Rheinische
Friedrich-Wilhelms-
Universität Bonn

Institute for Computer Science
Department VI
Autonomous Intelligent Systems

Rheinische Friedrich-Wilhelms-Universität Bonn

Master Thesis

Quadrupedal Footstep Planning Using Learned Motion Models of a Black-Box Controller

Author:
Ilyass Taouil

First Examiner:
Prof. Dr. Sven Behnke

Second Examiner:
Prof. Dr. Maren Bennewitz

Supervisor:
Daniel Schleich

Date: April 12, 2022

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, 12/04/22

Place, Date

Signature

Abstract

Autonomous robotic systems are increasingly entering new domains and applications, including search and rescue, inspection, and logistics. However, for such systems to be valuable in real-world scenarios, they must be able to autonomously navigate irregular terrains efficiently and robustly. As the real world presents itself unstructured and unknown.

Legged robots in particular offer a high degree of mobility and versatility compared to their wheeled counterparts, allowing them to practically go anywhere. In fact, legged locomotion has the potential to adapt to different types of terrain by deliberately stepping on discontinuous locations. In order to achieve this, such systems must be able to reason about the local terrain patch being traversed to find suitable stepping locations, which tend to be sparser in complex terrains compared to flat ones.

In this thesis, we present a local motion planning pipeline that enables quadrupeds to traverse irregular terrains consisting of different obstacles, such as steps and gaps. We do so by extending the capability of a walking controller that is able to track high-level reference velocities, but is unable to generate on-demand trajectories and torques for target footstep locations. More precisely, we learn a set of motion models for the controller that map high-level velocity commands to CoM and footstep displacements. We then integrate the learned models with a foot costmap computed from the output of an elevation mapping algorithm, and use a variant of the A^* algorithm to plan the CoM trajectory, footstep sequences, and corresponding high-level velocity commands.

We evaluate the method in simulation in three different environments, and present an evaluation of its performance compared to a blind whole-body-control locomotion. Our planner outperforms a blind locomotion in the number of unsafe contacts, minimum feet distances kept from closest obstacles, and terrain traversal success.

Contents

1 Introduction

Autonomous systems are continuously entering new domains and applications including manufacturing, search and rescue, inspection, and logistics. Significant leaps have been made in making these systems operational, demonstrating enormous potential for the future.

On the other hand, for such systems to be deployed on all sorts of settings —be it natural, urban, or industrial— they must be able to robustly navigate difficult terrains, as the real world presents itself unstructured and unknown. Therefore, an appropriate sensory suite that allows a robotic system to perceive the environment, localize within it, and execute the planning and control frameworks in real-time is required. Thanks to the progress made in hardware design, compute capability, and control algorithms in the last years, mobile robotic systems have come a step closer to being able to be deployed on all sorts of scenarios, including ones that present themselves with hurdles, unexpected obstacles, and irregularities in the terrain [2].

Legged systems in particular offer high mobility and versatility that enables them to overcome different obstacle sizes and patterns, such as stairs and gaps, while not suffering the same limitations of their wheeled counterparts. These include the need of large wheel diameters to overcome obstacles, and a stable ground support for the continuous rolling motion. In comparison, legged locomotion can adapt to the terrain through a selective and deliberate stepping maneuver at discontinuous locations [9], at the cost of added layers of complexity compared to the powerful simplicity of wheeled systems.

Many legged systems have been developed through the years for research purposes. These include bipedal human scale robots, like *Atlas* [17], *WALK-MAN* [40], and *Valkyrie* [28]. However, bipedal locomotion is generally still behind the current performance of multi-legged platforms, in terms of speed and energy efficiency [2]. Examples of multi-legged systems such as quadrupeds that showcase better locomotion performances than most humanoids include the *HyQ2Max* [36], *ETH's ANYmal* [12], Boston Dynamics' Spot [7], and Unitree Robotics' *AlienGo* [32].

The structure of the thesis is as follows: Chapter 2 describes the problem formulation, gives a high level overview of the approach, states the assumptions made,

and briefly introduces third party components used in the proposed method. Chapter 3 presents the main paradigms available in the literature relevant to this work. Chapter 4 explains in detail the overall approach of the thesis. Chapter 5 presents the metrics used for the evaluation and the results obtained. Finally, Chapter 6 summarizes the method presented in the thesis and offers a reflection on its limitations and possible extensions of the work.

2 Problem Formulation

Legged locomotion has the ability to adapt to different types of terrain by deliberately stepping on discontinuous locations. In planar and obstacle free environments footholds can be chosen freely as long as they respect the stability and kinematic constraints of the system. On the other hand, it is only in challenging terrains that legged robots' potential can start to unveil. In such scenarios, the legged system must be able to reason about the surrounding local area in order to select the best contact locations, which are inherently sparser compared to flat terrains.

This thesis addresses the development of perceptive locomotion skills for legged robots whose *Whole-Body Controller (WBC)* is unable to generate on-demand trajectories and torques given target feet locations. The AlienGo robot is an example of such system. Its internal controller determines the required footstep locations to keep the robot balanced given a reference velocity command, but it does so in a purely reactive manner using proprioceptive sensory data to maintain stability. We therefore seek to extend the capabilities of such quadrupeds by planning high-level velocity commands that will result in safe footstep locations in order to navigate complex terrains. To do so, we use exteroceptive sensory data and a set of learned motion models for the robot's internal controller treated as a black-box.

2.1 Approach and Contributions

The proposed solution is formulated as a motion planning problem that aims at finding the *Center of Mass (CoM)* trajectory, footstep sequences, and corresponding velocity commands that allow the AlienGo robot to autonomously navigate an irregular terrain. We design the planner to work locally. Hence, we leave global trajectory planning to third party algorithms whose output can be fed to our local planner. Furthermore, we do not assume any prior information about the terrain, which has to be sensed and reconstructed in real-time for the purpose of planning.

The motion planner has several modules to it. A set of trained motion models that map velocity commands —fed to the walking controller— to footstep and

CoM displacements in order to predict the next CoM and feet poses. A modified elevation mapping algorithm —that uses a linear filter to approximate the cell's height instead of a Gaussian distribution— that reconstructs the geometrical representation of the environment as a $2.5D$ grid map, and from which a foot costmap is computed describing which terrain locations are desirable to step onto and which are not. Finally, a search algorithm that integrates the trained motion models and foot costmap is used to plan locally optimal footstep sequences.

We therefore contribute with a computationally efficient local motion planner able to plan safe footstep locations for any legged system using learned motion prediction models and real-time perception.

2.2 Components

We identify three main components which are essential for the success of our approach, but which have not been developed for the purpose of this thesis: a walking controller that keeps the robot stable during motion, an elevation mapping algorithm that builds a geometric reconstruction of the local environment, and a graph search algorithm for the planning process. These modules are briefly introduced below as they form the basis of this work.

2.2.1 Whole-Body Controller

The AlienGo robot comes with an internal *Whole-Body Controller (WBC)* that maintains the system stable while in motion. The controller is able to balance using different dynamic gaits while being able to handle different terrain conditions and slopes. On the other hand, AlienGo's controller is closed-source and is only available with the real system. Hence, the controller proposed by Raiola et al. [29] is used for simulation and development purposes given its open-source nature. In this subsection, we briefly introduce the concept of *Whole-Body Control*, and give an overview of the controller presented in [29].

Growing research interest in robotics has led robots to become increasingly proficient in performing many different tasks. Running, jumping, stair climbing, and object manipulation being some of them. However, each of these tasks is addressed individually most of the times and this imposes a limitation to the real world usage of such systems. The ability of a robot to simultaneously and successfully execute multiple actions is an ongoing area of research. Whole-Body Control has been proposed as a promising research direction that aims to define a small set of low-dimensional rules that are enough to guarantee the execution of

any single task or simultaneous tasks whenever feasible, while exploiting the full capabilities of the entire body [37].

Typical WBC formulations for legged robots design controllers and planners for the locomotion problem in a loosely coupled fashion [19, 42], where the controller typically needs the CoM trajectory to ensure the stability during locomotion. This requires a state estimation algorithm to obtain the CoM's position and linear velocity [4, 25]. However, planning trajectories for the CoM is not trivial due to drifts introduced by the state estimation and the fact that multiple constraints have to be satisfied in the CoM planning process [29].

The controller described in [29] builds on top of *Hierarchical Cartesian Impedance* control with *Quadratic Programming (QP)* optimization approaches [15, 21]. It extends on them by proposing a novel locomotion framework that avoids the specification of a task for the CoM in terms of planning and control making the framework planner-free. This has the advantage of not requiring an input from a state estimation algorithm in order to obtain the CoM's position and linear velocity. This simplifies the formulation of the locomotion framework as the robot's constraints are not further limited by the CoM trajectory planning. Moreover, the controller keeps the robot in a kinematically appealing configuration by keeping the joints far from their limits, while maintaining good leg mobility and being robust on rough terrains. In order to reach these results the controller uses priorities. This is a strategy to deal with conflicting tasks, where some are more critical than others. The strategy ensures that the high priority tasks are achieved at the expense of the lower priority ones.

2.2.2 Elevation Mapping

The ability to perceive and map an environment is key to allow mobile robotic systems to navigate a complex terrain safely and efficiently. Fankhauser et al. [11] proposed a terrain mapping algorithm which is independent from any global localization relying solely on a relative state estimate and on-board sensor measurements. In classical *world-centric* mapping approaches the map is always associated with an inertial frame, requiring an accurate global localization to obtain a globally consistent map. Therefore, to remove the need of a global localization, the authors formulated the mapping approach as *robot-centric*, where the mapped terrain is associated with the current pose of the robot instead of an inertial frame. The robot-centric map is always a local representation of the surrounding terrain, where the regions inside the robot's current field of view have the highest accuracy, while older parts of the map accumulate uncertainty.

The underlying structure of the proposed elevation mapping algorithm is a grid-

based $2.5D$ height map representation of the terrain, where each cell (x, y) in the map describes the terrain height and variance at that location. Due to the simplicity of the grid map structure this type of terrain representation allows for better scalability and an efficient data access and processing, which is essential for our planning problem. The height map reconstruction consists of three phases: a measurement update, a motion update, and a fusion process.

Measurement Update. In the measurement update phase the latest range sensor measurements, consisting of spatially transformed $3D$ points from the sensor frame to the map frame, are used to update the height map. Each measurement is mapped to its corresponding $2D$ cell in the grid and its value is updated using a Gaussian probability distribution $\tilde{p} \sim \mathcal{N}(p, \sigma_p^2)$ with mean p and variance σ_p^2. To obtain the variance σ_p^2 of the height measurement the Jacobians for the sensor measurement J_S and the sensor frame rotation J_Φ are derived and used for the variance computation. If multiple measurements with different heights fall into the same $2D$ cell, the highest elevation is kept for the update, and the other measurements are dropped.

Motion Update. During the motion update phase, which happens whenever a motion of the robot relative to the inertial frame occurs, the elevation map data needs to be updated as it is defined relative to the pose of the robot. Therefore, the mean and variance values of each cell would have to undergo an update depending on the uncertainty of the motion as well as on the estimates of the surrounding cells. However, performing such a computation for each cell is computationally expensive. Instead, the elevation map structure is extended with an additional layer where the 3×3 spatial covariance matrix for each cell is stored, and the full computation is left out until required. The diagonal values of the covariance matrix store the variance of the height estimate, as well as the approximation of the horizontal uncertainty brought by the grid discretization defined as $(d/2)^2$, where d is the length of the grid cell size.

Fusion Process. In the map fusion phase, which can be triggered on request, the elevation map data is fused to obtain a more accurate representation of the terrain. Here, the mean height estimate of each cell is computed as the weighted mean from all cells within a 2σ confidence ellipse in the $xy - plane$. The lower and upper confidence bounds of the height estimation are also inferred from the surrounding cells within the 2σ confidence ellipse.

The above method is made available by the authors as an open-source *ROS* package [1]. For the purpose of this thesis, we use a modified version of this software that approximates the height value at each cell through a linear filter instead of a Gaussian distribution. Therefore, we do not require computing the expensive Jacobians which take significant CPU resources. For a similar reason,

we also do not make use of the fused elevation map, but instead work directly with the raw height map.

2.2.3 Discrete Planning

There is a fundamental need in robotics to have a set of algorithms that convert high-level specifications of general tasks from humans into low-level descriptions of how to move. *Motion planning* and *trajectory planning* are usually used to describe these kinds of problems [22].

Planning problems involve a state space that captures all possible situations in which a robot can find itself, defined as the minimum set of parameters that can uniquely identify the robot's position in the world. All planning problems involve a sequence of decisions that must be applied over time in order to generate actions that affect the state. Ultimately, the general framework of a motion planning problem entails starting in some initial state and trying to reach a destination goal through a careful action selection.

One of the main challenges in developing practical path planners is that the robot control space and trajectories are continuous. Much of the prior work for discrete state spaces neglects this fact, ultimately producing paths that are non-smooth and that generally do not satisfy kinematic feasibility. Typical discrete state space approaches make use of the A^* [14] algorithm. A^* is a graph search algorithm that seeks to find a path between a start and a goal node having the minimum cost through an iterative and optimal path extension. It does so by expanding the node, and hence the overall path, whose cost is the minimum. The costs used to decide which node n to expand are two: $g(n)$ and $h(n)$. $g(n)$ is the cost of the path from the start node to n with minimum cost found so far by A^*, and $h(n)$ is an admissible heuristic that estimates the cost of an optimal path from n to a goal node.

The method proposed in [6] addresses the problems mentioned above by using a variant of the A^* algorithm applied to the kinematic state space of the vehicle, but with a modified state-update rule that captures the continuous state space of the system in the search nodes of A^*. In fact, for each discrete node corresponding to the center of the cell the corresponding continuous state space is associated. The method also applies a non-linear optimization that leads to a local optimum to improve the quality of the solution obtained in the state-update step. The path obtained is not guaranteed to be the minimal-cost solution due to its continuous-coordinates states merging. However, the solution is guaranteed to be smooth instead of piecewise-linear as in the case of conventional A^*.

In this thesis, the motion planner is formulated discretely using the A^* algorithm

with the modified state-update rule presented in [6] but without the non-linear optimization, in order to find the locally best state configuration for the quadruped to overcome challenging terrains.

3 Related Work

3.1 Traditional Approaches

Kolter et al. [20] presented one of the first terrain aware control pipelines. The pipeline is based on three components: a *high-level planner*, a *low-level planner*, and a *low-level controller*. The high-level planner computes a foot costmap describing the safety of a terrain location through a linear combination of features extracted from the terrain. This is then used to find a set of footsteps that minimizes the traversal cost. The low-level planner plans the trajectories for the CoM and swinging feet in order to achieve the desired footsteps computed by the high level planner. The low-level controller executes the trajectories via inverse kinematics and a PD controller. Kalakrishnan et al. [18] proposed a similar architecture that applies learning to compute the foot costmap without the need of terrain feature engineering. The approach has several modules to it. A terrain reward map generator that learns a foothold ranking function from expert demonstrations. An approximate body path planner that plans a body path through the terrain. A footstep planner that computes the next four optimal footholds using a greedy search. A pose finder that generates the body pose given the swinging feet locations. A body trajectory generator that generates a smooth body trajectory. A foot trajectory planner that generates the swinging leg motions. Finally, a PID controller to track and execute desired trajectories. Both methods assume a static gait locomotion, a pre-scanned environment, and a motion capture based state estimation. Fankhauser et al. [10] introduced a perceptive rough-terrain static gait locomotion planner where footholds are selected from a binary foothold score map. The quality metrics used to compute the costmap with respect to the terrain are the slope and curvature, height standard deviation, and uncertainty interval. The costmap is then used by the planner to find the next feasible footstep using an iterated search over the nominal foothold search space. A pose optimizer then checks the kinematic feasibility of the foot location. Finally, a collision-free trajectory is computed for the swinging leg. The approach reconstructs the terrain in real-time as a $2.5D$ grid map, and uses a proprioceptive-based state estimation. Jenelten et al. [16] presented a foothold optimizer locomotion planner that finds locally optimal footholds in a height map generated in real-time, without the constraint

of a static gait locomotion. The task is split into four independent batch-search optimizations, one for each foot. Each optimization executes a batch search over neighboring cells of the nominal foot. Each iterated cell has an assigned objective cost to it, based on defined objectives and constraints criteria. The optimal footholds are the ones with the smallest cost.

3.2 End-to-End Approaches

Gangapurwala et al. [13] presented a model-based and data-driven approach to quadrupedal planning and control over uneven terrains. A *Reinforcement Learning (RL)* based footstep planning policy is trained that maps the robot state and terrain information to $2D$ feet location. Additionally, a domain adaptive tracker policy and recovery control policy are incorporated with the footstep planning. The former corrects aggregate errors in the joint trajectories execution to reach the planned footholds with corrective torques. The latter stabilizes the system in case of perturbations or unexpected events with desired joint positions. To account for terrain information in the footstep planning, exteroceptive feedback in the form of a local elevation map is encoded in the robot state along with the robot's proprioceptive sensory information. The encoding process is sped up by using a pre-trained convolutional neural network. Miki et al. [24] presented an encoder based solution that integrates both exteroceptive and proprioceptive perception. An encoder is trained in simulation end-to-end in three stages. First, a teacher policy is trained with RL to follow a given target velocity over randomly generated uneven terrains with random disturbances, while having privileged access to all information. Then, a student policy is trained to mimic the teacher policy's actions without access to privileged information. Thereby building a belief state about the unobserved information using a recurrent encoder. Height samples extracted from the elevation map can then be combined with proprioceptive data and fed to the network to obtain actuation commands. Tsounis et al. [41] presented a set of trained neural network policies that combine model-based motion planning and RL. A two-level hierarchical approach comprising a *Gait Planner (GP)* and a *Gait Controller (GC)* is used. Both components are RL based policies trained end-to-end. The GP uses proprioceptive and exteroceptive data to generate a finite sequence of support phases. The GC is a policy serving as a hybrid motion planner and controller, that uses the computed support phases in order to output joint position references and respective joint torques. Peng et al. [27] presented a similar two-level hierarchical control framework for bipedal locomotion. First, low-level controllers operating at fine timescale are trained in order to achieve robust

walking patterns that satisfy stepping-target and style. Then, high-level controllers are learned that plan at the timescale of steps by invoking desired footstep targets from the low-level controllers. Terrain map information are directly used by both controllers for the decision making.

3.3 Hybrid Approaches

The closest approaches to ours are the ones from Chestnutt et al. [5], and Schmitz et al. [35]. In [5], the authors presented a footstep planner for the humanoid robot *ASIMO* to navigate towards a goal while avoiding obstacles and unsafe stepping locations. The footstep choices are based on the current state of the robot, and are computed using the A^* search algorithm in order to find the optimal sequence of footstep locations up to a pre-defined planning horizon. Given the robot's level of control —that does not allow to specify desired foot locations—, the authors develop a motion model that is able to map the robot's state to the $2D$ displacement of the swinging leg using recorded data from a motion capture system. The environment is represented as a $2D$ grid where each entry describes if the location is a safe stepping space or not. The search of optimal actions is performed with the A^* algorithm using three heuristic costs. A footstep validity cost that accounts for the desirability of the step, a step cost that describes the effort of the swinging motion, and the estimated running cost which is the estimated remaining cost to the goal. In [35], the authors presented a method that generates ball approach trajectories for humanoids using footstep planning with the A^* algorithm. A set of pre-computed footstep sequences are used to learn a fast online policy in order to meet the requirements of resource constrained devices. To achieve this, the authors use a footstep prediction model [34] to map a gait velocity vector to a step location for the swinging foot in the Cartesian space. Using a small discrete set of actions, a near optimal trajectory to the target states is computed using a simple Euclidean distance heuristic. The computed trajectories are then used to train the online policy that outputs the desired velocity command to reach the next footstep location. By a successive recall of the policy the same footstep sequence that was initially planned with A^* will be reproduced. The footstep prediction model used in the search estimates the Cartesian location and orientation of the next footstep given the same inputs that control the walking controller of the robot. The footstep prediction model is learned using recorded data from a motion capture system, obtained by sending random walking speeds and direction to the robot's walking controller. The recorded data is then post-processed to extract footstep data used for model fitting, where a footstep is defined as a

step where the feet have approximately the same height and velocity. Then, three independent one-dimensional functions are fitted in order to predict the foot pose in the $xy - plane$ as well as orientation. Linear regression is used to fit all three functions due to the strong linear character in the recorded data. Magana et al. [23] presented a real-time dynamic foothold adaptation strategy based on visual feedback. The proposed method adjusts the landing position of the feet in a fully reactive manner, through a corrective trajectory along the swing phase, generated by a reactive controller framework, using the input of a self-supervised foothold classifier convolutional neural network.

4 Method

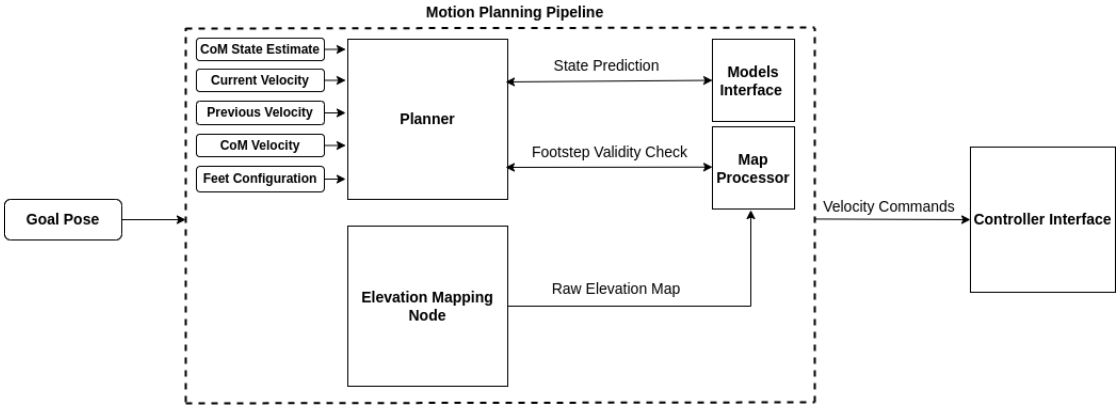


Figure 4.1: Overview of the method.

In this chapter, the motion planning pipeline, whose overview is shown in Fig. 4.1, is explained in its details. At a high level the pipeline consists of five blocks. A *models interface* that uses the learned motion models to predict the robot's CoM and swinging feet displacements. An *elevation mapping* module that reconstructs the local terrain traversed as a $2.5D$ grid map in real-time. A *map processor* that receives as input the generated height map of the terrain and processes it to compute a foot costmap describing safe and unsafe stepping locations. A *planner* module that implements the A^* variant algorithm and integrates it with the models interface and map processor to plan local footstep sequences and corresponding high-level velocities. Finally, a *controller interface* that receives a high-level velocity command and generates respective joint trajectories and torques.

We develop and integrate the various modules using the *Robot Operating System (ROS)* [31] middleware and its communication paradigm. The elevation mapping runs on its own ROS node and publishes the elevation map at a frequency of $30Hz$, while the planner, models interface, and map processor are executed on a different node. However, the map processor runs on a separate thread and makes available the required terrain information via caching.

4.1 Prediction Models

The AlienGo's internal controller uses proprioceptive sensory data in order to maintain balance. The controller is able to generate joint trajectories and torques for a given high-level velocity command, but is unable to do the same for target feet locations. This, ultimately hinders the capability of the system to navigate complex terrains.

Therefore, we propose to learn a set of motion models that can predict the robot's CoM and feet displacements given the current state of the robot and a velocity command. These models can then be used to plan a sequence of footsteps and its respective high-level commands that allow the quadruped to safely navigate irregular terrains.

The motion models are necessarily mapping functions that map an input vector to displacement quantities for the CoM and moving feet. More precisely, such mappings are fundamentally regression tasks as we seek to predict continuous values representing the displacements. As such, we define the models as a set of regression problems and use machine learning to learn the mappings.

There are different approaches that can be used to learn our models. One possibility is to build a mapping table, where for each velocity command we assign the mean displacement that the CoM and swinging feet would undergo. Although such an approach might work for simple terrains, it comes with a significant disadvantage as ambiguities cannot be easily modeled. For example, given the same input vector it could be that the respective displacements vary depending on the velocity at which the robot is moving at or other correlated variables. We could extend the table to include all possible mapping combinations, but it is intractable to do so. A more general approach could be to use a *Neural Network (NN)* [3]. This is a biologically inspired method that can learn arbitrary high dimensional functions in order to fit the data using gradient descent optimization techniques. However, for the purpose of our planning problem using a NN is not the best option, as it cannot be easily interpreted due to the black-box nature of the algorithm. Making it difficult to understand which predictors are statistically more significant. Moreover, due to the strong linear character between the velocity commands and displacement quantities, a better approach is found in *Regression Analysis (RA)* [39] methods. RA is a statistical process that estimates the relationship between the dependent variable or response, and the independent variables or predictors. One of the most common form of RA is *Multivariate Linear Regression (MLR)* [38], where a complex linear combination in the form of a first-degree polynomial that fits the data is computed. The strong linear character of the data, powerful simplicity of the MLR method, ease of integration and quick computation, and the

ability to deal with prediction ambiguities through complex linear combinations, makes MLR the best option to train the motion models.

We define the CoM models to predict the absolute displacement of the CoM with respect to the world frame. Similarly, we also define the footstep models to predict the absolute displacements of the swinging feet with respect to the world frame. However, we also evaluate the possibility of predicting the relative pose of the moving feet with respect to the CoM. An empirical evaluation for all models and a comparison between the two possible prediction choices for the footstep models is given in Chapter 5.

The input used for the CoM and swinging feet predictions is the same and consists of: the previous velocity command, the current velocity command, the CoM velocity, and the relative feet poses with respect to the CoM. The velocity commands are a 3-dimensional vector $V_{cmd} = [V_x, V_y, V_\Theta]$ whose components are: a linear velocity in x or forward speed, a linear velocity in y or lateral speed, and an angular velocity around the z axis or rotational speed. The CoM velocity is a 6-dimensional vector describing the linear and angular velocities of the robot. The relative feet poses are an 8-dimensional vector consisting of the $2D$ position in the $xy - plane$ for each foot with respect to the CoM.

Finally, in order to train the motion models and learn the underlying behavior of the robot, representative data samples need to be collected and processed for the training. In the coming sections we present the overall pipeline for the training process. This consists of the following steps: *data acquisition*, *footstep extraction*, *dataset generation*, and *training*.

4.1.1 Data Acquisition

The data acquisition step consists of gathering the quadruped's high-level state information, while it executes various motion commands, so that we can generate the dataset to train the motion models. The data collection happens on a simulated flat terrain, as shown in Fig. 4.2, with the assumption that the controller behavior does not drastically change on steeper terrains. Furthermore, we acquire data for a set footstep height of 6.5cm.

To acquire the data we use the robot's controller interface that receives a 3-dimensional velocity command as input. Based on this interface, we define a strategy to gather the data such that they reflect all possible behaviors of interest to the planning process. This entails exhausting all possible input permutations that can be commanded to the robot in terms of movement types and velocity magnitudes. In particular, the movements considered are: forward movements, lateral movements, and rotational movements. Backward movements are not considered

as the robot does not have back sensors that allow a safe backward motion. The velocity magnitudes, representing the percentage of the maximum velocity to use, range from 0.1 to 1.0 for forward movements, and from 0.1 to 0.9 for non-forward movements as the controller looses stability at 1.0. We use a step size of 0.1 to create the set of velocities.

For the robot to efficiently overcome irregular terrains that include steps and gaps, it has to continuously move between acceleration, deceleration, and constant speed patterns. Acceleration and deceleration phases are necessary to overcome unsafe stepping locations. For example, in the case of a large gap the robot would have to decelerate while approaching the hole, only to then accelerate taking a big enough step to overcome the gap. Constant speed patterns are also necessary in case of continuous and obstacle-free terrain patches, where the robot can move as fast as possible commanding constant velocities. Therefore, the motion models must be able to predict the displacements for all three patterns. To do so, we acquire data that include continuous, accelerating, and decelerating behaviors for all movements of interest and velocity magnitudes. More precisely, for each movement and each velocity magnitude a continuous velocity command is fed to the controller for 2 minutes while high-level state information are gathered. Similarly, for each movement and starting velocity magnitude, we command an acceleration to each higher velocity magnitude for 1.5 seconds, followed by a deceleration to the starting velocity magnitude of 1.5 seconds. This alternation continues for 2 minutes and is repeated for each allowed velocity magnitude.

The commands are sent to the controller at a frequency of $1KHz$. Every logged sample consists of the velocity command sent, the state estimation of robot's CoM, the CoM velocity vector, and the relative $2D$ pose of each foot with respect to the CoM.

4.1.2 Footstep Extraction

The acquired high-level state information is a high frequency time series of data samples. However, what we are interested in for the training of the models is footsteps. These are transitions between feet configurations where one of the diagonally opposite foot-pair swings in order to move the robot in the commanded direction. Therefore, we define a footstep as a feet configuration where each foot is in contact with the ground at the end of the swinging transition.

In the footstep extraction step, we seek to find all instances belonging to the data samples acquired that match the given footstep definition. We can identify such instances using force or height information of each foot, as well as a combination of these. More precisely, when the height difference between the feet is minimal

Figure 4.2: Simulated terrain in Gazebo [30] where the data is acquired.

or the sensed force at each foot is significant, the end of a footstep is identified.

Using a combination of height and force measurements would be necessary if the extraction process is carried on data acquired on uneven terrains. As only using a height difference constraint would not be sufficient to identify a footstep because the feet can be in contact at different heights. Similarly, only using force measurements can cause a lot of misdetections.

However, we acquire the data on a flat terrain, assuming that the walking controller does not drastically change its behavior in terms of CoM and swinging feet displacements when navigating uneven terrains. Thus, we only use height measurements for the extraction process. Given a sequence S of collected data, a sample $s \in S$ is a footstep if the difference between the absolute height differences of the front-left and rear-left as well as the front-right and rear-right is below a threshold ϵ:

$$(|h_{fl} - h_{fr}| - |h_{rl} - h_{rr}|) < \epsilon,$$

where h_{fl}, h_{fr}, h_{rl}, and h_{rr} are respectively the *front-left foot height*, *front-right foot height*, *rear-left foot height*, and *rear-right foot height*. We use a height difference threshold ϵ of 0.002m.

Furthermore, due to irregularities in the kinematic configuration adopted by the quadruped in simulation, the above criterion is not enough to detect the end of a footstep. This is because in order to correct the irregular posture the controller commands corrective trajectories that might break the cyclic alternation of the

swinging pair or that does not allow the moving feet to reach the set footstep height. If enough of these instances are present in the dataset used for the training, it could hinder the models' accuracy. Therefore, while performing the footstep extraction, additional criteria are imposed. Namely, whenever the minimal height difference criterion is met, we also check that the swinging pair is diagonally opposite and that the maximum footstep height reached is in line with the set footstep height in the controller. These measures help to filter out footsteps that are consequences of irregular behaviors. Moreover, as we use an inequality constraint for the height criterion, there exist multiple instances that meet it. Hence, a minimum time has to pass before another footstep can be extracted. This time is defined as $1/f_{sw}$ where f_{sw} is the *swing frequency*.

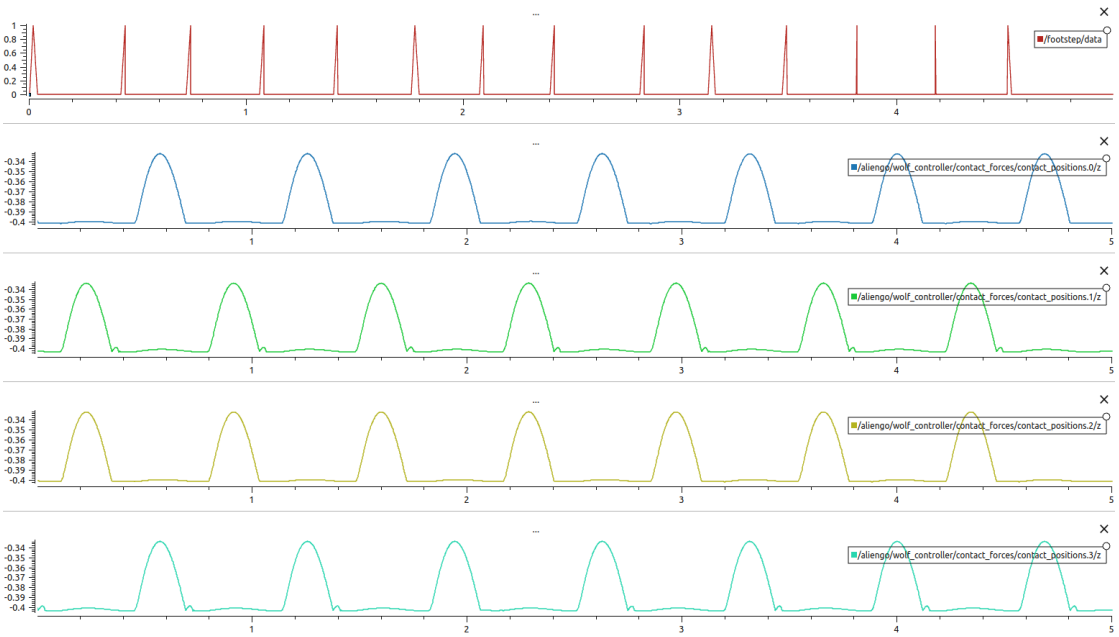


Figure 4.3: Examples of extracted footsteps on a time series of foot heights plotted using PlotJuggler [8]. The top row represents detected footsteps with a boolean value of 1. The remaining rows are respectively the heights of the front-left foot, rear-left foot, front-right foot, and rear-right foot with respect to the CoM.

4.1.3 Dataset Generation

Once the footsteps are extracted, an additional step is required before being able to train the models. This entails generating the input dataset X and the label dataset Y used for training and testing the models.

The dataset X is the same for all models, and contains all required information for the prediction. Given any two consecutive footsteps f_i and f_{i+1}, the set of predictors fed to the models at time i to predict the displacements brought by the footstep transition at time $i+1$ is: the velocity command at time i, the velocity command at time $i+1$, the relative $2D$ positions of the feet with respect to the CoM at time i, and the CoM velocity state at time i. The two velocity commands allow the model to learn to differentiate between continuous and accelerating patterns. The relative $2D$ positions of the feet are fundamental to the predictions, as a different step size might be taken based on the feet configuration. The CoM velocity improves the accuracy of the models in situations of ambiguity. Such as when the same two consecutive velocity commands and feet positions would yield different displacements based on the current speed of the robot. The input dataset X is obtained by aggregating the input vector described above for each consecutive footstep pair identified in the extraction process.

The label dataset Y contains the CoM and feet displacements for each pair of sequential footsteps. Given any two consecutive footsteps f_i and f_{i+1}, the CoM displacement vector $\Delta_{CoM} = [\Delta X_{CoM}, \Delta Y_{CoM}, \Delta\Theta_{CoM}]$ is given by the difference between $odom_{i+1}$ and $odom_i$. Where $odom_{i+1}$ and $odom_i$ are the odometry estimates of the robot's CoM at time $i+1$ and i, defined as $3D$ vectors describing the (x, y) position and Θ orientation of the CoM with respect to the world frame. Similarly, the displacement for each swinging foot $\Delta F_{swing} = [\Delta X_{fs}, \Delta Y_{fs}]$ is computed by taking the difference between the foot pose at time $i+1$ and time i. A foot pose is defined as the relative $2D$ vector describing the (x, y) coordinate of the foot with respect to the CoM. However, as we are interested in the absolute displacement of the swinging feet, and not the relative one, the CoM movement has to be accounted for. Therefore, the respective CoM displacement is added to the displacements of the swinging feet.

When computing the displacements for both the CoM and moving feet, the CoM rotation with respect to the world frame has to be considered. As in the case of significant rotation around the z-axis of the robot, the displacement vectors for all non-rotational motions would have a significant component in both the x and y direction, which is wrong. Hence, before computing the displacements, the odometry estimates used to compute the CoM movement are rotated by applying the inverse of the rotation matrix R^{OI}, which represents the rotation of the world frame I relative to the robot reference frame O. There is no need to apply the same rotation to the feet poses as these are always relative to the CoM and not the world frame.

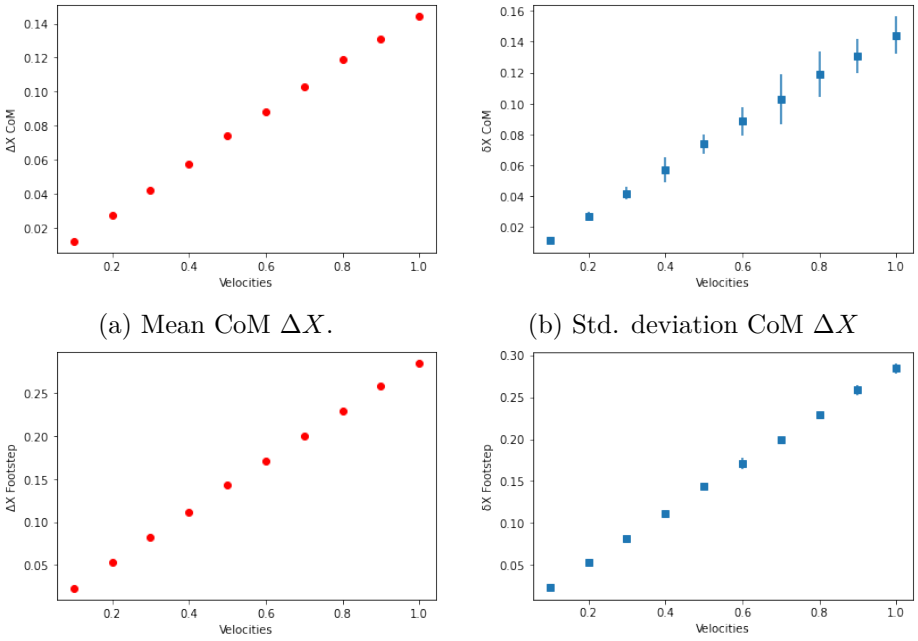
4.1.4 Training

The aim of the motion models is to accurately predict the displacements of the CoM and swinging feet given a set of predictors. Due to behavioral differences for the CoM depending on which diagonal foot-pair is swinging, two separate models are identified, one for each foot-pair. Similarly, we also identify four different models for the feet, one per foot, in order to account for intrinsic differences in their behaviors.

For each CoM model, three separate models are trained for the x, y, and Θ displacements. For each foot, two separate models are trained for the x and y displacements. A total of 14 models are trained.

Given a strong linear relationship between velocity commands and the mean displacements as well as low standard deviations for the mean values, for all movements of interest and velocity magnitudes, a *Multivariate Linear Regression* method is used to fit the data. This consists of a complex linear combination of the predictors in the form of a first-degree polynomial. Fig. 4.4, 4.5, 4.6, 4.7, and 4.8 show the linear relationships between the velocity magnitudes and mean displacements for the CoM and feet, for all movements of interest. However, we only show the displacements for the most significant axis. For example, for the forward movement we only show the mean and standard deviation for the x axis, as this is the direction with a meaningful displacement.

We use 80% of the data to fit the linear regression models. The remaining 20% is used to evaluate the accuracy of the learned models. The training and testing dataset are further split into two subsets, one for each swinging foot-pair. Depending on which model is being trained the corresponding subset is used for the fitting. A total of approximately 19000 data points are used for training.



(a) Mean CoM ΔX.

(b) Std. deviation CoM ΔX

(c) Mean footstep ΔX.

(d) Std. deviation footstep ΔX

Figure 4.4: CoM and footstep mean displacement and standard deviation for a forward motion.

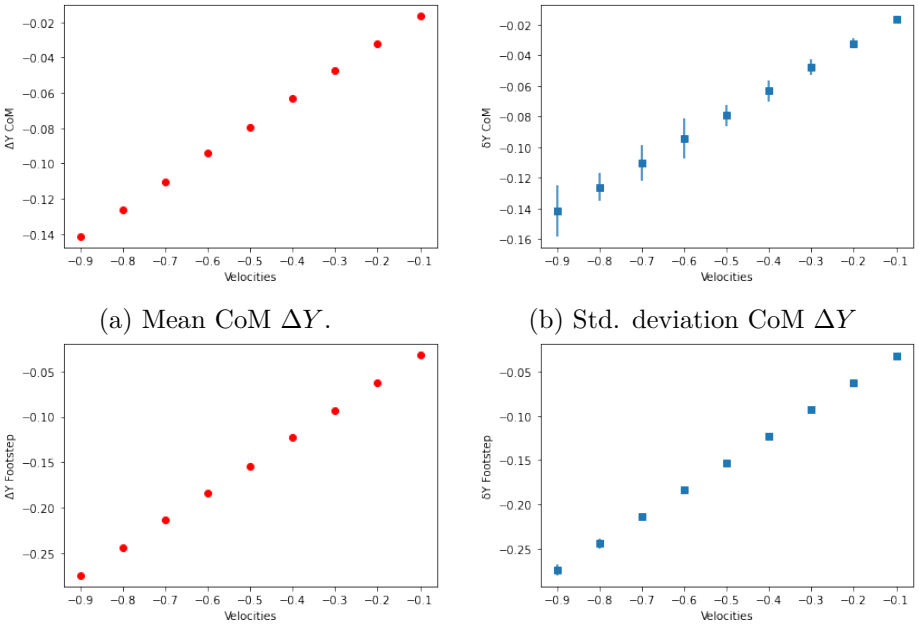
4.2 Planning

The motion planning process consists of finding high-level velocity commands that can be fed to the quadruped's controller in order to overcome complex terrains. To do so, we utilize a variant of the A^*. The graph search algorithm searches over a discrete set of velocities and uses the trained motion models and terrain information to find safe footstep sequences and respective velocity commands.

In this section, we explain the process used to obtain relevant information about the traversed terrain. Furthermore, we also present how we integrate the motion models, the terrain information, and the graph search algorithm together.

4.2.1 Height Map Processing

The elevation mapping algorithm provides a $2.5D$ height map representation of the local environment patch being traversed by the robot. The goal of the map processing module is to polish the raw height map values from erroneous or missed measurements, and compute a similar grid structure describing which cell locations



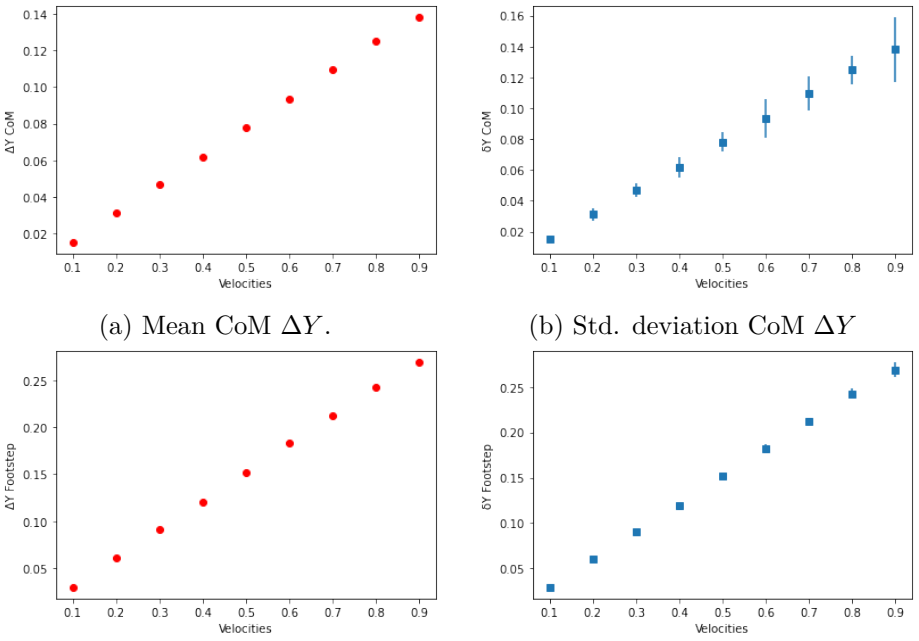
(a) Mean CoM ΔY.

(b) Std. deviation CoM ΔY

(c) Mean footstep ΔY.

(d) Std. deviation footstep ΔY

Figure 4.5: CoM and footstep mean displacement and standard deviation for a right lateral motion.



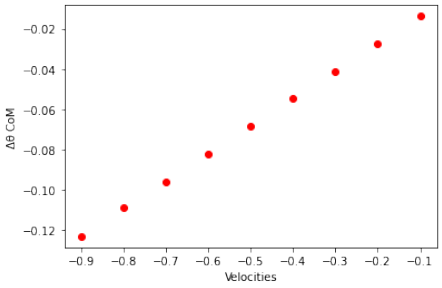
(a) Mean CoM ΔY.

(b) Std. deviation CoM ΔY

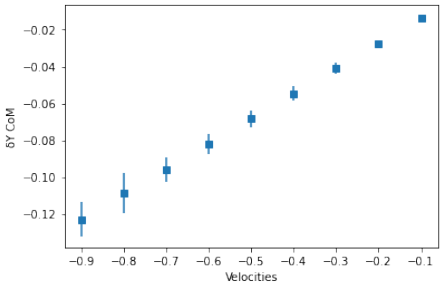
(c) Mean footstep ΔY.

(d) Std. deviation footstep ΔY

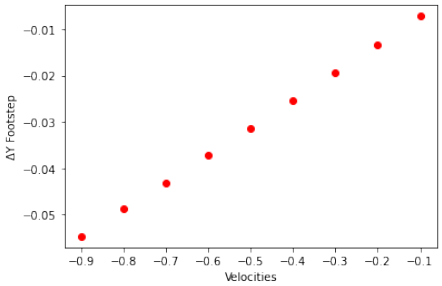
Figure 4.6: CoM and footstep mean displacement and standard deviations for a left lateral motion.



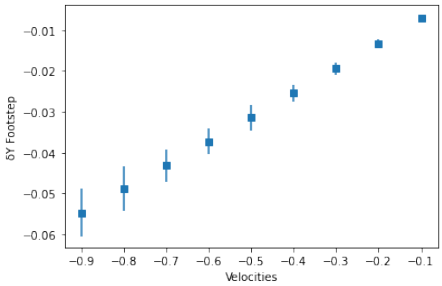
(a) Mean CoM $\Delta\Theta$.



(b) Std. deviation CoM $\Delta\Theta$

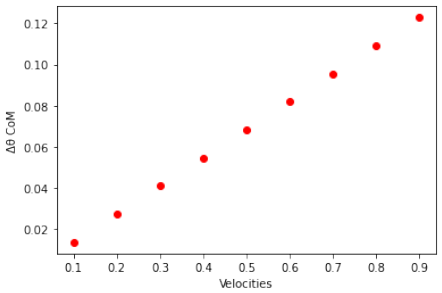


(c) Mean footstep ΔY.

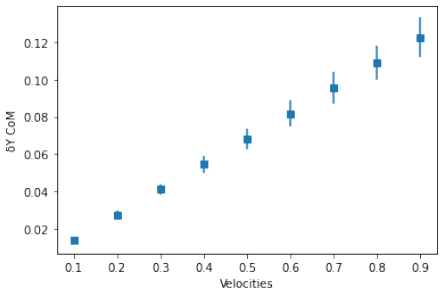


(d) Std. deviation footstep ΔY

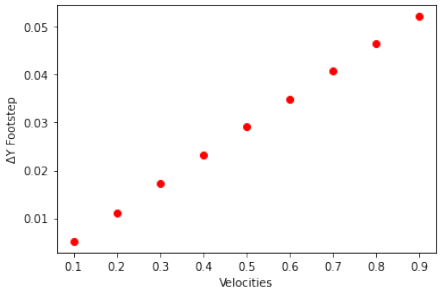
Figure 4.7: CoM and footstep mean displacement and standard deviation for a clockwise motion.



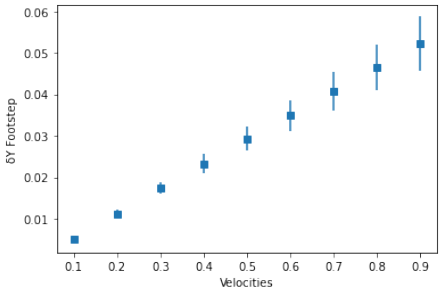
(a) Mean CoM $\Delta\Theta$.



(b) Std. deviation CoM $\Delta\Theta$



(c) Mean footstep ΔY.



(d) Std. deviation footstep ΔY

Figure 4.8: CoM and footstep mean displacement and standard deviations for a counter clockwise motion.

are safe to step onto and which are not. An example of a typical processing results in simulation is shown in Fig. 4.9.

The *ROS* middleware, and its message transport protocol, is used to subscribe to the output published by the elevation mapping node. At every received height map the respective data is stored into a cache. The cache gives a flexible way to store the incoming data with the possibility of extracting specific height map messages based on a desired timestamp. Although in our case we always use the latest height map received. All computational steps in this module are delegated to a separate thread to avoid blocking the main footstep planning logic.

At every thread cycle the latest height map is retrieved from the cache and is converted to an *OpenCV* [26] matrix structure. This conversion is necessary in order to apply common $2D$ computer vision techniques to the grid data using the OpenCV library. In particular, we apply in consecutive order a *dilation filter*, an *erosion filter*, a *median filter*, and a *sobel filter*.

The dilation operator fills sparse regions in the grid map using neighboring cell height values. The output of the dilation is then filtered by an erosion operator in order to remove extraneous sensor measurements. The erosion output is further filtered by a median filter that smooths the grid map while preserving the borders. A sobel filter operator then computes the gradients in the x and y direction used to compute the magnitude of the overall gradient at each grid location. We then clip the obtained magnitudes around a defined threshold to either 0 or 1 in order to create a foot costmap. A value of 1 signifies a safe stepping location, while a value of 0 an unsafe one.

However, directly using this costmap is not enough to guarantee that the robot is not going to step in precarious locations, as the costmap only carries information of where sharp edges appear in the grid map, but does not take into account the robot's foot radius. Moreover, due to errors in the prediction or inaccuracies in the robot's motion, the quadruped might not precisely step at the desired locations. Therefore, we introduce an additional buffer zone that marks neighboring locations to high gradient magnitude cells as unsafe to avoid stepping too close to the edges.

The buffer zone is obtained by first computing a distance transform, which is a $2D$ matrix where each (x, y) cell carries the *L2* distance to the closest cell with a value of 1 in the foot costmap. These distances are then used to create a new *buffered foot costmap*, where each cell location whose distance to the closest obstacle is less than a defined minimum distance is considered an unsafe stepping location.

This buffered costmap is generated for every processed height map and is added as an additional layer to the grid map data structure from the elevation mapping algorithm. Similarly, we also store the output of the median filter in a separate

layer. Both layers can be easily accessed by the planner in order to check if a predicted footstep location is valid and if the maximum footstep height allowed is respected.

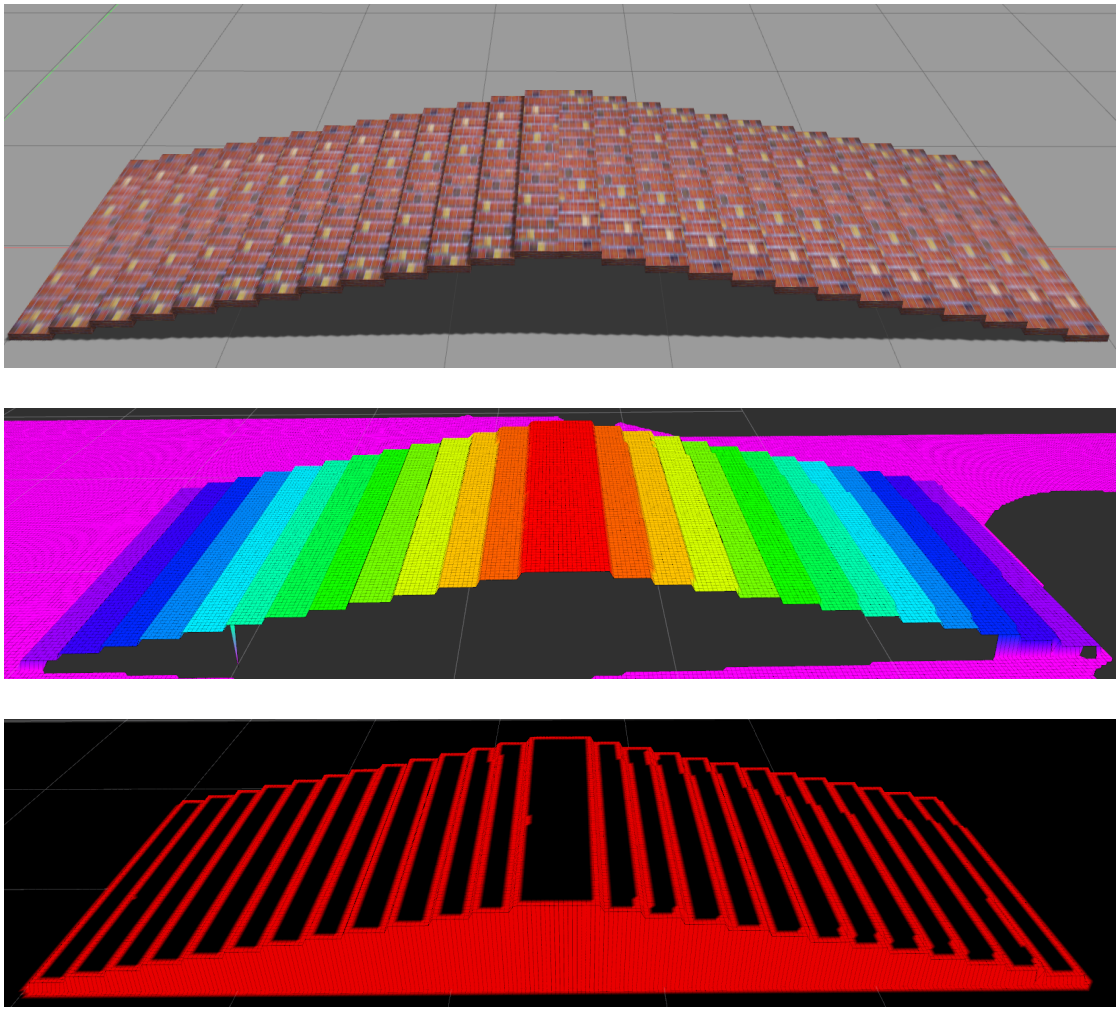


Figure 4.9: (Top) The simulated terrain in Gazebo. (Middle) The raw elevation map, generated by the mapping algorithm, color coded based on the elevation values. (Bottom) Our processing output that colors in red unsafe locations to step on, and in black safe ones using a minimum obstacle distance of 2cm.

4.2.2 Models Interface

To be able to use the trained models within the planning process these have to be made available through some sort of interface.

The *ROS* middleware offers a possible solution to this with its *service* protocol. This, can receive a prediction request and return the predicted displacements as a response. However, the overhead introduced by such a protocol is substantial, with one prediction call taking up to 1 second. Yielding it unusable for our planning time constraints.

Another possible solution is to use a ROS subscription protocol instead, where we would publish whenever needed the set of predictors for the prediction processes and obtain in return the CoM and swinging feet displacements over a topic. However, this would required creating a separate node for the models interface which comes at the cost of an increased communication overhead. Although, it is not as severe as the service protocol.

Therefore, we opt to tightly integrate the motion models using a *C++* interface. More precisely, we extract the 15 parameters for each fitted model, one parameter for every predictor plus the intercept, and store them in a vector structure. Whenever a prediction is needed, the models' vector structures of interest are fetched, and the respective model displacement can be obtained by a simple dot product with the predictors input vector. The aggregated output of the dot products is an 11-dimensional vector storing the displacements of the CoM and swinging feet. This output can then be used to update the CoM and feet poses in the world frame accordingly.

By using a C++ interface, we are able to perform multiple prediction calls with minimal computation time and communication overhead thanks to the tight integration of the models with the planner.

4.2.3 Logic

In this section, we give an overview of the logic behind the motion planning pipeline. First, we describe the input parameters of the planner. Then, we explain the main steps of the planning procedure, whose pseudo-code is shown in Algorithm 1.

The motion planner receives as input 6 parameters: the *previous velocity command*, the *start robot pose*, the *goal robot pose*, the *current CoM velocity*, and the *relative feet poses with respect to the CoM*.

The previous velocity command corresponds to the last executed high-level velocity. In the case of an idle start —i.e. the robot is not moving— the previous velocity is considered 0 for all components. The start robot pose, which describes the current CoM pose, is obtained by extracting the latest state estimate from the odometry cache. The goal pose, that represents the destination to be reached by the robot, is obtained directly from the user. Both the start and goal poses are

defined as $3D$ vectors describing the (x, y) coordinates and the Θ rotation of the robot in the world frame. The relative feet poses are obtained from the kinematic tree of the robot.

The start and goal poses are then converted to height map grid coordinates in order to obtain the respective *row* and *column* entries in the grid map for the two poses. This is in order to facilitate boundary violations and duplicate node expansion checks during the search process. Additionally, before the search process can start, a state representation for the expanded node is created that contains all required information for the prediction processes, the footstep validity checks, and the path reconstruction. This consists of 7 attributes: the respective *parent node state representation*, *velocity command*, *costs*, *CoM grid coordinates*, *CoM world coordinates*, *relative feet coordinates with respect to the CoM frame*, and *relative feet coordinates with respect to the world frame*.

The search process is carried in the world frame, where we search over a state lattice graph unrolled from the initial robot state representation. Moreover, by iterating over each allowed velocity command, we build a graph describing the evolution of the robot's states. Where each node corresponds to a state representation and each edge corresponds to a state transition as predicted by the absolute motion models. The process to build this graph consists of four main steps: *CoM prediction*, *footstep prediction*, *footstep validity check*, and *cost update*.

CoM Prediction. Given an expanded node's state representation or state, in the CoM prediction step we compute the next CoM pose in the world frame using the output of the CoM motion models and the state's CoM pose in the world frame. The CoM models used are chosen based on which foot-pair is swinging next, which we derive by observing the state's relative feet poses with respect to the CoM. If the expanded state, updated with the new CoM pose, is already present in the visited list of our search process —i.e. the same CoM grid coordinates, current velocity command, and CoM rotation were expanded before— or if it violates the boundaries of the map, we stop the search and move to the next iteration. Otherwise, we move to the next step.

Footstep Prediction. During the footstep prediction, we compute the prediction of the next swinging feet locations. Similarly to the CoM prediction, we choose which footstep models to use based on which foot-pair is going to swing next. The absolute displacements obtained by the footstep prediction are added to the state's relative feet coordinates with respect to the world frame to obtain the predicted moving feet world frame locations. The support feet are considered to have no displacement. The obtained moving feet locations are then checked for validity.

Footstep Validity Check. In the footstep validity check, we make sure that

the predicted swinging feet locations are valid. We define a footstep to be valid if two conditions uphold. First, the predicted feet locations need to fall on safe cells in the buffered costmap. Second, the height difference between the pre-prediction and post-prediction moving feet locations need to be less than the maximum footstep height allowed. To carry out these checks the height map processor module is used. The map processor takes as input the world coordinates of the swinging feet before the prediction and after the prediction. These are then converted to their corresponding grid coordinates in order to access the respective height value in the processed height map, and assess if the footstep height is within the limit allowed. Moreover, we use the same grid coordinates to check that the predicted feet locations do not fall on an unsafe cell using the buffered costmap. If any of the two conditions is false, the footstep is considered invalid and we stop the search in order to move to the next iteration. Otherwise, we move to the last step.

Cost Update. In the final step, we compute the costs for the predicted state transition node and construct a new state representation for it, which we either add or substitute into the graph. The heuristic costs computed are two: a Euclidean distance cost in order to move in the direction of the goal, and a feet configuration cost in order to maximize the feet distances to the closest unsafe cell. In fact, solely relying on the Euclidean cost and the footstep validity check might cause the robot to step too close to the edges due to prediction errors or motion inaccuracies. However, by maximizing the feet distances we can mitigate such errors. The feet configuration cost is set to 0 if the swinging foot distance to the closest unsafe cell is above a set maximum distance. Otherwise, the foot cost (C_{foot}) is the difference between the set maximum distance (D_{max}) and the respective foot distance (D_{foot}) as shown below:

$$C_{foot} = \begin{cases} D_{max} - D_{foot}, & \text{if } D_{foot} < D_{max}. \\ 0, & \text{otherwise.} \end{cases}$$

.

We aggregate the computed foot cost for each swinging foot to obtain the total feet configuration cost. We use a maximum distance (D_{max}) of 7cm for the cost computation. The total cost for the state transition node is given by adding the Euclidean and feet configuration costs together. Once the total cost is computed, we can construct the state representation for the new node by filling in the respective attributes. We then check if the node's state representation is already present in the graph. This is done by comparing the CoM grid coordinates, current velocity command, and CoM rotation attributes of the new state with each state present in the graph. If the new state has a match in the graph, we compare the states' total cost and keep the state with the least cost. Otherwise, we simply

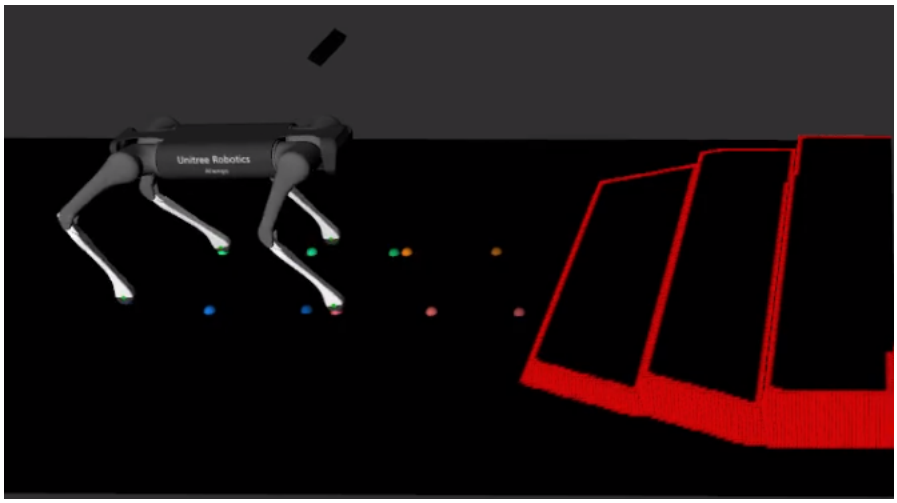
add it to the graph.

The search process continues in an iterative way by repeating the steps explained above for each expanded node. We always expand the node with the least aggregated cost. The search stops either when the expanded node is the goal, a footstep horizon has been planned for, or all the states in the graph are expanded.

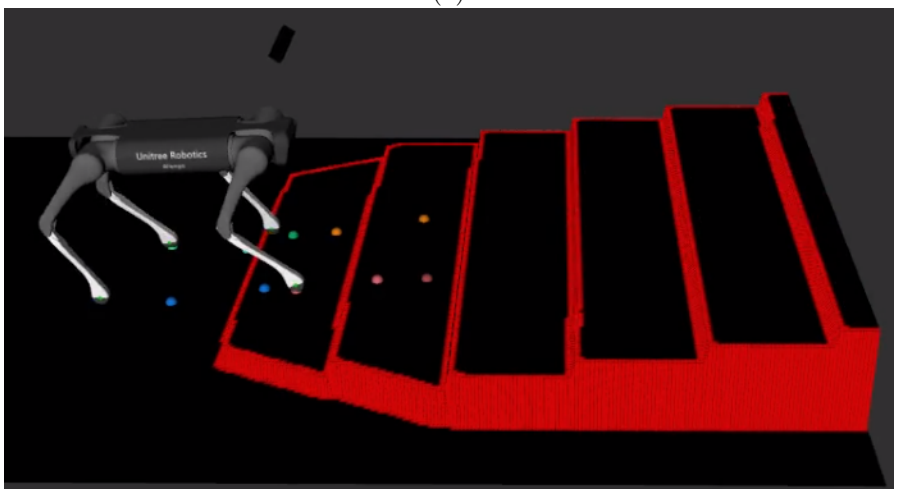
Finally, if a path is found, the planner outputs a sequence of state representations describing the CoM trajectory, footstep sequences, and respective velocity commands. This sequence is obtained by following the last expanded state's parent attribute recursively. We then use a simple control strategy that executes the very first velocity command and re-plans the next footstep horizon as soon as the feet are back in contact with the ground. If no path was found, the returned sequence is empty, and we simply bring the robot to an idle position.

Algorithm 1 Overall logic of the motion planning pipeline.

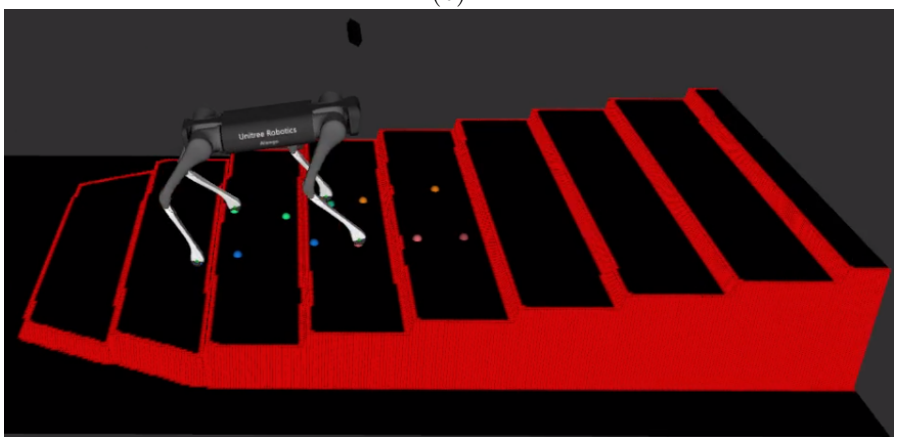
procedure FOOTSTEP PLANNING(v_{prev}, $start$, $goal$, v_{CoM}, $feet$)
    $f \leftarrow 0$
    $s_{start} \leftarrow$ **construct initial-state representation**
    $S \leftarrow \{s_{start}\}$
    $n \leftarrow$ **expand least-cost state representation from S**
    if $n ==$ *goal* or $f ==$ *horizon* or $S == \emptyset$ **then return** path
    end if
    for each velocity **do**
        for each movement **do**
CoM Prediction:
            Predict CoM for state n
            if invalid CoM **then**
                goto 9.
            end if
Footstep Prediction:
            Predict footstep for state n
Footstep Validity Check:
            if footstep height > max height or invalid feet locations **then**
                goto 9.
            end if
Cost Update:
            Construct state transition representation $n^{'}$
            Compute total cost for $n^{'}$
            if $n^{'}$ is a duplicate in S **then**
                keep state with lowest cost
            else
                $S \leftarrow S \cup n^{'}$
            end if
            $f \leftarrow f + 1$
            goto 5.
        end for
    end for
end procedure



(a)



(b)



(c)

Figure 4.10: Examples of online footstep re-planning with a horizon of 4. Orange marker represent the front-left foot, pink markers represent the front-right foot, green markers represent the rear-left foot, and blue markers represent the rear-right foot. Red regions correspond to inflated unsafe cells, while black regions correspond to safe cells.

5 Evaluation

In this chapter, we evaluate the motion models and the motion planning pipeline presented in Chapter 4. All experiments are executed on a system equipped with an AMD Ryzen 9 3900X@3.8GHz and an NVIDIA GeForce RTX 3060 Ti with 8GB of RAM.

First, we present an empirical evaluation of the prediction models and offer a comparison between the different design choices that led us to the final models. Afterwards, we evaluate the full motion planning pipeline using the Gazebo simulator on two types of environments and six different scenarios containing different obstacles. Including steps of different widths and heights as well as various gap lengths.

5.1 Models Evaluation

In this section, we evaluate and compare empirically the various design choices concerning the trained linear regression models. We use two metrics for the evaluation: the *R2* score and the *Mean Absolute Error (MAE)*. The R2 score is a percentage value between 0 and 100 that describes the proportion of the variance in the dependent variable that is predictable from the independent variables. The MAE is the average of the absolute errors between the ground truth and predicted displacements.

5.1.1 CoM Models

We identify four possible design choices for the CoM models predictors. In the first design choice (**C1**), we only use the current velocity command and the most significant feet coordinates for the CoM's component displacement to be predicted. For example, only the x coordinates of the feet are used to predict the CoM displacement in x, and only the y coordinates of the feet are used for the y and Θ components. This is due to the underlying behavior of the motions, where in order to obtain a substantial CoM displacement in x, there needs to be a substantial displacement of the moving feet in x. Similarly, to obtain a substantial CoM displacement in y or Θ, there needs to be a substantial displacement of the

moving feet in y. In the second design choice (**C2**), we drop the assumption that the displacement is only influenced by the most significant axis, and we use the full feet poses to predict all CoM components. In the third design choice (**C3**), we introduce the previously commanded velocity as an additional input to better model discontinuous motion patterns. Finally, in the fourth design option (**C4**) we also include the CoM velocity of the robot to deal with prediction ambiguities.

We carry out the evaluation and comparison of the CoM design choices in a cascading hierarchical manner as shown in Fig. 5.1. We start by comparing the first two design choices against each other. We then pick the best model design based on the R2 and MAE scores, and use it for comparison against the third design choice. The best model out of this comparison is finally used for the last evaluation against the fourth design choice.

We only present the results for one of the swinging foot-pair for each comparison, in this case the front-left and rear-right pair, as the results are similar.

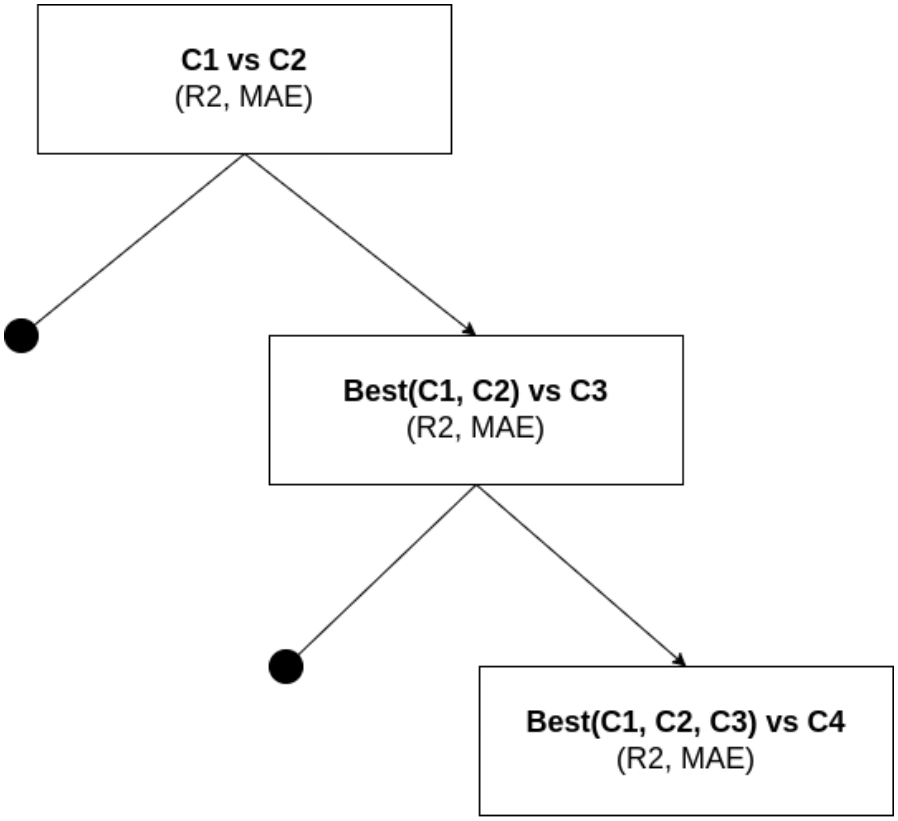


Figure 5.1: CoM models design choices comparison flow.

Including Full Feet Poses

Table 5.1: Comparison of design option 1 and 2. We state the R2 and MAE values for the prediction of the CoM.

	Partial Feet Poses		Full Feet Poses	
Component	R2 (%)	MAE (mm)	R2 (%)	MAE(mm)
X	97.0	3.81	**97.1**	**3.78**
Y	95.0	4.76	**96.3**	**3.78**
Θ	94.0	**2.80**	**94.2**	2.88

Table 5.1 compares the R2 and MAE values of the CoM models for the first two design choices, where both predictor sets consist of the currently commanded velocity command, but differ in which feet poses coordinates are used for the CoM's component prediction. It is evident from the results that using the full pose of the feet improves the dependence between the input and output variables for all three components. The average error also decreases when considering the full feet pose, with the exception of the Θ component where it slightly increases. Overall, the second design option performs better than the first one. This is most likely due to the fact that there exist slight deviations in the CoM displacement caused directly by the least significant foot coordinate. Hence, by introducing the full feet poses the predictors are better suited to predict more reliably such deviations. We therefore use the model definition using the full feet poses for the subsequent comparison.

Including Previous Velocity

Table 5.2: Comparison of design option 2 and 3. We state the R2 and MAE values for the prediction of the CoM.

	W/o Prev. Vel.		With Prev. Vel.	
Component	R2 (%)	MAE (mm)	R2 (%)	MAE(mm)
X	97.1	3.78	**97.2**	**3.70**
Y	96.3	3.78	**96.4**	**3.69**
Θ	94.2	2.88	**96.3**	**2.03**

Next, we compare the second and third design choices. Table 5.2 shows that introducing the previous velocity command improves both the R2 and MAE scores

for all components. This can be attributed to the fact that having both previous and current velocity commands facilitates the prediction of accelerating and decelerating patterns in the data. Therefore, we use the model definition of the third design option for the last comparison.

Including CoM Velocity

Table 5.3: Comparison of design option 3 and 4. We state the R2 and MAE values for the prediction of the CoM.

	W/o CoM Vel.		With CoM Vel.	
Component	R2 (%)	MAE (mm)	R2 (%)	MAE(mm)
X	97.2	3.70	**98.1**	**3.62**
Y	96.4	3.69	**97.9**	**3.53**
Θ	96.3	2.03	**96.9**	**1.86**

Finally, in Table 5.3 we compare the third and fourth design choices for the predictors. We note how adding the CoM velocity vector to the predictors set improves the model significantly. This might be explained by the fact that possible ambiguities in the components prediction are mitigated by adding the current CoM velocity. More precisely, given the last and current velocity command and the current feet configuration of the robot, the CoM displacement can differ depending on the velocity at which the robot is currently moving. Therefore, we use the model definition of the fourth design option as our final prediction model for the CoM.

5.1.2 Footstep Models

Similarly to the CoM models evaluation, we identify four possible design choices for the footstep prediction models which we compare in a hierarchical fashion. In the first design choice, we consider only the current velocity command and the swinging feet poses, with the assumption that the swinging feet displacements only depend on the moving feet poses. In the second design choice, we drop the previously made assumption and use both the swinging and support feet poses. In the third design option, we add the previous velocity command to better model acceleration patterns. In the fourth and final design choice, we also include the CoM velocity vector to better deal with prediction ambiguities.

Furthermore, in contrast to the CoM models, the footstep prediction models can have two possible prediction outputs. A *relative* prediction and an *absolute* prediction. The relative prediction outputs a relative $2D$ foot pose with respect

to the CoM, whereas the absolute prediction outputs a displacement relative to a world frame.

Therefore, we first evaluate the design choices for the case of the absolute prediction. Then, we compare the best absolute model found with the respective relative model, where we keep the same predictors for both models and only change the output accordingly.

Given that the footstep models share similar results, we only present the evaluations for one foot model. In this case, for the front-left (FL) foot.

Including All Feet Poses

Table 5.4: Comparison of design option 1 and 2. We state the R2 and MAE values for the prediction of the front-left foot.

	Swing Feet Poses		All Feet Poses	
Component	R2 (%)	MAE (mm)	R2 (%)	MAE(mm)
X	98.7	5.76	**99.2**	**4.49**
Y	98.7	4.39	**98.8**	**3.91**

Table 5.4 compares the R2 and MAE values of the FL footstep model for the first two design choices. Both include the current velocity command, but the former uses only the swinging feet poses while the latter uses both the swinging and support feet poses. The inclusion of both swinging and supporting feet poses increases the R2 scores and decreases the MAE for both components of the model. This can be explained by the fact that the swinging feet displacements are dependent on the displacements of the previous swinging pair. For example, if the displacement of the previous step was too small due to inaccurate motion execution, the current step will also be adapted to respect the kinematic constraints and maintain stability. Therefore, we use the model that includes all feet poses for the subsequent comparison.

Including Previous Velocity

Next, we compare the second and third design choices. Table 5.5 shows that introducing the previous velocity command improves both the R2 and MAE values for all predicted components of the foot. Similarly to the CoM models, this is due to the fact that by including both the previous and current velocity command the footstep model is better suited to deal with accelerating and decelerating patterns

Table 5.5: Comparison of design option 2 and 3. We state the R2 and MAE values for the prediction of the front-left foot.

	W/o Prev. Vel.		With Prev. Vel.	
Component	R2 (%)	MAE (mm)	R2 (%)	MAE(mm)
X	99.2	4.49	**99.4**	**3.33**
Y	98.8	3.91	**99.0**	**3.15**

in the dataset. Hence, we use the model definition of the third design option for the last comparison.

Including CoM Velocity

Table 5.6: Comparison of design option 3 and 4. We state the R2 and MAE values for the prediction of the front-left foot.

	W/o CoM Vel.		With CoM Vel.	
Component	R2 (%)	MAE (mm)	R2 (%)	MAE(mm)
X	99.4	3.33	**99.5**	**3.29**
Y	99.0	3.15	**99.4**	**2.86**

Finally, in Table 5.6 we compare the third and fourth design options. We note how introducing the CoM velocity vector in the predictors set improves both the R2 and MAE values for all the predicted components. Once again, similarly to the CoM models, the addition of the CoM velocity helps to improve ambiguous instances where given the same set of predictors the displacement of the components also depends on the velocity at which the robot is currently moving at. Therefore, we use the predictors defined by the fourth design choice as the final set of independent variables used for the footstep models.

Relative Prediction vs Absolute Prediction

Table 5.7 shows the comparison between a relative and absolute footstep prediction, where we use the same set of final predictors for both models but change the output predicted, as explained at the beginning of this section. Both R2 and MAE values are significantly worse for the relative model. This is because in order to predict the next relative foot pose with respect to the CoM, we need to implicitly predict the next CoM. Which is more difficult compared to predicting the foot and

Table 5.7: Comparison between relative and absolute footstep prediction. We state the R2 and MAE values for the prediction of the front-left foot.

	Relative Pred.		Absolute Pred.	
Component	R2 (%)	MAE (mm)	R2 (%)	MAE(mm)
X	97.5	3.77	**99.5**	**3.29**
Y	95.9	3.04	**99.4**	**2.86**

CoM displacements separately. Therefore, we utilize an absolute model definition for our footstep prediction task.

5.2 Planner Evaluation

In this section, we evaluate the motion planning pipeline on three simulated environments to test its capability to climb stairs, overcome gaps, and follow a global plan trajectory. In the first environment, the robot has to climb up and down three different staircase scenarios, where the step width or height varies. In the second environment, the quadruped needs to traverse discontinued flat terrains with gaps of varying length on which it cannot step. Finally, in the third environment, we simulate the input of a global plan in order to showcase lateral movements.

For each terrain, we present the *planning times*, *CoM prediction errors*, and *footstep prediction errors* for our planner. Moreover, we also compare the performance of the planner against a reactive WBC locomotion at different velocities, ranging from 0.2 to 1.0 with a step size of 0.2 describing the percentage of the maximal velocity to be used. In this comparison, we show the number of unsafe contacts for each foot and the minimum distance recorded over all feet to the closest obstacle. We define an unsafe contact to be any foot location whose distance to the closest unsafe cell is less than 1.5cm in the $xy - plane$ from the center of the foot. For each experiment, we always start the quadruped at the origin of the simulated scenario.

All presented results for the planner are obtained using a planning horizon of two steps. Furthermore, the motion models used for the evaluation are trained using footstep data with a maximum step height of 6.5cm, as described in Sec. 4.1.1.

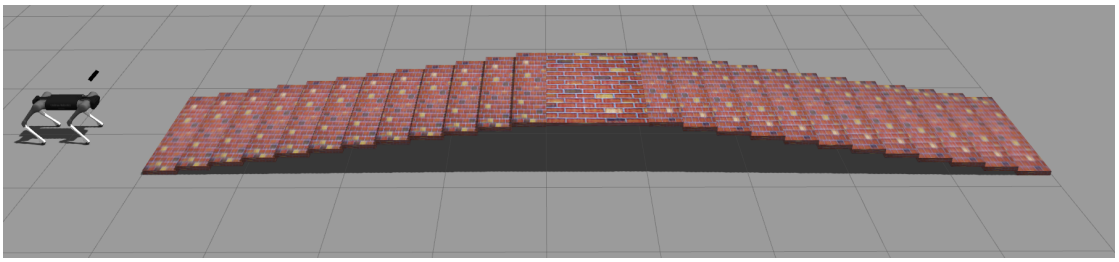
5.2.1 Staircase Environment

The first environment consists of three different staircase scenarios, as shown in Fig. 5.2. All three scenarios have 12 steps up and down but differ in the step width or height as shown in Table 5.8.

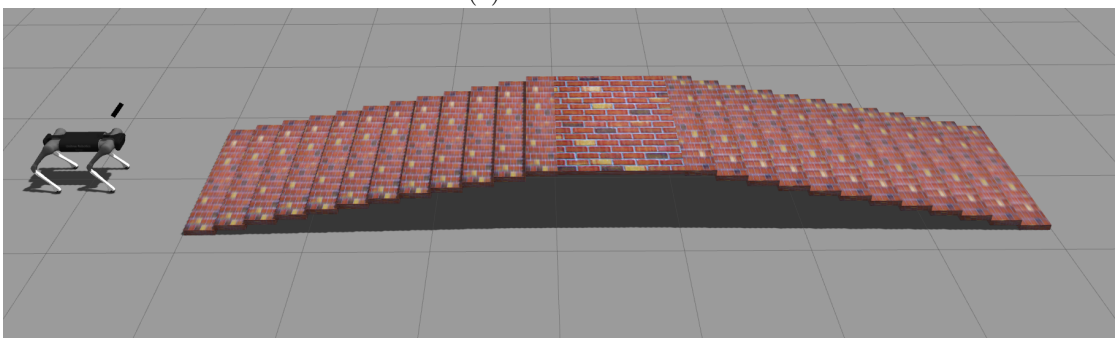
Table 5.8: Step parameters for each scenario of the Staircase Environment.

Scenario	Step Width	Step Height
Scenario 1.1	4.2cm	33cm
Scenario 1.2	4.2cm	25cm
Scenario 1.3	5cm	20cm

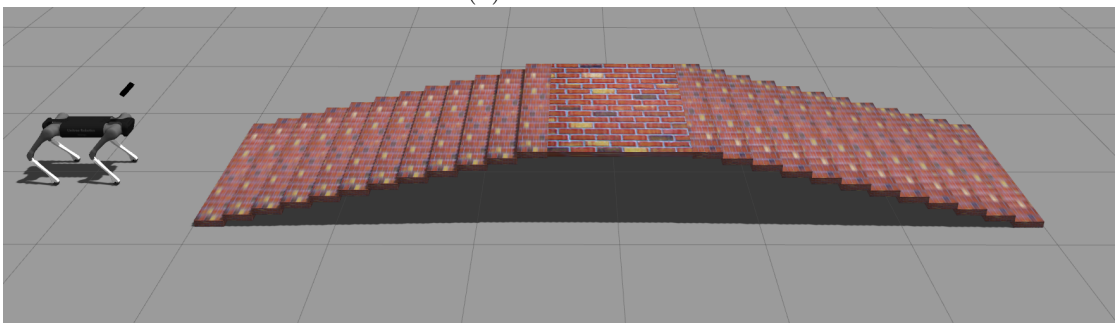
The step width and height for each scenario are chosen in order to increase the complexity of the scenarios, while respecting the limitations of the controller. In fact, for steeper slopes or larger step heights, the controller is not able to maintain stability.



(a) Scenario 1.1



(b) Scenario 1.2



(c) Scenario 1.3

Figure 5.2: The three scenarios evaluated for the Staircase Environment. All consisting of 12 steps up and down but different step width or height as shown in Table 5.8.

Planning Times

Table 5.9 shows the average and maximum planning times of three runs for each staircase scenario. All planning times are below 6ms, while on average, the next footsteps were generated within 1.5ms to 2ms. Hence, the planner is able to plan the next footstep horizon before the feet get out of contact with the ground, which happens on average after 9ms since the last footstep finishes. Furthermore, the fluctuations in the runtimes are mainly due to the CPU load of the system as several necessary software components are running during the experiments, including Gazebo and RViz [33].

Table 5.9: Average and maximum planning times for the Staircase Environment.

Experiment	Average	Maximum
Run1 (Scenario 1.1)	2004.25μs	5848μs
Run2 (Scenario 1.1)	1997.63μs	4998μs
Run3 (Scenario 1.1)	1956.81μs	4055μs
Run1 (Scenario 1.2)	2273.80μs	5321μs
Run2 (Scenario 1.2)	1678.04μs	3664μs
Run3 (Scenario 1.2)	1588.28μs	5903μs
Run1 (Scenario 1.3)	1577.04μs	2681μs
Run2 (Scenario 1.3)	2074.84μs	5911μs
Run3 (Scenario 1.3)	1636.82μs	3311μs

Prediction Errors

Table 5.10: Average CoM prediction errors in its X, Y, and Θ components for Scenario 1.1, Scenario 1.2, and Scenario 1.3.

	Mean Absolute Error		
Experiment	X	Y	Θ
Run1 (Scenario 1.1)	2.53cm	1.18cm	0.00388rad
Run2 (Scenario 1.1)	2.43cm	1.19cm	0.00376rad
Run3 (Scenario 1.1)	2.62cm	1.34cm	0.00355rad
Run1 (Scenario 1.2)	2.64cm	1.22cm	0.00322rad
Run2 (Scenario 1.2)	2.55cm	1.18cm	0.00321rad
Run3 (Scenario 1.2)	2.33cm	1.38cm	0.00333rad
Run1 (Scenario 1.3)	2.92cm	1.50cm	0.00315rad
Run2 (Scenario 1.3)	3.13cm	1.71cm	0.00326rad
Run3 (Scenario 1.3)	3.24cm	1.52cm	0.00324rad

The MAE between the predicted and real CoM poses after every velocity command execution, for each staircase scenario run, is shown in Table 5.10. Similarly, we also present the MAE between the predicted and real swinging feet locations, averaged over all feet and all steps of the corresponding run, in Table 5.11. The MAE error for both models notably increased in the X and Y components compared to their respective test set MAE, shown in Table 5.3 for the CoM model and Table 5.6 for the front-left footstep model. Moreover, the prediction errors are substantially different between the first two scenarios and the third one. This

Table 5.11: Footstep prediction errors averaged over the feet in the X and Y components for Scenario 1.1, Scenario 1.2, and Scenario 1.3.

	Mean Absolute Error	
Experiment	X	Y
Run1 (Scenario 1.1)	1.37cm	1.09cm
Run2 (Scenario 1.1)	1.25cm	1.08cm
Run3 (Scenario 1.1)	1.47cm	1.03cm
Run1 (Scenario 1.2)	1.49cm	1.11cm
Run2 (Scenario 1.2)	1.31cm	1.05cm
Run3 (Scenario 1.2)	1.20cm	1.09cm
Run1 (Scenario 1.3)	1.86cm	1.44cm
Run2 (Scenario 1.3)	1.80cm	1.72cm
Run3 (Scenario 1.3)	1.83cm	1.56cm

is due to the difference between the scenario where we train the models and the scenario where we use these models. The former is a flat terrain while the latter is a set of non-flat terrains. The Θ component for the CoM does not change significantly as it is not affected by the elevation change of the scenarios.

Planner vs WBC

Table 5.12 compares our planner and a blind WBC locomotion at different velocities. The number of unsafe contacts for the blind locomotion decreases as the velocity gets higher. At higher velocities, the robot executes larger footsteps. Thus, the number of required footsteps reduces and so does the probability of stepping on unsafe cells. However, the minimum distances for the blind WBC locomotion remain constantly low and close to 0 at all velocities. This is because the WBC does not consider the terrain, but only uses proprioceptive information. On the other hand, our planner performs better compared to a blind locomotion. In particular, the number of unsafe contacts drastically reduces as information about the terrain is now used by the planner. Moreover, even in the occasion where the safety distance of 1.5cm is not respected by the planner due to errors in the prediction or inaccuracies in the motion execution, the minimum distance to an obstacle is never below 1cm from the center of the foot.

In Fig. 5.3 we show a top-down view example of predicted (green) and real (blue) CoM and footstep sequences for Scenario 1.3. The red regions correspond to inflated edges and represent a safety area. Therefore, directly stepping inside

Table 5.12: Comparison of the number of unsafe contacts and the minimum recorded distance over all feet between our planner and a blind WBC locomotion at different velocities for Scenario 1.1, Scenario 1.2, and Scenario 1.3. Experiments marked with an asterisk (*) signify a failed traversal of the terrain.

		Unsafe Contacts				Min. Dist.
	Planner	FL Foot	FR Foot	RL Foot	RR Foot	All Feet
Scenario 1.1	Ours (Run1)	**0/78**	**0/78**	**0/78**	**0/78**	1.81cm
	Ours (Run2)	1/79	**0/79**	1/79	**0/78**	1.28cm
	Ours (Run3)	**0/78**	**0/78**	**0/78**	**0/78**	**2.82cm**
	WBC (v=0.2)	11/201	19/201	13/201	19/201	<0.01cm
	WBC (v=0.4)	11/92	1/92	12/92	2/92	<0.01cm
	WBC (v=0.6)	2/61	5/61	8/61	3/61	<0.01cm
	WBC (v=0.8)	3/45	2/45	3/45	3/45	<0.01cm
	WBC (v=1.0)	2/37	3/37	2/37	4/37	<0.01cm
Scenario 1.2	Ours (Run1)	1/57	**0/57**	**0/57**	**0/57**	1.10cm
	Ours (Run2)	**0/58**	**0/58**	**0/58**	**0/58**	**3.38cm**
	Ours (Run3)	**0/57**	**0/57**	**0/57**	**0/57**	2.26cm
	WBC (v=0.2)	13/164	16/164	16/164	15/164	<0.01cm
	WBC (v=0.4)	8/75	6/75	6/75	4/75	<0.01cm
	WBC (v=0.6)	6/61	8/61	6/61	8/61	<0.01cm
	WBC (v=0.8)	2/49	2/49	3/49	2/49	<0.01cm
	WBC (v=1.0)	4/37	1/37	1/37	4/37	<0.01cm
Scenario 1.3	Ours (Run1)	**0/39**	**0/39**	1/39	**0/39**	1.00cm
	Ours (Run2)	**0/40**	1/40	1/40	**0/40**	**1.20cm**
	Ours (Run3)	**0/40**	2/40	**0/40**	**0/40**	1.03cm
	WBC (v=0.2)	12/145	14/145	15/145	13/145	<0.01cm
	WBC (v=0.4)	8/67	7/67	6/67	5/67	<0.01cm
	WBC (v=0.6)	5/44	5/44	4/44	6/44	<0.01cm
	WBC (v=0.8)	1/34	3/34	4/34	1/34	<0.01cm
	WBC (v=1.0)*	2/21*	1/21*	1/21*	1/21*	<0.01cm

these zones does not necessarily result in a fall or slip.

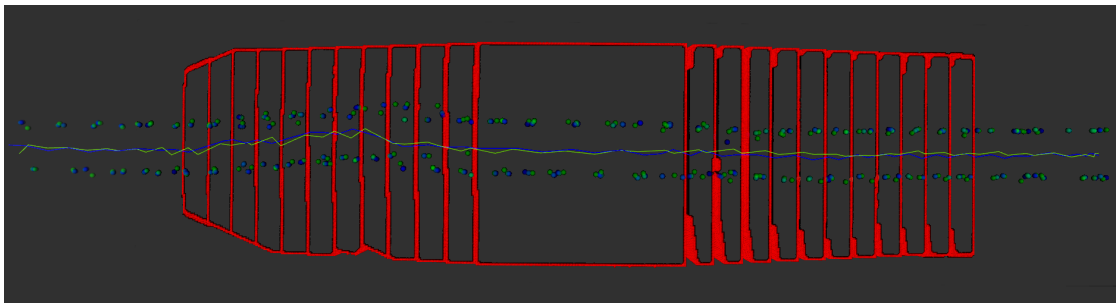


Figure 5.3: Top-down view example of predicted (green) and real (blue) CoM trajectory and footstep sequences for Scenario 1.3. Red regions correspond to inflated edges, and directly stepping on such zones does not signify a fall or slip.

5.2.2 Gaps Environment

In the second environment we test the planner's ability to deal with gaps of varying sizes, while taking in consideration the physical capabilities of the quadruped which can overcome gaps of at most 12.5cm width. Fig. 5.4 shows the two scenarios belonging to the second environment.

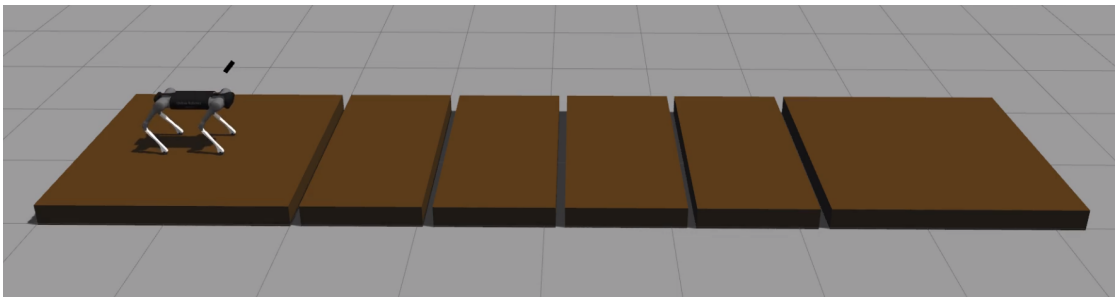
Both scenarios consist of discontinued elevated terrain blocks with a height of 18cm, but differ in the number of blocks and gaps sizes range as shown in Table 5.13. We set the maximum gap size to 9cm as we consider a security margin of 1.5cm at edges. Hence requiring a total stepping length of 12cm, which is slightly below the kinematic constraints of the robot.

Table 5.13: Scenarios parameters for Gaps Environment.

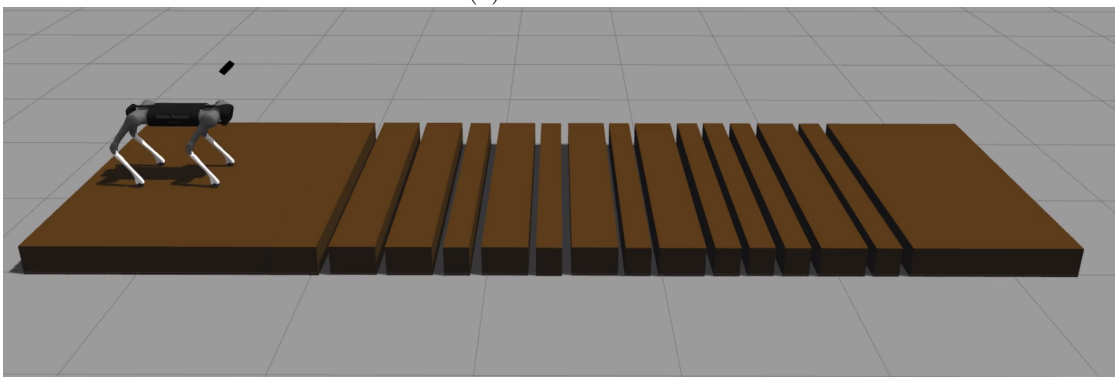
Scenario	Number of Gaps	Gaps Size Range
Scenario 2.1	5	7cm - 9cm
Scenario 2.2	14	1.5cm - 7.1cm

Planning Times

The average and maximum planning times for the Gaps Environment are shown in Table 5.14. The results are similar to the Staircase Environment, with the exception of slightly higher average runtimes. However, this is only due to the CPU load slightly affecting the planning times. The highest average and maximum planning times are 2.3ms and 5.9ms respectively, which is sufficient to finish planning before the next footstep execution starts.



(a) Scenario 2.1



(b) Scenario 2.2

Figure 5.4: The two scenarios evaluated for the Gaps Environment, each with a different number of gaps and gaps sizes as shown in Table 5.13.

Table 5.14: Average and maximum planning times for Scenario 2.1, 2.2, and 2.3.

Experiment	Average	Maximum
Run1 (Scenario 2.1)	2118.52μs	4721μs
Run2 (Scenario 2.1)	2208.13μs	4384μs
Run3 (Scenario 2.1)	1965.48μs	5492μs
Run1 (Scenario 2.2)	1819.34μs	4231μs
Run2 (Scenario 2.2)	2356.31μs	5987μs
Run3 (Scenario 2.2)	2008.11μs	4986μs

Prediction Errors

The MAE between the predicted and real CoM pose for the Gaps Environment is shown in Table 5.15. Similarly, we show the MAE between the predicted and real swinging feet locations averaged over all feet in Table 5.16. Both models' prediction errors decrease compared to the Staircase Environment for the X and Y components, while the Θ component remains mostly unvaried. This is because the scenarios evaluated in the second environment are flat, thereby obtaining a

Table 5.15: Average CoM prediction errors in its X, Y, and Θ components for Scenarios 2.1 and 2.2.

	Mean Absolute Error		
Experiment	X	Y	Θ
Run1 (Scenario 2.1)	1.19cm	1.15cm	0.00201rad
Run2 (Scenario 2.1)	1.29cm	1.18cm	0.00199rad
Run3 (Scenario 2.1)	1.15cm	1.12cm	0.00200rad
Run1 (Scenario 2.2)	1.17cm	1.14cm	0.00192rad
Run2 (Scenario 2.2)	1.26cm	1.11cm	0.00193rad
Run3 (Scenario 2.2)	1.14cm	1.10cm	0.00199rad

Table 5.16: Footsteps prediction errors averaged over the feet in the X and Y components for Scenarios 2.1 and 2.2.

Experiment	X	Y
Run1 (Scenario 2.1)	1.06cm	1.01cm
Run2 (Scenario 2.1)	0.998cm	1.02cm
Run3 (Scenario 2.1)	1.05cm	0.998cm
Run1 (Scenario 2.2)	0.992cm	0.999cm
Run2 (Scenario 2.2)	1.05cm	1.03cm
Run3 (Scenario 2.2)	1.01cm	1.00cm

similar controller behavior to the scenario in which we trained the motion models. However, the models' errors for the two scenarios are still higher compared to the MAE of the respective test sets. This can be explained by the frequent acceleration patterns that happen over a smaller timescale than what we considered in the training set, corresponding to 1.5s.

Planner vs WBC

Table 5.17 presents the number of unsafe contacts and the minimum recorded distance over all feet to the closest unsafe cell, for all scenarios. All executed experiments for the blind WBC locomotion, for both scenarios, result in a failed traversal as the robot always steps inside the gaps. Therefore, we report the statistics of the trajectory part before the failure, where we note how a blind locomotion leads to stepping very close to the edges of the gaps. On the other hand, our planner is able to successfully traverse all terrains on all three distinct runs, with very few unsafe contacts and a minimum distance of 1.1cm from the

Table 5.17: Comparison of the number of unsafe contacts between our planner and a blind WBC locomotion carried at different velocities for Scenario 2.1 and 2.2. Experiments marked with an asterisk (*) signify a failed traversal of the terrain.

		Unsafe Contacts				Min. Dist.
	Planner	FL Foot	FR Foot	RL Foot	RR Foot	All Feet
Scenario 2.1	Ours (Run1)	**0/44**	**0/44**	**0/44**	**0/44**	3.37cm
	Ours (Run2)	**0/44**	**0/44**	**0/44**	**0/44**	**4.86cm**
	Ours (Run3)	**0/45**	**0/45**	**0/45**	**0/45**	3.91cm
	WBC (v=0.2)*	2/14*	0/14*	0/14*	0/14*	1.06cm*
	WBC (v=0.4)*	1/9*	0/9*	0/9*	0/9*	<0.01cm*
	WBC (v=0.6)*	1/8*	1/8*	1/8*	0/8*	<0.01cm*
	WBC (v=0.8)*	0/6*	0/6*	0/6*	0/6*	2.37cm*
	WBC (v=1.0)*	0/4*	0/4*	0/4*	0/4*	2.53cm*
Scenario 2.2	Ours (Run1)	1/57	**0/57**	**0/57**	**0/57**	1.10cm
	Ours (Run2)	**0/58**	**0/58**	**0/58**	**0/58**	**3.38cm**
	Ours (Run3)	**0/57**	**0/57**	**0/57**	**0/57**	2.26cm
	WBC (v=0.2)*	1/16*	0/16*	0/16*	0/16*	<0.01cm*
	WBC (v=0.4)*	0/9*	0/9*	1/9*	0/9*	<0.01cm*
	WBC (v=0.6)*	0/6*	0/6*	1/6*	0/6*	<0.01cm*
	WBC (v=0.8)*	0/5*	0/5*	0/5*	0/5*	2.40cm*
	WBC (v=1.0)*	0/4*	0/4*	0/4*	0/4*	2.54cm*

center of the foot on the most difficult scenario (Scenario 2.2).

Fig. 5.5 shows a top-down view of the CoM trajectory and footstep sequences for Scenario 2.2. The red regions correspond to inflated edges, and directly stepping on such zones does not signify a fall or slip.

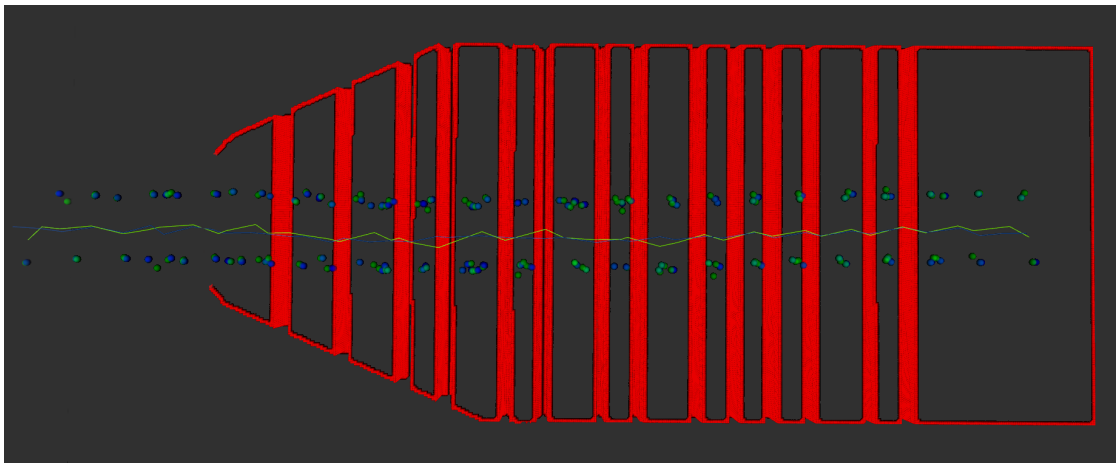


Figure 5.5: Predicted (green) and real (blue) CoM trajectory and footstep sequences for Scenario 2.2. Red regions correspond to inflated edges, and directly stepping on such zones does not signify a fall or slip.

5.2.3 Waypoint Environment

In the third and final environment, we test the planner on a course of mixed obstacles where we simulate a global planner input in order to showcase lateral stepping motions. Fig. 5.6 shows the scenario belonging to the third environment.

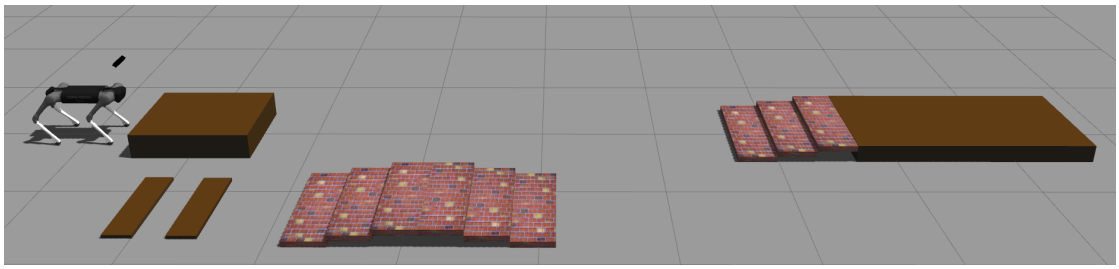


Figure 5.6: The scenario evaluated for the Waypoint Environment.

Planning Times

Table 5.18: Average and maximum planning times for Scenario 3.1.

Experiment	Average	Maximum
Run1 (Scenario 3.1)	2005.12μs	4333μs
Run2 (Scenario 3.1)	2350.56μs	4734μs
Run3 (Scenario 3.1)	2420.14μs	5892μs

The average and maximum planning times for the Waypoint Environment, shown in Table 5.18, does not differ from the planning times of the two previous environments. In fact, we note a similar range of values with the highest average and maximum planning time of 2.4ms and 5.9ms respectively. Once again, such times are sufficient to finish the planning before the next footstep sequence starts.

Prediction Errors

Table 5.19: Average CoM prediction errors in its X, Y, and Θ components for Scenario 3.1.

	Mean Absolute Error		
Experiment	X	Y	Θ
Run1 (Scenario 3.1)	0.997cm	0.862cm	0.00203rad
Run2 (Scenario 3.1)	1.02cm	0.834cm	0.00199rad
Run3 (Scenario 3.1)	1.01cm	0.863cm	0.00201rad

Table 5.20: Footsteps prediction errors averaged over the feet in the X and Y components for Scenario 3.1.

Experiment	X	Y
Run1 (Scenario 3.1)	0.983cm	0.968cm
Run2 (Scenario 3.1)	0.986cm	0.971cm
Run3 (Scenario 3.1)	0.991cm	0.972cm

The last evaluated environment has the lowest MAE for the CoM and swinging feet predictions among all environments, as shown in Table 5.19 and 5.20 respectively. This is because compared the to the Staircase Environment, the Waypoint Environment is mostly flat, thereby not affecting much the controller behavior. Moreover, compared to the Gaps Environment, the Waypoint Environment does not require as many acceleration patterns, as the majority of the traversed terrain is flat.

Planner vs WBC

Table 5.21 presents the number of unsafe contacts and the minimum distance recorded to the closest unsafe cell over the feet for the last evaluated scenario. The results are similar to the Staircase Environment, where all runs for the blind

Table 5.21: Comparison of the number of unsafe contacts between our planner and a blind WBC locomotion executed at different velocities for Scenario 3.1.

		Unsafe Contacts				Min. Dist.
	Planner	FL Foot	FR Foot	RL Foot	RR Foot	All Feet
Scenario 3.1	Ours (Run1)	**0/88**	**0/88**	**0/88**	**0/88**	3.37cm
	Ours (Run2)	**0/87**	**0/87**	**0/87**	**0/87**	**4.86cm**
	Ours (Run3)	**0/89**	**0/89**	**0/89**	**0/89**	3.91cm
	WBC (v=0.2)	8/254	7/254	9/254	9/254	<0.01cm
	WBC (v=0.4)	7/111	6/111	4/111	5/111	<0.01cm
	WBC (v=0.6)	3/81	4/81	2/81	3/81	<0.01cm
	WBC (v=0.8)	5/56	4/56	3/56	4/56	<0.01cm
	WBC (v=1.0)	2/33	2/33	3/33	2/33	<0.01cm

WBC locomotion reach successfully the destination, but with a high number of unsafe contacts and low minimum distances. On the other hand, our planner is also able to successfully traverse the scenario, but with significantly less unsafe contacts and fairly high minimum distances. In fact, the number of unsafe contacts for each foot is 0, and a minimum recorded distance of 3.31cm.

Fig. 5.5 shows a side view of the CoM trajectory and footstep sequences for Scenario 3.1. The red regions correspond to inflated edges, and directly stepping on such zones does not signify a fall or slip.

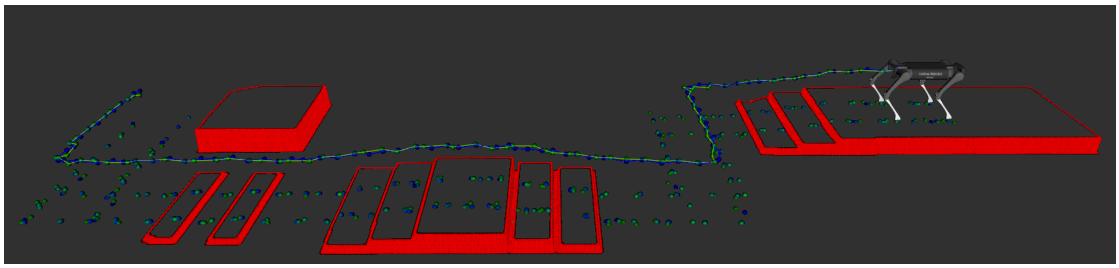


Figure 5.7: Side-view example of predicted (green) and real (blue) CoM trajectory and footstep sequences for Scenario 3.1. Red regions correspond to inflated edges, and directly stepping on such zones does not signify a fall or slip.

6 Conclusions

This thesis presented a method that allows quadrupeds to navigate irregular terrains, including steps and gaps, by extending the capability of a WBC that is able to track high-level velocity commands, but is unable to generate trajectories and torques for target footstep locations.

We achieve this by formulating the method as a local motion planning problem, whose task is to plan the CoM trajectory, footstep sequences, and relative high-level velocity commands that allow the quadruped to traverse the terrain. The planner uses a set of learned motion models, a foot costmap, and a variant of the A^* algorithm. The motion models are linear regression functions that map an input vector consisting of the the previous and current velocity commands, the relative feet position with respect to the CoM, and the CoM velocity to the respective absolute displacements of the CoM and swinging feet. We use the output of a modified elevation mapping algorithm that reconstructs a local patch of the traversed terrain in real-time as a $2.5D$ grid to compute the foot costmap. The foot costmap describes which cells are safe to step onto using gradient information extracted from the height map. The motion models and the costmap are then integrated with a variant of the A^* algorithm that uses a modified state-update rule where we associate the continuous state of the robot with the discrete cell. We use two costs for the search. A heuristic Euclidean cost to guide the CoM towards the goal, and a footstep margin cost that seeks to maximize the swinging feet distances from the closest unsafe cell location.

We evaluated our method on three different environments. A Staircase Environment, consisting of three staircase scenarios with different step width and height. A Gaps Environment, containing two scenarios consisting of a series of elevated and discontinued flat terrains with gaps of varying sizes. And a Waypoint Environment, composed of a mixed obstacle course scenario where we simulated a global planner input and showcased lateral stepping motions. We compared our planner with a reactive WBC locomotion, and showed that our method is able to safely navigate all scenarios while never getting too close to unsafe regions. On the other hand, a blind-reactive locomotion failed all scenarios of the Gaps Environment and constantly stepped on unsafe regions on the remaining environments.

We also found some limitations to the presented method. First of all, the planner

is as good as the controller. Meaning that if the controller is not able to keep stability on different slopes or handle sharp velocity changes, the proposed method would not benefit the system. Secondly, the planner requires accurate training data in order to model the controller behavior for the terrain types to be traversed. Including learning separate models for each significant change in elevation, as assuming that the controller behavior does not change for very steep slopes can lead to meaningful prediction inaccuracies.

There are also possible extensions to the method. First, we could integrate a global planner in our approach that computes at lower frequencies intermediate waypoints which can be fed to the local planner. Second, we can introduce a trajectory collision checking that makes sure the swinging feet trajectories are not colliding with the environment. Finally, we could learn several motions models for different footstep heights, and condition our method on the terrain height ahead. This would have the benefit to improve the prediction accuracy but also yield a more energy efficient navigation, as we would use high footstep heights only when required.

List of Figures

List of Tables

Bibliography

[1] ETH Zurich Autonomous Systems Lab Robotic Systems Lab Anybotics. "Https://github.com/anybotics/elevation$_m$*apping*." In: ().

[2] C Dario Bellicoso, Marko Bjelonic, Lorenz Wellhausen, Kai Holtmann, Fabian Günther, Marco Tranzatto, Peter Fankhauser, and Marco Hutter. "Advances in real-world applications for legged robots." In: *Journal of field robotics* 35.8 (2018), pp. 1311–1326.

[3] Chris M Bishop. "Neural networks and their applications." In: *Review of scientific instruments* 65.6 (1994), pp. 1803–1832.

[4] Michael Bloesch, Marco Hutter, Mark A Hoepflinger, Stefan Leutenegger, Christian Gehring, C David Remy, and Roland Siegwart. "State estimation for legged robots-consistent fusion of leg kinematics and imu." In: *Robotics* 17 (2013), pp. 17–24.

[5] Joel Chestnutt, Manfred Lau, German Cheung, James Kuffner, Jessica Hodgins, and Takeo Kanade. "Footstep planning for the honda asimo humanoid." In: *Proceedings of the 2005 ieee international conference on robotics and automation*. IEEE. 2005, pp. 629–634.

[6] Dmitri Dolgov, Sebastian Thrun, Michael Montemerlo, and James Diebel. "Practical search techniques in path planning for autonomous driving." In: *Ann arbor* 1001.48105 (2008), pp. 18–80.

[7] Boston Dynamics. *Spot*. 2022. URL: `https://www.bostondynamics.com/products/spot`.

[8] Davide Faconti. *Plotjuggler*. 2022. URL: `https://www.plotjuggler.io/`.

[9] Péter Fankhauser. "Perceptive locomotion for legged robots in rough terrain." PhD thesis. ETH Zurich, 2018.

[10] Péter Fankhauser, Marko Bjelonic, C Dario Bellicoso, Takahiro Miki, and Marco Hutter. "Robust rough-terrain locomotion with a quadrupedal robot." In: *2018 ieee international conference on robotics and automation (icra)*. IEEE. 2018, pp. 5761–5768.

[11] Péter Fankhauser, Michael Bloesch, and Marco Hutter. "Probabilistic terrain mapping for mobile robots with uncertain localization." In: *Ieee robotics and automation letters* 3.4 (2018), pp. 3019–3026.

[12] Péter Fankhauser and Marco Hutter. "Anymal: a unique quadruped robot conquering harsh environments." In: *Research features* 126 (2018), pp. 54–57.

[13] Siddhant Gangapurwala, Mathieu Geisert, Romeo Orsolino, Maurice Fallon, and Ioannis Havoutis. "Rloc: terrain-aware legged locomotion using reinforcement learning and optimal control." In: *Arxiv preprint arxiv:2012.03094* (2020).

[14] Peter E Hart, Nils J Nilsson, and Bertram Raphael. "A formal basis for the heuristic determination of minimum cost paths." In: *Ieee transactions on systems science and cybernetics* 4.2 (1968), pp. 100–107.

[15] Enrico Mingo Hoffman, Arturo Laurenzi, Luca Muratore, Nikos G Tsagarakis, and Darwin G Caldwell. "Multi-priority cartesian impedance control based on quadratic programming optimization." In: *2018 ieee international conference on robotics and automation (icra)*. IEEE. 2018, pp. 309–315.

[16] Fabian Jenelten, Takahiro Miki, Aravind E Vijayan, Marko Bjelonic, and Marco Hutter. "Perceptive locomotion in rough terrain–online foothold optimization." In: *Ieee robotics and automation letters* 5.4 (2020), pp. 5370–5376.

[17] Matthew Johnson, Brandon Shrewsbury, Sylvain Bertrand, Tingfan Wu, Daniel Duran, Marshall Floyd, Peter Abeles, Douglas Stephen, Nathan Mertins, Alex Lesman, et al. "Team ihmc's lessons learned from the darpa robotics challenge trials." In: *Journal of field robotics* 32.2 (2015), pp. 192–208.

[18] Mrinal Kalakrishnan, Jonas Buchli, Peter Pastor, Michael Mistry, and Stefan Schaal. "Fast, robust quadruped locomotion over challenging terrain." In: *2010 ieee international conference on robotics and automation*. IEEE. 2010, pp. 2665–2670.

[19] Mrinal Kalakrishnan, Jonas Buchli, Peter Pastor, Michael Mistry, and Stefan Schaal. "Learning, planning, and control for quadruped locomotion over challenging terrain." In: *The international journal of robotics research* 30.2 (2011), pp. 236–258.

[20] J Zico Kolter, Mike P Rodgers, and Andrew Y Ng. "A control architecture for quadruped locomotion over rough terrain." In: *2008 ieee international conference on robotics and automation*. IEEE. 2008, pp. 811–818.

[21] Arturo Laurenzi, Enrico Mingo Hoffman, Matteo Parigi Polverini, and Nikos G Tsagarakis. "Balancing control through post-optimization of contact forces." In: *2018 ieee-ras 18th international conference on humanoid robots (humanoids)*. IEEE. 2018, pp. 320–326.

[22] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[23] Octavio Antonio Villarreal Magana, Victor Barasuol, Marco Camurri, Luca Franceschi, Michele Focchi, Massimiliano Pontil, Darwin G Caldwell, and Claudio Semini. "Fast and continuous foothold adaptation for dynamic locomotion through cnns." In: *Ieee robotics and automation letters* 4.2 (2019), pp. 2140–2147.

[24] Takahiro Miki, Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. "Learning robust perceptive locomotion for quadrupedal robots in the wild." In: *Science robotics* 7.62 (2022), eabk2822.

[25] Simona Nobili, Marco Camurri, Victor Barasuol, Michele Focchi, Darwin G Caldwell, Claudio Semini, and Maurice F Fallon. "Heterogeneous sensor fusion for accurate state estimation of dynamic legged robots." In: *Robotics: science and systems*. 2017.

[26] *Opencv*. 2022. URL: `https://opencv.org/`.

[27] Xue Bin Peng, Glen Berseth, KangKang Yin, and Michiel Van De Panne. "Deeploco: dynamic locomotion skills using hierarchical deep reinforcement learning." In: *Acm transactions on graphics (tog)* 36.4 (2017), pp. 1–13.

[28] Nicolaus A Radford, Philip Strawser, Kimberly Hambuchen, Joshua S Mehling, William K Verdeyen, A Stuart Donnan, James Holley, Jairo Sanchez, Vienny Nguyen, Lyndon Bridgwater, et al. "Valkyrie: nasa's first bipedal humanoid robot." In: *Journal of field robotics* 32.3 (2015), pp. 397–419.

[29] Gennaro Raiola, Enrico Mingo Hoffman, Michele Focchi, Nikos Tsagarakis, and Claudio Semini. "A simple yet effective whole-body locomotion framework for quadruped robots." In: *Frontiers in robotics and ai* (2020), p. 159.

[30] Open Robotics. *Gazebo*. 2022. URL: `http://gazebosim.org/`.

[31] Open Robotics. *ROS*. 2022. URL: `https://www.ros.org/`.

[32] Unitree Robotics. *Aliengo*. 2022. URL: `https://www.unitree.com/products/aliengo/`.

[33] *Rviz*. 2022. URL: `http://wiki.ros.org/rviz`.

[34] Andreas Schmitz, Marcell Missura, and Sven Behnke. "Learning footstep prediction from motion capture." In: *Robot soccer world cup*. Springer. 2010, pp. 97–108.

[35] Andreas Schmitz, Marcell Missura, and Sven Behnke. "Real-time trajectory generation by offline footstep planning for a humanoid soccer robot." In: *Robot soccer world cup*. Springer. 2011, pp. 198–209.

[36] Claudio Semini, Victor Barasuol, Jake Goldsmith, Marco Frigerio, Michele Focchi, Yifu Gao, and Darwin G Caldwell. "Design of the hydraulically actuated, torque-controlled quadruped robot hyq2max." In: *Ieee/asme transactions on mechatronics* 22.2 (2016), pp. 635–646.

[37] IEEE Robotics Automation Society. *Wbc-ieee*. 2022. URL: `https://www.ieee-ras.org/whole-body-control`.

[38] Xiaogang Su, Xin Yan, and Chih-Ling Tsai. "Linear regression." In: *Wiley interdisciplinary reviews: computational statistics* 4.3 (2012), pp. 275–294.

[39] Alan O Sykes. "An introduction to regression analysis." In: (1993).

[40] Nikolaos G Tsagarakis, Darwin G Caldwell, Francesca Negrello, Wooseok Choi, Lorenzo Baccelliere, Vo-Gia Loc, J Noorden, Luca Muratore, Alessio Margan, Alberto Cardellino, et al. "Walk-man: a high-performance humanoid platform for realistic environments." In: *Journal of field robotics* 34.7 (2017), pp. 1225–1259.

[41] Vassilios Tsounis, Mitja Alge, Joonho Lee, Farbod Farshidian, and Marco Hutter. "Deepgait: planning and control of quadrupedal gaits using deep reinforcement learning." In: *Ieee robotics and automation letters* 5.2 (2020), pp. 3699–3706.

[42] Alexander W Winkler, Carlos Mastalli, Ioannis Havoutis, Michele Focchi, Darwin G Caldwell, and Claudio Semini. "Planning and execution of dynamic whole-body locomotion for a hydraulic quadruped on challenging terrain." In: *2015 ieee international conference on robotics and automation (icra)*. IEEE. 2015, pp. 5148–5154.