints are solved by sequential quadratic programming, which is about 6 times slower compared to the previous approach. However, by parallelizing the foothold optimization and the hierarchical controller, it is still possible to generate motion plans online.

The Cheetah 2 robot (Figure 2.1), presented in [45] from MIT, uses an online MPC controller to optimize its locomotion to run and jump over obstacles with heights up to 40 cm. The presented approach divides the jumping motion into 4 phases (front stance phase, air phase, back stance phase, and jump phase) and performs gait planning using a multiple horizon MPC problem solved for each phase by QP.

The authors in [16] present an optimized jumping trajectory for the Cheetah 3 quadruped robot (Figure 2.1). The jumping trajectory optimization uses a simplified 2D model of the robot with 5 limbs on a vertical plane to generate joint commands at low update rates (100 Hz), which are then linearly interpolated to obtain the desired joint commands at high update rates (1 kHz). Landing is handled separately based on a force control method that enforces PD control over the CoM and body orientation while satisfying friction constraints. The authors successfully validated the method by repeatedly jumping onto and down from a desk with a height of 76 $^{\text{cm}}/$. However, the simplified planar model only allows the method to be used in certain environments with flat ground plane.

In [21], the authors present a MPC approach for computing ground reaction forces for the Cheetah 3 robot (Figure 2.1). For this purpose, they simplified the dynamics model of the robot to a single rigid body by assuming massless legs. The mass and inertia of the body are considered constant and the body is assumed to have only yaw angular velocities. Considering a prediction horizon of up to 0.5 seconds, they formulated the MPC problem as a convex QP optimization with constraints on the ground reaction force and an objective function that penalizes tracking error and control effort. The resulting MPC runs in up to 50 Hz while joint torques are computed in 1 kHz according to the last MPC control output, state estimation data, and swing leg schedules. Despite the use of a simplified model, they achieved robust locomotion with different gaits (trot, running-trot, pronk, bound, pace and a full 3D gallop) and a maximum forward speed of 3 m/s[1]. However, the dynamics of the links are ignored and the body tracking relies only on the MPC with low update rates.

The authors of [12] extended the above work by combining the previous MPC method with a WBC. They defined 6 virtual joints for body orientation and position in addition to leg joints. The WBC computes joint positions, velocity and acceleration commands for the body and legs by tracking the commanded body and foot states, taking into account the optimal reaction force profiles found by the MPC. To this end, they optimize a QP, which is constrained to the full-body dynamics, using the computed accelerations of the body joints and the reaction force found by the MPC. During the optimization, the body acceleration commands are relaxed according to the specified weights to account for the optimal reaction forces. The joint torques are then found using the full-body dynamics formulations. With this approach, the control of the swing leg is still possible during the full flight phases by relaxing the body accelerations. In addition, a longer horizon length for the MPC is possible because the WBC applies the instantaneous dynamics of the robot with high update rates. The authors tested this approach on the Mini-Cheetah quadruped (Figure 2.1) robot with different gaits and rough terrain and achieved a top speed of 3.7 m/s in the lab and 1 m/s outdoor[2].

Hybrid Locomotion

The legged locomotion methods described above rely on non-slip contact points with unmovable location of forces. However, in a wheeled legged system, the rolling wheels allow the controller to change the location of forces along the rolling direction simply by rotating the wheels in contact, which can increase control performance. There are various quadruped robots with wheels in several institutes that either use the wheels only for driving and the legs for climbing in unstructured terrain, or formulate the controllers to integrate the wheels into the dynamics

[1] https://youtu.be/q6zxCvCxhic
[2] https://youtu.be/6JlVol3eyNI

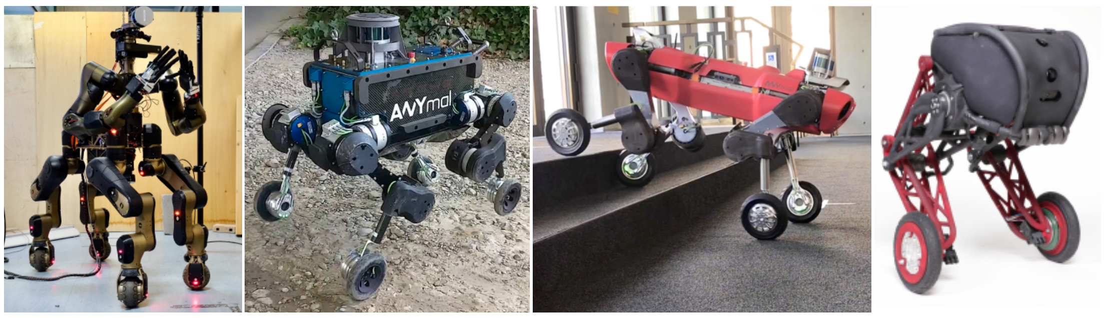


Figure 2.2: **Hybrid Robots.** From left: CENTAURO (from [13]), Wheeled ANYmal 2018 (from [2]), Wheeled ANYmal 2020 (from [1]), Ascento (from [15])

model of the system.

The work in [24] presents a hybrid driving-stepping locomotion for Momaro robot that can walk or drive in simulation, but only performs quasi-static motions without considering the dynamics of the robot. The CENTAURO robot (Figure 2.2) from [13] is a quadruped robot with a humanoid upper body, and wheels at the end of each leg, capable of automatic footstep placement using a linear MPC, but it uses the wheels only for driving and performs walking motions separately. There are several hybrid locomotion platforms that use TO to precompute complex trajectories over a time horizon offline and plan a motion on flat terrain by solving a NLP problem. [5, 22].

In [7], authors discuss a hierarchical whole-body controller that tracks the motion trajectories including the rolling conditions of the wheels. However, the robot cannot walk and drive at the same time, so it is forced to stop and switch to a walk-only mode to overcome obstacles[3]. The work in [6] extends the aforementioned work by computing base and wheel trajectories in a single optimization framework using linearized ZMP constraints, which has a low update rate of 50 Hz and does not perform experiments on real robot[4]. A generalized approach to motion planning is presented in [22] for the Skaterbots[5] by solving a nonlinear programming problem. However, due to the high computational cost, it is not practical to perform online in a receding horizon.

The work in [2] presents an optimization framework that incorporates the additional degrees of freedom introduced by the wheels into the motion generation. The optimization problem is split into end-effector and base trajectory planning, similar to [65], to make the locomotion planning more manageable for high-dimensional robots with wheels and legs. The problem is solved in realtime in a MPC fashion

[3]https://youtu.be/nGLUsyx9Vvc
[4]https://youtu.be/I1aTCTc0J4U
[5]https://youtu.be/TcTD0rRPG_k

to be robust against disturbances[6].

The authors in [1] propose a MPC method based on a kinodynamic model of hybrid quadruped robots with moving ground contacts to simultaneously optimize joint velocity and ground reaction forces. The robot dynamics are approximated by a SRBD with predefined inertia at the nominal robot configuration, treating the wheels as moving ground contacts. They increase the Cost of Transport (CoT) by proposing a leg utility which assigns a value for each leg that describes how useful the leg is. According to the leg utility values, the robot switches between pure driving, static walking and trotting. However, the robot remains in pure driving mode until one or two legs reach their kinematic limits. Then it discretely switches to walking or trotting to recover the legs, and other gaits (e.g. bounding or pronking) are not used to recover the legs. This method is evaluated on the ANYmal robot (Figure 2.2), which shows robust locomotion at driving speeds up to $4\,\mathrm{m/s}$[7].

The Ascento robot (Figure 2.2) in [15] is a compact bipedal robot with wheels that is able to navigate quickly, balance on flat terrain, and overcome obstacles by jumping[8]. The dynamics of the robot is modeled using SRBD and controlled by LQR method. However, the dynamics of the leg links and motors were neglected, which is significant due to the weight of the wheels relative to the body.

Most of the above methods neglected the dynamics of the legs and simplified the model of the robot by approximating the dynamics by a SRBD with constant inertia during the prediction horizon. However, the wheels add more weight to the end effector of each leg and increase the mass of the feet, which should be made as light as possible to satisfy the above simplifications. Therefore, in this thesis, we incorporate the kinematics and the mass of the legs into the dynamics model used in the MPC and formulate a time-varying SRB that simplifies the robot dynamics into a different SRBD at each time step of the control horizon to obtain a predictive controller that can accurately track various dynamic motions.

State Estimation

State estimation plays an important role for systems that require constant stabilization, such as quadruped robots. Robots can localize their environment using visual or GPS based methods [60, 66, 68]. However, for a legged robot without perceptual units, state estimation relies on data from IMU and the robot's interaction with the environment over multiple intermittent ground contacts.

[6]https://youtu.be/ukY0vyM-yfY
[7]https://youtu.be/_rPvKlvyw2w
[8]https://youtu.be/U8bIsUPX1ZU

The state estimation approach in this thesis is an extension of the approach presented by the Mini Cheetah open platform robot [11], which estimates the robot's position and orientation in the world coordinate system without perceptual units. Therefore, we focus on related works that use contact points on the ground to estimate the robot's state.

In [67], data from IMU is fused with the leg-based odometer of a hexapod robot, assuming that the robot is in contact with three of its six feet at all times and that the terrain is completely flat. However, the method is limited to walking and running on three legs.

In [52], an extended KF method is presented that fuses kinematic encoder data with IMU measurements by including the absolute position of all feet in the filter state to accurately capture the uncertainties related to the ground contacts. The resulting filter simultaneously estimates the pose of the main body and the positions of the footholds without making assumptions about the shape of the terrain, but is only tested in simulation.

The above contact-based methods require accurate knowledge of the contact state of each leg and therefore need a system with dedicated sensors on each foot to detect ground contact. However, for a legged wheeled robot, the use of accurate contact sensors could be costly and require a complex hardware design, motivating state estimation methods that do not rely on such sensors. To this end, a probabilistic contact detection algorithm is presented in [20] which uses a KF to fuse pre-calculated probabilities: the contact probability from gait swing phase, from foot height, and from foot force. The work in [29] estimates the probability of reliable contact and detects foot impacts using internal force sensing. This knowledge is then used to improve the estimation of the kinematic-inertial state of the robot base, resulting in an approach that has comparable performance to systems with foot sensors.

The approach followed in this thesis fuses IMU and kinematic encoder data by including the contribution of driving with the wheels in robot pose and the position of all end effectors in world coordinates in the filter state. The fusion is based on the contact probability given by the gait sequence of the robot.

3 Backgrounds

This chapter gives the background related to our methods. Our approach relies heavily on MPC, which is often formulated as a QP problem.

3.1 Model Predictive Controller

MPC is a control technique that is often applied in practice due to its ability to handle constraints on control inputs. MPC has evolved into a mature control technique based on a well-established theoretical foundation, especially after its success in the process industry in the 1980s. The basic concepts of MPC were introduced in the 1960s, and since then, numerous researchers have published applications and theoretical aspects around it. There has been an increasing interest in developing fast MPC methods in recent years, enabling MPC to be used for systems with slow dynamics, such as chemical processes and high-speed sampled systems like turbine control.

While the basic idea of MPC is well established, there exist many variants for guaranteeing closed-loop feasibility, stability, robustness or reference tracking. This section introduces MPC as an expansion over LQR, the closed-loop control of a plant with MPC, the condensed formulation of MPC, and its formulation as a QP problem.

3.1.1 Linear Quadratic Regulator

Control objectives describe the performance expected from the system being controlled and are usually stated as desired output values, such as the torque output of a joint, the speed of the wheels of a car, etc. Given a set of possible control actions, the controller chooses the one that minimizes the deviation of the system from the specified target values. Optimal control deals with several control objectives simultaneously, whose relative importance is weighted in the so-called cost function, allowing for complex control objective formulations. The goal of optimal control is to minimize the value of the cost function over time by utilizing available knowledge about the system's dynamics to predict its future behavior.

The LQR simplifies the formulations of optimal controls by discretizing the system dynamics, by making the cost function a quadratic polynomial and convex, and by considering the system dynamics as the only constraints in the problem [73]. We can formulate the *finite-horizon* version of LQR as:

$$\begin{array}{ll} \min\limits_{\boldsymbol{U},\boldsymbol{X}} & \sum_{k=0}^{N-1} \left\{ \boldsymbol{x}_k^\top \mathbf{Q} \boldsymbol{x}_k + \boldsymbol{u}_k^\top \mathbf{R} \boldsymbol{u}_k \right\} + \boldsymbol{x}_N^\top \mathbf{P} \boldsymbol{x}_N \\ \text{s.t.} & \boldsymbol{x}_{k+1} = \mathbf{A}\boldsymbol{x}_k + \mathbf{B}\boldsymbol{u}_k, \quad k = 0 \dots N-1 \\ & \boldsymbol{x}_0 = \widetilde{\boldsymbol{x}}(t), \end{array} \tag{3.1}$$

where $N > 0$ is referred to as the *prediction horizon*, $\boldsymbol{u}_k \in \mathbb{R}^m$ is the input vector at step k, $\boldsymbol{x}_k \in \mathbb{R}^n$ is the state vector at step k, and $\widetilde{\boldsymbol{x}}(t)$ is the state estimate at time t. There are m inputs and n states in the system. The matrices $\mathbf{A} \in \mathbb{R}^{n\times n}$ and $\mathbf{B} \in \mathbb{R}^{n\times m}$ represent the discrete time system dynamics over state and control variables. $\mathbf{B}$ describes the influence of the control input on the following state, and $\mathbf{A}$ describes the impact of the current state on the next state. $\mathbf{R} \in \mathbb{R}^{m\times m}, \mathbf{Q} \in \mathbb{R}^{n\times n}$ and $\mathbf{P} \in \mathbb{R}^{n\times n}$ are the input, state and terminal state penalty matrices. They weigh the relative importance of different deviations from the control goals. These penalty matrices are considered positive definite, a mathematical condition that guarantees the problem is strictly convex and has a unique solution by minimizing the cost [51].

In this thesis, we use capital letters $\boldsymbol{U} \in \mathbb{R}^{N\cdot m}$ and $\boldsymbol{X} \in \mathbb{R}^{N\cdot n}$ to represent the sequences of inputs and states, respectively, as:

$$\begin{aligned} \boldsymbol{U} &:= \begin{bmatrix} \boldsymbol{u}_0^\top & \boldsymbol{u}_1^\top & \dots & \boldsymbol{u}_{N-1}^\top \end{bmatrix}^\top \\ \boldsymbol{X} &:= \begin{bmatrix} \boldsymbol{x}_0^\top & \boldsymbol{x}_1^\top & \dots & \boldsymbol{x}_N^\top \end{bmatrix}^\top \end{aligned}$$

The optimal analytical solution for this problem is expressed as the feedback law $\boldsymbol{u}_k^* := -\mathbf{L}_k \boldsymbol{x}_k$ where $\mathbf{L}_k$ is the feedback matrix obtained from *Discrete Algebraic Riccati Equation* explained in [28, p. 168]. $\boldsymbol{u}_k^*$ stands for the optimal input at time step k, and $\boldsymbol{U}^*$ stands for the optimal *sequences of inputs* up to the prediction horizon length.

Simplifications of LQR along with the absence of constraints over inputs and states are the reasons why this method only solves a few real-world problems. However, most of the time, constraints need to be considered for the problem. For example, the joints of a robot arm can only be operated over a limited range of angles (state constraints) and can be controlled only by a bounded torque (control constraints), which LQR cannot capture. Model Predictive Control expands LQR to be more feasible in real-world applications. Even though this expansion increases the computational costs of the problem, it is necessary to achieve a more realistic control over the plant.

3.1.2 Model Predictive Control

Physical systems naturally involve constraints by definition (e.g., opening a valve in range of 0% to 100%), because of limitations (e.g., the maximum possible torque of a joint in a robot arm), or because of the control goals defined in the model (e.g., the top speed of a car on streets for safety reasons). Suppose a model neglects such limitations (which is the case of traditional PID controllers). In that case, the generated control actions might violate some constraints and pose a danger or even completely lose control of the plant.

MPC expands LQR by including constraints in the state and control variables with a receding horizon, allowing the models to represent the behavior of more complex dynamics systems. These models predict the changes in their state variables caused by the state-independent control actions or the measured disturbances. The task of MPC is to hold the state variables close to reference, while fulfilling constraints over state and control variables. Therefore, the cost function includes control sequence and error on reference state tracking.

In linear MPC, the problem to be solved at each point has linear dynamics and a quadratic objective function, which sets a classical optimization problem using QP, which is convex since the quadratic objective function is convex. For linear MPC, the feasible sets for states and control are based on only linear constraints, resulting the following formulation of the problem:

$$
\begin{aligned}
\min_{\boldsymbol{U},\boldsymbol{X}} \quad & \textstyle\sum_{k=0}^{N-1} \left\{ \boldsymbol{x}_k^\top \mathbf{Q} \boldsymbol{x}_k + \boldsymbol{u}_k^\top \mathbf{R} \boldsymbol{u}_k \right\} + \boldsymbol{x}_N^\top \mathbf{P} \boldsymbol{x}_N \\
\text{s.t.} \quad & \begin{array}{ll}
\boldsymbol{x}_{k+1} = \mathbf{A}\boldsymbol{x}_k + \mathbf{B}\boldsymbol{u}_k, & k = 0 \dots N{-}1 \\
\mathbf{F}_u \boldsymbol{u}_k \leq f_u, & k = 0 \dots N{-}1 \\
\mathbf{F}_x \boldsymbol{x}_k \leq f_x, & k = 0 \dots N{-}1 \\
\mathbf{F}_t \boldsymbol{x}_N \leq f_t & \\
\boldsymbol{x}_0 = \widetilde{\boldsymbol{x}}(t), &
\end{array}
\end{aligned}
\tag{3.2}
$$

where the matrices $\mathbf{F}_u \in \mathbb{R}^{\kappa_u \times m}, \mathbf{F}_x \in \mathbb{R}^{\kappa_x \times n}$ and $\mathbf{F}_t \in \mathbb{R}^{\kappa_t \times n}$ are the coefficient matrices for the *input constraints*, *state constraints* and *terminal constraints*, respectively. The state sequence is different from LQR (Section 3.1.1) by eliminating $\boldsymbol{x}_0$ as:

$$
\boldsymbol{X} := \begin{bmatrix} \boldsymbol{x}_1^\top & \boldsymbol{x}_2^\top & \dots & \boldsymbol{x}_N^\top \end{bmatrix}^\top
$$

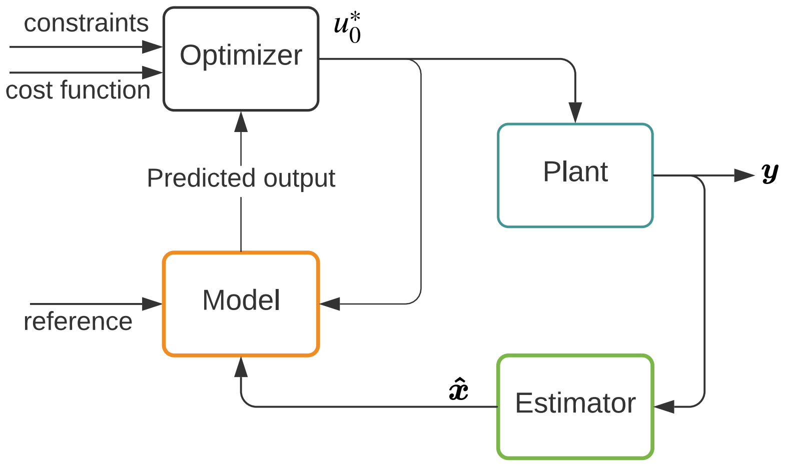


Figure 3.1: **Model predictive controller block diagram.** The estimator observes the plant state $\boldsymbol{y}$ and generates state estimate $\hat{\boldsymbol{x}}$, the dynamics model predicts future trajectory based on reference trajectory, the estimated state, and the latest control output. The optimizer finds a series of optimal control vectors for the horizon length N ($\boldsymbol{u}_0^* \dots \boldsymbol{u}_{N-1}^*$) by minimizing the cost function while considering constraints. The first control action $\boldsymbol{u}_0^*$ is applied to the plant resulting new state $\boldsymbol{y}$. The whole process repeats in a loop.

Closed-loop Control

Equation (3.2) computes a sequence of optimal control inputs for a predicted evolution of the system model over a finite horizon. This process is an open-loop control with the optimal solution $\boldsymbol{U}^*$, which is optimal for the corresponding state $\boldsymbol{X}^*$. But, in the presence of external disturbances or model inaccuracies, the system might deviate from its predicted optimal trajectory. In that case, the control inputs are no longer optimal, leading to degraded performance or instabilities. With short horizons, minimizing the cost function might generate an optimal control sequence but it might drive the system into uncontrollable states. Although increasing the prediction horizon can circumvent this issue and avoid such a scenario, it would be more computationally expensive to solve.

To address the aforementioned issues, the *receding horizon* idea is introduced [73, pp. 7-9]. This so-called receding horizon strategy introduces feedback to the system, making the controller closed-loop, and allows for compensation of potential modeling errors or disturbances acting on the system, resulting in improved and more robust control [28, pp. 243–274]. Figure 3.1 describes the closed-loop control structure.

With the receding horizon approach, depicted in Figure 3.2, a finite-horizon optimization problem is solved at each sampling instant. However, only the first

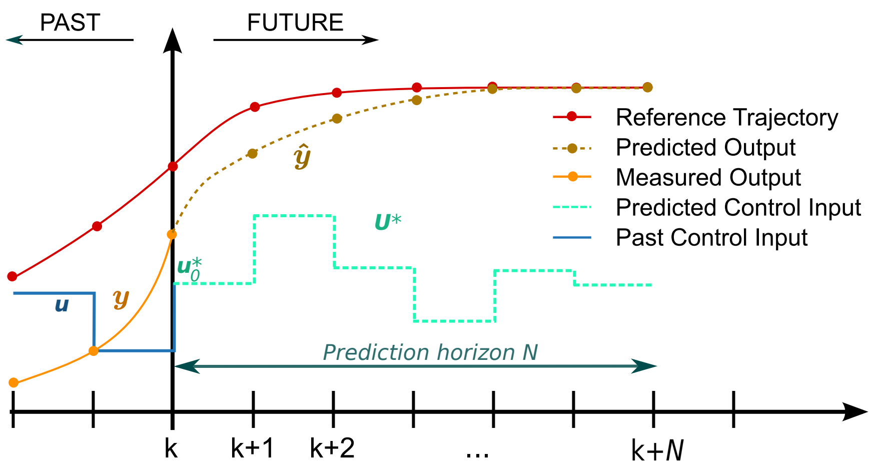


Figure 3.2: **Model predictive controller tracking graph.** The measured state $\boldsymbol{y}$ of the plant in past are already influenced by the past control actions $\boldsymbol{u}$. At current time step k, the controller solves an optimization problem over the prediction horizon N that moves the plant toward the reference trajectory by generating control actions $\boldsymbol{U}^* := (\boldsymbol{u}_0^* \dots \boldsymbol{u}_{N-1}^*)$ and expected future states $\widehat{\boldsymbol{y}}$ for all steps of the control. (modified picture[1])

element of the control sequence u_0^* is applied (the rest of the control sequence is ignored). Then, the system's state is measured again at the next sampling time, repeating the whole process with a new optimization problem that uses the new state estimate as its initial state. Therefore, the propagated error between the actual and predicted states is reduced.

Tracking Problem

The nominal MPC Equation (3.2) typically regulates the system state $\boldsymbol{x}(k)$ to the origin while minimizing the control effort and respecting constraints on inputs and states. But, many control applications in practice require tracking the desired sequence of steady states rather than regulation around the origin or a particular steady-state. The tracking of piecewise constant references can be achieved by modifying the MPC Equation (3.2), so that the variables penalize the *deviation* of the state from the input reference (rather than simply penalizing the state itself)[73]. However, these deviations should be small to steer the system to the new reference point. Otherwise, it might not be reachable from the current state, given the horizon length.

The tracking approach is possible by the introduction of an artificial reference into the optimization problem. An optimal artificial reference and control inputs

[1] https://en.wikipedia.org/wiki/File:MPC_scheme_basic.svg

are then computed in one optimization problem, allowing the artificial reference to deviate from the actual reference if the latter is not a feasible target from the current state. This provides recursive feasibility and renders the shifted solution computed at the previous time instant feasible for the current state measurement.

The tracking approach can be formulated in the form of an MPC problem, in which the cost penalizes the deviation from the states and inputs to the artificial reference instead of the real reference. Penalizing such deviations sometimes causes the system to oscillate around the reference due to model inaccuracies and disturbances. However, one can circumvent this by also punishing the input change rate. The resulting cost function of the Equation (3.2) for a simple reference tracking is given by:

$$\min_{\boldsymbol{U},\boldsymbol{X}} \sum_{k=0}^{N-1} \left\{ \boldsymbol{u}_k^\top \mathbf{R} \boldsymbol{u}_k + (\boldsymbol{x}_k - \boldsymbol{x}_{k,r})^\top \mathbf{Q} (\boldsymbol{x}_k - \boldsymbol{x}_{k,r}) \right\} + (\boldsymbol{x}_N - \boldsymbol{x}_{N,r})^\top \mathbf{P} (\boldsymbol{x}_N - \boldsymbol{x}_{N,r}), \tag{3.3}$$

where $\boldsymbol{x}_{k,r}$ and $\boldsymbol{x}_{N,r}$ denote the artificial reference state at time k and N.

3.1.3 Quadratic Programming Formulation

In this thesis, a linear time-variant version of MPC (LTV-MPC) is used. Hence, in this section, we consider a model predictive controller problem with time horizon length k in the standard linear form, without constraining states variables, which leads to the QP problem:

$$\begin{array}{rl} \min\limits_{\boldsymbol{U},\boldsymbol{X}} & \sum_{k=0}^{N-1} \left\{ \boldsymbol{x}_k^\top \mathbf{Q} \boldsymbol{x}_k + \boldsymbol{u}_k^\top \mathbf{R} \boldsymbol{u}_k \right\} + \boldsymbol{x}_N^\top \mathbf{P} \boldsymbol{x}_N \\ & \boldsymbol{x}_{k+1} = \mathbf{A}_k \boldsymbol{x}_k + \mathbf{B}_k \boldsymbol{u}_k, \; k = 0 \dots N{-}1 \\ \text{s.t.} & \underline{\boldsymbol{c}}_k \le \mathbf{C}_k \boldsymbol{u}_k \le \bar{\boldsymbol{c}}_k, \; k = 0 \dots N{-}1 \; , \\ & \mathbf{D}_k \boldsymbol{u}_k = \mathbf{0}, \; k = 0 \dots N{-}1 \end{array} \tag{3.4}$$

where $\mathbf{C}_k$ represents inequality constraints on the control input within the lower $\underline{\boldsymbol{c}}_{\boldsymbol{k}}$ and upper $\bar{\boldsymbol{c}}_k$ bounds, and $\mathbf{D}_k$ is the equality matrix. $\mathbf{C}_k$ and $\mathbf{D}_k$ are set specifically for each problem to select and scale the aspects of the control input that needs to be constrained. For example, consider a robot joint where the torque on the joint is the control input, so the maximum physical limits can be set as the upper-bound for the constraints.

We focus on QP formulation of the nominal MPC Equation (3.4) which regulates the system state to the origin ($\boldsymbol{x}_k \to \mathbf{0}$). However, the tracking problems can also be explained similarly by defining the model state as deviations from the reference trajectory $\boldsymbol{\delta}_k = (\boldsymbol{x}_k - \boldsymbol{x}_{k,r})$ resulting $\boldsymbol{\delta}_k \to \mathbf{0}$.

Equality constraints can be applied using the inequality constraints by setting upper and lower bounds to zero. Hence we remove the equality constraints ($\mathbf{D}_k \boldsymbol{u}_k = \mathbf{0}$). Then, by taking out $\boldsymbol{x}_0$ separately, we can rewrite the Equation (3.4) as:

$$\begin{aligned} \min_{\boldsymbol{U},\boldsymbol{X}} \quad & \boldsymbol{X}^\top \bar{\mathbf{Q}} \boldsymbol{X} + \boldsymbol{U}^\top \bar{\mathbf{R}} \boldsymbol{U} + \boldsymbol{x}_0^\top \mathbf{Q} \boldsymbol{x}_0 \\ \text{s.t.} \quad & \begin{matrix} \boldsymbol{X} = \bar{\mathbf{A}} \boldsymbol{X} + \bar{\mathbf{B}} \boldsymbol{U} + \boldsymbol{A}_\circ \boldsymbol{x}_0 \\ \underline{\boldsymbol{c}} \le \mathbf{C}\boldsymbol{U} \le \bar{\boldsymbol{c}} \end{matrix} \; , \end{aligned} \tag{3.5}$$

where $\mathbf{C}$ is the block matrix consisting of all the constraint matrices during the prediction horizon, and the block matrices $\bar{\mathbf{Q}}$, $\bar{\mathbf{R}}$, $\bar{\mathbf{A}}$, $\bar{\mathbf{B}}$, and the block vectors $\boldsymbol{X}$, $\boldsymbol{U}$, $\boldsymbol{A}_\circ$ are defined as:

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_{N-1} \\ \boldsymbol{x}_N \end{bmatrix} \quad \boldsymbol{U} = \begin{bmatrix} \boldsymbol{u}_0 \\ \boldsymbol{u}_1 \\ \vdots \\ \boldsymbol{u}_{N-1} \end{bmatrix} \quad \boldsymbol{A}_\circ = \begin{bmatrix} \mathbf{A}_0 \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{bmatrix}$$

$$\bar{\mathbf{A}} = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{A}_1 & \mathbf{0} & \dots & \mathbf{0} \\ \vdots & \mathbf{A}_2 & \ddots & \vdots \\ \mathbf{0} & \dots & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{A}_{N-1} \end{bmatrix} \quad \bar{\mathbf{Q}} = \begin{bmatrix} \mathbf{Q} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{Q} & \mathbf{0} & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \dots & \mathbf{0} & \mathbf{Q} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{P} \end{bmatrix}$$

$$\bar{\mathbf{B}} = \begin{bmatrix} \mathbf{B}_0 & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{B}_1 & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \dots & \mathbf{0} & \mathbf{B}_{N-1} \end{bmatrix} \quad \bar{\mathbf{R}} = \begin{bmatrix} \mathbf{R} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{R} & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \dots & \mathbf{0} & \mathbf{R} \end{bmatrix}$$

We disregard the term corresponding to the initial state $\boldsymbol{x}_0^\top \mathbf{Q} \boldsymbol{x}_0$ in Equation (3.5) because it is just a constant offset in the cost function. By combining state and control vectors into one vector $\boldsymbol{w} = \begin{bmatrix} \boldsymbol{X} & \boldsymbol{U} \end{bmatrix}^\top$, we can formulate the optimization problem as a minimization of the following pure quadratic form:

$$\begin{aligned} \min_{\boldsymbol{w}} \quad & \boldsymbol{w}^\top \begin{bmatrix} \bar{\mathbf{Q}} & \\ & \bar{\mathbf{R}} \end{bmatrix} \boldsymbol{w} \\ \text{s.t.} \quad & \bar{\bar{\mathbf{A}}} \boldsymbol{w} + \bar{\bar{\mathbf{b}}} \boldsymbol{U} = \mathbf{0} \; , \end{aligned} \tag{3.6}$$

where $\bar{\bar{\mathbf{A}}} = \begin{bmatrix} (\bar{\mathbf{A}} - \mathbf{I}) & \bar{\mathbf{B}} \end{bmatrix}$ and $\bar{\bar{\mathbf{b}}} = \boldsymbol{A}_\circ \boldsymbol{X}$. Clearly, Equation (3.6) with linear constraints renders our problem perfectly suitable for quadratic solvers (e.g. quadprog [79]).

Condensed Approach

Condensing is an approach that eliminates the state variables $\boldsymbol{X}$ from the optimization problem. This is achieved by expressing the state variables as a function of the initial state $\boldsymbol{x}_0$ and the input sequence $\boldsymbol{U}$. With recursive iterations over the discrete-time horizon as:

$$\begin{aligned}
\boldsymbol{x}_1 &= \mathbf{A}_0\boldsymbol{x}_0 + \mathbf{B}_0\boldsymbol{u}_0\\
\boldsymbol{x}_2 &= \mathbf{A}_1\boldsymbol{x}_1 + \mathbf{B}_1\boldsymbol{u}_1 = \mathbf{A}_0\mathbf{A}_1\boldsymbol{x}_0 + \mathbf{A}_1\mathbf{B}_0\boldsymbol{u}_0 + \mathbf{B}_1\boldsymbol{u}_1\\
\boldsymbol{x}_3 &= \mathbf{A}_0\mathbf{A}_1\mathbf{A}_2\boldsymbol{x}_0 + \mathbf{A}_1\mathbf{A}_2\mathbf{B}_0\boldsymbol{u}_0 + \mathbf{A}_2\mathbf{B}_1\boldsymbol{u}_1 + \mathbf{B}_2\boldsymbol{u}_2\\
&\vdots\\
\boldsymbol{x}_k &= \prod_{i=0}^{k-1}\mathbf{A}_i\boldsymbol{x}_0 + \begin{bmatrix}\prod_{i=1}^{k-1}\mathbf{A}_i\mathbf{B}_0 & \prod_{i=2}^{k-1}\mathbf{A}_i\mathbf{B}_1 & \dots & \prod_{i=k-1}^{k-1}\mathbf{A}_i\mathbf{B}_{k-2} & \mathbf{B}_{k-1}\end{bmatrix}\boldsymbol{U},
\end{aligned}$$

the resulting dynamics of the system becomes a function of the initial state $\boldsymbol{x}_0$ and the control vector $\boldsymbol{U}$ over all the steps in the optimization horizon:

$$\boldsymbol{X} = \bar{\mathbf{C}}\boldsymbol{U} + \widehat{\boldsymbol{A}}\boldsymbol{x}_0, \tag{3.7}$$

where $\bar{\mathbf{C}}$ and $\widehat{\boldsymbol{A}}$ are called the controllablity matrix and the dynamics vector, respectively. $\bar{\mathbf{C}}$ converts the control input into the state vector trajectory during the whole prediction horizon. $\mathbf{c}_{r,c} = \prod_{i=c+1}^{r}\mathbf{A}_i\mathbf{B}_c, (c \leq r)$ defines element of lower-triangular matrix $\bar{\mathbf{C}}$ at row r and column c (zero-based indexing), and $\boldsymbol{a}_r = \prod_{i=0}^{r}\mathbf{A}_i$ is the element of $\widehat{\boldsymbol{A}}$ on each row r. The controllablity matrix ends up as:

$$\bar{\mathbf{C}} = \begin{bmatrix}
\mathbf{B}_0 & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0}\\
\mathbf{A}_1\mathbf{B}_0 & \mathbf{B}_1 & \vdots & \dots & \mathbf{0}\\
\mathbf{A}_1\mathbf{A}_2\mathbf{B}_0 & \mathbf{A}_2\mathbf{B}_1 & \vdots & \vdots & \mathbf{0}\\
\vdots & \vdots & \ddots & \dots & \vdots\\
(\mathbf{A}_1\dots\mathbf{A}_{N-1})\mathbf{B}_0 & (\mathbf{A}_2\dots\mathbf{A}_{N-1})\mathbf{B}_1 & \dots & \mathbf{B}_{N-2} & \mathbf{0}\\
(\mathbf{A}_1\dots\mathbf{A}_N)\mathbf{B}_0 & (\mathbf{A}_2\dots\mathbf{A}_N)\mathbf{B}_1 & \dots & \mathbf{A}_N\mathbf{B}_{N-2} & \mathbf{B}_{N-1}
\end{bmatrix}
\quad
\widehat{\boldsymbol{A}} = \begin{bmatrix}\mathbf{A}_0\\ \mathbf{A}_0\mathbf{A}_1\\ \vdots\\ \mathbf{A}_0\dots\mathbf{A}_N\end{bmatrix}$$

We can substitute into the cost function we derived above, as:

$$\begin{aligned}
J(\boldsymbol{U}, \boldsymbol{x}_0) &= (\bar{\mathbf{C}}\boldsymbol{U} + \widehat{\boldsymbol{A}}\boldsymbol{x}_0)^\top \bar{\mathbf{Q}}(\bar{\mathbf{C}}\boldsymbol{U} + \widehat{\boldsymbol{A}}\boldsymbol{x}_0) + \boldsymbol{U}^\top \bar{\mathbf{R}}\boldsymbol{U} + \boldsymbol{x}_0^\top \mathbf{Q}\boldsymbol{x}_0 \\
&= \tfrac{1}{2}\boldsymbol{U}^\top \underbrace{2(\bar{\mathbf{C}}^\top \bar{\mathbf{Q}}\bar{\mathbf{C}} + \bar{\mathbf{R}})}_{\mathbf{H}} \boldsymbol{U} + \underbrace{\boldsymbol{x}_0^\top 2\widehat{\boldsymbol{A}}^\top \bar{\mathbf{Q}}\bar{\mathbf{C}}}_{\boldsymbol{g}^\top} \boldsymbol{U} + \underbrace{\frac{1}{2}\boldsymbol{x}_0^\top 2(\widehat{\boldsymbol{A}}^\top \bar{\mathbf{Q}}\widehat{\boldsymbol{A}} + \mathbf{Q})\boldsymbol{x}_0}_{\text{constant offset} \Rightarrow \text{ignored}} \\
J(\boldsymbol{U}) &= \tfrac{1}{2}\boldsymbol{U}^\top \mathbf{H}\boldsymbol{U} + \boldsymbol{U}^\top \boldsymbol{g}.
\end{aligned}$$

Using the cost function $J(\boldsymbol{U})$ we achieve the following optimization program:

$$\begin{aligned}
\min_{\boldsymbol{U}} \quad & \boldsymbol{U}^\top \mathbf{H}\boldsymbol{U} + \boldsymbol{U}^\top \boldsymbol{g} \\
\text{s.t.} \quad & \underline{\boldsymbol{c}} \leq \mathbf{C}\boldsymbol{U} \leq \bar{\boldsymbol{c}} \ ,
\end{aligned} \tag{3.8}$$

where $\mathbf{H}$ and $\boldsymbol{g}$ are given as:

$$\begin{aligned}
\mathbf{H} &= 2(\bar{\mathbf{C}}^\top \bar{\mathbf{Q}}\bar{\mathbf{C}} + \bar{\mathbf{R}}) \\
\boldsymbol{g} &= 2\bar{\mathbf{C}}^\top \bar{\mathbf{Q}}(\widehat{\boldsymbol{A}}\boldsymbol{x}_0).
\end{aligned} \tag{3.9}$$

For a tracking MPC however, we will reach to $\boldsymbol{g} = 2\bar{\mathbf{C}}^\top \bar{\mathbf{Q}}(\widehat{\boldsymbol{A}}\boldsymbol{x}_0 - \boldsymbol{X}_r)$ where $\boldsymbol{X}_r$ is the reference trajectory for the whole horizon.

The optimum solution $\boldsymbol{U}^*$ to the minimization of $J(\boldsymbol{U})$, which does not consider any constraints, is obtained by zeroing the gradient of the cost function:

$$\nabla_{\boldsymbol{U}} J(\boldsymbol{U}) = \mathbf{H}\boldsymbol{U} + \boldsymbol{g} = \mathbf{0} \quad \Rightarrow \quad \boldsymbol{U}^* = -\mathbf{H}^{-1}\boldsymbol{g}. \tag{3.10}$$

Since $\bar{\mathbf{C}}$ is a lower block triangular matrix, the symmetry of the result could be exploited to reduce the cost of operations for the matrix-matrix multiplications. For the system of linear equations, the symmetric positive definite (SPD) dense matrices help the problem to be solved using an unstructured Cholesky factorization [59].

Multiple practical approaches to find solutions to the Equation (3.8), which includes constraints, are introduced. Among those methods are: *Gradient Projection* (e.g. [71, 77]), *Active Set* (e.g. [37, 76]), and *Interior Point* (e.g. [9, 34, 62]) methods.

3.2 Whole-Body Control

Control trajectories used for planners typically include position, velocity, orientation, and angular velocity of the robot's CoM. While behavioral planners typically decide on control trajectories in relatively large time discretizations, whole-body control architectures augment the planned trajectory with feedback strategies to achieve a fast control loop over all robot joints.

Tracking the CoM of a robot has often been performed by position control. This leads to a stiff behavior of the robot, which is effective in the case of a well-estimated ground geometry, but not very robust in the case of a premature or delayed ground contact. On the other hand, torque-controlled systems with the possibility of force control include architectures for compliant behavior and more robustness when the robot interacts with an uncertain environment [40, 46, 49, 63].

Methods like force control directly governs the forces which are actively applied by the robot [74]. For this, position and force tasks in the control architecture are separated into strictly prioritized sub-tasks [53, 56, 69]. Hence, the control behavior of the high-priority tasks are applied first, without the lower-priority tasks' influence, later on, the controller tries to fulfill the lower-priority tasks within the remaining solution space. Constraints such as actuator limits or friction force can be handled within such hierarchies by solving optimization problems which use quadratic programming approaches [50, 58].

In this thesis, a whole-body control (WBC) is formulated for the mini-cheetah robot [12]. This control method is explained in the following section.

3.2.1 Whole-Body Impulse Control

Whole-body impulse control (WBIC) formulation is similar to existing whole-body controllers [23, 55], with the incorporation of pre-computed reaction forces by relaxing the floating base control inputs. This plays a vital role in dynamic locomotion control. Figure 3.3 depicts the control overview of the WBIC.

In the formulation of WBIC, the ground reaction forces generated from a higher-level controller, such as MPC (e.g., Cheetah 3 [21], and Mini-Cheetah [11]) are tracked, rather than the body trajectories. These high-level controllers usually consider the dynamics model of the system and predict the future states and control inputs that stabilize the robot. However, the prediction process requires high computational cost, resulting in limited update frequency; hence, by introducing simplifications on the plant model, algorithms can decide on a trade-off between the higher update rate and more accurate control inputs. Nevertheless, the modeling inaccuracies resulted from the simplifications add a fundamental limitation in the position control.

The WBIC architecture solves this limitation by running a high-frequency feedback loop over the whole-body dynamics, taking into account the reaction forces of the contact points generated at a lower update rate. However, since the WBC can

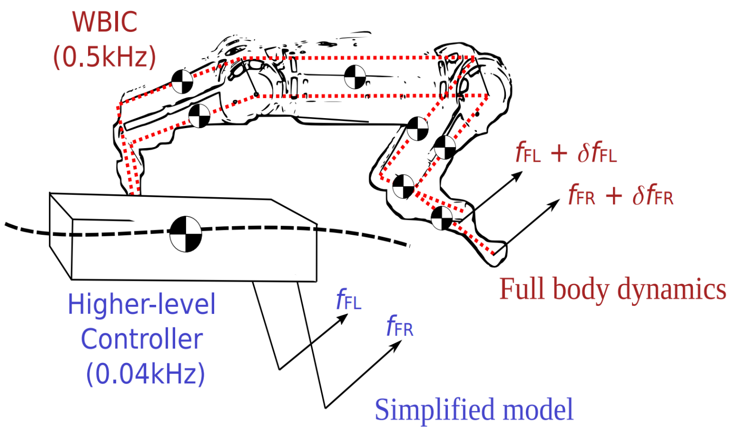


Figure 3.3: **Floating-base body control.** This figure depicts the floating-base body control using contact reaction forces. Higher-level controller (usually MPC) uses a simplified model of the robot and computes desired reaction forces $\mathbf{f}_r$ (blue). The WBIC uses the full body dynamics in high update rate and corrects the desired $\mathbf{f}_r$ commands of the higher-level controller (red). Picture taken from [12].

only consider a single time step, the prediction horizon of the high-level controllers perfectly complements the WBC. The single time step issue was addressed in [32] and [42], which develop MPC formulation using full-body dynamics. However, they only reached up to a 200 Hz update frequency, and the results shown are not as dynamic compared to other controllers presented for the same robots.

The WBIC implementation, which integrates MPC and WBC to take advantage of both worlds, is fast and reliable because the solver uses convex optimization without getting stuck in strange local minima. The integration of the MPC with WBC is done by modifying the formulations presented in [23] to utilize the results of reaction forces produced by MPC in the WBC. Figure 3.4 depicts the block diagram of the WBIC approach.

In [10], an approach to accomplish this integration is presented that attempts to follow the CoM trajectory of the higher-level MPC controller and uses the reaction forces determined by the higher-level controller only for regulating internal forces. However, the WBIC is formulated to use the higher-level reaction forces as the desired reaction forces rather than attempting to follow the corresponding CoM trajectories.

The WBIC allows the floating base motion to differ from the higher level commanded trajectory by accounting for relaxation variables during body posture and foot swing control. This relaxation will enable behaviors with uncontrollable CoM

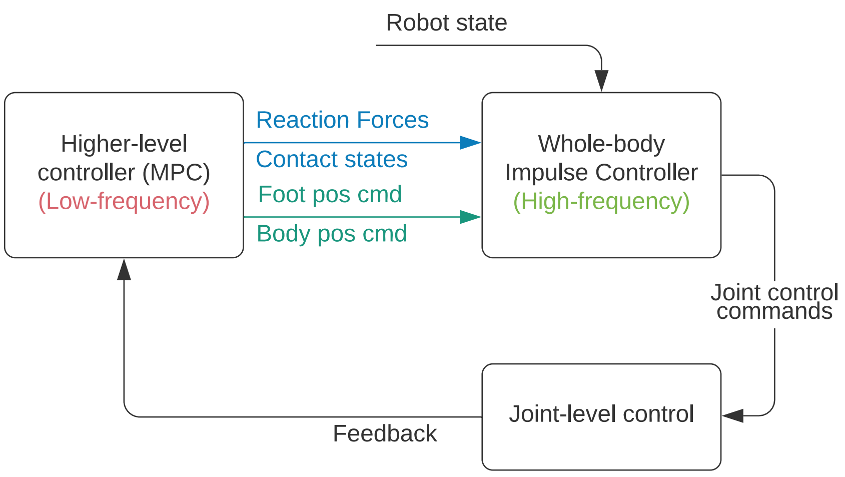


Figure 3.4: **Whole-body impulse control block diagram.** The higher-level controller observes joint feedbacks and computes the reaction forces considering long-term future controls and decides on CoM and foot position commands. WBIC receives this information along with the current robot state (generalized mass matrix, foot position, etc.) and computes joint control commands (τ, q, $\dot{q}$) that are sent to the joint-level controller.

(during periods of flight e.g., jumping) to be executed by controlling the higher-level reaction forces and ignoring the body posture trajectories.

The hierarchical tasks in the WBIC formulation can be extended to account for additional limbs (e.g., manipulation arm); this makes the architecture flexible for different robots. This approach can be implemented on a real robot to demonstrate highly dynamic locomotion while maintaining the robot's balance, even in the presence of large disturbances. Authors of the WBIC approach claim their method demonstrates the most agile locomotion of any quadruped robot [12]. They also implemented this algorithm successfully using efficient dynamics engine including rotor dynamics [54, pp. 229-243] [12].

Full-body Dynamics

The WBIC approach uses full-body dynamics with high-frequency feedback and therefore determines more accurate torque commands than the higher-level controller, with the following formulation:

$$\mathbf{A}\begin{pmatrix}\ddot{\boldsymbol{q}}_f\\ \ddot{\boldsymbol{q}}_j\end{pmatrix}+\boldsymbol{b}+\boldsymbol{g}=\begin{pmatrix}\mathbf{0}_6\\ \boldsymbol{\tau}\end{pmatrix}+\mathbf{J}_c^{\top}\boldsymbol{f}_r\,, \qquad (3.11)$$

where $\ddot{\boldsymbol{q}}_f \in \mathbb{R}^6$ is the acceleration of the floating base and $\ddot{\boldsymbol{q}}_j \in \mathbb{R}^{n_j}$ is the vector of joint accelerations (with number of joints defined by n_j). $\mathbf{A}$, $\boldsymbol{b}$, $\boldsymbol{g}$, $\boldsymbol{\tau}$, $\boldsymbol{f}_r$, and

$\mathbf{J}_c$ are the generalized mass matrix, coriolis force, gravitation force, joint torque, augmented reaction force and contact Jacobian, respectively.

An inverse kinematics algorithm that strictly holds task priority, computes joint positions $\boldsymbol{q}$, velocities $\dot{\boldsymbol{q}}$ and accelerations $\ddot{\boldsymbol{q}}$. Resulted $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$ are used within joint controllers (as depicted in Figure 3.4) to stabilize robot posture. This is beneficial for dynamic locomotion, since the joint controller runs in very high frequency [3].

WBIC uses operational space control to directly control the impedance, reducing the control delay.

Prioritized Task Execution

As explained earlier in this section, a strict task hierarchy is executed by using a null-space projection technique. Having $\boldsymbol{q} = \begin{bmatrix} \boldsymbol{q}_f^\top & \boldsymbol{q}_j^\top \end{bmatrix}^\top$ representing full configuration space, the iterative process is formulated as [12]:

$$\begin{aligned} \Delta\boldsymbol{q}_i &= \Delta\boldsymbol{q}_{i-1} + \mathbf{J}^\dagger_{i|pre}\left(\mathbf{e}_i - \mathbf{J}_i\Delta\boldsymbol{q}_{i-1}\right), & (3.12)\\ \dot{\boldsymbol{q}}_i^{\text{cmd}} &= \dot{\boldsymbol{q}}_{i-1}^{\text{cmd}} + \mathbf{J}_{i|pre}{}^\dagger\left(\dot{\boldsymbol{x}}_i^{\text{des}} - \mathbf{J}_i\dot{\boldsymbol{q}}_{i-1}^{\text{cmd}}\right), & (3.13)\\ \ddot{\boldsymbol{q}}_i^{\text{cmd}} &= \ddot{\boldsymbol{q}}_{i-1}^{\text{cmd}} + \overline{\mathbf{J}_{i|pre}^{\text{dyn}}}\left(\ddot{\boldsymbol{x}}_i^{\text{cmd}} - \dot{\mathbf{J}}_i\dot{\boldsymbol{q}} - \mathbf{J}_i\ddot{\boldsymbol{q}}_{i-1}^{\text{cmd}}\right), & (3.14)\end{aligned}$$

where (with $i \geq 1$)

$$\begin{aligned} \mathbf{J}_{i|pre} &= \mathbf{J}_i\mathbf{N}_{i-1},\\ \mathbf{N}_{i-1} &= \boldsymbol{N}_0\boldsymbol{N}_{1|0}\cdots\mathbf{N}_{i-1|i-2}, \end{aligned} \qquad (3.15)$$

$$\begin{aligned} \mathbf{N}_0 &= \mathbf{I} - \mathbf{J}_c^\dagger\mathbf{J}_c,\\ \mathbf{N}_{i|i-1} &= \mathbf{I} - \mathbf{J}^\dagger_{i|i-1}\mathbf{J}_{i|i-1}, \end{aligned} \qquad (3.16)$$

$$\begin{aligned} \Delta\boldsymbol{q}_0, \dot{\boldsymbol{q}}_0^{\text{cmd}} &= \mathbf{0},\\ \ddot{\boldsymbol{q}}_0^{\text{cmd}} &= \overline{\mathbf{J}_c^{\text{dyn}}}\left(-\dot{\mathbf{J}}_c\dot{\boldsymbol{q}}\right), \end{aligned} \qquad (3.17)$$

there the position error $\boldsymbol{e}_i = \boldsymbol{x}_i^{des} - \boldsymbol{x}_i$, $\mathbf{J}_c$ defines a contact Jacobian and is the same as the one in Equation (3.11) $\mathbf{J}_{i|pre}$ is the projection of the i-th task Jacobian into the null space of the prior tasks, the SVD-based[2] pseudo-inverse is denoted by $\{\cdot\}^\dagger$, the dynamically consistent pseudo-inverse is defined as $\bar{\mathbf{J}} = \mathbf{A}^{-1}\mathbf{J}^\top(\mathbf{J}\mathbf{A}^{-1}\mathbf{J}^\top)^{-1}$ which is used for computing acceleration (Equation 3.14), and the acceleration

[2]Singular Value Decomposition

command of i-th task is defined by:

$$\ddot{\boldsymbol{x}}_i^{cmd} = \ddot{\boldsymbol{x}}^{des} + \boldsymbol{K}_p(\boldsymbol{x}_i^{des} - \boldsymbol{x}_i) + \boldsymbol{K}_d(\dot{\boldsymbol{x}}^{des} - \dot{\boldsymbol{x}}) \tag{3.18}$$

where $\boldsymbol{K}_p$ and $\boldsymbol{K}_d$ are the position and velocity feedback gains.

The computed joint commands from Equations (3.12) and (3.13) along with the current joint positions are sent to the joint-level PD controller, and the acceleration commands $\ddot{\boldsymbol{q}}^{cmd}$ are used with the QP optimization (explained in the following section) to compute accurate torque commands $\boldsymbol{\tau}$.

The purpose of the WBC is to produce accurate joint torques $\boldsymbol{\tau}$. However, the PD joint controller, which receives $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$, could differ with the torque command and affect the overall performance (the difference comes from the relaxation of floating base acceleration which is explained in the next section). For this reason, the gains for the joint PD controller are set to relatively small values. The small gains allow the torque commands to apply the desired joint torques $\boldsymbol{\tau}$ efficiently while the joint's current state (position and velocity) is not too far from the commanded one.

Quadratic Programming

The joint torque commands are computed using quadratic programming. The QP in this formulation reduces the error in acceleration command tracking and reaction force command tracking while satisfying inequality constraints on the resultant reaction forces. The optimization is solved using an efficient open-source QP solver [79]. The QP problem is formulated as:

$$\begin{aligned}
\min_{\boldsymbol{\delta}_{\mathbf{f}_r}, \boldsymbol{\delta}_f} \quad & \boldsymbol{\delta}_{\mathbf{f}_r}^\top \mathbf{Q}_r \boldsymbol{\delta}_{\mathbf{f}_r} + \boldsymbol{\delta}_f^\top \mathbf{Q} \boldsymbol{\delta}_f \\
\text{s.t.} \quad & \mathbf{S}_f(\mathbf{A}\ddot{\boldsymbol{q}} + \boldsymbol{b} + \boldsymbol{g}) = \mathbf{S}_f \mathbf{J}_c^\top \mathbf{f}_r && \text{(floating base dynamics)} \\
& \ddot{\boldsymbol{q}} = \ddot{\boldsymbol{q}}^{\text{cmd}} + \begin{bmatrix} \boldsymbol{\delta}_f \\ \mathbf{0}_{n_j} \end{bmatrix} && \text{(acceleration)} \\
& \mathbf{f}_r = \mathbf{f}_r^{\text{cmd}} + \boldsymbol{\delta}_{\mathbf{f}_r} && \text{(reaction forces)} \\
& \mathbf{W}\mathbf{f}_r \geq \mathbf{0} && \text{(contact force constraints),}
\end{aligned} \tag{3.19}$$

where $\mathbf{Q}_r$ and $\mathbf{Q}$ are the penalty matrix for reaction forces and the floating base relaxations, respectively. $\boldsymbol{S}_f$ is the floating base selection matrix (to limit the operations for floating base related variables), $\mathbf{f}_r^{cmd}$ is the higher-level reaction force commands generated in low frequency, $\mathbf{W}$ is the contact constraint matrix, $\mathbf{J}_c$ is the contact Jacobian similar to Equation (3.11), and $\boldsymbol{\delta}_{\mathbf{f}}$ and $\boldsymbol{\delta}_f$ are the relaxation variables for the reaction forces and the floating base acceleration.

Relaxation of floating base acceleration causes the task accelerations to be different from the ones computed in Equation (3.14). This difference allows for the base to freely move during flight phase without being controlled; However it will introduce tracking errors to other tasks, which is why a smaller gain values for the PD joint controllers are favored. Also, during flight phase where there is no contact points with ground, the WBIC completely relaxes the floating base dynamics rather than strictly executing the prioritized tasks (e.g., hierarchical quadratic programming in [47]).

Finally, the joint torque commands $\boldsymbol{\tau}_j$ are generated using the following formulation:

$$\begin{bmatrix} \boldsymbol{\tau}_f \\ \boldsymbol{\tau}_j \end{bmatrix} = \mathbf{A}\ddot{\boldsymbol{q}} + \boldsymbol{b} + \boldsymbol{g} - \mathbf{J}_c^\top \mathbf{f}_r. \tag{3.20}$$

4 Modeling and State Estimator

In this thesis, an approach for a hybrid driving-stepping and jumping control framework for a four-legged wheeled robot is proposed, which is able to perform a jumping motion or stepping with different gait patterns while driving.

The wheels change the kinematics and dynamics of the robot due to their shape, inertia tensor, mass, and free rolling along the x-axis. We tightly integrate the wheels into the state estimator and control system of the open platform MIT mini-cheetah open platform [11]. We achieved the aforementioned integration into the state estimator by modifying the kinematics and dynamics model of the robot to include the dimensions, mass, and inertia tensor of the wheels, by calculating the exact ground contact positions and velocities, and by considering the contribution of the rolling wheels in the state estimation.

Figure 4.1 depicts the overall diagram of our whole-body control framework. The control module receives the user input commands and executes correspondingly a dynamic motion: a hybrid driving-stepping motion or a hybrid driving-jump motion. Chapter 5 describes the structure of the control module.

This chapter defines the coordinate systems in use, the robot's orientation and angular velocity, and the robot's pose, and explains the robot kinematic, robot dynamic, and state estimator modules.

Coordinate Systems

We use the notation ${}_{\square}x$ with the letters $\mathcal{G}$, $\mathcal{B}$, $\mathcal{Y}$ and $\mathcal{R}$ for the global, body, yaw and relative coordinate systems, respectively (e.g. ${}_{\mathcal{B}}\boldsymbol{x}$). The notation ${}^{B}\square_{A}$ denotes a transformation matrix from coordinate system A to B (e.g., ${}^{\mathcal{G}}\mathbf{R}_{\mathcal{B}}$ rotation from body to world coordinate system).

Global. The global coordinate system is a fixed inertial reference system whose origin is on the ground where the robot starts its operation, and whose z-axis points upward. The roll and pitch angles are 0 when the robot is up-right. The yaw angle is 0 when the robot is facing the positive x-axis. Most of the equations for the controller and the state estimator in this thesis use the global coordinate system.

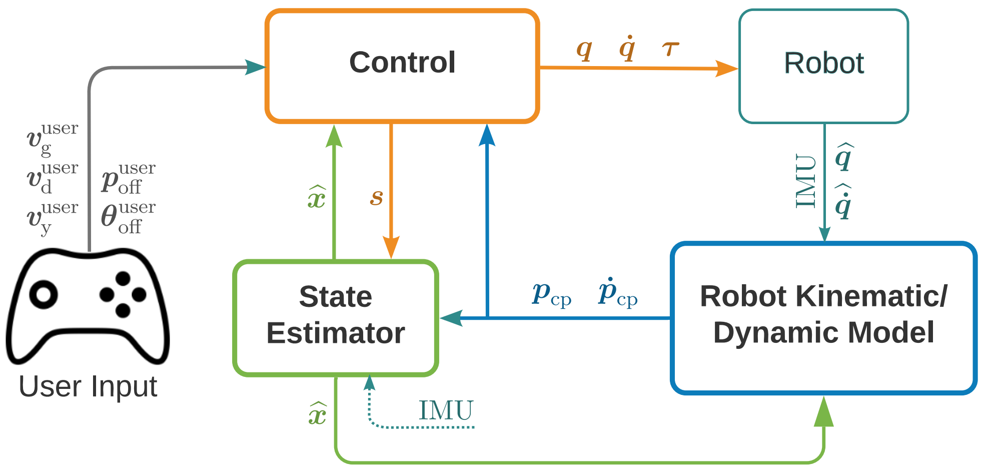


Figure 4.1: **General framework diagram.** The measured joint position and velocities ($\hat{\boldsymbol{q}}$ and $\hat{\dot{\boldsymbol{q}}}$) and the IMU data are sent to the robot model modules which compute the position and velocity of each foot ($\boldsymbol{p}_{\text{cp}}$ and $\dot{\boldsymbol{p}}_{\text{cp}}$). The state estimator module uses the feet information, IMU data, and the foot contact states ($\boldsymbol{s}$) to compute the body position and orientation state ($\hat{\boldsymbol{x}}$). The user input includes the desired driving, stepping and turning velocities ($\boldsymbol{v}_{\text{d}}^{\text{user}}$, $\boldsymbol{v}_{\text{g}}^{\text{user}}$ and $\boldsymbol{v}_{\text{y}}^{\text{user}}$) and the robot pose offsets ($\boldsymbol{\theta}_{\text{off}}^{\text{user}}$ and $\boldsymbol{p}_{\text{off}}^{\text{user}}$) which are used by the control module along with $\hat{\mathbf{x}}$ to compute joint position, velocity and torque commands ($\boldsymbol{q}$, $\dot{\boldsymbol{q}}$ and $\boldsymbol{\tau}$).

Body. The body coordinate system originates at the geometric center of the body and rotates with the robot in the global coordinate system to maintain the same orientation as the robot.

Yaw. The yaw reference system originates from the projection of the geometric center of the robot onto the global ground plane ($z = 0$). Roll and pitch coincide with the orientation of the global coordinate system, but yaw coincides with the orientation of the robot body. This coordinate frame is used for user commands because it is easy for the operator to identify this reference frame by looking at the robot.

Relative The relative reference frame is similar to the yaw reference frame, but the orientation fully coincides with the orientation of the global coordinate system. This reference frame is mainly used in predictive control to define the positions of the feet relative to the kinematic center of the robot.

Orientation and Angular Velocity

The orientation of the robot is given as a vector of Euler angles $\boldsymbol{\theta} = \begin{bmatrix}\phi & \theta & \psi\end{bmatrix}^\top$, which contains the roll (ϕ), pitch (θ) and yaw (ψ) angles of the robot. The transformation from body to global coordinates is expressed by a sequence of rotations as:

$$^{\mathcal{G}}\mathbf{R}_{\mathcal{B}} = \mathbf{R}_z(\psi)\mathbf{R}_y(\theta)\mathbf{R}_x(\phi), \tag{4.1}$$

where $\mathbf{R}_n(\alpha)$ represents a positive rotation of α about the n-axis. The order in which the Euler angular rotations are defined is important [**euler_angles**]. The angular velocity in the global coordinates is formulated using the rate of change $\dot{\boldsymbol{\theta}}$, where the first Euler angle (ϕ) undergoes two additional rotations, the second angle (θ) undergoes one rotation, and the last Euler angle (ψ) undergoes no additional rotations:

$$\begin{aligned}\boldsymbol{\omega} &= \mathbf{R}_z(\psi)\mathbf{R}_y(\theta)\begin{bmatrix}\dot{\phi}\\0\\0\end{bmatrix} + \mathbf{R}_z(\psi)\begin{bmatrix}0\\\dot{\theta}\\0\end{bmatrix} + \begin{bmatrix}0\\0\\\dot{\psi}\end{bmatrix}\\ &= \underbrace{\begin{bmatrix}\cos\theta\cos\psi & -\sin\psi & 0\\ \cos\theta\sin\psi & \cos\psi & 0\\ 0 & 0 & 1\end{bmatrix}}_{\mathbf{R}'}\begin{bmatrix}\dot{\phi}\\\dot{\theta}\\\dot{\psi}\end{bmatrix}.\end{aligned} \tag{4.2}$$

By inverting the Equation (4.2) for the case where the robot is not vertically oriented ($\cos\theta \neq 0$), we can compute the rotational velocity in the body frame as:

$$\dot{\boldsymbol{\theta}} = \underbrace{\begin{bmatrix}\cos\psi/\cos\theta & \sin\psi/\cos\theta & 0\\ -\sin\psi & \cos\psi & 0\\ \cos\psi\tan\theta & \sin\psi\tan\theta & 1\end{bmatrix}}_{\mathbf{R}'^{-1}}\boldsymbol{\omega}. \tag{4.3}$$

Robot Pose

We define the pose of the robot parameterized by $\boldsymbol{\theta}^{\text{pose}}$ and $\boldsymbol{p}^{\text{pose}}$ for orientation and position, respectively. The orientation pose of the robot for roll and pitch angles is obvious, but for yaw angle we need to rotate the ground contact positions with respect to the body so that the robot can drive and step in the actual direction specified before applying the pose. Figure 4.2 represents different poses of the robot.

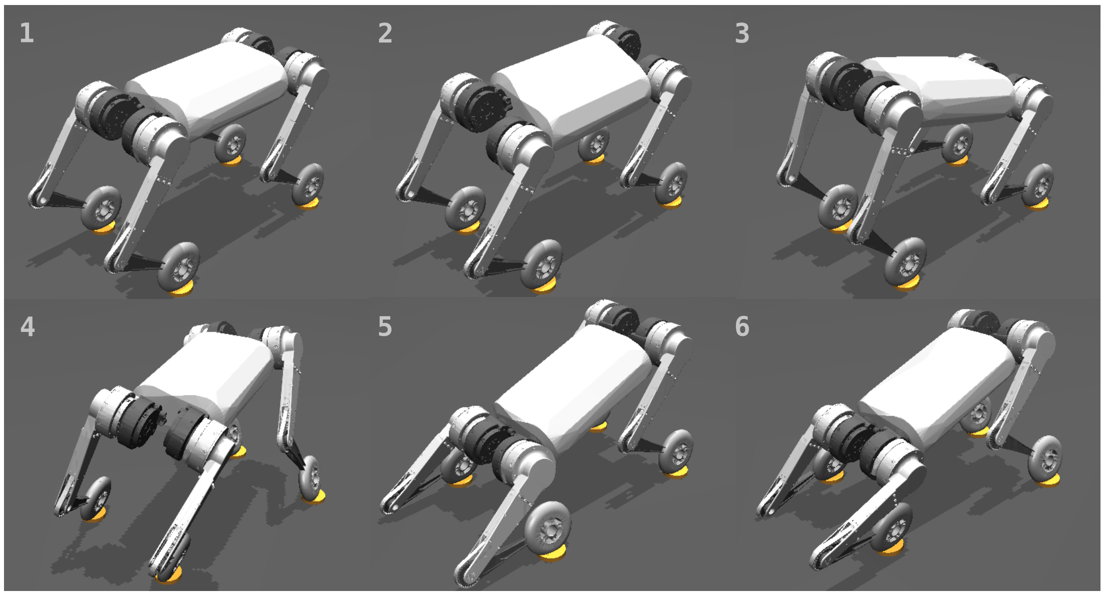


Figure 4.2: **Robot pose examples.** 1. default pose ($\boldsymbol{\theta}^{\text{pose}} = \mathbf{0}$, ${}_z\boldsymbol{p}^{\text{pose}} = 0.3$), 2. positive roll offset, 3. positive pitch offset, 4. positive yaw offset, 5. negative height and pitch angle offsets, 6. positive yaw and negative height and pitch offsets.

4.1 Dynamic Model

Whole-Body Control is a robust dynamic motion control with a general framework that makes it easy to extend to different systems and tasks (see 3.2). In this work, we use the WBIC (described in 3.2.1) to control the robot. The WBIC uses a floating-base model with rigid-body dynamics for legged robots that includes the rotor dynamics of the BLDC motors, and generates the mass matrix and composite inertia values for each link in the robot structure [54, pp. 57-66, 229-243]. The efficient dynamics engine developed by [12] uses linear algebra optimizations through template formatting [70]. With this efficient engine, we can achieve a closed-loop update rate of over 500 Hz when controlling joints.

We modeled the mass and inertia of the added wheels into the rigid-body dynamics of the WBIC. In our setup, a wheel consists of a static part (stater) and a rotating part, which is lighter compared to the static part. All the off-diagonal inertia elements of the rotating part are close to zero; therefore, the total inertia of the added wheel remains the same for any angle of the wheel joint.

With the above assumptions, we simply updated the dynamics of the shank link to include the mass and moment of inertia tensor of the wheels. However, this simple integration ignores the dynamic effect of the wheels during the rotational acceleration of their moving parts. We neglected this for two main reasons: the mass of the wheel's rotating part is negligible compared to the total weight of the shank, and the wheels are mainly used for driving and maintain a constant rotational speed relative to the commanded linear driving speed.

4.2 Kinematic Model

The mini-cheetah quadruped has a rubber ball at the end of each leg as an end effector. The size and shape of these robber balls are small (radius of 1.5 cm); therefore, the authors of related works [2, 1, 12, 7, 21] have previously ignored them and considered the middle of the end effectors as single ground contact points. However, in this work, the wheels that replaced the robber balls have a radius of 5 cm, which is considerable compared to the shank length of 19 cm. Thus, we can no longer consider the ground contact points in the middle of the end effector (Figure 4.3 right).

Our method does not rely on 3D structural information about the contact surface. Hence, we define the ground contact point, as shown in Figure 4.3, using the touch down position $\boldsymbol{p}_{\text{cp}}$ of the end effector on the arbitrary horizontal ground. However, this assumes that the ground surface is flat. The possible errors between the actual contact point and $\boldsymbol{p}_{\text{cp}}$ are negligible and ignored in this work.

We define a specific geometry for the end effector as depicted in Figure 4.3, with radius a_{end} and width b_{end} assuming that $b_{\text{end}} \leq a_{\text{end}}$, giving the touch-down position in the relative coordinate system ($\mathcal{R}$) as:

$$\boldsymbol{p}_{\text{cp}} = \begin{bmatrix} L_3 S_{23} + L_2 S_2 \\ L_1 C_1 + L_3(S_1 C_{23}) + L_2 C_2 S_1 + rS_1 \\ L_1 S_1 - L_3(C_1 C_{23}) - L_2 C_1 C_2 - (rC_1 + b_{\text{end}}) \end{bmatrix}, \tag{4.4}$$

where $r = a_{\text{end}} - b_{\text{end}}$ indicates the length of the last link of the chain assuming the touch-down point, S_i and C_i are the short terms for $\sin q_i$ and $\cos q_i$, $C_{23} = C_2 C_3 - S_2 S_3$ and $S_{23} = S_2 C_3 + C_2 S_3$. The highlighted terms in the equation are resulted from end effector geometry in the kinematics.

The kinematic contribution of leg into the touch-down velocity in world frame is given as:

$$\dot{\boldsymbol{p}}_{\text{cp}_k} = \mathbf{J}\dot{\boldsymbol{q}}, \tag{4.5}$$

where $\mathbf{J}$ is the Jacobian matrix of the kinematic chain of the leg including the end

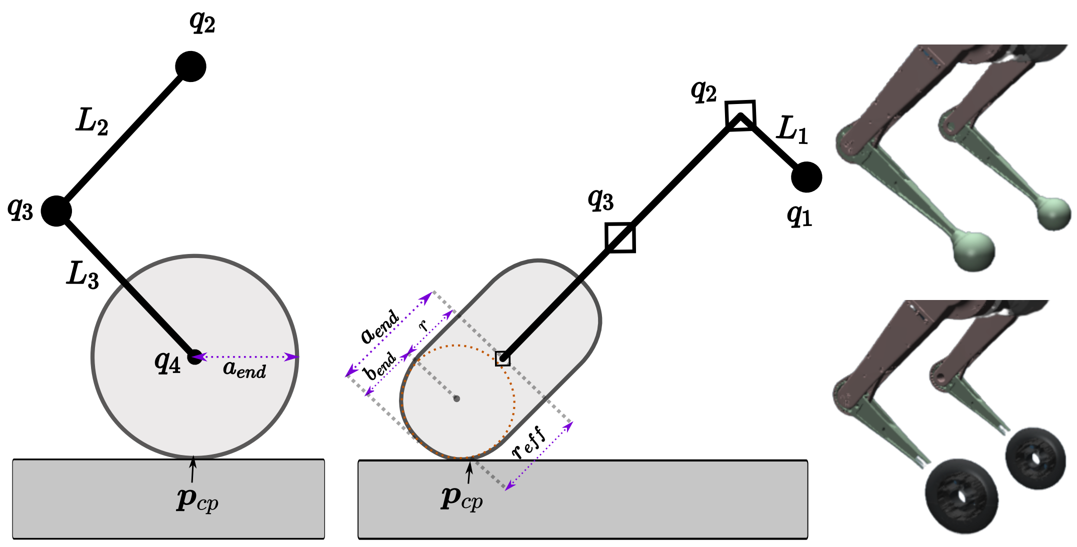


Figure 4.3: **Robot leg kinematics.** Left: Side view of a leg with a wheel as end effector. Middle: front view of the leg. Right: the comparison between the rubber ball and the wheels. Joints are labeled with$q$ and shown as black circles or white rectangles, and links are labeled with L. a_{end} and b_{end} are two radii defining the geometry of the end effector, and $\boldsymbol{p}_{\text{cp}}$ is the ground contact position.

effector geometry, given as:

$$\mathbf{J} = \begin{bmatrix} 0 & L_3C_{23} + L_2C_2 + rC_1 & L_3C_{23} + rS_1 \\ L_3C_1C_{23} + L_2C_1C_2 - L_1S_1 & -L_3S_1S_{23} - L_2S_1S_2 & -L_3S_1S_{23} \\ L_3S_1C_{23} + L_2C_2S_1 + L_1C_1 & L_3C_1S_{23} + L_2C_1S_2 & L_3C_1S_{23} \end{bmatrix}, \tag{4.6}$$

where the highlighted terms refer to the end effector, based on the assumption of touch-down described earlier. We have addressed the kinematic singularity at joint space, which leads to two solutions for the knee joint during the inverse kinematic operation.

We also define r_{eff} as the effective radius of the end effector used by the state estimator (Section 4.3) and the wheel controller (Section 5.6) to accurately convert linear velocity and force into rotational velocity and torque, and vice versa, and formulated as a function of the hip angle in the world coordinate:

$$r_{\text{eff}} = a_{\text{end}} - b_{\text{end}} \sin(q_1 + \phi), \tag{4.7}$$

where ϕ is the roll angle of the body.

4.3 State Estimator

The task of the state estimator is to estimate the position and orientation of the robot body in the world coordinate system. A good state estimator for a quadruped robot should be robust to different types of terrain, gaits, locomotion modes, and travel speeds. It should also take into account the fact that the motions are periodic over an interval and that the system interacts with the environment through multiple intermittent ground contacts.

The legs in contact are the reference points for the state estimation. The estimator relies on these reference points and solves for the position and velocity of the body using forward kinematics and the Jacobian matrix of the leg in contact. Clearly, the estimated position cannot be trusted if a leg is supposed to be in contact but is actually swinging. In the opposite case, if the leg is planned to be swinging but is not, the estimator will ignore it. Neither situation is desirable, but both occur frequently while traversing unstructured terrain or encountering unexpected obstacles.

The control method in this work relies on accurate state estimation of the robot to achieve a hybrid driving stepping locomotion. Both the MPC and WBIC directly use the residual between the desired and estimated robot positions. The MPC calculates the reaction forces and the WBIC refines the reaction forces and finds the joint torques, both of which aim to reduce the aforementioned residual and bring the joint position close to the desired position. However, if the estimated position is incorrect, the controllers will generate inappropriate joint torques, resulting in an undesirable body posture that could cause the robot to fall.

With the addition of wheels, the contact points can roll along the x-axis either by drive commands or by disturbances. State estimation errors are larger when driving with wheels than when stepping because the wheels are continuously turning in one direction and the errors accumulate, leading to larger deviations from the estimated position. However, during stepping, the end effector moves to some extent relative to the body and returns in periodic cycles, causing some errors to cancel each other out. Therefore, accurate modeling of the wheels (as explained in Sections 4.2 and 4.3.2) is important to achieve higher state estimation performance.

We use a KF that combines the end effector contact positions and the comprehensive contact velocities (Section 4.3.2), the wheels' rotational velocity, and the linear acceleration values from the IMU to obtain a position estimate. Furthermore, we obtain the orientation estimate directly from the IMU hardware.

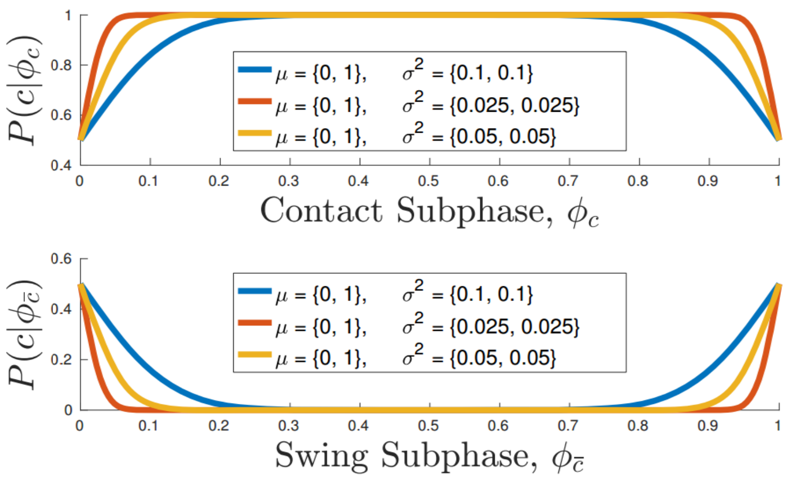


Figure 4.4: **Phase-based probability of contact.** (from [20]) Various values of σ^2 defining the probability of contact given the scheduled state and the percentage progress through the contact and swing state subphases.

4.3.1 Phase-based Contact Probability Model

While most research groups working on state estimation for walking robots use contact/pressure sensors, the method we use in this work does not rely on such sensors. Instead, the expected ground contact trajectory (explained in Section 5.1) is used for each end effector. At each instant, each leg of the robot assumes one of two distinct states, contact or swing, defined by the Boolean variable $s \in \{0 = \text{swing}, 1 = \text{contact}\}$. However, the true value of s is unknown to the robot. The ground contact trajectory switches each leg between two states. $s_\phi \in \{0 = \text{swing}, 1 = \text{contact}\}$, which is the expected value of s at each time step during the control cycle.

Under ideal conditions, we expect the contact state of the leg s to follow and change simultaneously with s_ϕ. In reality, however, there is a small difference between s_ϕ and s because we cannot assume that the ground is completely flat and may contain unforeseen surface heights due to unseen obstacles or rough terrain. In addition, the robot is subject to possible timing delays in its control system that result in early or late takeoff or landing of the leg, as well as unexpected contacts due to inaccurate tracking of the trajectory of the swing leg. Therefore, we cannot simply trust the control system to maintain the planned contact trajectory near the contact change times.

To address these issues, the authors of [20] created a phase-based probabilistic model for the expectation of contact (presented in Figure 4.4). The contact and swing phase variables $\phi_c, \phi_{\bar{c}} \in [0, 1)$ are defined over the period of the ground contact trajectory as a linear function of time:

$$\phi_c = \tfrac{t-t_c}{T_c}, \quad \phi_{\bar{c}} = \tfrac{t-t_{\bar{c}}}{T_{\bar{c}}}, \tag{4.8}$$

where t is the current time, t_c and T_c are the start time and duration of the contact phase, and $t_{\bar{c}}$ and $T_{\bar{c}}$ are similar values for the swing phase. The probability of contact given the current contact state (s_ϕ) and phase values (ϕ_c and $\phi_{\bar{c}}$) is formulated (taken from [20]) as:

$$\begin{aligned} P\left(c|s_\phi,\phi\right) =& \frac{1}{2}\Bigg(s_\phi\left[\operatorname{erf}\left(\frac{\phi-\mu_{c_0}}{\sigma_{c_0}\sqrt{2}}\right)+\operatorname{erf}\left(\frac{\mu_{c_1}-\phi}{\sigma_{c_1}\sqrt{2}}\right)\right]+ \\ & \bar{s}_\phi\left[2+\operatorname{erf}\left(\frac{\mu_{\bar{c}_0}-\phi}{\sigma_{\bar{c}_0}\sqrt{2}}\right)+\operatorname{erf}\left(\frac{\phi-\mu_{\bar{c}_1}}{\sigma_{\bar{c}_1}\sqrt{2}}\right)\right]\Bigg), \end{aligned} \tag{4.9}$$

where s_ϕ chooses between stance or swing, the σ^2 parameters are the variance determined by the variability of the contact phase value, the μ parameters are the expected phase value at the contact switch, and erf is the error function defined for probability theory [75]. Figure 4.4 represents the contact probability for a given scheduled leg contact state during the swing and stance phases for different variances. The contact probability is 0 in the middle of the swing phase and 1 in the middle of the contact phase, high at the beginning and end of the swing phase, but low at the beginning and end of the contact phase.

4.3.2 Comprehensive Contact-point Velocity

The contact point velocity $\dot{\boldsymbol{p}}_{\mathrm{cp}_k}$, given in Equation (4.5), takes into account the robot kinematics and the shape and size of the end effector. However, additional terms must be considered. The body pitch velocity $\dot{\theta}$, the hip velocity $\dot{q}_2$, and the knee velocity $\dot{q}_3$, as well as the body roll velocity $\dot{\phi}$ and the hip-roll velocity $\dot{q}_1$ directly force additional rotational velocity of the end effector (i.e., wheels) along the x and y axes, respectively. These forced rotations result in an additional velocity $\dot{\boldsymbol{p}}_{\mathrm{cp}_w}$ for the contact point in addition to the current rotational velocity $\dot{q}_4$ of the wheel read by encoders. We define $\dot{\boldsymbol{p}}_{\mathrm{cp}_w}$ as the contribution of the end effector to the global velocity of the contact point:

$$\dot{\boldsymbol{p}}_{\mathrm{cp}_w} = \begin{bmatrix}(\dot{\theta}C_1+\dot{q}_2+\dot{q}_3+\dot{q}_4)r_{\mathrm{eff}} & (\dot{\phi}+\dot{q}_1)b_{\mathrm{end}} & 0\end{bmatrix}^\top, \tag{4.10}$$

where r_{eff} (Equation 4.7) is the effective radius of the wheel represented in Figure 4.5. Finally, the comprehensive contact point velocity $\dot{\boldsymbol{p}}_{\text{cp}}$ is written as follows:

$$\dot{\boldsymbol{p}}_{\text{cp}} = \dot{\boldsymbol{p}}_{\text{cp}_k} + \dot{\boldsymbol{p}}_{\text{cp}_w}. \tag{4.11}$$

$\dot{\boldsymbol{p}}_{\text{cp}}$ accurately formulates the velocity at the contact position in world coordinates by taking into account the kinematic shape of the end effector, the velocity feedback from the wheel joint, and the forced velocity terms from other joints on the end effector.

4.3.3 Probabilistic Contact Model Fusion

The KF satisfies the requirements for performing state estimation on legged robots and can use the contact probability (Section 4.3.1) and the various measurements of our robot. One advantage of using probabilities instead of discrete, binary contact states in estimation is to set a confidence value for each contact point and update the KF covariance matrices accordingly.

We extended the state estimator of the mini-Cheetah robot [11] to accurately include the wheels in the estimation process.

In the following, we first define the KF state vector and then further explain the KF model and equations. The notation $k|k-1$ in the formulas of this section stands for estimation at time k considering observations up to and including time $k-1$. The identity matrix $\mathbf{I}_3$ and the zero matrix $\mathbf{0}_3$ are of size 3×3, and $\mathbf{I}_{12}$ and $\mathbf{0}_{12}$ are of size 12×12.

Filter State

The state vector of the filter must be chosen such that the corresponding prediction and measurement equations can be stated consistently. In the approach followed in this thesis, we compose the state vector of the quadruped robot from the position and velocity of the body, the contribution of driving into the position and velocity of the body, and the contact point positions for all legs relative to the body, which take into account the kinematics of the legs. This results in the following state vector:

$$\boldsymbol{x} := \begin{pmatrix} \boldsymbol{p} & \dot{\boldsymbol{p}} & \boldsymbol{p}_w & \dot{\boldsymbol{p}}_w & \boldsymbol{p}_{\text{cp}}^1 \dots \boldsymbol{p}_{\text{cp}}^4 \end{pmatrix} \tag{4.12}$$

where all terms of $\boldsymbol{x}$ are in the world frame, $\boldsymbol{p}$ and $\dot{\boldsymbol{p}}$ are the global position and velocity of the robot, $\boldsymbol{p}_w$ and $\dot{\boldsymbol{p}}_w$ are the contribution of driving with wheels into position and velocity of the robot, and $\boldsymbol{p}_{\text{cp}}^i$ is the touchdown position of leg i relative to the body (Figure 4.3).

Although the robot dynamics are complex, the above state vector allows us to use a simple linear KF. The inclusion of foot contact positions in the filter state, similar to [52], as well as the contribution of driving with wheels into position and velocity, are key points in our filter design that allows for a simple and consistent representation of the model equations. The KF is able to simultaneously correct for the position of the foot contacts, the pose of the main body, and the contribution of driving.

The KF presented represents the uncertainties of the estimated state vector via the covariance matrix $\mathbf{P}$ of the corresponding state error vector $\delta\boldsymbol{x}$

$$\begin{aligned} \mathbf{P} &= \operatorname{Cov}(\delta\boldsymbol{x}) \\ \delta\boldsymbol{x} &= \begin{pmatrix}\delta\boldsymbol{p} & \delta\dot{\boldsymbol{p}} & \delta\boldsymbol{p}_w & \delta\dot{\boldsymbol{p}}_w & \delta\boldsymbol{p}^1_{\mathrm{cp}}\ldots\delta\boldsymbol{p}^4_{\mathrm{cp}}\end{pmatrix} \end{aligned} \tag{4.13}$$

Noise Covariance Update Gain

The phase-based contact probability model in Section 4.3.1 explains how a probability is assigned for each ground contact point. We use this probability information in the KF to define the noise covariance update gain for the prediction and correction steps. We define the contact probability matrix as follows:

$$\underset{12\times 12}{\mathbb{P}_c} = \begin{bmatrix} \mathbb{P}^1_c & \mathbf{0}_3 & \ldots & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbb{P}^2_c & \vdots & \vdots \\ \vdots & \ldots & \mathbb{P}^3_c & \mathbf{0}_3 \\ \mathbf{0}_3 & \ldots & \mathbf{0}_3 & \mathbb{P}^4_c \end{bmatrix} \tag{4.14}$$

where $\underset{3\times 3}{\mathbb{P}^i_c} = P^i\left(c|s_\phi,\phi\right)\mathbf{I}_3$ is a diagonal matrix of the contact probability for leg i. The gain of the noise covariance update is then given by:

$$\underset{12\times 12}{\boldsymbol{\xi}} = \mathbf{I}_{12} + \kappa(\mathbf{I}_{12} - \mathbb{P}_c), \tag{4.15}$$

where κ is a hand-tuned value defined as high suspect number. For the setup in this work, we chose $\kappa = 120$. $\boldsymbol{\xi}$ encodes the confidence value for all contact points and is used at each prediction and correction step of the KF to update the covariance matrices of the process and measurement noise such that only the legs that are in contact contribute to the estimation process.

Prediction Model

Using the prediction equations, we propagate the state from one time step to the next. We use the acceleration measurements of IMU in the prediction model of the sensor fusion method (similar to [52, 44]) defined with:

$$\boldsymbol{u} = {}^{\mathcal{G}}\mathbf{R}_{\mathcal{B}}(\boldsymbol{a}_{\mathrm{IMU}} - \boldsymbol{g}), \tag{4.16}$$

where $\boldsymbol{a}_{\mathrm{IMU}}$ is the linear acceleration read by the sensor, and $\mathbf{R}$ applies the orientation of the body to obtain the acceleration in the global coordinate system.

The standard prediction equations for the KF are shown here as follows:

$$\widehat{\boldsymbol{x}}_{k|k-1} = \mathbf{F}_k \widehat{\boldsymbol{x}}_{k-1} + \mathbf{B}_k \boldsymbol{u}_k, \tag{4.17}$$

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k \mathbf{P}_{k-1} \mathbf{F}_k^\top + \mathbf{Q}_k, \tag{4.18}$$

where $\widehat{\boldsymbol{x}}$ is the estimate of $\boldsymbol{x}$, $\mathbf{P}$ is the covariance matrix of the state vector identical to Equation (4.13), and $\mathbf{F}$ is the state transition matrix applying the system dynamics and is defined as follows:

$$\mathbf{F} = \begin{bmatrix} \begin{matrix} \mathbf{I}_3 & \Delta t\mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{I}_3 & \Delta t\mathbf{I}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{I}_3 \end{matrix} & \mathbf{0}_{12} \\ \mathbf{0}_{12} & \mathbf{I}_{12} \end{bmatrix}. \tag{4.19}$$

$\mathbf{B}$ applies the body acceleration to update body velocity, and is given as:

$$\mathbf{B} = \begin{bmatrix} \mathbf{0}_3 & \Delta t\mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 \end{bmatrix}^\top. \tag{4.20}$$

$\mathbf{Q}$ is the covariance matrix of the process noise that encodes the confidence we place in the accuracy of the phase-based switching model, and is defined as:

$$\mathbf{Q} = \begin{bmatrix} \begin{matrix} \omega_{\boldsymbol{p}}\mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \omega_{\dot{\boldsymbol{p}}}\mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \omega_{\boldsymbol{p}_w}\mathbf{I}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \omega_{\dot{\boldsymbol{p}}_w}\mathbf{I}_3 \end{matrix} & \mathbf{0}_{12} \\ \mathbf{0}_{12} & \omega_{\boldsymbol{p}_{\mathrm{cp}}}\boldsymbol{\xi}\mathbf{I}_{12} \end{bmatrix}, \tag{4.21}$$

where the ω parameters are the process noise values for the corresponding variables in the state vector $\boldsymbol{x}$.

The foot contacts are assumed to remain stationary, but in reality they may slip. Therefore, we have included the white noise terms $\omega_{\boldsymbol{p}_{\text{cp}}}$ in the covariance matrix $\mathbf{Q}$ to account for some degree of foot slippage. In addition, the noise covariance update gain $\boldsymbol{\xi}$ is applied to the confidence in the foot contact positions. $\boldsymbol{\xi}$ sets the noise parameter of each foot to a very large value when it has no ground contact. This allows the corresponding foot to shift and reset its position estimate when it makes ground contact again, and remove the old foot position from the estimation process. This process deals with the contact switches in the ground contact trajectory during hybrid driving-stepping.

Correction: Measurement Model

Knowing that the prediction is likely to contain inaccuracies in the hybrid locomotion scheme, we can use available measurements to correct the prediction and obtain a more educated estimate for the position. Since the wheels are integrated into the hybrid quadruped robot, the measurement vector $\boldsymbol{z}$ includes the contact position and kinematic contribution, as well as the driving contribution to the contact point velocity for all legs, as follows:

$$\boldsymbol{z} := \left(\boldsymbol{p}^1_{\text{cp}} \cdots \boldsymbol{p}^4_{\text{cp}} \quad \dot{\boldsymbol{p}}^1_{\text{cp}_k} \cdots \dot{\boldsymbol{p}}^4_{\text{cp}_k} \quad \dot{\boldsymbol{p}}^1_{\text{cp}_w} \cdots \dot{\boldsymbol{p}}^4_{\text{cp}_w} \right), \tag{4.22}$$

where all terms of $\boldsymbol{z}$ are in the global frame, $\boldsymbol{p}^i_{\text{cp}}$ is the measured contact position of leg i relative to the body, $\dot{\boldsymbol{p}}^i_{\text{cp}_k}$ and and $\dot{\boldsymbol{p}}^i_{\text{cp}_w}$ are the measured linear contact point velocity from the kinematics and driving, respectively (Equations (4.5) and (4.10)).

The separate inclusion of the kinematic and driving contributions in the measurement vector $\boldsymbol{z}$ plays a key role in the estimation process to correctly capture the effects of stepping and driving on the position of the hybrid quadruped robot. The correction equations of the standard KF are presented as follows:

$$\widetilde{\boldsymbol{y}}_k = \boldsymbol{z}_k - \mathbf{H}_k \widehat{\boldsymbol{x}}_{k|k-1} \tag{4.23}$$

$$\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^\top + \mathbf{R}_k \tag{4.24}$$

$$\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{H}_k^\top \mathbf{S}_k^{-1} \tag{4.25}$$

$$\widehat{\boldsymbol{x}}_{k|k} = \widehat{\boldsymbol{x}}_{k|k-1} + \mathbf{K}_k \widetilde{\boldsymbol{y}}_k \tag{4.26}$$

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_{k|k-1}, \tag{4.27}$$

where $\widetilde{\boldsymbol{y}}$ is the measurement residual (or innovation), $\mathbf{S}$ is the covariance of the residual (or innovation covariance), and $\mathbf{K}$ is the Kalman gain.

$\mathbf{R}$ is the covariance matrix of the measurement noise, which encodes the confidence we place in the accuracy of the measurements, and is defined as follows:

$$\mathbf{R} = \boldsymbol{\xi} \begin{bmatrix} \nu_{\boldsymbol{p}_{\mathrm{cp}}} \mathbf{I}_{12} & \mathbf{0}_{12} & \mathbf{0}_{12} \\ \mathbf{0}_{12} & \nu_{\dot{\boldsymbol{p}}_{\mathrm{cp}_k}} \mathbf{I}_{12} & \mathbf{0}_{12} \\ \mathbf{0}_{12} & \mathbf{0}_{12} & \nu_{\dot{\boldsymbol{p}}_{\mathrm{cp}_w}} \mathbf{I}_{12} \end{bmatrix}, \tag{4.28}$$

where the ν parameters are the measurement noise values for the corresponding variables in $\boldsymbol{z}$. The noise covariance update gain $\boldsymbol{\xi}$ is applied to the measurement confidence. Similar to the process noise covariance, $\boldsymbol{\xi}$ sets the noise parameter of each foot to a very large value when it is not in contact with the ground, which prevents the corresponding foot measurements from participating in the estimation process. Therefore, we can rotate the wheels at a different speed during the swing phase without affecting the estimation process. In Section 5.6 we talk about how we exploit this to achieve better control.

Finally, we define the observation matrix $\mathbf{H}$ with the following Jacobian:

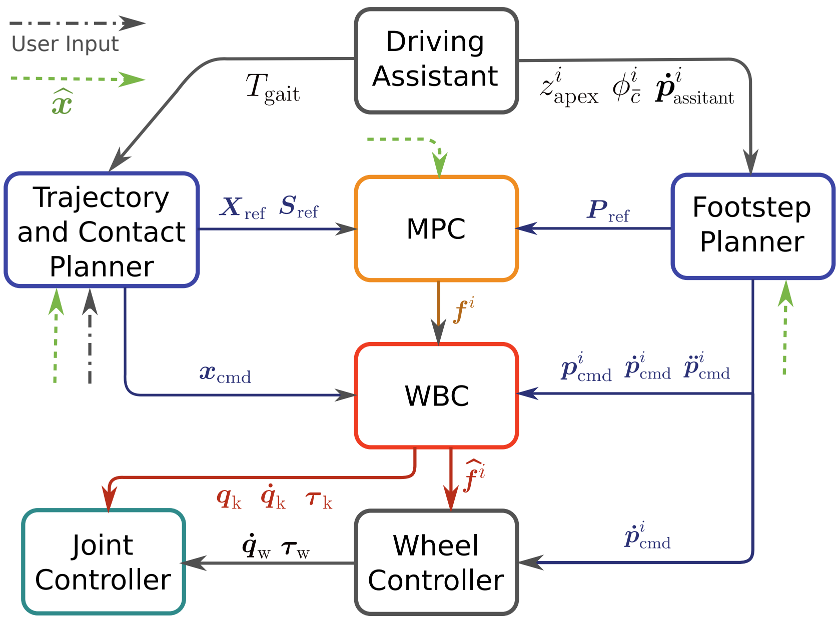
$$\mathbf{H} = \left[\begin{array}{cccc|c} \mathbf{I}_3 & \mathbf{0}_3 & -\mathbf{I}_3 & \mathbf{0}_3 & \\ \mathbf{I}_3 & \mathbf{0}_3 & -\mathbf{I}_3 & \mathbf{0}_3 & -\mathbf{I}_{12} \\ \mathbf{I}_3 & \mathbf{0}_3 & -\mathbf{I}_3 & \mathbf{0}_3 & \\ \mathbf{I}_3 & \mathbf{0}_3 & -\mathbf{I}_3 & \mathbf{0}_3 & \\ \mathbf{0}_3 & \mathbf{I}_3 & \mathbf{0}_3 & -\mathbf{I}_3 & \\ \mathbf{0}_3 & \mathbf{I}_3 & \mathbf{0}_3 & -\mathbf{I}_3 & \mathbf{0}_{12} \\ \mathbf{0}_3 & \mathbf{I}_3 & \mathbf{0}_3 & -\mathbf{I}_3 & \\ \mathbf{0}_3 & \mathbf{I}_3 & \mathbf{0}_3 & -\mathbf{I}_3 & \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{I}_3 & \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{I}_3 & \mathbf{0}_{12} \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{I}_3 & \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{I}_3 & \end{array}\right]. \tag{4.29}$$

The experimental results presented in Chapter 6 evaluate the performance of the hybrid driving-stepping position tracking.

5 Control Framework

Dynamic motions such as jumps are challenging because they involve a flight phase and require the robot's body to deviate from ground plane—the body tilts significantly during the jump—which requires a full 3D orientation model. In addition, a controller that anticipates the upcoming changes in the contact sequence is required to compute the ground reaction forces before the robot enters the flight phase. For example, at the beginning of the jump, the controller should ensure that the base of the robot has an upward velocity while it tilts backward by applying large ground reaction forces to the front feet. After the front feet leave the ground, large reaction forces should be applied to the rear feet, resulting in a jumping motion. The reaction forces should also be optimized considering the flight duration so that the robot lands on the floor plane with low inclination. After landing, the controller must apply enough force to counteract gravity without the robot's height and orientation deviating too far from the commanded pose. Without knowledge of the impending contact pattern of the feet, it is very hard to effectively stabilize these types of motions. When less than three feet are on the ground, finding reaction forces that would simultaneously control the orientation and position of the body would be unfeasible. The resulted underactuation is another challenge to the control of a dynamic motion. Another challenge to locomotion is the constraints imposed by contact with the ground. The robot is only allowed to exert a limited amount of force on its feet otherwise the robot might slip, which limits the force that can be exerted on the body. In addition, the forces along the z-axis cannot be negative because the robot cannot pull itself closer to the ground.

We define a model predictive method that optimizes the future state and control sequence to minimize the given objective function (Section 5.4). We developed the driving-stepping and jumping behaviors to generate realistic reference trajectories which are used by the MPC to control the robot. The MPC uses a time-varying model of the robot that includes the shape, mass, and inertia of the wheel and generates approximate values for the composite inertia tensor, effective mass, and CoM of the robot for the given trajectories. The time-varying model in the MPC is accurate for the first time step, but it is an approximation for future time steps because it is based on the expected robot state in the future, which will change



User Input
$\widehat{x}$
Driving Assistant
T_{gait}
z^i_{apex} $\phi^i_{\bar{c}}$ $\dot{\boldsymbol{p}}^i_{\text{assitant}}$
Trajectory and Contact Planner
$\boldsymbol{X}_{\text{ref}}$ $\boldsymbol{S}_{\text{ref}}$
MPC
$\boldsymbol{P}_{\text{ref}}$
Footstep Planner
$\boldsymbol{f}^i$
$\boldsymbol{x}_{\text{cmd}}$
WBC
$\boldsymbol{p}^i_{\text{cmd}}$ $\dot{\boldsymbol{p}}^i_{\text{cmd}}$ $\ddot{\boldsymbol{p}}^i_{\text{cmd}}$
$\boldsymbol{q}_{\text{k}}$ $\dot{\boldsymbol{q}}_{\text{k}}$ $\boldsymbol{\tau}_{\text{k}}$
$\widehat{\boldsymbol{f}}^i$
Joint Controller
$\dot{\boldsymbol{q}}_{\text{w}}$ $\boldsymbol{\tau}_{\text{w}}$
Wheel Controller
$\dot{\boldsymbol{p}}^i_{\text{cmd}}$

Figure 5.1: **Control framework diagram.** $\widehat{\boldsymbol{x}}$ is the body state vector (position and orientation) from the state estimation module (Section 4.3). The superscript $i \in \{1 \;\ldots\; 4\}$ denotes the leg number. The driving assistant module dynamically defines gait cycle time (T_{gait}), and swing height, swing duration, and stance leg correction velocity (z^i_{apex}, $\phi^i_{\bar{c}}$ and $\dot{\boldsymbol{p}}^i_{\text{assistant}}$, respectively) for each leg, based on the leg utility values. The user input (Figure 4.1) is processed by the trajectory and contact planner module to generate state and contact reference trajectories ($\boldsymbol{X}_{\text{ref}}$ and $\boldsymbol{S}_{\text{ref}}$) for the prediction horizon, and the state commands ($\boldsymbol{x}_{\text{cmd}}$) at the current control time. The footstep planner module generates feet reference trajectory and commands ($\boldsymbol{P}_{\text{ref}}$ and $\boldsymbol{p}^i_{\text{cmd}}$) based on robot's state and the information given by the driving assistant module. The MPC module uses the reference trajectories and computes the optimal reaction forces ($\boldsymbol{f}^i$) for all contact points. The WBC module executes the prioritized tasks for body orientation, body position, and foot positions using the corresponding commands received from other modules, and generates commands for legs' joint position, velocity, and torque ($\boldsymbol{q}_{\text{k}}$, $\dot{\boldsymbol{q}}_{\text{k}}$, and $\boldsymbol{\tau}_{\text{k}}$) as well as the refined reaction forces ($\widehat{\boldsymbol{f}}^i$). The wheel controller module uses the commanded foot velocities from the footstep planner and the refined reaction forces from WBC modules to compute the rotational speed and torque for the wheels ($\dot{\boldsymbol{q}}_{\text{w}}$ and $\boldsymbol{\tau}_{\text{w}}$). Finally, the joint controller module applies the joint commands to the robot.

due to optimizations or disturbances. We enhance the control output of the MPC method with a WBC to achieve accurate and fast joint control. Figure 5.1 shows the diagram of the control framework.

Our framework allows different types of behaviors to be defined and incorporated into the system. Each behavior receives the higher level commands (or the operator's commands) and satisfies the behavior's goals by generating the control commands for the MPC with a low update rate (80Hz) and for the WBIC with a high update rate (500Hz).

Control commands are generated using the trajectory, contact, and footstep planning modules, which are discussed in more detail in this chapter, and consist of the body state command and the trajectories for the body state, contact state, and foot positions. These trajectories used by the MPC are defined for the horizon length of N. The duration between each time step of the trajectory is denoted by Δt and is determined by each behavior based on the its duration and the horizon length. The value of N can be configured by the user based on the processing power of the computer, and should be coarse enough for the controller to achieve a reasonable update rate. However, for future work, N could be set dynamically during runtime to take advantage of all available processing capacity at any given time. In this work, we have implemented two types of behaviors:

Hybrid Driving-Stepping The goal of the driving-stepping behavior is to allow the robot to perform arbitrary gaits while simultaneously driving with the wheels. When the operator commands only driving (without stepping), the controller aims for a minimum amount of corrective swing actions of each foot to perform a more natural driving locomotion. To this end, we introduce the driving assistant in Section 5.3.

Since the trajectories for driving-stepping behavior are periodic, the horizon window shifts in time and maintains a constant size of N at all times.

The default pose parameters ($\boldsymbol{\theta}_{\text{def}}$ and $\boldsymbol{p}_{\text{def}}$) are configured to reduce the CoT, i.e., the body is parallel to the ground (with pitch and roll angles set to zero), and the pose height is set manually (for our robot, 32cm).

Hybrid Driving-Jumping The goal of the driving-jumping behavior is to perform a jumping motion while driving with wheels, which allows the robot to jump onto a path with a different height or to overcome upcoming obstacles by jumping over them. By maintaining the travel speed instead of stopping at a height difference (or in front of an obstacle), the robot achieves a higher average locomotion speed.

Unlike the driving-stepping behavior, the trajectories for the driving-jumping are not periodic. Instead, they are executed only once by the jump motion. For this reason, the size of the horizon window at the beginning of the jump motion is N and gets reduced to $N-s$ according to the elapsed time steps s of the jump motion, keeping the time step duration Δt constant for each step of the control. Reducing the size of the horizon window during the jump is ideal because the controller only operates effectively at the beginning of the jump when there are ground contacts, but as the jump motion continues and the robot flies, the controller no longer regulates the system, so the size of the horizon window no longer matters. Lower body height is configured in $\boldsymbol{p}_{\mathrm{def}}$ for the default pose to give the body more time to accelerate, resulting in an increase in the maximum jump height.

In the following sections we explain the trajectory and contact planner, footstep planner, and driving assistant modules which are responsible for generating control commands used for the MPC and WBC modules described in Sections 5.4 and 5.5.

5.1 Trajectory and Contact Planner

The trajectory and contact planner module generates body state commands as well as body state and contact trajectories for the duration of each behavior. To this end, we first explain the input modifier, which aims to filter user input data and compute acceleration commands for driving and stepping, and then discuss the above commands and trajectories for the driving-stepping and driving-jumping behaviors.

5.1.1 Input Modifier

User input is provided in the yaw reference frame, because expressing this information in the yaw frame is more intuitive for the operator. User input data consists of: desired gait velocity $\boldsymbol{v}_{\mathrm{g}}^{\mathrm{user}}$, desired drive speed $\boldsymbol{v}_{\mathrm{d}}^{\mathrm{user}}$, desired turning velocity $\boldsymbol{v}_{\mathrm{y}}^{\mathrm{user}}$, pose offsets $\boldsymbol{\theta}_{\mathrm{off}}$ and $\boldsymbol{p}_{\mathrm{off}}$ (Section 4), locomotion information (gait), and jump information (jump height). The gait type is set by the user and the jump height h_J is determined based on the size of the obstacle (height and depth).

Since the desired velocity commands could be changed erratically by the user, and the pose offset and desired gait cannot be used directly by the controller, the input modifier converts and augments the user input data into appropriate commands for the controller module. It takes into account the current velocities of the robot and accelerates them toward desired velocities using filtered stepping,

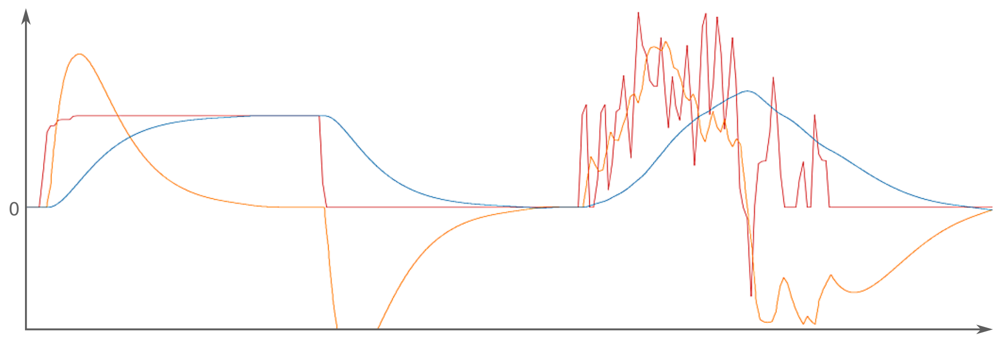


Figure 5.2: **Input modifier.** The user velocity command in red and the filtered acceleration and velocity commands resulting from Equation (5.1) in orange and blue, respectively. The operator's velocity commands are filtered on the right side of the graph.

driving, and turning acceleration commands $\boldsymbol{a}_{\mathrm{g}}$, $\boldsymbol{a}_{\mathrm{d}}$, and $\boldsymbol{a}_{\mathrm{y}}$. Filtering acceleration helps to smooths the jerk value. The filtering formulation is defined as follows:

$$\begin{aligned} \boldsymbol{a}_{j_k} &= (1-\beta)\boldsymbol{a}_{j_{k-1}} + \beta\frac{\boldsymbol{v}_j^{\mathrm{user}} - \boldsymbol{v}_{j_{k-1}}}{\boldsymbol{v}_j^{\mathrm{max}}}, \quad \left|\boldsymbol{a}_{j_k}\right| \leq \boldsymbol{a}_j^{\mathrm{max}}, \\ \boldsymbol{v}_{j_k} &= \boldsymbol{v}_{j_{k-1}} + \boldsymbol{a}_{j_k}\Delta t, \quad \left|\boldsymbol{v}_{j_k}\right| \leq \boldsymbol{v}_j^{\mathrm{max}}, \end{aligned} \tag{5.1}$$

$$\begin{aligned} \boldsymbol{a}_j^{\mathrm{cmd}} &= {}^{\mathcal{G}}\mathbf{R}_{\mathcal{Y}}\boldsymbol{a}_{j_k}, \\ \boldsymbol{v}_j^{\mathrm{cmd}} &= {}^{\mathcal{G}}\mathbf{R}_{\mathcal{Y}}\boldsymbol{v}_{j_k}, \end{aligned} \tag{5.2}$$

where the index $j \in \left\{ g:\text{stepping} \quad d:\text{driving} \quad y:\text{turning} \right\}$ indicates the stepping, driving or turning formulation, the index k is the time step, β is the filter value, $\boldsymbol{v}^{\mathrm{max}}$ and $\boldsymbol{a}^{\mathrm{max}}$ are the defined maximum velocity and acceleration parameters in the yaw reference frame, $\boldsymbol{a}$ and $\boldsymbol{v}$ are the current filtered accelerations and velocities in the yaw reference frame, and $\boldsymbol{a}^{\mathrm{cmd}}$ and $\boldsymbol{v}^{\mathrm{cmd}}$ are in the global coordinate system rotated by ${}^{\mathcal{G}}\mathbf{R}_{\mathcal{Y}} = \mathbf{R}_z(\psi)$. The user commands only the xy-axis speed and yaw rotation rate, so the z-axis elements of driving and stepping ($\boldsymbol{v}_{\mathrm{g}}$, $\boldsymbol{v}_{\mathrm{d}}$, $\boldsymbol{a}_{\mathrm{g}}$, and $\boldsymbol{a}_{\mathrm{d}}$) and xy-axis elements of turning ($\boldsymbol{v}_{\mathrm{y}}$ and $\boldsymbol{a}_{\mathrm{y}}$) are set to zero. Figure 5.2 shows the example of the resulting smooth trajectory for acceleration and velocity. The user only enters the offsets for the position $\boldsymbol{p}_{\mathrm{off}}^{\mathrm{user}} := \begin{pmatrix} 0 & 0 & h_{\mathrm{off}} \end{pmatrix}$ and orientation $\boldsymbol{\theta}_{\mathrm{off}}^{\mathrm{user}}$ to the default position $\boldsymbol{p}_{\mathrm{def}} := \begin{pmatrix} 0 & 0 & h_{\mathrm{def}} \end{pmatrix}$ and orientation $\boldsymbol{\theta}_{\mathrm{def}}$ of the controller's pose. The input modifier computes the position and orientation for the target pose ($\boldsymbol{p}^{\mathrm{pose}}$ and $\boldsymbol{\theta}^{\mathrm{pose}}$) by summing these offsets to the default pose, taking into account the posture limits that the robot can hold:

$$\begin{aligned} \boldsymbol{p}_k^{\text{pose}} &= \boldsymbol{p}_{\text{def}} + \boldsymbol{p}_{\text{off}_k} \\ \boldsymbol{\theta}_k^{\text{pose}} &= \boldsymbol{\theta}_{\text{def}} + \boldsymbol{\theta}_{\text{off}_k} \end{aligned}, \tag{5.3}$$

where $\boldsymbol{p}_{\text{off}_k}$ and $\boldsymbol{\theta}_{\text{off}_k}$ are the pose offset variables at time k approaching the desired pose offsets $\boldsymbol{p}_{\text{off}}^{\text{user}}$ and $\boldsymbol{\theta}_{\text{off}}^{\text{user}}$ while taking into account the controller's maximum pose velocities $\dot{\boldsymbol{p}}^{\max}$ and $\dot{\boldsymbol{\theta}}^{\max}$, as:

$$\boldsymbol{p}_{\text{off}_k} = \boldsymbol{p}_{\text{off}_{k-1}} + \dot{\boldsymbol{p}}_k^{\text{pose}} \Delta t, \quad {}_i\dot{\boldsymbol{p}}_k^{\text{pose}} = \begin{cases} {}_i\dot{\boldsymbol{p}}^{\max}, & \text{if } {}_i\boldsymbol{p}_{\text{off}_{k-1}} < {}_i\boldsymbol{p}_{\text{off}}^{\text{user}} \\ 0, & \text{if } {}_i\boldsymbol{p}_{\text{off}_{k-1}} = {}_i\boldsymbol{p}_{\text{off}}^{\text{user}} \\ -{}_i\dot{\boldsymbol{p}}^{\max}, & \text{otherwise} \end{cases}, \tag{5.4}$$

$$\boldsymbol{\theta}_{\text{off}_k} = \boldsymbol{\theta}_{\text{off}_{k-1}} + \dot{\boldsymbol{\theta}}_k^{\text{pose}} \Delta t, \quad {}_i\dot{\boldsymbol{\theta}}_k^{\text{pose}} = \begin{cases} {}_i\dot{\boldsymbol{\theta}}^{\max}, & \text{if } {}_i\boldsymbol{\theta}_{\text{off}_{k-1}} < {}_i\boldsymbol{\theta}_{\text{off}}^{\text{user}} \\ 0, & \text{if } {}_i\boldsymbol{\theta}_{\text{off}_{k-1}} = {}_i\boldsymbol{\theta}_{\text{off}}^{\text{user}} \\ -{}_i\dot{\boldsymbol{\theta}}^{\max}, & \text{otherwise} \end{cases}, \tag{5.5}$$

where left-subscript i denotes the i-th element of the vectors.

5.1.2 State Trajectory and Command

We define the body state with its orientation, position, rotational velocity and linear velocity in the global coordinate system with:

$$\boldsymbol{x} := \begin{pmatrix} \boldsymbol{\theta} & \boldsymbol{p} & \boldsymbol{\omega} & \dot{\boldsymbol{p}} \end{pmatrix}. \tag{5.6}$$

The state trajectory is the reference trajectory over the horizon window that the predictive controller (Section 5.4) tracks, and is defined by the sequence of body states in the global coordinate system as:

$$\boldsymbol{X}_{\text{ref}} := \begin{pmatrix} \boldsymbol{x}_1 & \dots & \boldsymbol{x}_N \end{pmatrix}. \tag{5.7}$$

We design the state trajectory based on the generated commands from the input modifier (Section 5.1.1), rather than directly using the user input, resulting in smooth trajectories.

The state command is used for WBIC tasks (see Section 5.5) and is defined as follows:

$$\boldsymbol{x}_{\text{cmd}} := \begin{pmatrix} \boldsymbol{\theta} & \boldsymbol{p} & \dot{\boldsymbol{\theta}} & \dot{\boldsymbol{p}} & \ddot{\boldsymbol{\theta}} & \ddot{\boldsymbol{p}} \end{pmatrix}, \tag{5.8}$$

where $\ddot{\boldsymbol{p}}$ and $\ddot{\boldsymbol{\theta}}$ are the linear and angular accelerations of the body, respectively.

These accelerations are not directly modeled by the MPC, but are used by WBIC to refine the output of the MPC and more accurately track the state commands.

At each iteration of the controller, we recalculate the state trajectory and initialize it with the global tracked position and heading. In the following, we first introduce the global track of position and heading, and then explain how the hybrid driving-stepping and driving-jumping behaviors construct the state trajectory and state command to satisfy their behavioral goals.

Hybrid Driving-Stepping

Global Track. We track the robot's position and heading in the global coordinate system by integrating the commands of the input modifier while maintaining a maximum distance from the position and heading estimates of the state estimator at all times (Section 4.3) by:

$$\begin{aligned} \boldsymbol{p}_k^{\text{track}} &= \boldsymbol{p}_{k-1}^{\text{track}} + (\boldsymbol{v}_g^{\text{cmd}} + \boldsymbol{v}_d^{\text{cmd}})\Delta t, \quad \left|\boldsymbol{p}_k^{\text{track}} - \boldsymbol{p}^{\text{estimate}}\right| \leq \boldsymbol{p}^{\text{dist}} \\ \boldsymbol{\theta}_k^{\text{track}} &= \boldsymbol{\theta}_{k-1}^{\text{track}} + \boldsymbol{v}_y^{\text{cmd}}\Delta t, \quad \left|\boldsymbol{\theta}_k^{\text{track}} - \boldsymbol{\theta}^{\text{estimate}}\right| \leq \boldsymbol{\theta}^{\text{dist}}, \end{aligned} \tag{5.9}$$

where the index $k-1$ indicates the previous time step, $\boldsymbol{p}^{\text{dist}}$ and $\boldsymbol{\theta}^{\text{dist}}$ are the configured maximum distance between the tracked and estimated values for position and orientation, respectively, and $\boldsymbol{v}^{\text{cmd}}$ is calculated in Equation (5.1). When the robot is pushed externally, it will recover its previous position and heading, as long as the maximum distance is not violated. In our setup, the maximum distances are 15 cm for the position and 6 degrees for the heading.

Horizontal Compensation. In reality, the predicted reference trajectory may differ from the executed one due to violations of the flat ground assumption or to approximation errors resulted from the discrete-time implementation of the dynamics (especially for the simultaneous translation and rotation velocity commands). This difference becomes even larger at higher locomotion speeds. Therefore, the reaction forces output by the MPC may not result in the desired pitch and roll angles of the body and adversely affect the control performance.

The horizontal compensation tracks the errors in the pitch and roll angles between the desired and estimated values for the commanded velocities in real time, and sums these errors (using leaky integration) to find the correction parameters ϕ^{int} and θ^{int} for the roll and pitch angles, respectively, and construct the compensation vector:

$$\boldsymbol{\theta}^{\text{comp}} = \alpha \begin{pmatrix} \phi^{\text{int}} & \theta^{\text{int}} & 0 \end{pmatrix}, \tag{5.10}$$

where α is the gain defining the effect of compensation and it is based on the norm of commanded velocities ($\alpha \approx ||\boldsymbol{v}_{\mathrm{d}}^{\mathrm{cmd}} + \boldsymbol{v}_{\mathrm{g}}^{\mathrm{cmd}}||$). $\boldsymbol{\theta}^{\mathrm{comp}}$ changes dynamically at runtime to account for the inevitable errors over the desired roll and pitch resulted from the aforementioned causes.

Formulation. During the driving-stepping locomotion, the linear and angular velocities of the robot body are defined as follows:

$$\dot{\boldsymbol{\theta}}_k = \dot{\boldsymbol{\theta}}_k^{\mathrm{pose}} + \boldsymbol{v}_{\mathrm{y}_k}, \tag{5.11}$$

$$\dot{\boldsymbol{p}}_k = \dot{\boldsymbol{p}}_k^{\mathrm{pose}} + \boldsymbol{v}_{\mathrm{g}_k}^{\mathrm{cmd}} + \boldsymbol{v}_{\mathrm{d}_k}^{\mathrm{cmd}}, \tag{5.12}$$

where $\dot{\boldsymbol{\theta}}^{\mathrm{pose}}$, $\dot{\boldsymbol{p}}^{\mathrm{pose}}$, $\boldsymbol{v}_{\mathrm{y}}$, $\boldsymbol{v}_{\mathrm{d}}^{\mathrm{cmd}}$, and $\boldsymbol{v}_{\mathrm{g}}^{\mathrm{cmd}}$ are generated by iteration over Equations (5.5, 5.4, 5.1, and 5.2).

The state trajectory is generated based on the commanded velocities and accelerations for driving, stepping, and turning, and the commanded orientation and position of the target pose by the following equations:

$$\boldsymbol{x}_k = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{R}_{k-1}^{-1}\Delta t & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \Delta t \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} \boldsymbol{x}_{k-1} + \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{R}_{k-1}\dot{\boldsymbol{\theta}}_{k-1} \\ \dot{\boldsymbol{p}}_{k-1} \end{bmatrix}, \tag{5.13}$$

$$\boldsymbol{x}_0 = \begin{bmatrix} \boldsymbol{\theta}^{\mathrm{track}} + \boldsymbol{\theta}_0^{\mathrm{pose}} + \boldsymbol{\theta}^{\mathrm{comp}} \\ \boldsymbol{p}^{\mathrm{track}} + \boldsymbol{p}_0^{\mathrm{pose}} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \tag{5.14}$$

where the subscript k stands for the time step in the trajectory ($1 \leq k \leq N$), $\boldsymbol{x}_0$ contains the current pose and the current tracked position and orientation, $\mathbf{R}_{k-1}$ is the rotation matrix computed similarly to $\mathbf{R}'$ in Equation (4.2) from the orientation in $\boldsymbol{x}_{k-1}$, $\boldsymbol{\theta}^{\mathrm{track}}$ and $\boldsymbol{p}^{\mathrm{track}}$ are the tracked heading and position at the current time step, and $\boldsymbol{\theta}_0^{\mathrm{pose}}$ and $\boldsymbol{p}_0^{\mathrm{pose}}$ are given by Equation (5.3) at the current time step.

The state command vector is constructed using the velocity and acceleration values in the current time step of the controller as follows:

$$\boldsymbol{x}_{\text{cmd}} = \begin{bmatrix} \boldsymbol{\theta}^{\text{track}} + \boldsymbol{\theta}_0^{\text{pose}} + \dot{\boldsymbol{\theta}}_0 \Delta t \\ \boldsymbol{p}^{\text{track}} + \boldsymbol{p}_0^{\text{pose}} + \dot{\boldsymbol{p}}_0 \Delta t \\ \dot{\boldsymbol{\theta}}_0 \\ \dot{\boldsymbol{p}}_0 \\ \boldsymbol{a}_{\text{y}} \\ \boldsymbol{a}_{\text{g}}^{\text{cmd}} + \boldsymbol{a}_{\text{d}}^{\text{cmd}} \end{bmatrix}. \tag{5.15}$$

Hybrid Driving-Jumping

During the hybrid driving-jumping motion, locomotion consists only of driving without stepping, and all acceleration and velocity commands from the input modifier except the driving velocity ($\boldsymbol{v}_{\text{d}}^{\text{cmd}}$) are set to zero. In this way, the robot follows a constant direction and velocity towards the target.

When the robot encounters an obstacle, the jump is triggered either by the user command (or by perception modules) at the distance to the target defined by the jump duration T_{J} and the robot's driving velocity (distance $= \boldsymbol{v}_{\text{d}}^{\text{cmd}} T_{\text{J}}$). At the end of the plan, we switch back to the driving-stepping behavior after waiting for the configured switching delay, and let the controller settle the landing.

To achieve a jumping trajectory, we accelerate the robot's body along the z-axis for the specified duration t_{acc}, which should cause the robot to fly and to continue decelerating under the influence of gravity until the end of the jumping behavior. By adding a tilt trajectory to the body (for pitch orientation), the robot tilts back to achieve the configured tilt angle $\theta_{\text{J}}^{\text{tilt}}$ for the duration t_{tilt} before flight and tilts forward by the end of the jump. In order for the robot to tilt forward, we need to consider $t_{\text{tilt}} < t_{\text{acc}}$, which gives the controller enough time to apply appropriate reaction forces before the flight phase. Although tilting does not increase the maximum jump height, it does allow for jumping over higher obstacles since the legs will have a higher ground clearance when passing over the obstacle. The tilting trajectory is based on two consecutive cubic splines connected at the time step t_{tilt} with the pitch angle $\theta_{\text{J}}^{\text{tilt}}$ defining the tilt angle θ_k^{tilt}, tilt velocity $\dot{\theta}_k^{\text{tilt}}$ and tilt acceleration $\ddot{\theta}_k^{\text{tilt}}$ at each time step k.

The tilt angle ($\theta_{\text{J}}^{\text{tilt}}$), the duration of acceleration and tilt (t_{acc} and t_{tilt}) and the jump height gain (k_{J}) are manually configured to achieve a correct jump motion. The acceleration value during t_{acc} is computed at every control time step with:

$$a_{\text{J}} = k_{\text{J}} \left(2 \frac{h_{\text{J}}^{\text{cmd}} - \widehat{h}}{(t_{\text{acc}} - s)^2} - \frac{2\widehat{\dot{h}}}{t_{\text{acc}} - s} \right), \tag{5.16}$$

where $h_{\mathrm{J}}^{\mathrm{cmd}}$ is the commanded jump height, s is the elapsed time step from the start of the jumping behavior, and $\widehat{h}$ and $\widehat{\dot{h}}$ are the current body position and velocity along z-axis received from the state estimator.

The state trajectory $\boldsymbol{X}_{\mathrm{ref}}$ is generated using the Equation (5.13) by considering the following formulations for the initial state, and the linear and angular velocities of the robot body:

$$\dot{\boldsymbol{\theta}}_k = \dot{\boldsymbol{\theta}}_k^{\mathrm{pose}} + \begin{bmatrix} 0 & \dot{\theta}_{k+s}^{\mathrm{tilt}} & 0 \end{bmatrix}^\top, \tag{5.17}$$

$$\dot{\boldsymbol{p}}_k = \dot{\boldsymbol{p}}_k^{\mathrm{pose}} + \boldsymbol{v}_{\mathrm{d}}^{\mathrm{cmd}} + \boldsymbol{v}_k, \tag{5.18}$$

$$\boldsymbol{v}_k = \boldsymbol{v}_{k-1} + \boldsymbol{a}_k \Delta t, \quad \boldsymbol{v}_0 = \boldsymbol{0}, \tag{5.19}$$

$$\boldsymbol{x}_0 = \begin{bmatrix} \widehat{\boldsymbol{\theta}} & \widehat{\boldsymbol{p}} & \boldsymbol{0} & \boldsymbol{0} \end{bmatrix}^\top, \tag{5.20}$$

where the variables $\dot{\boldsymbol{\theta}}^{\mathrm{pose}}$ and $\dot{\boldsymbol{p}}^{\mathrm{pose}}$ are generated by iterating over Equations (5.5, 5.4), s is the elapsed time steps of the jump motion, $\dot{\theta}_{k+s}^{\mathrm{tilt}}$ is the first derivative of the tilt trajectory (from splines) at time step $k+s$, $\boldsymbol{v}_k$ is the aggregate velocity from acceleration at time step k, $\widehat{\boldsymbol{\theta}}$ and $\widehat{\boldsymbol{p}}$ are the current orientation and position from state estimator, and

$$\boldsymbol{a}_k = \begin{cases} \begin{bmatrix} 0 & 0 & a_{\mathrm{J}}^{\mathrm{cmd}} \end{bmatrix}^\top, & \text{if } k+s \leq t_{\mathrm{acc}} \\ \begin{bmatrix} 0 & 0 & -g \end{bmatrix}^\top, & \text{otherwise} \end{cases}. \tag{5.21}$$

The state trajectory for the driving-jumping behavior applies the desired body pose (body orientation and height) at the beginning of the motion. The time steps required to apply the desired body pose depend on $\dot{\boldsymbol{p}}^{\max}$ and $\dot{\boldsymbol{\theta}}^{\max}$ (Equations (5.4) and (5.5)) and the difference in the desired poses between the driving-stepping and driving-jumping controllers. To reduce the required time steps, we prefer higher values for $\dot{\boldsymbol{p}}^{\max}$ and $\dot{\boldsymbol{\theta}}^{\max}$ and similar default poses between the driving-jumping and the driving-stepping behaviors.

The state command vector is constructed as follows:

$$\boldsymbol{x}_{\mathrm{cmd}} = \begin{bmatrix} \widehat{\boldsymbol{\theta}} + \dot{\boldsymbol{\theta}}_0 \Delta t \\ \widehat{\boldsymbol{p}} + \dot{\boldsymbol{p}}_0 \Delta t \\ \dot{\boldsymbol{\theta}}_0 \\ \dot{\boldsymbol{p}}_0 \\ \begin{bmatrix} 0 & \ddot{\theta}_s^{\mathrm{tilt}} & 0 \end{bmatrix}^\top \\ \boldsymbol{a}_s \end{bmatrix}, \tag{5.22}$$

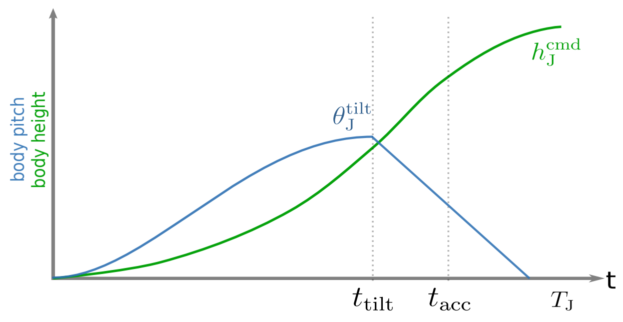


Figure 5.3: **Expected jump trajectory.** The height offset of the robot in green and the tilt trajectory in blue with a spline configured as line. The robot is expected to experience acceleration a_J in Equation (5.21) for the duration of t_acc and then gravity. the tilt angle $\theta_\mathrm{J}^\mathrm{tilt}$ is expected to be reached at t_tilt before the robot leaves the ground at t_tilt.

where $\dot{\boldsymbol{\theta}}_0$ and $\dot{\boldsymbol{p}}_0$ are given by Equations (5.11) and (5.12) at the current time step, $\boldsymbol{a}_s$ is given by Equation (5.21), and $\ddot{\theta}_s^\mathrm{tilt}$ is the second derivative of the tilting trajectory (from splines) at the time step s.

Jumping is very dynamic, which means that the expected state trajectory may differ from the optimized trajectory in the MPC. Therefore, we trust the state commands ($\boldsymbol{x}_\mathrm{cmd}$) less during the jump by setting lower penalties for floating-base body relaxations (Section 5.5).

5.1.3 Contact Trajectory

Contact trajectory is denoted by:

$$\boldsymbol{S}_\mathrm{ref} := \begin{pmatrix} \boldsymbol{s}_1^\top & \boldsymbol{s}_2^\top & \dots & \boldsymbol{s}_N^\top \end{pmatrix}$$

where $\boldsymbol{s}_i := \begin{pmatrix} s_\phi^1 & s_\phi^2 & s_\phi^3 & s_\phi^4 \end{pmatrix}$ is the vector of expected contact states s_ϕ for all legs at time step i, explained in Section 4.3.1.

Hybrid Driving-Stepping

The contact trajectory $\boldsymbol{S}_\mathrm{ref}$ for hybrid driving-stepping locomotion is provided by the gait. We define a gait as a periodic sequence of contact states for all legs. The default gaits designed for the robot are: walk, trot, pace, bound, pronk, and amble. A feature of all of these gaits (except walk) is that the period is relatively short, and the robot goes through a cycle in 350 to 500 milliseconds. Figure 5.4 shows the contact trajectory and duration of the default gaits. During walk, three legs are in contact simultaneously and one leg swings. Trot is a diagonal gait with

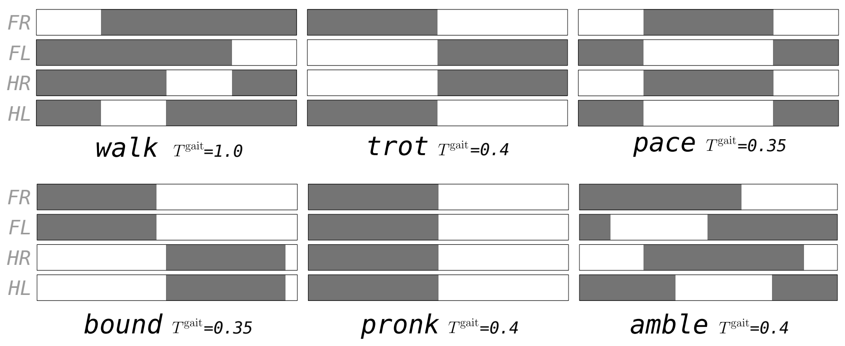

FR
FL
HR
HL
walk T^{gait}=1.0
trot T^{gait}=0.4
pace T^{gait}=0.35
bound T^{gait}=0.35
pronk T^{gait}=0.4
amble T^{gait}=0.4

Figure 5.4: **Stepping contact trajectory.** This figure shows the diagram of the contact trajectory for each default gait. T^{gait} is the configured duration of the gaits in seconds, which is dynamically changed at runtime based on commanded gait dynamic character (Figure 5.5) and gait frequency received from the driving assistant module (Section 5.3). The filled areas in each row indicate the contact for one leg. The right and left legs are shown as *FR* and *FL* for the front legs, and *HR* and *HL* for the hind legs, respectively.

two strokes, where the diagonal pairs of legs are in swing or in contact at the same time. Analogous to trot, in pace gait the legs on each side move together and in bound gait the front or back legs move together. In pronk, all four legs are in the contact or swing phase at the same time, resulting in a flight time without ground contact. Amble is similar to pace with longer contact duration, but front legs shift to swing phase earlier (by about 25% of the gait duration) than the hind legs.

We call a gait static when legs in contact wait until the swinging legs touch the ground and settle before changing to the swinging phase $\phi_{\bar{c}}$. Similarly, in a dynamic gait, the legs in contact change to the swing phase before the swinging legs touch the ground. Dynamic gaits look more natural but cause flight times without ground contact points. On the other hand they allow better swing control of the swinging legs. Static gaits, however, are better suited for imperfect ground surfaces.

In this thesis, we do not design multiple static or dynamic gaits (e.g., trot-walk and trot-run for the trot gait), but only the default gaits mentioned above and give them a dynamic character value d_g that defines the ratio of contact phase to swing phase duration of the gait (except for the walk gait). Negative and positive values of d_g increase the duration of the contact ϕ_c and swing $\phi_{\bar{c}}$ phases of all legs. Figure 5.5 shows a discrete-time diagram of the contact trajectory for the trot gait. Increasing d_g results in decreased gait controllability because the contact phase is shorter compared to the swing phase and control is only possible if there is at least one ground contact point. In contrast, a lower d_g increases the duration of gait's contact phase and results in a more robust controlled gait. This gives the operator

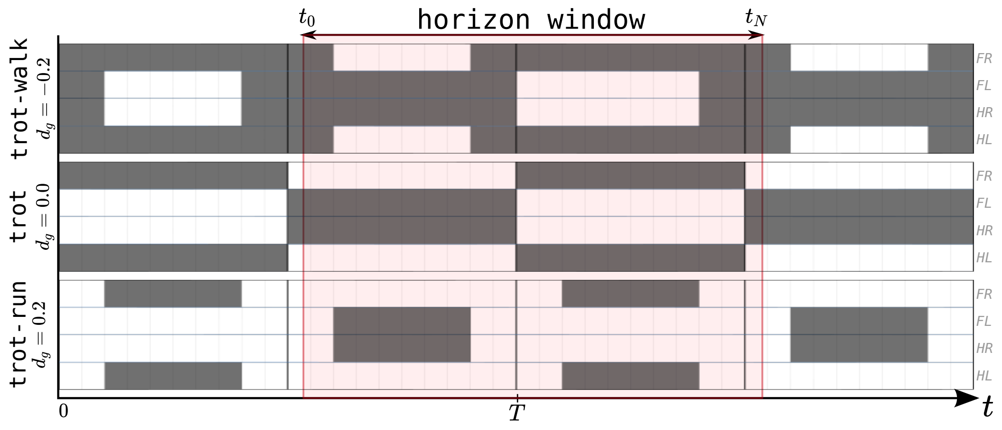


Figure 5.5: **Gait diagram.** This figure shows the diagram of the contact trajectory of two successive periods of trot gait with different dynamic characters (d_g) leading to trot-walk and trot-run. T is the gait's cycle time steps or the horizon length. The controller obtains the contact trajectory (highlighted area in the figure) at time t_0 for the size of the horizon window N.

the ability to change the value of d_g at any time during the operation of the robot based on the ground traversability.

We always include a full cycle of gait in the controller's plan, which ensures uninterrupted periodic control output. Figure 5.5 represents this with the highlighted area as the horizon window (or a full cycle of gait).

Hybrid Driving-Jumping

To construct the contact trajectory during online jumping, we define two contact durations t_c^{front} and t_c^{hind} for front and hind legs, respectively, where each leg pair in front and hind share the same contact state during the jump. Since the trajectories for the jump motion are not periodic, the horizon window becomes shorter with each elapsed time step.

For the robot to follow the tilting trajectory, the contact duration for the front legs should be less than or equal to that for the hind legs ($t_c^{\text{front}} \leq t_c^{\text{hind}}$), otherwise the controller cannot satisfy the desired trajectory. For example, in the case of $t_{\text{acc}} > t_c^{\text{front}}$, the controller generates ground reaction forces to satisfy the acceleration command a_{J}, and since the hind legs have no contact, any reaction forces acting on the front legs along the z-axis would interrupt the tilting trajectory.

Initially, t_c^{front} and t_c^{hind} are set to t_{tilt} and t_{acc}, respectively (Section 5.1.3) and further adjusted to improve the jump performance.

5.2 Footstep Planner

The footstep planner module generates foot commands and trajectories for the duration of the defined behavior. In this section, we explain how the step offsets are computed using the symmetry and centrifugal equations to formulate foot trajectories and commands, and explain the generation of swing foot trajectories.

At each control step, the target landing offset for each swing foot is determined in the relative coordinate system by extending the approach in [12] as:

$$\boldsymbol{p}_k^{\text{off}} = \boldsymbol{p}_k^{\text{symmetry}} + \boldsymbol{p}_k^{\text{centrifugal}}, \tag{5.23}$$

$$\boldsymbol{p}_k^{\text{symmetry}} = \frac{t_{\text{stance}}}{2}\left(\widehat{\boldsymbol{v}}_{\text{g}_k} + \boldsymbol{v}_{\text{sh}_k}\right) + k_1\left(\widehat{\boldsymbol{v}}_{\text{g}_k} - \boldsymbol{v}_{\text{g}_k}^{\text{cmd}}\right) + k_2\boldsymbol{v}_{\text{sh}_k}, \tag{5.24}$$

$$\boldsymbol{p}_k^{\text{centrifugal}} = \frac{h}{2g}\dot{\boldsymbol{p}}_k \times \boldsymbol{\omega}_k, \tag{5.25}$$

where t_{stance} is the duration of the contact phase, $\dot{\boldsymbol{p}}_k$ and $\boldsymbol{\omega}_k$ are the linear and angular velocities of the robot at time step k, k_1 and k_2 are the feedback gains for the errors related to the linear and angular motions, respectively. $\boldsymbol{v}_{\text{g}}^{\text{cmd}}$ is the stepping velocity command from the input modifier at time step k, and $\widehat{\boldsymbol{v}}_{\text{g}_k}$, the estimated gait velocity (excluding driving velocity), and $\boldsymbol{v}_{\text{sh}_k}$, the linear velocity induced by the current yaw rate of the robot, are defined as:

$$\widehat{\boldsymbol{v}}_{\text{g}_k} = \dot{\boldsymbol{p}}_k - \boldsymbol{v}_{\text{d}_k}^{\text{cmd}}, \tag{5.26}$$

$$\boldsymbol{v}_{\text{sh}_k} = \boldsymbol{\omega}_k \times \left(\mathbf{R}(\boldsymbol{\theta}_k)\boldsymbol{p}_{\text{sh}}\right), \tag{5.27}$$

where $\boldsymbol{v}_{\text{d}_k}^{\text{cmd}}$ is the driving velocity command at time k, $\boldsymbol{p}_{\text{sh}}$ is the location of the shoulder in the body ($\mathcal{B}$) coordinate system, and $\mathbf{R}(\boldsymbol{\theta}_k)$ is the rotation matrix from the body angles $\boldsymbol{\theta}_k$ at time k.

$\boldsymbol{p}^{\text{symmetry}}$ applies the Raibert heuristic, which reduces the leg extensions during the stepping by forcing the landing and leaving angles of the leg to be identical [78]. The symmetric motion of the legs looks more natural, and the feedback gain terms greatly increase the robustness of the control to external pushes. The applied pushes change the velocity of the robot ($\dot{\boldsymbol{p}}$), increase the error between $\widehat{\boldsymbol{v}}_{\text{g}}$ and $\boldsymbol{v}_{\text{g}}^{\text{cmd}}$, and subsequently target a more distant landing step location that counteracts the acting push. This is based on the concept of the capture point, which adjusts the placement of the foot on the ground to come to a complete stop or change velocity without disturbing its orientation [4, 64]. We extend the approach used in [12] by including the rotation of the robot in the $\boldsymbol{p}^{\text{symmetry}}$; Using Equation (5.27), we compute the linear velocity at each shoulder location ($\boldsymbol{v}_{\text{sh}}$)

resulted from robot's angular velocity ($\dot{\boldsymbol{\omega}}$), and use it in Equation (5.24) to obtain more natural turning motions and increase the robustness of the control to external yaw torques.

While driving without stepping, the $\boldsymbol{v}_{\mathrm{g}}^{\mathrm{cmd}}$ is 0, so the $\boldsymbol{p}^{\mathrm{symmetry}}$ term in Equation (5.23) results only from external pushes and yaw torques. This is ideal because in pure driving we prefer a minimum amount of leg swings, and only perform stepping when the robot encounters disturbances. Equation (5.23) is also effective for jumping behavior, as it takes into account the disturbances that affect the robot during the jump and determines the correct foot positions for landing to increase stability after landing.

Equation (5.25) finds a target foot position ($\boldsymbol{p}^{\mathrm{centrifugal}}$) where the torque is 0. These target landing positions make the robot look more natural when turning, and play an important role in keeping the robot upright when a combination of linear and turning velocities are commanded.

5.2.1 Feet Trajectory

We define the feet trajectory $\boldsymbol{P}_{\mathrm{ref}}$ with a sequence of block vectors containing all foot positions in the relative coordinate system:

$$\boldsymbol{P}_{\mathrm{ref}} := \left(\begin{bmatrix} \boldsymbol{p}_1^1 \\ \vdots \\ \boldsymbol{p}_1^4 \end{bmatrix} \begin{bmatrix} \boldsymbol{p}_2^1 \\ \vdots \\ \boldsymbol{p}_2^4 \end{bmatrix} \dots \begin{bmatrix} \boldsymbol{p}_N^1 \\ \vdots \\ \boldsymbol{p}_N^4 \end{bmatrix} \right), \tag{5.28}$$

where $\boldsymbol{p}_k^i$ is the i-th foot location at the the time step k as:

$$\boldsymbol{p}_k^i = \mathbf{R}(\boldsymbol{\theta}_k)\boldsymbol{p}_{\mathrm{sh}}^i + \widehat{\boldsymbol{p}}_k^i, \tag{5.29}$$

where $\boldsymbol{\theta}_k$ is the body orientation from the state trajectory $\boldsymbol{X}_{\mathrm{ref}}$ in Equation (5.7), $\mathbf{R}(\boldsymbol{\theta}_k)$ rotates the local shoulder position $\boldsymbol{p}_{\mathrm{sh}}^i$ into the global coordinate system, and $\widehat{\boldsymbol{p}}_k^i$ is the expected foot position relative to the global shoulder position:

$$\widehat{\boldsymbol{p}}_k^i = \begin{cases} \widehat{\boldsymbol{p}}_{k-1}^i - \boldsymbol{v}_{\mathrm{g}_k}^{\mathrm{cmd}} \Delta t, & \text{during contact phase } \phi_c \\ f_{\mathrm{swing}}\left(\frac{k - t_{\bar{c}}}{T_{\bar{c}}}, \widehat{\boldsymbol{p}}_{t_{\bar{c}}}^i, \boldsymbol{p}_k^{i,\mathrm{off}} \right), & \text{during swing phase } \phi_{\bar{c}} \end{cases} \tag{5.30}$$

where $\boldsymbol{v}_{\mathrm{g}_k}^{\mathrm{cmd}}$ is the commanded stepping velocity at time k, $t_{\bar{c}}$ and $T_{\bar{c}}$ are the start time and the duration of the swing phase, $\widehat{\boldsymbol{p}}_{t_c}^i$ and $\widehat{\boldsymbol{p}}_{t_{\bar{c}}}^i$ are the measured and expected foot positions relative to the global shoulder location at the start of the contact and swing phases, $\boldsymbol{p}_k^{i,\text{off}}$ is the commanded target foot offset given by Equation (5.23) using $\dot{\boldsymbol{p}}_k$, $\boldsymbol{\theta}_k$ and $\boldsymbol{\omega}_k$ from the state trajectory $\boldsymbol{X}_{\text{ref}}$, and the function f_{swing} (described in Section 5.2.3) defines the foot trajectory during the swing phase.

Equation (5.29) accounts for the rolling wheels condition for the foot trajectory, since the body position $\boldsymbol{p}_k$ in the state trajectory already accounts for both the driving and the stepping velocities (Equations (5.12) and (5.18)).

Equation (5.30) describes the motion of the i-th foot in the relative coordinate system ($\mathcal{R}$). During the contact phase, the foot moves only with the driving velocity $\boldsymbol{v}_{\text{d}_k}^{\text{cmd}}$ in the global frame, therefore it moves with the negative stepping velocity $-\boldsymbol{v}_{\text{g}_k}^{cmd}$ in the relative frame, which is accumulated for the duration of the contact. During the swing phase, the foot moves with the trajectory defined in the function f_{swing} towards the target landing foot location $\boldsymbol{p}_k^{i,\text{off}}$.

5.2.2 Feet Command

The feet command is a matrix containing the position, velocity, and acceleration of each foot in the global coordinate system and is used for the WBIC foot tasks to generate the joint torque commands. The i-th foot commands are:

$$\boldsymbol{p}_{\text{cmd}}^i = \boldsymbol{p}_{\text{cmd}} + \boldsymbol{p}_1^i, \tag{5.31}$$

$$\dot{\boldsymbol{p}}_{\text{cmd}}^i = \begin{cases} \boldsymbol{v}_{\text{d}}^{\text{cmd}} + \dot{\boldsymbol{p}}_{\text{assistant}}^i, & \text{during contact} \\ \dot{\boldsymbol{p}}_{\text{cmd}} + f'_{\text{swing}}\left(\frac{t-t_{\bar{c}}}{T_{\bar{c}}}, \widehat{\boldsymbol{p}}_{t_{\bar{c}}}^i, \boldsymbol{p}_t^{i,\text{off}}\right), & \text{during swing} \end{cases}, \tag{5.32}$$

$$\ddot{\boldsymbol{p}}_{\text{cmd}}^i = \begin{cases} \boldsymbol{a}_{\text{d}}^{\text{cmd}}, & \text{during contact} \\ \ddot{\boldsymbol{p}}_{\text{cmd}} + f''_{\text{swing}}\left(\frac{t-t_{\bar{c}}}{T_{\bar{c}}}, \widehat{\boldsymbol{p}}_{t_{\bar{c}}}^i, \boldsymbol{p}_t^{i,\text{off}}\right), & \text{during swing} \end{cases}, \tag{5.33}$$

where $\boldsymbol{p}_1^i$ if the first target position in the feet trajectory ($\boldsymbol{P}_{\text{ref}}$), $\dot{\boldsymbol{p}}_{\text{assistant}}^i$ is the corrective velocity from the driving assistant defined by Equation (5.37), $\boldsymbol{p}_{\text{cmd}}$, $\dot{\boldsymbol{p}}_{\text{cmd}}$ and $\ddot{\boldsymbol{p}}_{\text{cmd}}$ are the position, velocity and acceleration of the robot in the state command ($\boldsymbol{x}_{\text{cmd}}$) given by Equation (5.8), and f' and f'' give the velocity and acceleration of the swing trajectory at the current phase time t.

5.2.3 Swing Trajectory

The swing trajectory is defined by the function $f_{\text{swing}}(\phi, \boldsymbol{p}_1, \boldsymbol{p}_2)$, ϕ is the swing phase (from 0 to 1), and $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ are the start and end foot positions. f_{swing}

uses cubic splines for each axis of motion starting from $\boldsymbol{p}_1$ and ending at $\boldsymbol{p}_{\text{target}}$:

$$\boldsymbol{p}_{\text{target}} = \boldsymbol{p}_2 + \boldsymbol{p}_{\text{planner}}, \tag{5.34}$$

where $\boldsymbol{p}_{\text{planner}}$ is the target position configured for each behavior. For jumping behavior, the target locations of each swing foot are configured to achieve the highest ground clearance during the jump. For the driving-stepping behavior, a small offset on the z-axis can be configured to stop the z-swing slightly higher than the ground, reducing the velocity on impact with the ground. Figure 5.6 shows the swing trajectories [1].

The velocity and acceleration of the trajectory are calculated by the first and second derivatives of the function f_{swing} at each phase.

Swing Leg Retraction The swing leg retraction technique from [48], which is exhibited by animals in which the swing legs move backwards before touchdown, has motivated us to finish xy-swings earlier than z-swings. This gives the foot enough time to reach the target velocity of 0. In the case of early touchdown due to disturbances, this prevents the robot from pushing the leg in the direction of locomotion.

Driving-Stepping Behavior. The z-swing trajectory for the driving-stepping behavior is different. Instead of one cubic spline, two quartic splines are used, connected in phase ϕ_{apex} with height z_{apex}. ϕ_{apex}, and the parameters of the quartic splines are manually configured to accelerate the foot faster when it leaves the ground but slower when it lands on the ground, which reduces the drag of xy-swings when the foot is still in contact due to disturbances. We use $\phi_{\text{apex}} = 0.38$ for our experiments. z_{apex} is the swing height determined dynamically by the driving assistant (Section 5.3.2).

5.3 Driving Assistant

Even though our method is a hybrid locomotion with stepping, we would like to maintain the desired robot kinematics and avoid unnecessary swings of the legs when purely driving to achieve a natural motion and reduce the energy consumption of the robot due to the effort required to move the heavy wheel at the end of each leg. However, when a foot encounters a rough or high obstacle, the driving is interrupted and the foot must step over the obstacle to continue the locomotion.

[1]Live example at: `https://www.desmos.com/calculator/rffmsyigqg`

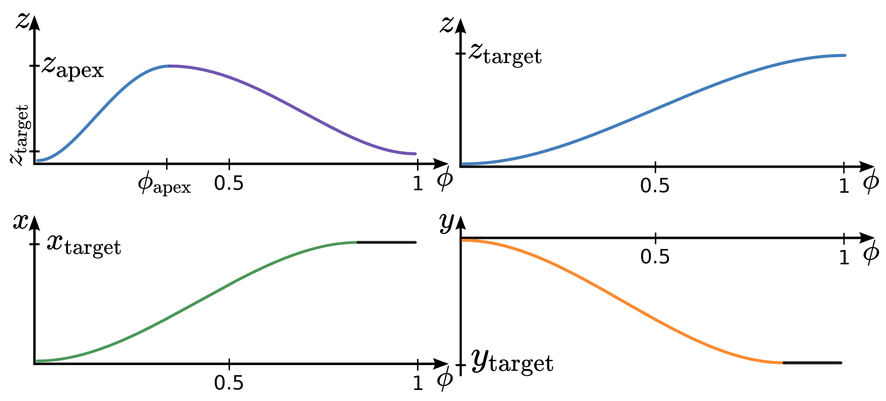


Figure 5.6: **Swing trajectory.** Top left: z-swing trajectory for hybrid driving-stepping behavior, reaching the apex of z_{apex} in phase ϕ_{apex} with the blue quartic spline and landing on the ground at z_{target} via the second quartic spline. Top right: z-swing trajectory for hybrid driving-jumping behavior with a cubic spline. Bottom: xy-swing trajectories with shorter swing duration compared to z-swing.

Despite the commanded stepping velocity, the turning commands and disturbances (i.e., external push or yaw torque) continue to force the robot to step because the non-steerable wheels limit the robot to turn only through stepping and the disturbances induce the foot command in Equation (5.23) to account for the symmetry and centrifugal step sizes.

The idea of leg utility presented in [1] inspired us to propose the driving assistant approach in this section. The leg utility value u is a measure for the usability of a leg during the contact phase. The driving assistant minimizes leg swings when the robot is in an ideal state, but induces leg swings when the leg utility is low. The utility of a leg approaches 0 immediately after hitting an obstacle.

The error between the current and the targeted foot position (during the contact phase) changes the desired robot kinematics and is measured in the body coordinate system by:

$$\boldsymbol{p}^i_{\mathrm{error}} := f\left(\mathbf{R}\left(\boldsymbol{p}^i_{\mathrm{cmd}} - \boldsymbol{p}\right) - \boldsymbol{p}^i_{\mathrm{cur}}\right), \tag{5.35}$$

where $\boldsymbol{p}$ is the current body position in the global frame, $\boldsymbol{p}^i_{\mathrm{cmd}}$ is the commanded foot location from Equation (5.31), $\mathbf{R}$ is a rotation matrix from the global to the body frame, $\boldsymbol{p}^i_{\mathrm{cur}}$ is the measured location of the foot in the body coordinate system, and f is a low-pass filter. Filtering the above errors improves the effectiveness of the driving assistant.

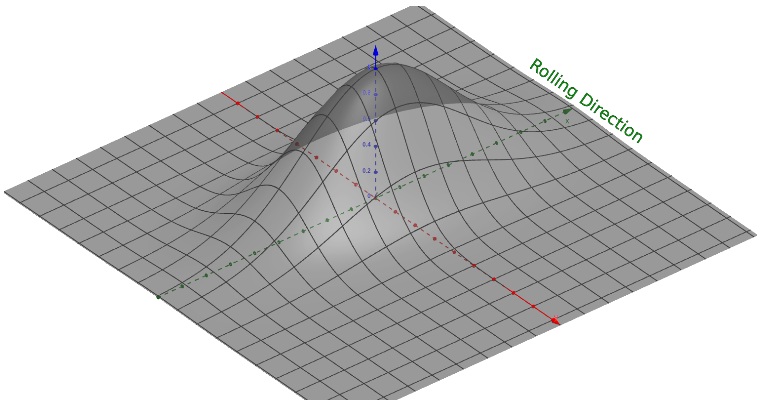


Figure 5.7: **Leg utility graph.** This graph shows the leg utility value $u(x_e, y_e)$ for the filtered foot position errors on xy-axes. Since the foot in contact is constrained to roll along x-axis without slipping on y-axis, the errors perpendicular to the rolling direction (y-errors) reduce the leg utility quicker compared to the x-errors.

The leg utility value u^i of the i-th leg is:

$$u^i(x_e, y_e) = \exp\left(\frac{-x_e^2}{2\sigma_x^2} + \frac{-y_e^2}{2\sigma_y^2}\right) \tag{5.36}$$

where x_e and y_e are the elements of the filtered foot position error ($\boldsymbol{p}^i_{\text{error}}$) along and perpendicular to the roll direction, and σ_x^2 and σ_y^2 are the configured variances defining the influence of errors on the leg utility. Figure 5.7 represents the leg utility graph[2].

We keep the desired driving kinematics of the robot and reduce the swings by changing the swing duration $\phi_{\bar{c}}$ and swing height z_{apex} of each leg depending on the utility of that leg. A higher utility of the leg decreases the swing height and increases the feet contact duration. The increased contact durations improve the performance of the controller by reducing the underactuation periods. When the leg's utility approaches 0, the leg is recovered by a larger swing duration and height to increase its utility.

This approach is not limited to a particular gait type (i.e., trot, pace, bound, etc.). Depending on the gait frequency, the reaction forces on each leg periodically approach 0 for the duration of the swing phase. Therefore, despite the low swing height values, the feet can still move on the ground towards the target step locations in two ways: by moving freely along the rolling direction with the help of the wheels or by being pulled perpendicular to the rolling direction with little

[2]Live graph at: `https://www.geogebra.org/3d/y9hqumya`

ground friction. This is especially useful for slow turning commands that do not significantly reduce the utility of the legs (resulting in low swing heights), allowing the robot to turn slowly without taking the legs off the ground. Section 1.1 highlights the difference of this approach and the related work in [1].

5.3.1 Stance Leg Correction

While the robot is driving, the feet in the contact phase may not follow the desired rolling velocity due to the disturbances or imperfect control, resulting in foot errors. We add a corrective velocity along the rolling direction to each foot during the contact phase, which reduces the foot position error and subsequently increases the utility of the corresponding leg without inducing any swings. The corrective velocity is used by Equation (5.32) and is defined for the i-th foot as follows:

$$\dot{\boldsymbol{p}}^i_{\text{assistant}} = \begin{bmatrix} k_p & 0 & 0 \end{bmatrix}^\top \odot \boldsymbol{p}^i_{\text{error}}, \tag{5.37}$$

where k_p is the configured p-gain for the rolling direction, $\odot$ is the element-wise multiplication operator, and $\boldsymbol{p}^i_{\text{error}}$ is the filtered position error defined in Equation (5.35).

5.3.2 Dynamic Variables

The swing height command z_{apex} for the i-th leg is defined proportional to the utility of the leg and the gait swing height $z^{\text{gait}}_{\text{apex}}$:

$$z^i_{\text{apex}} = z^{\text{gait}}_{\text{apex}} \left(1 - u^i\right). \tag{5.38}$$

The current gait of the robot defines phase values for contact and swing ($\phi^{i,\text{gait}}_c$ and $\phi^{i,\text{gait}}_{\bar{c}}$ for the i-th leg) based on the gait's dynamic character d_g (Figure 5.5), which are used in the following equation to calculate the dynamic values of the corresponding phases:

$$\begin{aligned} \phi^i_c &= \phi^{i,\text{gait}}_c + \phi^{i,\text{gait}}_{\bar{c}} u^i, \\ \phi^i_{\bar{c}} &= \phi^{i,\text{gait}}_{\bar{c}} (1 - u^i). \end{aligned} \tag{5.39}$$

The gait frequency is increased by reducing the duration of the gait cycle T according to the average utility of all legs, while maintaining at least the minimum cycle time T_{minimum}:

$$T = \frac{\sum u^i}{4} T^{\text{gait}}, \quad T > T_{\text{minimum}}. \tag{5.40}$$

5.4 Model Predictive Controller

The MPC (Section 3.1) considers the dynamic model of the robot over a finite time horizon N defined by each behavior to anticipate the future periods of flight or underactuation, and finds the optimal reaction forces for the duration of the prediction horizon to make the robot follow the given reference trajectory $\boldsymbol{X}_{\mathrm{ref}}$.

Our approach extends the work in [12] by removing the assumption of massless legs and incorporating rolling wheels into the modeling to obtain a hybrid driving-stepping and running quadruped. Despite using a time-varying model of the robot, our hierarchical parallel implementation allows the model predictive controller to have higher update rates (80 Hz compared to the previous 30 Hz). We include the expected $\boldsymbol{\omega}$ in the state trajectory provided by the selected behavior, which improves the tracking of the reference trajectory by allowing the controller to immediately solve for the output reaction forces that could generate such rotational velocities (see matrix $\mathbf{B}_t$ in Equation (5.48)).

5.4.1 Whole-Body Kinematic Model

We use the computed trajectories (i.e., state, contact, and foot trajectories) and use a whole-body kinematic model of the robot to generate the robot's effective mass (m), composite inertia tensor (${}_{\mathcal{B}}\mathbf{I}$), and CoM offset $\boldsymbol{p}^{\mathrm{CoM}}$.

At each time step k, the robot's orientation $\boldsymbol{\theta}_k$, foot positions $\boldsymbol{p}_k^i$, and contact state $\boldsymbol{s}_k$ are determined from the state trajectory $\boldsymbol{X}_{\mathrm{ref}}$, feet trajectory $\boldsymbol{P}_{\mathrm{ref}}$, and contact trajectory $\boldsymbol{S}_{\mathrm{ref}}$, respectively. The angles of the leg joints computed from the inverse kinematics over the foot positions with respect to the body are used in a forward kinematics to translate the inertia tensor and CoM of each limb in the robot to its parent frame of reference and determine the composite inertia and average CoM. This process is performed for all links to obtain the composite inertia and CoM of the entire robot at time step k.

The mass of the end effector (i.e., the wheel) for a leg in the swing phase changes the CoM significantly, but for a leg in the contact phase, the wheel is on the ground and its mass is not important. Therefore, we set the mass of the wheel to 0 during the contact time to obtain a realistic model (with the effective mass m) at each time step.

5.4.2 Time-Varying Dynamic Model

We extend the approach presented in [12] by using the whole-body kinematic model from the previous section, by obtaining a reduced single rigid body at each time step k of the prediction horizon and by modeling its dynamics for the predictive controller. Similar to formulation in [12], the resulting lumped mass model for each time step k follows the following dynamics:

$$m_k \ddot{\boldsymbol{p}}_k = \sum_{i=1}^{n_{c,k}} \boldsymbol{f}_k^i - \boldsymbol{g}, \tag{5.41}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}(\mathbf{I}_k \boldsymbol{\omega}) = \sum_{i=1}^{n_{c,k}} \boldsymbol{r}_k^i \times \boldsymbol{f}_k^i, \tag{5.42}$$

where $\ddot{\boldsymbol{p}}_k$, $\boldsymbol{f}_k^i$ and $\boldsymbol{c}_g$ are the vectors representing the acceleration, reaction force and gravitational acceleration of the robot in the global coordinate system, m_k and $\mathbf{I}_k$ are the effective mass and the composite inertia tensor of the whole body in the global frame, $n_{c,k}$ is the number of contacts, $\boldsymbol{\omega}$ is the angular velocity of the body in the global coordinate system, and $\boldsymbol{r}_k^i$ is the moment arm representing the position of the i-th contact point with respect to the CoM of the robot:

$$\boldsymbol{r}_k^i = \boldsymbol{p}_k^i - \boldsymbol{p}_k^{\mathrm{CoM}}. \tag{5.43}$$

Equation (5.42) shows the relationship between the rate of change of angular momentum and the reaction forces at the contact points. By assuming small off-diagonal terms for the inertia tensor, we approximate Equation (5.42) as [12]:

$$\frac{d}{dt}(\mathbf{I}_k \boldsymbol{\omega}) = \mathbf{I}_k \dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times (\mathbf{I}_k \boldsymbol{\omega}) \approx \mathbf{I}_k \dot{\boldsymbol{\omega}}. \tag{5.44}$$

Ideally, the rotational velocity of the robot is only significant for yaw during the driving-stepping behavior and for pitch during the jumping behavior, which reduces the nonlinearity of the orientation dynamics. However, we use the $\boldsymbol{\theta}_k$ in the state trajectory ($\boldsymbol{X}_{\mathrm{ref}}$) to linearize the orientation dynamics at each time step k, which defines how $\boldsymbol{\omega}$ in the global frame affects the local rotational velocity $\dot{\boldsymbol{\theta}}$ of the robot during the prediction horizon. We also rotate the inertia tensor of the body with the same known angles:

$$\dot{\boldsymbol{\theta}} \approx \tilde{\mathbf{R}}(\boldsymbol{\theta}_k)\boldsymbol{\omega}, \tag{5.45}$$

$${}_{\mathcal{G}}\mathbf{I}_k \approx \mathbf{R}(\boldsymbol{\theta}_k)\, {}_{\mathcal{B}}\mathbf{I}_k \mathbf{R}(\boldsymbol{\theta}_k)^{\top}, \tag{5.46}$$

where $\tilde{\mathbf{R}}$ is the same as $\mathbf{R}'^{-1}$ in Equation (4.3) for the known roll, pitch, and yaw body angles, $\dot{\boldsymbol{\theta}}$ and $\boldsymbol{\omega}$ are the values used by the optimizer during the minimization process, $\mathbf{R}$ is the rotation matrix from the body to the world frame generated by the known orientations, and ${}_{\mathcal{B}}\mathbf{I}_k$ is the inertia tensor at time step k generated from the whole-body model.

The continuous dynamics of the time-varying model is defined in [21] as:

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix}\boldsymbol{\theta}\\ \boldsymbol{p}\\ \boldsymbol{\omega}\\ \dot{\boldsymbol{p}}\end{bmatrix} = &\begin{bmatrix}\mathbf{0}_3 & \mathbf{0}_3 & \tilde{\mathbf{R}}(\boldsymbol{\theta}) & \mathbf{0}_3\\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{I}_3\\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3\\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3\end{bmatrix}\begin{bmatrix}\boldsymbol{\theta}\\ \boldsymbol{p}\\ \boldsymbol{\omega}\\ \dot{\boldsymbol{p}}\end{bmatrix} + \\ &\begin{bmatrix}\mathbf{0}_3 & \dots & \mathbf{0}_3\\ \mathbf{0}_3 & \dots & \mathbf{0}_3\\ {}_{\mathcal{G}}\mathbf{I}[\boldsymbol{r}^1]_\times & \dots & {}_{\mathcal{G}}\mathbf{I}[\boldsymbol{r}^4]_\times\\ \mathbf{I}_3/m & \dots & \mathbf{I}_3/m\end{bmatrix}\begin{bmatrix}\boldsymbol{f}^1\\ \vdots\\ \boldsymbol{f}^4\end{bmatrix} + \begin{bmatrix}\mathbf{0}\\ \mathbf{0}\\ \mathbf{0}\\ \boldsymbol{g}^\top\end{bmatrix},\end{aligned} \tag{5.47}$$

where the skew-symmetric matrix $[\boldsymbol{r}]_\times$ is the cross-product matrix of $\boldsymbol{r}$ that satisfies $\boldsymbol{r} \times \boldsymbol{f} = [\boldsymbol{r}]_\times \boldsymbol{f}$. We add the extra gravity term to the state and rewrite Equation (5.47) to bring the dynamics into the practical state space form:

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{x}_t = \mathbf{A}_t(\boldsymbol{\theta})\,\boldsymbol{x}_t + \mathbf{B}_t\left(\boldsymbol{r}^1, \boldsymbol{r}^2, \boldsymbol{r}^3, \boldsymbol{r}^4,\ {}_{\mathcal{G}}\mathbf{I}, m\right)\boldsymbol{u}_t, \tag{5.48}$$

where $\mathbf{A}_t \in \mathbb{R}^{13\times 13}$ and $\mathbf{B}_t \in \mathbb{R}^{13\times 12}$. The addition of gravity increases the state dimensions to 13 (from 12) and sets the value of $\mathbf{A}_t$ in the last column and row to 1, defining the effect of gravity on the rate of body velocity along the z-axis. This form depends only on the orientation of the robot $\boldsymbol{\theta}_k$, the moment arm $\boldsymbol{r}_k^i$ ($i \in \{1 \dots 4\}$) of each contact, the composite inertia tensor ${}_{\mathcal{G}}\mathbf{I}_k$, and the effective mass m_k of the model for each time step k. Since these are computed in advance the dynamics become linear time-varying, leading to a convex formulation for the model predictive control [21].

5.4.3 QP Formulation

We apply the LTV-MPC formulation in Equation (3.4) to construct the QP using the approach given in [21]. The matrix $\mathbf{D}_k$ is set to select the forces corresponding to the feet during the swing phase at time step k, in order to constrain the reaction forces for the legs during the swing phase to be 0. The matrix $\mathbf{C}_k$ is used to select the forces corresponding to the feet during the contact phase at time step k, with the following inequality constraints for each foot:

$$\begin{aligned} 0 \leq f_z \leq f_{\max}, \\ -\mu f_z \leq \pm f_x \leq \mu f_z, \\ -\mu f_z \leq \pm f_y \leq \mu f_z, \end{aligned} \tag{5.49}$$

where $f_{\max}$ is the configured maximum force the robot can exert, and μf_z terms define a square pyramid approximation of the friction cone. $\mathbf{Q}_k$ and $\mathbf{R}_k$ are the positive diagonal matrices of the weights that penalize the deviation from the reference trajectory $\boldsymbol{x}_k$ at each time step and the amount of applied control value, respectively. We use similar penalty matrices for the entire prediction horizon. The controller in this form attempts to find a sequence of control inputs that guides the system along the state trajectory, balancing tracking accuracy and control overhead while respecting the constraints [21].

By assuming a time-invariant system between each time step (for the duration of Δt), we compute the approximated matrices $\mathbf{A}_k$ and $\mathbf{B}_k$ at each time step k of the prediction horizon from the $\mathbf{A}_t(\boldsymbol{\theta})$ and $\mathbf{B}_t\left(\boldsymbol{r}^1, \boldsymbol{r}^2, \boldsymbol{r}^3, \boldsymbol{r}^4, {}_{\mathcal{G}}\mathbf{I}, m\right)$ matrices in Equation (5.48) using the corresponding values given by the state trajectory and the Equations (5.43) and (5.46). We convert these matrices to a zero-order hold discrete time model using the state transition matrix in Equation (5.48):

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{u} \end{bmatrix} = \underbrace{\begin{bmatrix} \mathbf{A}_k & \mathbf{B}_k \\ \mathbf{0} & \mathbf{0} \end{bmatrix}}_{\mathbf{M}} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{u} \end{bmatrix}. \tag{5.50}$$

The discrete time matrices $\hat{\mathbf{A}}_t$ and $\hat{\mathbf{B}}_t$ are computed by solving the following block matrix exponential:

$$\begin{bmatrix} \hat{\mathbf{A}}_k & \hat{\mathbf{B}}_k \\ \mathbf{0} & \mathbf{I} \end{bmatrix} = e^{\Delta t \mathbf{M}}, \tag{5.51}$$

where Δt is the duration between each time step and $\mathbf{M}$ is the block matrix defined in Equation (5.50). The discrete time form of the dynamics is expressed by:

$$\boldsymbol{x}_{k+1} = \hat{\mathbf{A}}_k \boldsymbol{x}_k + \hat{\mathbf{B}}_k \boldsymbol{u}_k. \tag{5.52}$$

The $\hat{\mathbf{B}}$ matrix is inaccurate by design because it is based on the time-varying model computed according to the commanded trajectories for the behavior while the resulted optimized state trajectory from the MPC differ from the commanded one. In the presence of external or terrain disturbances, the robot may deviate significantly from the reference trajectory, resulting in a more inaccurate $\hat{\mathbf{B}}$ matrix at further time steps and making the approximation in Equation (5.52) inaccurate. However, the $\hat{\mathbf{B}}$ matrix at the first time step is computed based on the current state of the robot and is always accurate. Since the MPC control loop runs periodically at high update rates (about 80 Hz), the reference trajectory is always recalculated based on the perturbed robot to compensate for the perturbation.

The QP formulation is achieved by the formulation explained in Section 3.1.3 using the discrete time matrices $\hat{\mathbf{A}}$ and $\hat{\mathbf{B}}$.

5.5 Whole-Body Controller

The task of the whole-body controller is to refine the reaction forces from the MPC and generate the joint position, velocity and torque commands ($\boldsymbol{q}$, $\dot{\boldsymbol{q}}$ and $\boldsymbol{\tau} \in \mathbb{R}^{12}$) for all feet according to the robot state and the output commands (body and foot commands) for the behavior (see Sections 5.1 and 5.2).

We use the WBIC explained in Section 3.2.1 by including the prioritized tasks for body orientation, body position, and swing foot positions. The feet during the contact phase are used to create the contact Jacobian $\mathbf{J}_c$ to initialize the null space $\mathbf{N}_0$ in Equation (3.16), which is updated after the execution of each task.

Equation (3.18) defines the acceleration for the above tasks using the state and foot commands produced for the behavior (see Sections 5.1 and 5.2), and the configured gains $\mathbf{K}_p$ and $\mathbf{K}_d$. The body orientation task uses the $\boldsymbol{\theta}$, $\dot{\boldsymbol{\theta}}$, and $\ddot{\boldsymbol{\theta}}$ commands in the $\boldsymbol{x}_{\text{cmd}}$ state command vector, the body position task uses $\boldsymbol{p}$, $\dot{\boldsymbol{p}}$ and $\ddot{\boldsymbol{p}}$ in Equation (5.8), and the swing foot position task uses $\boldsymbol{p}^i_{\text{cmd}}$, $\ddot{\boldsymbol{p}}^i_{\text{cmd}}$, and $\ddot{\boldsymbol{p}}^i_{\text{cmd}}$ foot commands in Equations (5.31), (5.32), and (5.33). For each of these tasks, the gains $\mathbf{K}_p$ and $\mathbf{K}_d$ are configured separately for each behavior. The jumping behavior sets a larger gain for body position along the z-axis for the corresponding elements of the gain matrices, while the driving-stepping behavior sets larger gains for the foot tasks.

The joint space is configured with 6 degrees for the orientation and position of the floating-base and with 12 degrees for the joint positions of all feet. The execution of the above prioritized tasks solves the joint positions $\boldsymbol{q}$, velocities $\dot{\boldsymbol{q}}$ and accelerations $\ddot{\boldsymbol{q}}$ using the Equations 3.12, 3.13 and 3.14.

The WBIC receives the reaction forces calculated by the MPC and uses the resulted floating-base accelerations mentioned above, to solve the QP problem in Equation (3.19) and find the optimized reaction forces. Finally, the feet joint torque $\boldsymbol{\tau}$ is calculated by Equation (3.20).

For each behavior, we configure the weight matrices $\mathbf{Q}_r$ and $\mathbf{Q}$ differently for the reaction forces and the floating-base relaxations in Equation (3.19). For the jumping behavior, we penalize the floating-base relaxations with a smaller $\mathbf{Q}$ than for the driving-stepping behavior. This is because the optimized trajectory of the MPC deviates more from the expected reference trajectory of the jumping behavior than for the stepping-driving behavior.

5.6 Wheel Controller

The wheel joints are controlled by a DI controller that follows the input velocities. However, the low resolution of the encoders (84 pulses per revolution) does not allow perfect velocity control, resulting in unwanted wheel motion and reducing the effectiveness of the controller. We solve this problem by converting the reaction forces from the WBC to the wheel joint torque τ and passing it to the joint controller. The wheel joints are controlled based on the linear velocity of the corresponding foot and the reaction forces received from the WBC:

$$\tau = \begin{bmatrix} r_{\text{eff}} & 0 & 0 \end{bmatrix}^\top \cdot \mathbf{R}\boldsymbol{f}^i, \tag{5.53}$$

$$\dot{q} = \begin{bmatrix} \frac{1}{r_{\text{eff}}} & 0 & 0 \end{bmatrix}^\top \cdot \mathbf{R}\dot{\boldsymbol{p}}^i_{\text{cmd}}, \tag{5.54}$$

where r_{eff} is the effective radius of the wheel from Equation (4.7), $\mathbf{R}$ is a rotation matrix to convert from the global to body reference frame, $\boldsymbol{f}^i$ is the current reaction force from WBC, and $\dot{\boldsymbol{p}}^i_{\text{cmd}}$ is the commanded foot velocity from Equation (5.32).

Since the wheel joint velocities are set according to the foot velocities, the tracking of swings along the rolling direction is greatly improved and even allows the robot to step along its x-axis with 0 swing height. If the wheel collides with a surface during the swing, it continues the x-swing to bypass the height differences without additional swing heights.

6 Evaluation

We modified the hardware of the MIT Mini Cheetah open platform described in Chapter 1 by adding wheels and shank links. The joints of this robot produce a torque of 2.1 Nm for the wheel, 27 Nm for the knee, and 18 Nm for other joints. These torques are sufficient for the robot to jump with a weight of 12.5 kg.

We transferred the open-source software version of the above robot to the ROS echo system, which meets our requirements for the tests in this chapter. We performed the tests on real hardware and simulated them in the MuJoCo multibody simulator to accurately simulate the contact dynamics and rolling friction of the additional wheels.

The robot performed the commanded pose offsets (Section 4) shown in Figure 6.1, and the applied yaw offset allowed the robot to walk in the commanded direction while facing in a different direction according to the requested yaw offset angle. However, due to the non-steerable wheels, the rolling direction always matches robot's yaw pose, resulting in errors from the commanded direction while driving, which are corrected by additional stepping.

We evaluated several contributions of this work and presented the results in several diagrams. In the following, we discuss the evaluations presented and refer to the corresponding graphs.

The tracking accuracy of the state estimator is demonstrated in Figure 6.2, which compares our contribution to the state estimator with the previous work in [11]. We set the robot in the simulator to walk forward and turn left at the same time, and compared the true value received from the simulator with the estimated values for the robot's position and velocity.

The previous method ignores the shape of the end effector assuming that its size is negligible. However, the rubber ball at the end of each leg has a radius of 2.5 cm, which is considerable compared to the 19 cm length of the shank link. The errors in leg position and velocity resulted from the assumption above, accumulates over time decreasing the estimation accuracy.

The above errors gets more pronounced with the wheels with a radius of 5 cm. Our method, accurately computes the ground contact point position and velocity considering the shape of the wheel (Section 4.2) and the effect of other joint velocities and the robot's angular velocity into the velocity of the contact point,

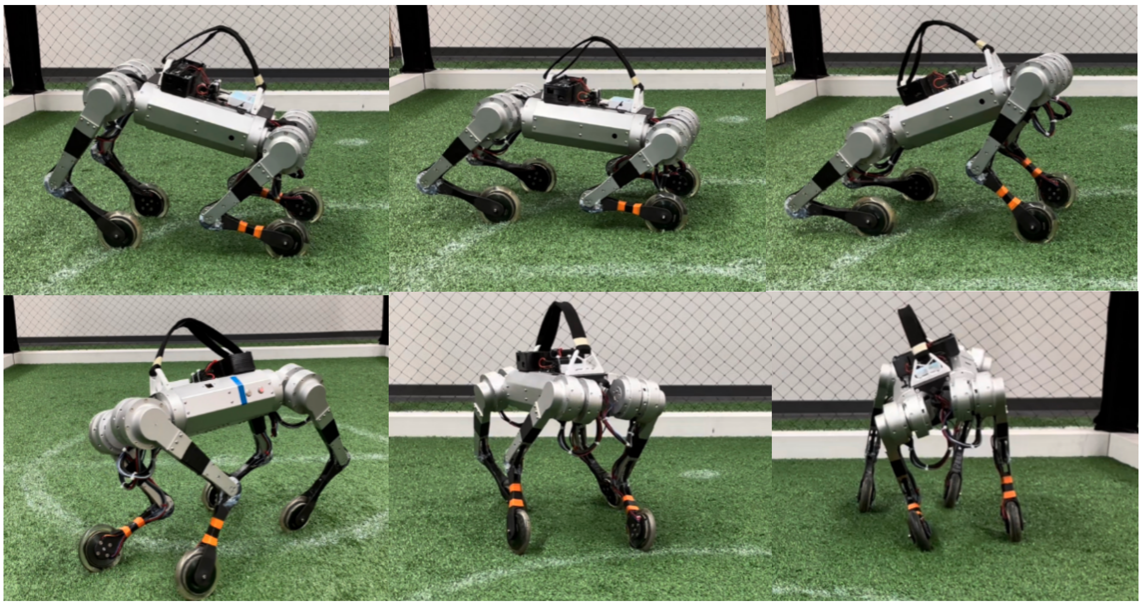


Figure 6.1: **Hardware pose offset.** This figure shows the modified hardware of Mini Cheetah robot performing different poses according to the pose offsets commanded by the user. With the yaw pose offset (in the second row images), the robot still walks and drives in the actual commanded direction before the offset.

resulting an accurate position and velocity estimations shown in Figure 6.2.

The driving assistant plays an important role in controlling the robot to demonstrate a natural driving style and reduce the energy consumption of the robot. It steers the robot along the rolling direction of the wheels and minimizes the number of swings by manipulating the gait characteristics (gait frequency, foot swing heights, and duration of foot contact).

The transition of gait characteristics is seamless and executed simultaneously with the gait commands requested by the user. When the robot receives pushes or yaw torques while driving with wheels, the leg utilities immediately decrease, so the driving assistant responds by allowing a lower gait frequency (Equation 5.40) and more swing height and duration (for each leg using the Equations 5.39 and 5.38).

Figure 6.3 demonstrates the seamless transition described above, which allows the robot's agile locomotion to take control of the perturbed robot, maintain its balance, and perform corrective steps to follow the commanded track.

Figure 6.4 shows the foot height trajectory for the front-right and rear-rightlegs during a jump motion performed in the simulation.

The driving assistant is also responsible for recovering a stuck foot. As shown in Figure 6.5 , if a wheel encounters a large obstacle while the robot drives forward,

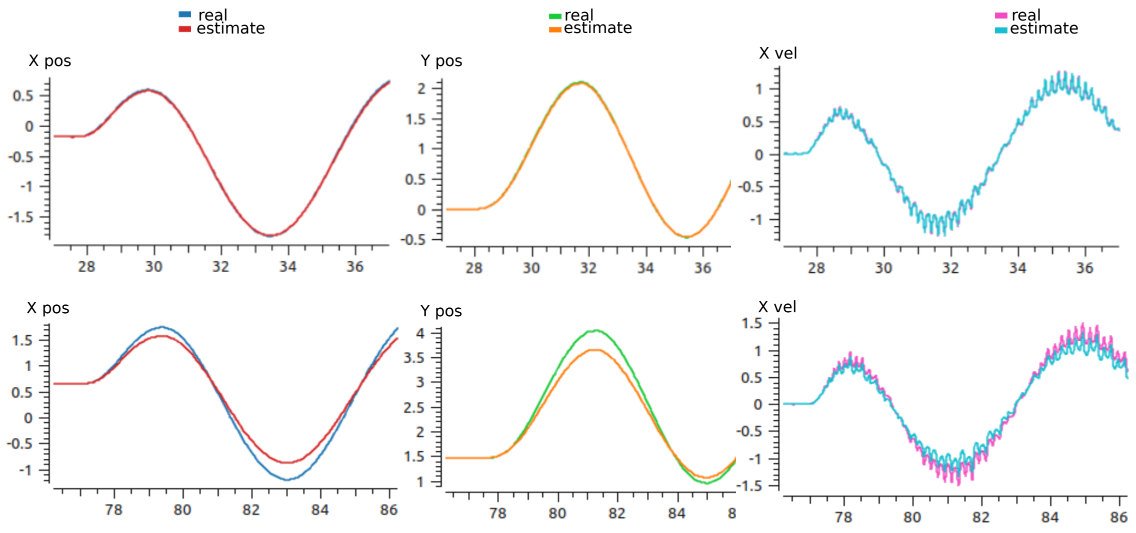


Figure 6.2: **State estimator evaluation graph.** This figure compare the accuracy of state estimation in tracking with the true values (in the simulator) for our method in the top row and the published version for the Mini Cheetah in the bottom row. In this experiment the robot walks forward turns left simultaneously. The shape of the end effector needs to be modeled. In previous work, the center of the rubber ball was assumed to be the contact point and its shape was ignored. With our method, the exact ground contact point between the end effector and the ground is calculated (Section 4.2) and the effect of the angular velocity of the body and the velocity of other joints on the velocity of the ground contact point (Section 4.3.2) is accurately computed to achieve accurate position and velocity tracking for our robot.

the error in the desired foot position increases, resulting in a lower leg utility (Equation 5.36) and subsequently a higher swing height and longer swing duration. This process, also with the help of wheel control (Section 5.6), helps the stuck foot to recover and overcome the obstacle.

We demonstrate the output of the time-varying RBD model used for the MPC in Figure 6.6. As shown, the effective mass, inertia, and CoM of the robot differ significantly depending on the gait.

The computation of the composite inertia tensor and CoM required inverse kinematics calculations over the future foot positions to first determine the joint positions and then apply the whole-body kinematics model. This process has to be repeated for the number of segments of the prediction horizon at each iteration of the control loop. However, since the computation of the composite inertia is considerably cheaper compared to the optimization of QP, the increased computational cost was not negligible.

Although we used a time-varying RBD model and updated the WBC controller with 500 Hz, we were still able to achieve an update rate of over 80 Hz for the MPC loop, which is three times higher compared to the results on the open-source Mini Cheetah platform. This is due to the hierarchical implementation of the software.

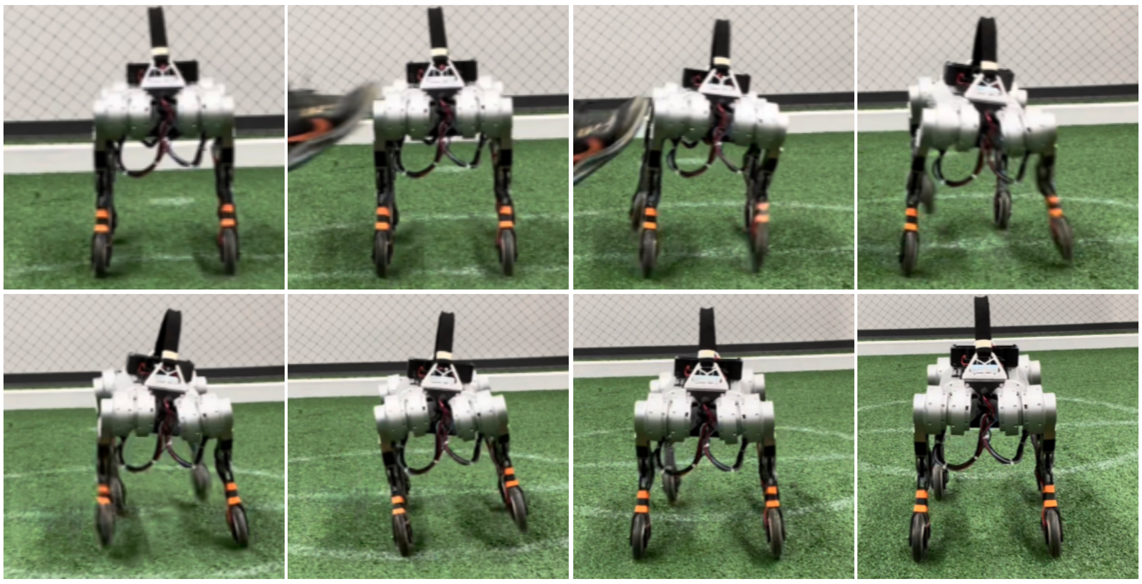


Figure 6.3: **Drive assistant output on perturbed robot.** This figure shows the robot while driving forward (the robot is held in the center of the image and the background shifts backward) under the control of the driving assistant (Section 5.3). While driving without walking commands, the controller sets a high gait frequency with the swing heights and durations near 0. When the robot is disturbed, it immediately reacts accordingly and performs the side-step shifts generated by Equation (5.23) to balance the robot and maintain its current track.

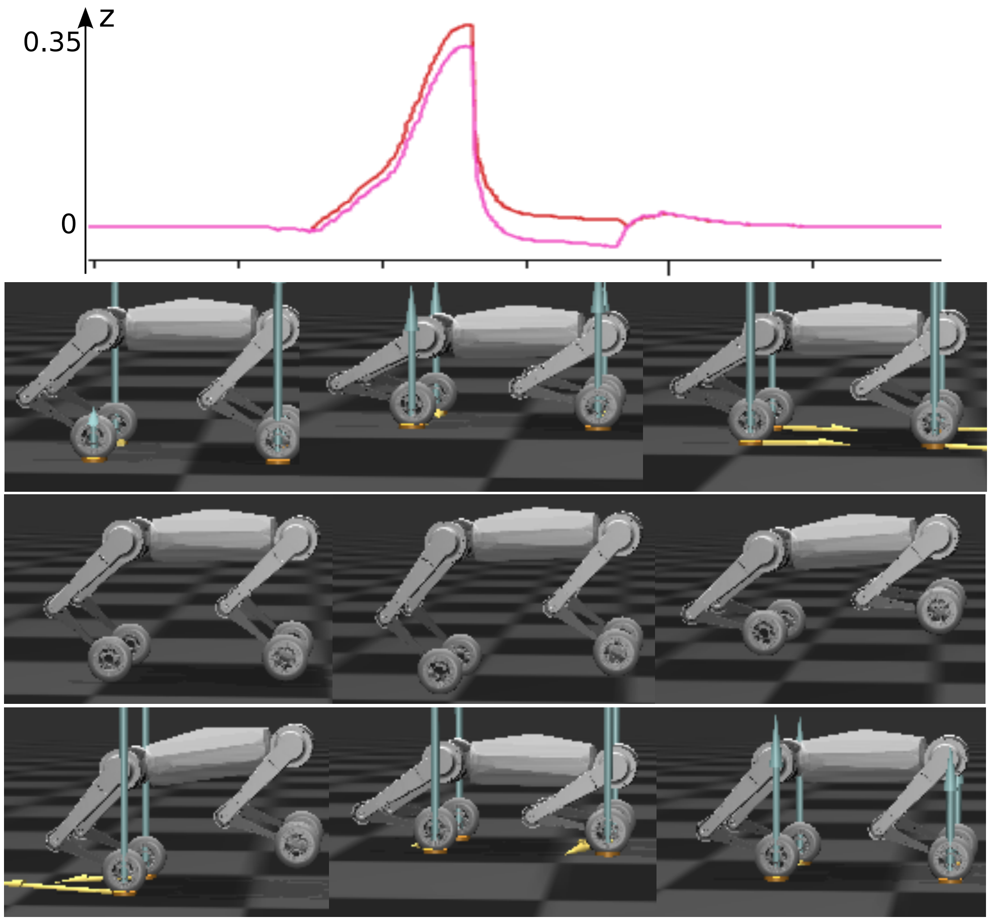


Figure 6.4: **Jumping motion.** The top graph shows the height of the front right foot in red and hind right foot in magenta. The generated reference trajectory for the jumping motion (Section 5.1.2) is passed to the MPC, which optimizes the trajectory and issues commands for the reaction force. The WBC plays an important role in keeping the roll and pitch velocity of the body close to zero.

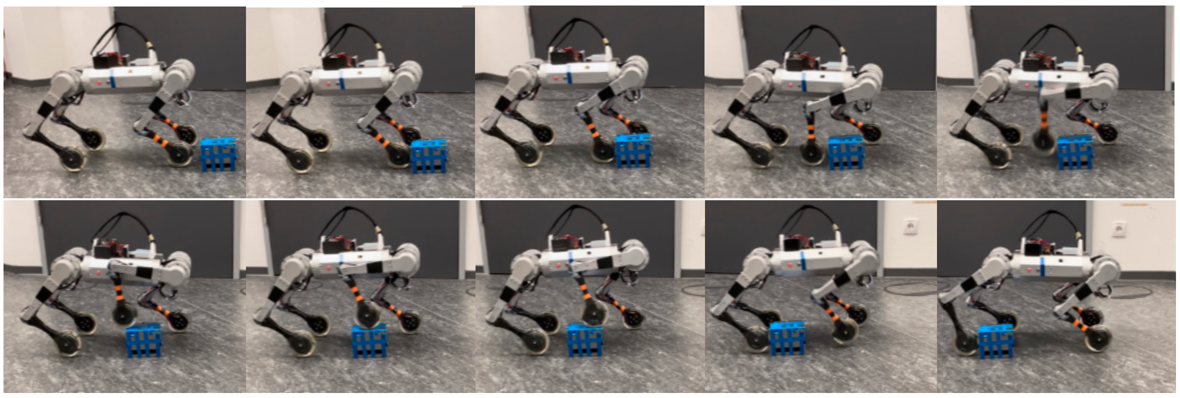


Figure 6.5: **Drive assistant with stuck wheel.** This figure shows the robot driving forward (starting at the top left and ending at the bottom right) under the control of the driving assistant (Section 5.3) with full contact duration for all feet until the right front wheel hits a large obstacle and gets stuck. The leg utility value decreases due to the error in the target foot position and therefore increases the swing height (Equation 5.38) and duration for the foot while the other feet remain in contact. The stalled wheel swings over the obstacle (second row left) to land on the ground after the obstacle.

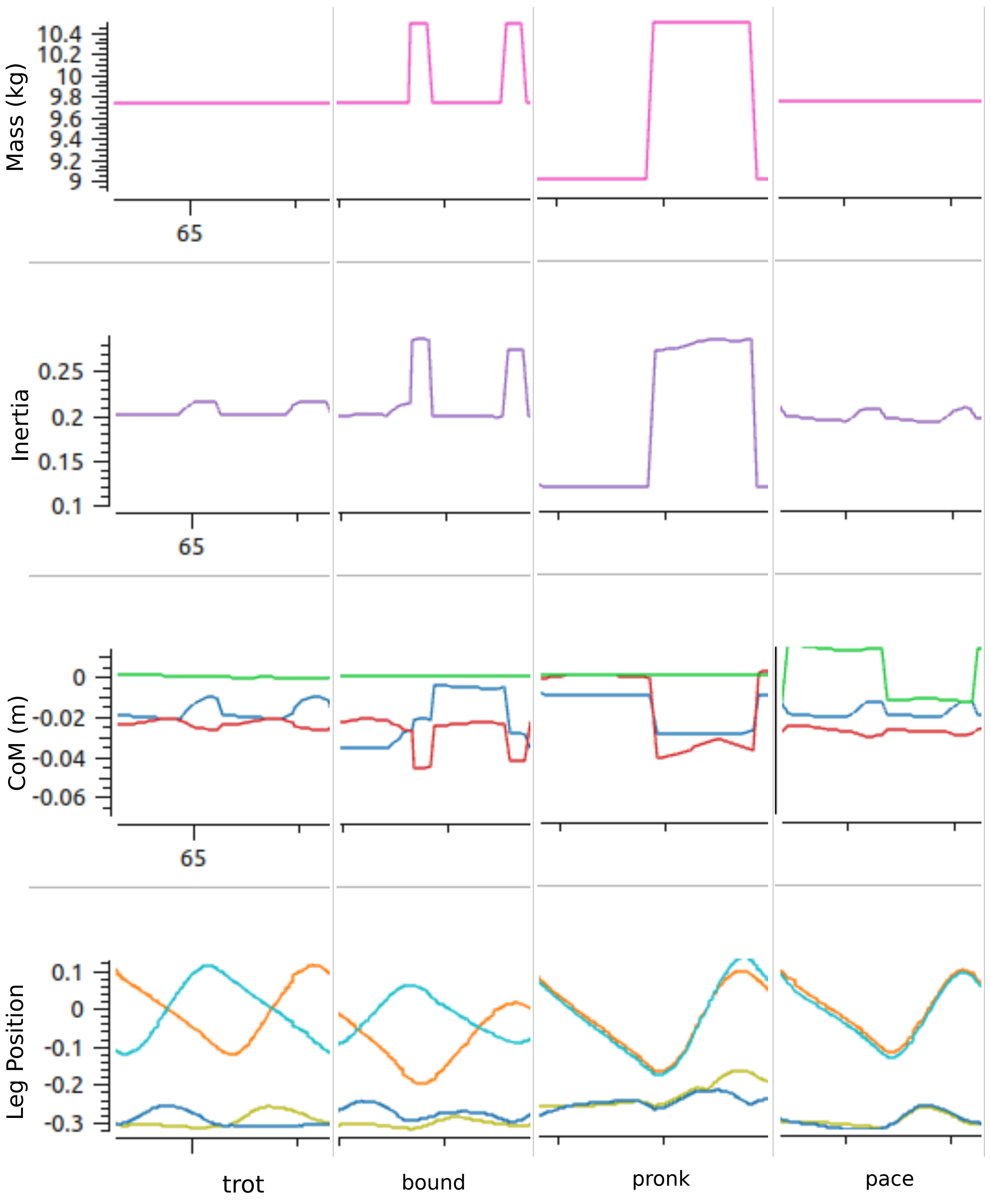


Figure 6.6: **Time-varying model output.** This figure shows the time-varying rigid-body model of the robot (Section 5.4.2) walking forward at a speed of 0.4 m/s in the trotting, bounding, pronking, and pacing gaits. The first row shows graphs for the effective mass trajectory. The plots in th second row show the trajectory of first diagonal element of the composite inertia (Ixx). The third row shows the CoM of the robot for the x, y, and z axes in blue, green, and red, respectively. The last row shows the x- and z-trajectories of the feet for the duration of one gait cycle: the position of the front and rear right foot on the x-axis in orange and blue, and the corresponding foot positions on the z-axis in olive and dark blue.

7 Conclusion

In this thesis, we integrated wheels and leg mass into the control framework of a hybrid quadruped robot to enable driving-steping and driving-jumping behaviors, and we achieved agile and long-range navigation for a hybrid quadruped robot over difficult terrain.

Although the works discussed in Chapter 2 mostly assume massless legs in their methods, in this work the wheels are heavy and large relative to the robot and could no longer be ignored. Therefore, we included the shape and size of the wheels in the kinematic model of the robot to obtain the exact ground contact point and velocity, which proved to be helpful in increasing the accuracy of the state estimation. We formulated the state estimation problem to include the velocities of the wheel joints and their contribution to the overall position of the robot in the filter state. This method allowed the simultaneous correction of the body position and the position of the foot contacts.

The presented time-varying RBD for the MPC proved to be successful in improving the accuracy of the robot in tracking the reference trajectories, especially for the non-symmetric gaits (i.e. walking). The improvement was more pronounced for more dynamic trajectories, e.g., jumping or simultaneous driving and turning. The predictive controller optimizes the given reference trajectory by trading off control effort and tracking accuracy, resulting in a different trajectory. To keep the optimization convex for our time-varying dynamics model, we needed to provide the body orientation and foot positions in advance for the duration of the prediction horizon. This requirement made it even more important to generate more accurate reference trajectories that would be close to the optimized trajectory. Therefore, we explained our methods for generating reference trajectories for locomotion and jumping with higher accuracy.

The driving assistant was able to minimize the number of swings, thus reducing the energy consumption of the robot. The robot was able to drive while keeping all legs in contact, and kept the legs in the desired positions by making small swings. It was able to free a stuck foot behind a large obstacle by stepping over it, and respond to disturbances by taking steps while driving with the wheels. The robot could drive while turning slowly with very small swings. Stepping commands requested by the user were executed simultaneously with seamless transitions of the gait properties (frequency, swing height, and swing duration).

Our approach, to the best of our knowledge, demonstrates a quadruped robot that can jump while driving with wheels for the first time.

In future work, terrain information will be used in planning the jumping motion and determining the target foot positions. The target foot positions can be tracked by the control framework with a relaxed area for the controller to shift the target foot position to optimize it for its balance. The trajectory generated for the jumping motion requires the involvement of the user to adjust the parameters. However, these will be optimized using an offline optimization method considering an accurate dynamics model of the robot.

List of Figures

Acronyms

LIPM	Linear Inverted Pendulum
ZMP	Zero Moment Point
RBD	Rigid Body Dynamics
SRBD	Single Rigid Body Dynamics
NLP	Nonlinear Programming
TO	Trajectory Optimization
QP	Quadratic Programming
LQR	Linear Quadratic Regulator
MPC	Model Predictive Control
LTV	Linear Time Variant
WBC	Whole-Body Control
WBIC	Whole-Body Impulse Control
CoM	Center of Mass
IMU	Inertial Measurement Unit
KF	Kalman Filter
CoT	Cost of Transport

Bibliography

[1] Marko Bjelonic, Ruben Grandia, Oliver Harley, Cla Galliard, Samuel Zimmermann, and Marco Hutter (2021). "Whole-body MPC and online gait sequence generation for wheeled-legged robots". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

[2] Marko Bjelonic, Prajish K. Sankar, C. Dario Bellicoso, Heike Vallery, and Marco Hutter (2020). "Rolling in the deep – hybrid locomotion for wheeled-legged robots using online trajectory optimization". In: *IEEE Robotics and Automation Letters (RA-L)*.

[3] Donghyun Kim, Steven Jens Jorgensen, Jaemin Lee, Junhyeok Ahn, Jianwen Luo, and Luis Sentis (2020). "Dynamic locomotion for passive-ankle biped robots and humanoids using whole-body locomotion control". In: *International Journal of Robotics Research (IJRR)*.

[4] Marcell Missura, Maren Bennewitz, and Sven Behnke (2020). "Capture steps: robust walking for humanoid robots". In: *International Journal of Humanoid Robotics (IJHR)*.

[5] Guillaume Bellegarda and Katie Byl (2019). "Trajectory optimization for a wheel-legged system for dynamic maneuvers that allow for wheel slip". In: *IEEE Conference on Decision and Control (CDC)*.

[6] Marko Bjelonic, Dario Bellicoso, Marco Hutter, Yvain Viragh, and Fabian Jenelten (2019). "Trajectory optimization for wheeled-legged quadrupedal robots using linearized ZMP constraints". In: *IEEE Robotics and Automation Letters (RA-L)*.

[7] Marko Bjelonic, Dario Bellicoso, Yvain Viragh, Dhionis Sako, F. Tresoldi, Fabian Jenelten, and Marco Hutter (2019). "Keep rollin'-whole-body motion control and planning for wheeled quadrupedal robots". In: *IEEE Robotics and Automation Letters (RA-L)*.

[8] Jan Carius, Rene Ranftl, Vladlen Koltun, and Marco Hutter (2019). "Trajectory optimization for legged robots with slipping motions". In: *IEEE Robotics and Automation Letters (RA-L)*.

[9] J. Gondzio and F. N. C. Sobral (2019). "Quasi-newton approaches to interior point methods for quadratic problems". In: *Computational Optimization and Applications*.

[10] Ruben Grandia, Farbod Farshidian, Alexey Dosovitskiy, Rene Ranftl, and Marco Hutter (2019). "Frequency-aware model predictive control". In: *IEEE Robotics and Automation Letters (RA-L)*.

[11] Benjamin Katz, Jared Di Carlo, and Sangbae Kim (2019). "Mini Cheetah: a platform for pushing the limits of dynamic quadruped control". In: *International Conference on Robotics and Automation (ICRA).*

[12] Donghyun Kim, Jared Carlo, Benjamin Katz, Gerardo Bledt, and Sangbae Kim (2019). "Highly dynamic quadruped locomotion via whole-body impulse control and model predictive control".

[13] Tobias Klamt, Diego Rodriguez, Lorenzo Baccelliere, Xi Chen, Domenico Chiaradia, Torben Cichon, Massimiliano Gabardi, Paolo Guria, Karl Holmquist, Malgorzata Kamedula, Hakan Karaoguz, Navvab Kashiri, Arturo Laurenzi, Christian Lenz, Daniele Leonardis, Enrico Mingo, Luca Muratore, Dmytro Pavlichenko, Francesco Porcini, and Sven Behnke (2019). "Flexible disaster response of tomorrow: final presentation and evaluation of the CENTAURO system". In: *IEEE Robotics and Automation Magazine.*

[14] Tobias Klamt et al. (2019). "Remote mobile manipulation with the Centauro robot: Full-body telepresence and autonomous operator assistance". In: *Journal of Field Robotics (JFR).*

[15] Victor Klemm, Alessandro Morra, Ciro Salzmann, Florian Tschopp, Karen Bodie, Lionel Gulich, Nicola Kung, Dominik Mannhart, Corentin Pfister, Marcus Vierneisel, Florian Weber, Robin Deuber, and Roland Siegwart (2019). "Ascento: a two-wheeled jumping robot".

[16] Quan Nguyen, Matthew Powell, Benjamin Katz, Jared Carlo, and Sangbae Kim (2019). "Optimized jumping on the MIT Cheetah 3 robot". In: *International Conference on Robotics and Automation (ICRA).*

[17] Dario Bellicoso, Marko Bjelonic, Lorenz Wellhausen, Kai Holtmann, Fabian Günther, Marco Tranzatto, Péter Fankhauser, and Marco Hutter (2018). "Advances in real-world applications for legged robots". In: *Journal of Field Robotics (JFR).*

[18] Dario Bellicoso, Fabian Jenelten, Christian Gehring, and Marco Hutter (2018). "Dynamic locomotion through online nonlinear motion optimization for quadrupedal robots". In: *IEEE Robotics and Automation Letters (RA-L).*

[19] Gerardo Bledt, Matthew Powell, Benjamin Katz, Jared Carlo, Patrick Wensing, and Sangbae Kim (2018). "MIT Cheetah 3: design and control of a robust, dynamic quadruped robot". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).*

[20] Gerardo Bledt, Patrick Wensing, Sam Ingersoll, and Sangbae Kim (2018). "Contact model fusion for event-based locomotion in unstructured terrains". In: *International Conference on Robotics and Automation (ICRA).*

[21] Jared Carlo, Patrick Wensing, Benjamin Katz, Gerardo Bledt, and Sangbae Kim (2018). "Dynamic locomotion in the MIT Cheetah 3 through convex model-predictive control". In: *IEEE Robotics and Automation Letters (RA-L).*

[22] Moritz Geilinger, Roi Poranne, Ruta Desai, Bernhard Thomaszewski, and Stelian Coros (2018). "Skaterbots: optimization-based design and motion

synthesis for robotic creatures with legs and wheels". In: *ACM Transactions on Graphics (TOG)*.

[23] Donghyun Kim, Jaemin Lee, Junhyeok Ahn, O. Campbell, Hochul Hwang, and Luis Sentis (2018). "Computationally-robust and efficient prioritized whole-body controller with contact constraints". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

[24] T. Klamt and S. Behnke (2018). "Planning hybrid driving-stepping locomotion on multiple levels of abstraction". In: *International Conference on Robotics and Automation (ICRA)*.

[25] Arturo Laurenzi, Enrico Mingo Hoffman, and Nikolaos G. Tsagarakis (2018). "Quadrupedal walking motion and footstep placement through linear model predictive control". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

[26] Alexander W. Winkler, C. Dario Bellicoso, Marco Hutter, and Jonas Buchli (2018). "Gait and trajectory optimization for legged systems through phase-based end-effector parameterization". In: *IEEE Robotics and Automation Letters (RA-L)*.

[27] Dario Bellicoso, Fabian Jenelten, Péter Fankhauser, Christian Gehring, Jemin Hwangbo, and Marco Hutter (2017). "Dynamic locomotion and whole-body control for quadrupedal robots". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

[28] Francesco Borrelli, Alberto Bemporad, and Manfred Morari (2017). "Predictive control for linear and hybrid systems". Cambridge University Press.

[29] Marco Camurri, Maurice Fallon, Stéphane Bazeille, Andreea Radulescu, Victor Barasuol, Darwin Caldwell, and Claudio Semini (2017). "Probabilistic contact estimation and impact detection for state estimation of quadruped robots". In: *IEEE Robotics and Automation Letters (RA-L)*.

[30] Michele Focchi, Andrea Prete, Ioannis Havoutis, Roy Featherstone, Darwin Caldwell, and Claudio Semini (2017). "High-slope terrain locomotion for torque-controlled quadruped robots". In: *Autonomous Robots*.

[31] Matthew P. Kelly (2017). "Transcription methods for trajectory optimization: a beginners tutorial".

[32] Michael Neunert, Markus Stäuble, Markus Giftthaler, Dario Bellicoso, Jan Carius, Christian Gehring, Marco Hutter, and Jonas Buchli (2017). "Whole-body nonlinear model predictive control through contacts for quadrupeds". In: *IEEE Robotics and Automation Letters (RA-L)*.

[33] Diego Pardo, Michael Neunert, Alexander Winkler, Ruben Grandia, and Jonas Buchli (2017). "Hybrid direct collocation and control in the constraint-consistent subspace for dynamic legged robot locomotion". In: *Robotics, Science and Systems (RSS)*.

[34] John W. Pearson and Jacek Gondzio (2017). "Fast interior point solution of quadratic programming problems arising from PDE-constrained optimization". In: *Numerische Mathematik*.

[35] Dario Bellicoso, Christian Gehring, Jemin Hwangbo, Péter Fankhauser, and Marco Hutter (2016). “Perception-less terrain adaptation through whole body control and hierarchical optimization”. In: *IEEE-RAS International Conference on Humanoid Robots (Humanoids).*

[36] Péter Fankhauser, Dario Bellicoso, Christian Gehring, Renaud Dube, Abel Gawel, and Marco Hutter (2016). “Free gait — an architecture for the versatile control of legged robots”. In: *IEEE-RAS International Conference on Humanoid Robots (Humanoids).*

[37] Anders Forsgren, Philip E. Gill, and Elizabeth Wong (2016). “Primal and dual active-set methods for convex quadratic programming”. In: *Mathematical Programming (MPC).*

[38] Christian Gehring, Stelian Coros, Marco Hutter, Dario Bellicoso, Huub Heijnen, Remo Diethelm, Michael Bloesch, Péter Fankhauser, Jemin Hwangbo, Mark A. Höpflinger, and Roland Y. Siegwart (2016). “Practice makes perfect: an optimization-based approach to controlling agile motions for a quadruped robot”. In: *IEEE Robotics and Automation Magazine (RAM).*

[39] Stefan Kohlbrecher, Alexander Stumpf, Alberto Romay, Philipp Schillinger, Oskar Von Stryk, and David Conner (2016). “A comprehensive software framework for complex locomotion and manipulation tasks applicable to different types of humanoid robots”. In: *Frontiers in Robotics and AI.*

[40] S. Feng, X. Xinjilefu, C. Atkeson, and Joohyung Kim (2015). “Optimization based controller design and implementation for the atlas robot in the DARPA robotics challenge finals”. In: *IEEE-RAS International Conference on Humanoid Robots (Humanoids).*

[41] Alexander Herzog, Nicholas Rotella, Stefan Schaal, and Ludovic Righetti (2015). “Trajectory generation for multi-contact momentum-control”. In: *IEEE-RAS International Conference on Humanoid Robots (Humanoids).*

[42] Jonas Koenemann, Andrea Del Prete, Yuval Tassa, E Todorov, Olivier Stasse, M Bennewitz, and Nicolas Mansard (2015). “Whole-body model-predictive control applied to the HRP-2 humanoid”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).*

[43] Scott Kuindersma, Robin Deits, Maurice Fallon, Andrés Valenzuela, Hongkai Dai, Frank Permenter, Twan Koolen, Pat Marion, and Russ Tedrake (2015). “Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot”. In: *Autonomous Robots.*

[44] Jeremy Ma, Max Bajracharya, Sara Susca, Larry Matthies, and Matt Malchano (2015). “Real-time pose estimation of a dynamic quadruped in GPS-denied environments for 24-hour operation”. In: *International Journal of Robotics Research (IJRR).*

[45] Hae-Won Park, Patrick Wensing, and Sangbae Kim (2015). “Online planning for autonomous running jumps over obstacles in high-speed quadrupeds”.

[46] Johannes Englsberger, Alexander Werner, Christian Ott, Bernd Henze, Maximo A. Roa, Gianluca Garofalo, Robert Burger, Alexander Beyer, Oliver

Eiberger, Korbinian Schmid, and Alin Albu-Schäffer (2014). "Overview of the torque-controlled humanoid robot TORO". In: *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*.

[47] Adrien Escande, Nicolas Mansard, and Pierre-Brice Wieber (2014). "Hierarchical quadratic programming: fast online humanoid-robot motion generation". In: *International Journal of Robotics Research (IJRR)*.

[48] J. Karssen, Matt Haberland, Martijn Wisse, and Sangbae Kim (2014). "The effects of swing-leg retraction on running performance: analysis, simulation, and experiment". In: *Robotica*.

[49] Marco Hutter, C. David Remy, Mark A. Hoepflinger, and Roland Siegwart (2013). "Efficient and versatile locomotion with highly compliant legs". In: *IEEE/ASME Transactions on Mechatronics (TMECH)*.

[50] Ludovic Righetti, Jonas Buchli, Michael Mistry, Mrinal Kalakrishnan, and Stefan Schaal (2013). "Optimal distribution of contact forces with inverse-dynamics control". In: *International Journal of Robotics Research (IJRR)*.

[51] Jasbir Arora (2012). "Introduction to optimum design". Elsevier Science, pp. 164–170.

[52] Michael Blösch, M. Hutter, M. Höpflinger, Stefan Leutenegger, Christian Gehring, C. Remy, and R. Siegwart (2012). "State estimation for legged robots - consistent fusion of leg kinematics and IMU". In: *Robotics: Science and Systems (RSS)*.

[53] Marco Hutter, Mark Hoepflinger, C Remy, and Roland Siegwart (2012). "Hybrid operational space control for compliant legged systems". In: *Robotics Science and Systems RSS*.

[54] Abhinandan Jain (2012). "Robot and multibody dynamics: analysis and algorithms".

[55] Ludovic Righetti and Stefan Schaal (2012). "Quadratic programming for inverse dynamics with optimal distribution of contact forces". In: *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*.

[56] Ludovic Righetti, Jonas Buchli, Michael Mistry, and Stefan Schaal (2011). "Inverse dynamics control of floating-base robots with external constraints: a unified view". In: *International Conference On Robotics And Automation (ICRA)*.

[57] Benjamin Stephens and Christopher Atkeson (2011). "Push recovery by stepping for humanoid robots with force controlled joints". In: *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*.

[58] Benjamin J. Stephens and Christopher G. Atkeson (2010). "Dynamic balance force control for compliant humanoid robots". In: *IEEE Robotics and Automation Letters (RA-L)*.

[59] Caroline N. Haddad (2009). "Cholesky factorization". In: *Encyclopedia of optimization*. Ed. by Christodoulos A. Floudas and Panos M. Pardalos. Boston, MA: Springer US, pp. 374–377.

[60] J. Cobano, Joaquin Estremera, and Pablo Gonzalez-de-Santos (2008). "Location of legged robots in outdoor environments". In: *Robotics and Autonomous Systems*.

[61] Marc H. Raibert, Kevin Blankespoor, Gabriel M. Nelson, and Robert Playter (2008). "Bigdog, the rough-terrain quadruped robot". In: *IFAC Proceedings Volumes (IFAC-PapersOnline)*.

[62] P.-A. Absil and André L. Tits (2007). "Newton-KKT interior-point methods for indefinite quadratic programming". In: *Computational Optimization and Applications*.

[63] Gordon Cheng, Sang-Ho Hyon, Jun Morimoto, Aleš Ude, Joshua G. Hale, Glenn Colvin, Wayco Scroggin, and Stephen C. Jacobsen (2007). "CB: a humanoid research platform for exploring neuroscience". In: *Advanced Robotics*.

[64] Jerry Pratt, John Carff, Sergey Drakunov, and Ambarish Goswami (2007). "Capture point: a step toward humanoid push recovery". In: *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*.

[65] John Rebula, Peter Neuhaus, Brian Bonnlander, Matthew Johnson, and Jerry Pratt (2007). "A controller for the LittleDog quadruped walking on rough terrain". In: *IEEE International Conference on Robotics and Automation (ICRA)*.

[66] Fung-ling Tong and Max Q.-H. Meng (2007). "Localization for legged robot with single low-resolution camera using genetic algorithm". In: *IEEE International Conference on Integration Technology (ICIT)*.

[67] Pei-Chun Lin, H. Komsuoglu, and D.E. Koditschek (2006). "Sensor data fusion for body state estimation in a hexapod robot with dynamical gaits". In: *IEEE Transactions on Robotics (T-RO)*.

[68] Surya P. N. Singh, Paul J. Csonka, and Kenneth J. Waldron (2006). "Optical flow aided motion estimation for legged locomotion". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

[69] Luis Sentis and Oussama Khatib (2005). "Synthesis of whole-body behaviors through hierarchical control of behavioral primitives". In: *International Journal of Humanoid Robotics (IJHR)*.

[70] David Abrahams and Aleksey Gurtovoy (2004). "C++ template metaprogramming: concepts, tools, and techniques from boost and beyond (c++ in depth series)". Addison-Wesley Professional.

[71] L.M. Graña Drummond and A. N. Iusem (2004). "A projected gradient method for vector optimization problems". In: *Computational Optimization and Applications*.

[72] Shuuji Kajita, Fumio Kanehiro, K. Kaneko, Kiyoshi Fujiwara, Kensuke Harada, Kazuhito Yokoi, and H. Hirukawa (2003). "Biped walking pattern generation by using preview control of zero-moment point". In: *IEEE International Conference on Robotics and Automation (ICRA)*.

[73] J. Maciejowski (2002). "Predictive control with constraints". Prentice Hall.

[74] T. Yoshikawa (2000). "Force control of robot manipulators". In: *International Conference on Robotics and Automation (ICRA)*.

[75] L.C. Andrews and Society of Photo-optical Instrumentation Engineers (1998). "Special functions of mathematics for engineers". SPIE Optical Engineering Press, p. 110.

[76] Thomas F. Coleman and Laurie A. Hulbert (1989). "A direct active set algorithm for large sparse quadratic programs with simple bounds". In: *Mathematical Programming (MPC)*.

[77] Paul H. Calamai and Jorge J. Moré (1987). "Projected gradient methods for linearly constrained problems". In: *Mathematical Programming (MPC)*.

[78] Marc H. Raibert, H. Benjamin Brown, and Michael Chepponis (1984). "Experiments in balance with a 3D one-legged hopping machine". In: *International Journal of Robotics Research (IJRR)*.

[79] D. Goldfarb and A. Idnani (1983). "A numerically stable dual method for solving strictly convex quadratic programs". In: *Mathematical Programming (MPC)*.