BiQu: Stair Climbing Robot Dog

A Major Qualifying Project (MQP) Report
Submitted to the Faculty of
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements
for the Degree of Bachelor of Science in

Robotics Engineering,
Computer Science

By:

Wyatt Harris
Adam Kalayjian
Sean Lendrum
Jared Morgan
Kai Nakamura
Owen Sullivan

Project Advisors:

Agheli Hajiabadi, Mohammad Mahdi
Jing Xiao
Guanrui Li

Date: April 30, 2025

Abstract

Legged robots are designed to tackle environments where wheels would otherwise fail. Quadrupedal robots, or "robot dogs," offer unique advantages in stability over humanoid, bipedal robots. Developments from organizations like Boston Dynamics and Unitree demonstrate wide applications for quadrupeds like inspection, surveying, logistics, etc. However, complex terrains that require precision, like stairs, still remain difficult. This work presents a comprehensive framework for real-time stair climbing, integrating terrain perception, footstep planning, and contact estimation. It utilizes LiDAR for SLAM, improving odometry and reducing map drift. Hardware upgrades expand robot capabilities, and novel integration of mapping confidence values improves terrain interaction. A dynamic footstep planner maps footholds onto convex planes, providing initial guesses to speed up trajectory optimization. Contact estimation combines foot height, force, gait timing, and terrain data, which improves robustness on challenging terrains. Validated on the Unitree Go1, this framework demonstrates capability in navigating stairs, showcasing suitability for complex environments.

Figure 1: Unitree Go1 climbing stairs around the WPI campus (multiple frames combined)

Acknowledgments

The BiQu team would like to thank our project advisors, Professor Agheli, Professor Xiao, and Professor Li whose guidance and support were invaluable throughout the project's development. Additionally, we would like to thank our sponsors, Unitree Robotics and MathAltitude School of Mathematics, whose contributions and belief in our vision made our project possible. Lastly, we would also like to thank Ethan Chandler, Akshay Jaitly, and the rest of the 2024 BiQu team for their mentorship and dedication.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Legged robots offer significant advantages over traditional wheeled robots for their inherent adaptability and agility. They are capable of navigating complex environments that would otherwise be impossible for robots with wheels or treads to overcome. Stair climbing is a fundamental requirement for robots operating in human-centric environments, yet it remains a challenging problem due to the need for precise coordination and adaptability. As underactuated systems, legged robots require more complex methods to effectively manage their movements and maintain stability [8].

In recent years, companies like Boston Dynamics, ANYbotics, and Unitree have made significant strides in advancing the capabilities of legged robots, including complex tasks like stair climbing (Figure 1.1). Our project introduces a unique pipeline that leverages novel methods in perception and control, enabling enhanced robustness and computational efficiency.

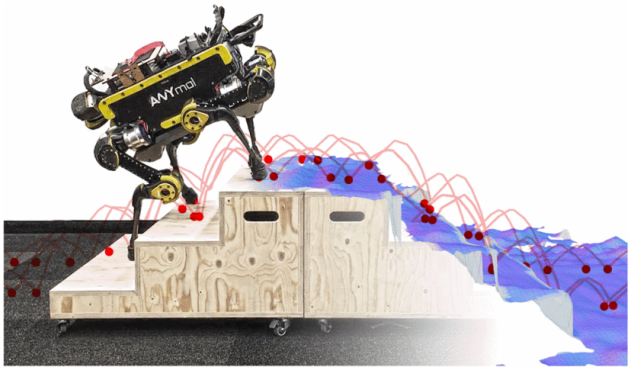


Figure 1.1: Example of a quadruped robot from ANYbotics climbing stairs [3]

1.2 Problem Statement

Our project aimed to achieve live climbing of full-sized human stairs on a Unitree Go1 quadruped robot (Figure 1.2). By live climbing, we mean that the robot has no prior knowledge of the terrain; trajectories and terrain maps are generated online. The full-sized stairs are 7 inches tall (about 30% of the Unitree Go1's leg length) and the steps will be parallel with the ground.

We chose the height of 7 inches because it is the height of a standard, human-sized stair. Also notably, the Unitree Go1's built-in "stair climbing" mode struggles with elevation changes greater than 6 inches [9]. The built-in controller simply walks with a gait where the legs are lifted higher than normal. However, achieving stairs with heights of 7 inches and taller with the Unitree Go1 is feasible with improved perception and trajectory planning.

The previous 2024 BiQu team achieved steps of around 2 inches tall onto large foam mats. Our project aims to achieve live climbing of full-sized 7-inch stairs in a robust and computationally efficient manner.

Figure 1.2: Unitree Go1 robot with attached Livox Mid-360, Intel NUC, and onboard battery

Chapter 2

Background

2.1 Perception

Effective perceptive locomotion is essential for quadrupedal robots to navigate unpredictable and dynamic terrains, where precise foothold placement directly impacts stability and efficiency. Perceptive locomotion integrates sensory information to handle the complexities in these environments. Current perceptive locomotion strategies include static gaits [10] or learning-based controllers [11]. Although due to incomplete perceptive data and dynamic constraints, there are significant challenges in efficiently optimizing footholds over rough terrain [6]. Although some strategies, [12], hierarchically calculate footholds before torso positioning, [6] uses a non-linear model predictive controller that optimizes over all degrees of freedom. [6] streamlines foothold optimization by simplifying terrain into primitive geometry and utilizing a signed-distance-field for obstacle avoidance. Additionally, [13] and [14] use deep-learning techniques to optimize the vision pipeline and foothold optimization, respectively.

2.1.1 Robot-Centric Elevation Mapping

Robot-centric elevation mapping focuses on dynamically maintaining a 2.5D elevation map relative to the robot's frame [6] (Figure 2.1). This approach minimizes reliance on precise global localization by leveraging onboard sensors, such as depth cameras or LiDAR, to directly represent terrain geometry relative to the robot. The current system uses odometry informed primarily by proprioceptive sensors—such as IMUs, joint encoders, and force sensors—to estimate the robot's pose. Then, the uncertainty of the pose of the robot and the sensor readings are incorporated into the resulting elevation map. While this reduces

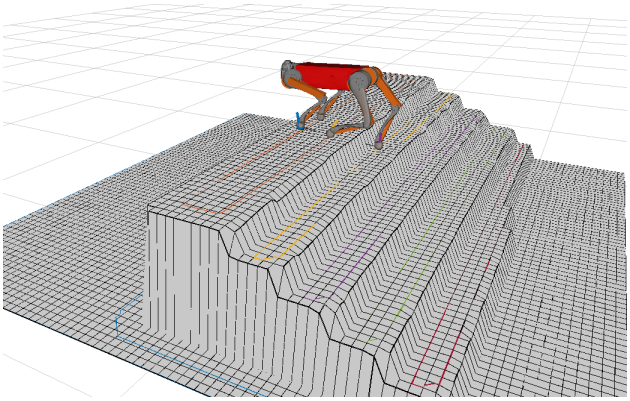


Figure 2.1: Example simulated environment using ANYbotics Elevation Mapping [4, 5] and ETH Zürich Convex Plane Decomposition [6]

dependency on external localization systems, future implementations could integrate SLAM for enhanced odometry accuracy.

The perception pipeline includes several key stages:

- **Filtering and Inpainting:** Noise reduction and gap filling in the elevation data.
- **Steppability Classification:** Classifying terrain regions based on slope and local geometry as steppable or non-steppable.
- **Plane Segmentation:** Extracting convex plane regions that can be used as feasible footholds.
- **Signed Distance Field (SDF):** Precomputing distances from robot parts to nearby terrain for collision avoidance.

The Elevation Mapping package by ANYbotics enables elevation mapping and convex plane decomposition using a variety of sensors [4, 5]. We utilize this module to enable perceptive locomotion on the Unitree Go1.

2.1.1.1 Map Cell Updates

In the elevation mapping process, the cell heights and individual cell variances are updated for each new sensor reading with the following linear kalman filter:

$$h = \frac{\sigma_p^2 h + \sigma_m^2 p_z}{\sigma_m^2 + \sigma_p^2} \tag{2.1}$$

$$\sigma_m^2 = \frac{\sigma_m^2 \sigma_p^2}{\sigma_m^2 + \sigma_p^2} \tag{2.2}$$

where h represents the estimated cell height, σ_m^2 is the elevation map cell variance, σ_p^2 is the sensor noise variance, and p_z is the measured height [4].

Then, ray casting is performed to account for dynamic obstacles by comparing the final height of rays cast by the sensor to the estimated elevation of that cell.

2.1.1.2 GPU Acceleration

In [4], the authors describe how this elevation mapping process can be accelerated with a GPU. The height update can be processed for each point at the same time instead of iteratively. Similarly, the ray casting step can also be processed for each ray simultaneously. Finally, they also keep this processed data in the GPU to run a traversability filter, which is important for foothold selection. Finally, the elevation map is sent back to the CPU and converted to a ROS message at a rate selected by the user.

[4] measures the performance of the elevation mapping node by measuring its point cloud processing time. They analyzed the processing time performance with the Jetson Xavier, both with and without GPU acceleration. They found that for a smaller point cloud, about 10% of the size a D435i would normally record, processing time was about 30 ms without GPU acceleration and about 4 ms with GPU acceleration. Using a point cloud of about the same size as a D435i would capture, processing time was about 25 ms with GPU acceleration and 175 ms without. The GPU acceleration seems to make a larger impact when processing larger point clouds. Although, using cropping and down-sampling filters could potentially reduce the need for GPU acceleration.

2.1.2 SLAM in Combination with Robot-Centric Elevation Mapping

Simultaneous localization and mapping (SLAM) can be used to improve the odometry estimates of autonomous robots. Using information from LiDAR sensors in conjunction with odometry can negate the cumulative drift that occurs when only using odometry. Information from the LiDAR generates a feature map, which can then be used to determine the location of the robot and correct drift from the state estimator [15]. A fast enough elevation mapping module may be able to overcome state estimation drift on its own [6], but some pipelines opt to utilize SLAM in combination with elevation mapping to overcome drift [16]. The setup shown in [16] utilizes LiDAR and depth camera sensors to collect information from its surroundings. Using both LiDAR and depth cameras allows the LiDAR to collect vast amounts of information useful for SLAM, while the depth camera can provide detailed information directed at key areas. There are many examples using both of these sensors, notably many setups with the Unitree Go2 [17, 18]. Since we have

access to both a LiDAR and depth camera, SLAM is a promising element that could be used to reduce the drift of state estimation. Additionally, [4] uses an Extended Kalman Filter that combines odometry data from the robot IMU and SLAM output, while using a time offset to ensure that no discrepancies occur.

There are many available 3D SLAM packages to choose from, one of the most popular is RTAB-map for its general purpose scope [19]. It provides many different features, including SLAM for the purpose of taking a 3D point cloud to improve odometry estimates. Despite the popularity of RTAB-map, it is shown to be slower than many similar packages such as FAST-LIO [20] or Google's Cartographer [21]. These two are more comparable in real-time performance, and are more commonly used in the context of robotics, where computational power is limited. In [16], they utilize a version of FAST-LIO, FAST-LIO2 [22], to achieve an overall perception pipeline refresh rate of about 20 Hz, which is also the refresh rate that we are targeting.

FAST-LIO2 [22] is a fast and efficient LiDAR-inertial odometry framework designed to improve accuracy and computational speed in SLAM tasks. Unlike methods that rely on extracting features like edges and planes from LiDAR data, FAST-LIO2 directly registers raw point cloud data to a global map. This direct method improves accuracy, especially in environments that lack distinct features, and eliminates the need for hand-tuning parameters. To manage the large amount of point cloud data efficiently, FAST-LIO2 introduces the ikd-Tree, an incremental k-d tree structure that allows for fast insertion, deletion, and dynamic rebalancing of map points. The module uses a tightly-coupled iterated Kalman filter to fuse IMU and LiDAR data, compensating for motion distortion and maintaining accurate state estimation. By combining direct point registration with ikd-Tree-based mapping, FAST-LIO2 achieves high-frequency updates (up to 100 Hz) with low computational load, making it best for real-time applications like ours. For these reasons, we decided to utilize FAST-LIO2 for SLAM.

2.1.3 Perceptive Locomotion Implementation on the Go1

The previous 2024 BiQu team implemented perceptive locomotion on the Unitree Go1 by integrating mapping, planning, and control modules in both simulation and hardware. Using ROS Noetic and Gazebo, they employed Qiayuan Liao's legged control framework [23] alongside the ANYbotics Elevation Mapping package. The pipeline facilitated real-time elevation mapping, convex plane segmentation, and foothold optimization, both in simulation and on the subsequently acquired Go1 robot. A RealSense D435i depth camera was used to capture terrain data, with the convex planes informing foothold planning. The team identified challenges such as map drift due to a 17 Hz update rate, which was below the recommended 20 Hz, and limited camera perspectives in complex environments.

On the physical robot, the perceptive stack was deployed with the D435i and a high-performance Intel NUC for real-time processing. Liao's legged framework was modified to incorporate filtered foot force sensor readings, enhancing locomotion stability. However, the camera's limited field-of-view constrained mapping of elevated surfaces.

This previous work established a framework for active perception and robust locomotion with the Go1, forming a solid foundation for future improvements. This year's work focuses on improving terrain representation by reducing odometry drift with FAST-LIO2, incorporating additional sensors, and improving elevation mapping fusion methods.

2.2 Control

Existing control systems for quadrupedal systems largely operate using multiple control layers formed together in a pipeline. Current control pipelines sometimes begin with an offline trajectory planner that solves for an optimal series of contact forces over an entire motion [24]. In [25], the pipeline connects an offline trajectory optimization program to a whole body controller (WBC) directly. In [24], the generated trajectory is connected through a model predictive controller (MPC) that continually combines the proposed trajectory with execution over a limited time horizon. Some pipelines omit offline planning in favor of a model predictive controller into a whole body controller based on data about the terrain fed into the model predictive controller [3, 26]. Others still use mixed integer optimization to solve the trajectory using binary representations of the foot contacts [27, 28].

2.2.1 Trajectory Optimization

Trajectory optimization is essential for controlling highly dynamic systems with simple objective functions [8]. Trajectory optimization methods are used to find the best trajectory choice, usually by selecting inputs to the system (i.e., controls) as functions of time [29]. The trajectory describes the system's path, including state and control as functions of time. It involves formulating an optimization problem where the objective is to minimize a cost function subject to some constraints, such as the system's dynamic equations and kinematic feasibility. In the context of legged robotics, trajectory optimization enables the generation of efficient and feasible paths for the robot to follow that adhere to various constraints such as actuator limits, robot dynamics, and terrain collision.

Model predictive control (MPC) is a common method for turning trajectory optimization into

a feedback policy [8]. The steps are straightforward:

- Measure the current state
- Optimize a trajectory from the current state to some goal state
- Execute the first control input from the optimized trajectory
- Let the system dynamics evolve for one iteration and repeat

A critical part of all trajectory optimization problems is *transcription*, which is the process of turning the original trajectory optimization problem into an easier-to-solve constrained parameter optimization problem [29]. In the original trajectory optimization problem, decision variables are vector functions that are difficult to optimize over, but in the constrained parameter optimization problem, the decision variables are all real numbers that are easier to optimize over. Additionally, the original trajectory optimization problem can contain differential and integral terms which makes it harder to solve than a constrained parameter optimization problem.

Transcription methods can largely be categorized into two groups: direct methods and indirect methods [29]. **Direct methods** discretize the trajectory optimization and convert it into a *non-linear program*, a specially constrained parameter optimization problem that has non-linear terms in the objective or constraint functions. These non-linear programs can then be optimized using a non-linear program solver, common choices include the sequential-quadratic programming (SQP) solver SNOPT [30] or the interior-point solver IPOPT [31] (See Section 2.2.4 for more details). **Indirect methods** instead start by formulating the mathematical conditions for optimality analytically. After this, they discretize the conditions and solve them numerically. So in short, direct methods discretize and then optimize, whereas indirect methods optimize and then discretize [29]. Indirect methods are more accurate and have less error when compared to direct methods; however, they also have some drawbacks. Indirect methods have a small region of convergence, which requires better initialization, and setting up the problem analytically can be challenging [32]. For these reasons, direct methods are generally better suited for the control of legged robots.

Transcription methods can also be characterized as shooting methods or collocation methods. **Shooting methods** are fundamentally based on simulation. Discretization of the trajectory optimization problem is done through explicit integration methods. The next state of the system is calculated based on the previous state of the system and its dynamic equations. **Collocation methods**, on the other hand, are fundamentally based on function approximations. Discretization of the optimization problem is handled with implicit integration methods. One approach to collocation is to split the trajectory into piecewise

polynomial functions or splines [8, 29]. Another approach is to use a single high-order polynomial to approximate the trajectory as a whole, this is called **pseudospectral collocation** (or *orthogonal collocation*) [8, 29]. Pseudospectral collocation solves globally over the entire trajectory, over both the state and controls simultaneously. While this makes implementation more difficult, it improves stability and accuracy when compared to other methods [33].

2.2.2 Footstep Planners

State-of-the-art trajectory optimization libraries typically optimize over fixed footstep contact sequences [34]. The purpose of a footstep planner is to find a list of feasible footstep locations that create a path through a constrained environment [27].

One method is to use a geometrical approach, as implemented by [3]. Firstly, the current position and orientation of the robot are determined by its current foot positions. An interpolation between the current pose and the goal pose with a default step size creates a path of stances to follow. A predetermined leg sequence is used to move the legs one by one to the positions of the next stance (Figure 2.2). The major benefits of this approach are that it has fast computation and easy implementation. However, footstep locations are only locally optimized.

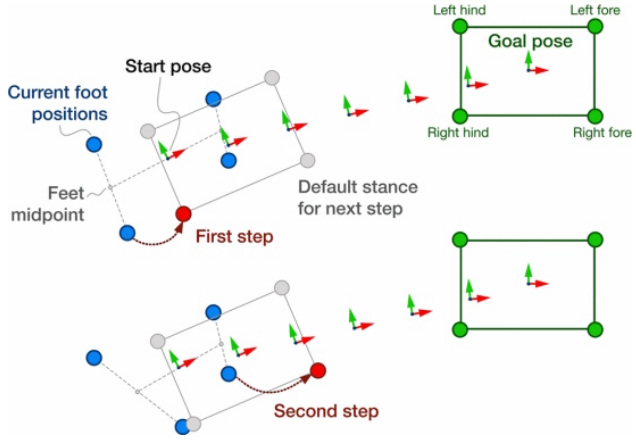


Figure 2.2: Illustration of geometric footstep planner from [3]

Another approach to footstep planning is to use **mixed-integer programming** (MIP), which is an optimization technique used to solve problems that involve both continuous and discrete variables. In [27], the problem is formulated as a single mixed-integer convex optimization, specifically a mixed-integer quadratically constrained quadratic problem (MIQCQP). This allows solving for a global optimal solution. Integer variables are used to absorb non-convex constraints. For example, solving for the orientation of

footstep placements depends on trigonometric functions (i.e., sine and cosine). So, piecewise linear approximations for sine and cosine are used instead, with integer variables to choose the appropriate approximation. Additionally, non-convex obstacle avoidance is achieved by enumerating a set of convex obstacle-free configuration space regions and using integer variables to assign footsteps to those regions. The use of integer constraints can make formulating the problem more complicated, but once formulated, a variety of mixed-integer program solvers can be used, such as Gurobi [35], GLPK [36], or lp_solve [37]. All of which provide globally optimal solutions. The mixed-integer approach from [27] was capable of planning short sequences of a few steps in under 1 second, and longer sequences of 10-30 steps in tens of seconds to minutes. However, some methods, such as $\ell 1$-norm minimization, have been shown to provide substantial speedups [38].

One approach that was recently published by Carnegie Mellon University is Diffusion-Inspired Annealing for Legged Model Predictive Control, or DIAL-MPC for short [39]. DIAL-MPC uses a novel diffusion-inspired annealing framework to optimize trajectories in real-time. It requires no prior training and achieves global convergence at 50 Hz. A major advantage of DIAL-MPC is its capability to generate highly dynamic motions, such as clambering onto a box in simulation. However, one drawback is that it struggles with long time horizons. We strongly considered using DIAL-MPC as a footstep generator, but ultimately decided against it because we anticipated it to be difficult to integrate into our existing pipeline. In the future, integrating DIAL-MPC or a similar diffusion-based implementation might be an interesting avenue to explore.

2.2.3 Trajectory Optimizer for Walking Robots (TOWR)

In the end, the method we decided to use for our footstep planner is based around the library TOWR, which stands for Trajectory Optimizer for Walking Robots. The TOWR library uses a novel phase-based parametrization of foot motion and forces, allowing the problem to be formulated as a single trajectory optimization problem [7]. The gait is defined by continuous phase durations, which avoids integer programming, allowing an NLP solver to optimize over the gait sequence. For a 1-second horizon, TOWR is able to generate a 4-footstep motion plan in 100 ms [40]. The TOWR library also comes neatly wrapped in a ROS package which makes development and integration easier.

Another benefit of using TOWR is that it generates a trajectory using a simplified dynamic model, which can be used as an initial guess for the MPC trajectory optimizer. The TOWR library uses a single rigid-body dynamic model which assumes that the legs have negligible inertia so they don't significantly contribute to the dynamics (i.e., light legs). Other trajectory solvers that use a more complex dynamic

model can use the simplified trajectory as an initial guess, speeding up computations and allowing for a faster control loop.

2.2.4 Nonlinear Programming Solvers

The TOWR framework makes use of IFOPT [41], an Eigen-based C++ interface to nonlinear programming solvers, namely SNOPT [30] and IPOPT [31]. SNOPT solves problems using sequential quadratic programming (SQP), a method that uses a series of quadratic programming subproblems, each of which approximates the original nonlinear problem. On the other hand, IPOPT implements an interior point method that solves optimization problems by navigating through the interior of the feasible region and employing a barrier term in the objective function to adhere to the problem constraints.

SNOPT excels in handling large-scale problems with "many thousands of constraints and variables" and can handle infeasible constraints, which allow it to "terminate without computing the nonlinear functions" [30, p. 979, 981]. However, SNOPT is designed for problems with a "moderate number of degrees of freedom," which may limit its applicability to problems with a high number of degrees of freedom [30, p. 979]. IPOPT promises "efficiency and robustness" and "global convergence," making it a reliable choice for problems requiring a strong convergence guarantee [31, p. 26, 35]. It also has the added benefit of being open source, which makes it easily accessible and modifiable [31, p. 47]. However, IPOPT may encounter issues if the problem is infeasible, and its sensitivity to scaling can also pose challenges [31, p. 32, 45].

IPOPT can also be run with several different linear solvers. The most basic of which is Mumps, which is the default setting of IFOPT. Within the IFOPT code, it says that "Mumps is the default because it comes with the precompiled Ubuntu binaries. However, the Coin-HSL solvers can be significantly faster and are free for academic purposes." This refers to the Harwell Subroutine Library (HSL) linear solvers. There are several different linear solvers, but we chose to examine HSL-MA97 since it is recommended for "general use," as well as HSL-MA57 since it is recommended for "small or highly sparse problems" [42].

2.2.5 Contact Estimation

One of the major limitations of the robot, as noted by the previous BiQu team, is issues of accurately representing the terrain and understanding its own location within that terrain. While the perception team is addressing drift and quicker updates for state estimation, the contact estimation can also be improved. The contact estimation informs the trajectory optimizer and whole body controller (WBC) of the current contacts of the robot's legs, allowing for the application, or lack thereof, of contact forces.

Currently, the contact estimation is done by the force sensors. The force sensors lie beneath the footpads, a spherical surface for transferring force from the floor to the sensor as well as providing stability at the contact point. These force sensors can be rife with problems such as large error bounds, drift, and the use of a threshold for measuring contact that does not account for historical data. Indeed, Unitree notes "the drift of this sensor is serious, and it needs to calibrate the zero point intermittently" during the swing phase [43].

These evident issues with the use of force sensors for contact estimation have inspired alternative methods of contact estimation and how to compensate for a contact mismatch, becoming a source of considerable research. [44] uses the forward pass of a Hidden Markov Model (HMM) to detect contact using a sampling-based method such as Monte-Carlo. Unfortunately, the lack of a guaranteed analytical solution as well as the necessity to measure joint accelerations, which cannot be done in our case without using finite differentiation that introduces delay and accuracy issues, remain major drawbacks. [45] uses artificial intelligence to learn the appropriate probability threshold for contact given the applied force. This solution, although effective for the given task, is unfortunately not very generalizable to different terrain types or gaits, something that will likely be very necessary when climbing stairs. [46] uses a Kernel Density Estimator to probabilistically model the contact surface's coefficient of friction and computes the contact probability accordingly from the IMU readings. Although the method appears effective when maneuvering across surfaces with varying or especially low coefficients of friction, it is slower than other methods, running at 500 Hz, and does not seem to add much for our task accordingly. [47] uses a generalized momentum (GM) observer model to estimate contact. Probability estimates of contact are calculated according to MPC-generated gait timing, foot height calculated from forward kinematics, and estimated terrain height from either vision or historical footsteps, and estimated force from the GM observer. This method was ultimately selected by the team because of its quick reaction speed, quick computation time, ability to adjust expected footstep height according to historical data, ability to assist in SLAM calculations, and ability to add measurements from the force sensor to create a more holistic calculation. The handling of these differences is typically done directly by the MPC, simply accounting for the lack of force for that leg in the trajectory optimization, but sometimes frameworks such as [6] introduce an impedance controller for early touchdown and pause the motion for the rest of the contacts in the event of a late touchdown.

2.2.6 Odometry

Odometry is highly important for any robotics task, but especially so for legged robots and even more so for a task such as stair climbing. For legged robots, odometry issues can severely impact the robot's

contact plan, leading to slippage, large changes in body height or orientation, and even catastrophic failure, such as falls or flips. The results of these odometry issues are more severe for stair climbing, where a slight change in leg position can make the difference between making contact with a stair or the stair below it, an overall change of around 7 in for a full-sized stair. A 7 in gap can mean that the leg isn't making contact for a significant portion of the trajectory optimizer's plan, easily leading to tips with the robot's already altered orientation and smaller support polygon. To minimize these odometry error effects, we implement a simplified combination of IMU readings and leg kinematics shown in [48]. Moreover, we implement vision data using SLAM in Section 2.1.2 to get a more external understanding of the robot's position from the surrounding elements. The odometry relies on an Extended Kalman Filter (EKF) to update and correct the robot's current state at each discrete time step. The extended Kalman filter equations are written as

$$\hat{x}_{k|k-1} = f(\hat{x}_{k-1|k-1}, u_{k-1}) \tag{2.3}$$

$$P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q_{k-1} \tag{2.4}$$

where $\hat{x}_{k|k-1}$ is the predicted state at time k from some update equation $f(\hat{x}_{k-1|k-1}, u_{k-1})$ and $P_{k|k-1}$ is the predicted covariance, F_k is the linearized error dynamics matrix, and Q_{k-1} is the process noise covariance matrix. Because we have the IMU measurement at every time, we do not rely on the foot forces from the previous timestep as part of our state estimation. Instead, we define

$$\hat{x}_{k-1|k-1} = \begin{bmatrix} r, v, p_1, p_2, ...p_N \end{bmatrix}^T \tag{2.5}$$

where r is the position of the robot in the global frame, v is the velocity of the robot in the global frame, and p_l is the position of the lth foot in the global frame, with N being the number of legs. We can therefore define

$$r_{k|k-1} = r_{k-1|k-1} + \Delta t v_{k-1|k-1} + \frac{\Delta t^2}{2} a_{k-1} \tag{2.6}$$

where a_{k-1} is the 3×1 vector of acceleration as measured by the IMU at time $k-1$. Moreover, the body velocity can therefore be understood as

$$v_{k|k-1} = v_{k-1|k-1} + \Delta t a_{k-1} \tag{2.7}$$

We can assume that the legs are stationary for the prediction section of the EKF, so we can therefore write $f(\hat{x}_{k-1|k-1}, u_{k-1})$ as

$$\hat{x}_{k|k-1} = A\hat{x}_{k-1|k-1} + Ba_{k-1} \tag{2.8}$$

$$A = \begin{bmatrix} I_{3\times3} & \Delta t I_{3\times3} & \dots & 0_{3\times3} \\ 0_{3\times3} & I_{3\times3} & \dots & 0_{3\times3} \\ \vdots & \ddots & \ddots & \vdots \\ 0_{3\times3} & \dots & \dots & I_{3\times3} \end{bmatrix} B = \begin{bmatrix} \frac{\Delta t^2}{2} I_{3\times3} \\ \Delta t I_{3\times3} \\ 0_{3\times3} \\ \vdots \\ 0_{3\times3} \end{bmatrix} \tag{2.9}$$

The EKF update equations can then be used to correct the estimations of the trunk from the IMU integration using leg kinematics and to incorporate end-effector movement as well. The correction equations can be written as

$$\tilde{y}_k = z_k - h(\hat{x}_{k|k-1}) \tag{2.10}$$

$$S_k = H_k P_{k|k-1} H_k^T + R_k \tag{2.11}$$

$$K_k = P_{k|k-1} H_k^T S^{-1} \tag{2.12}$$

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k \tilde{y} \tag{2.13}$$

$$P_{k|k} = (I - K_k H_k) P_{k|k-1} \tag{2.14}$$

where z_k is the sensor measurement at time k, S_k is the innovation covariance H_k is the measurement Jacobian, R_k is the measurement covariance, and K_k is the Kalman gain. We therefore introduce a measurement vector

$$z_k = \begin{bmatrix} s_1 & s_2 & \dots & s_N \end{bmatrix} \tag{2.15}$$

where

$$s_l = C_k \mathrm{lkin}_l(q) \tag{2.16}$$

where lkin_l represents the forward kinematics in the robot body frame of leg l from leg joint variables q as multiplied by C_k, which transforms from the body frame to the global frame using the robot trunk's orientation. We can therefore define the transform $h(\hat{x}_{k|k-1})$ as

$$\tilde{y}_k = z_k - D\hat{x}_{k|k-1} \tag{2.17}$$

$$D = \begin{bmatrix} I_{3\times3} & -I_{3\times3} & 0_{3\times3} & \dots & 0_{3\times3} \\ I_{3\times3} & 0_{3\times3} & -I_{3\times3} & \dots & 0_{3\times3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ I_{3\times3} & \dots & \dots & \dots & -I_{3\times3} \end{bmatrix} \tag{2.18}$$

The covariance estimate can be updated using the same matrix, redefining

$$H_k = D \tag{2.19}$$

It can therefore be observed that

$$\tilde{y}_k = \begin{bmatrix} s_{1,k} - s_{1,k-1} \\ s_{2,k} - s_{2,k-1} \\ \vdots \\ s_{N,k} - s_{N,k-1} \end{bmatrix} = \begin{bmatrix} \Delta s_{1,k} \\ \Delta s_{2,k} \\ \vdots \\ \Delta s_{N,k} \end{bmatrix} \tag{2.20}$$

and

$$K_k \tilde{y} \propto \begin{bmatrix} \sum_{l=1}^{N} \Delta s_{l,k} \\ 0_{3\times3} \\ -\Delta s_{1,k} \\ -\Delta s_{2,k} \\ \vdots \\ -\Delta s_{N,k} \end{bmatrix} \tag{2.21}$$

Following the update equations, this updates the trunk position at each time according to the change in the leg kinematics. Moreover, we change the leg kinematic sensor variance to a large number during the swing phase to prevent changes in foot position during swing phase from contributing to the change in the trunk position. The end result is that if the robot is predicted by the IMU to be lower than the leg kinematics allow, the leg kinematics push the trunk position upwards over time to accommodate. The EKF adjusts the gain on a running basis, leading to an adaptive trust according to historical measurements.

Chapter 3

Design and Implementation

3.1 Project Objectives

To achieve our goal, as outlined in Section 1.2, of live climbing full-sized 7-inch stairs in a robust and computationally efficient manner, our team set out to fulfill the objectives outlined below. We organized these objectives into two main areas of concern, **perception** and **control**, and delegated sub-teams to handle each.

The perception team focused on enhancing the sensors and hardware to improve the refresh rate of the perception module, while also exploring alternative strategies to optimize its performance. The objectives of the perception team were to:

- Integrate elevation mapping using additional sensors and compute
- Improve odometry using LiDAR
- Expose planar confidence data for MPC integration

The control team focused on integrating new elements to the control pipeline to handle live terrain interactions. The objectives of the control team were to:

- Create a dynamic footstep planner for live terrain response
- Develop contact estimation for improved terrain interaction

3.2 System Pipeline

The system pipeline proposed in this paper is shown in Figure 3.1. Note that the dynamic footstep planner, FPOWR, has not yet been integrated into the full system pipeline. This is because it was designed as a standalone library to enable the use of custom trajectory optimization frameworks in an MPC context (See Section 6.1 for more details).

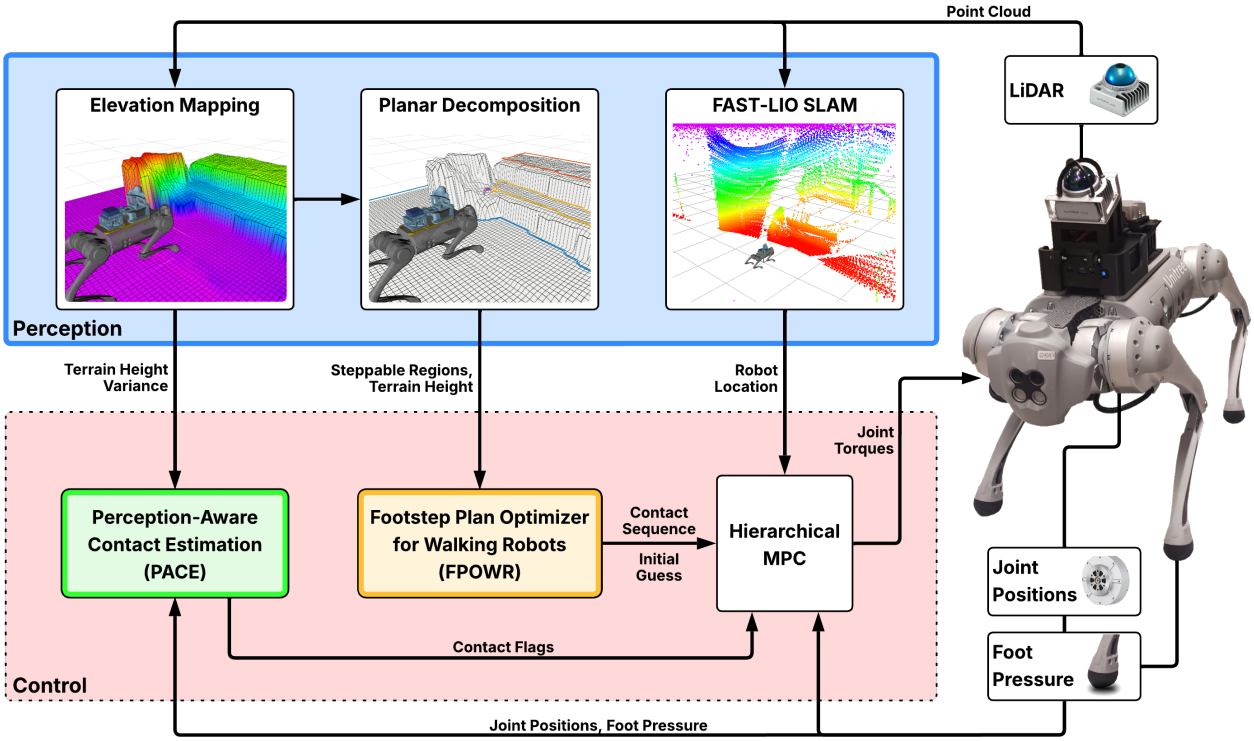


Figure 3.1: System Pipeline

3.3 Perception

3.3.1 Hardware Details

The previous 2024 BiQu team utilized the Intel RealSense D435i stereo depth camera but encountered limitations, including restricted field of view, environmental noise, and map drift. Adding a LiDAR sensor addresses these issues by offering broader coverage and enhancing the generated maps for downstream pipeline steps. Additionally, the LiDAR sensor contributes to more accurate odometry measurements within the perception pipeline. To improve the accuracy and refresh rate of the robot-centric elevation mapping pipeline, we incorporated additional sensors and compute resources on the Go1 robot.

Initially, we investigated methods to enhance the scope and detail of the elevation mapping pipeline using multiple sensors. Based on a survey of elevation mapping protocols and their integration strategies, we selected the Livox Mid-360 LiDAR for its high refresh rate and extensive field of view. One of the largest problems faced by the previous BiQu team was drift of the robot with respect to the map. The Mid-360's 360° horizontal and 59° vertical coverage made it particularly suitable for running SLAM. As described in Section 2.1.2, using SLAM combats drift in odometry. After purchasing the sensor and integrating its SDK, we experimented with various mounting configurations on the robot and in simulation to optimize coverage while ensuring minimal obstruction by the robot itself.

To develop our perception solution, we focused primarily on two sensors: the Intel RealSense D435i and the Livox Mid-360, a depth camera and a LiDAR, respectively. Both devices produce point clouds, but use different sensing methods and offer different strengths. The D435i captures a narrow but high-resolution view in the direction it's facing, with a high frame rate. The Livox Mid-360, in contrast, offers a panoramic 360° horizontal field of view at a lower refresh rate, better suited for global spatial awareness. The table below highlights the key differences:

Specification	**Livox Mid-360**	**Intel RealSense D435i**
Field of View (H × V)	360° × 59°	87° × 58°
Effective Range	Up to 70 m (at 80% reflectivity)	Up to 10 m
Sensing Method	Non-repetitive scanning LiDAR	Active infrared stereo vision
Data Rate	200,000 points/sec	Up to 1280 × 720 depth frames @ 30–90 FPS
Update Rate	10 Hz (typical)	Up to 90 Hz

Table 3.1: Comparison of Livox Mid-360[1] and Intel RealSense D435i[2] specifications

In addition, we tried two separate compute solutions to run the perception pipeline. We use the Intel NUC i7 11th generation mounted on top of the robot to run the pipeline with a powerful CPU. The other solution we tried was the NVIDIA Jetson Orin NX, which features a powerful GPU. Following the implementation described in [4], we adopted a CUDA-accelerated elevation mapping module to parallelize ray casting operations. Implementing GPU acceleration enables more frequent updates to the elevation map, reducing accumulated variance and ensuring better alignment with true terrain heights. Although, the powerful CPU that the NUC features is still extremely valuable, as it runs FAST-LIO, as well as other modules of the pipeline, at increased speed compared to the Jetson.

3.3.2 Wireless Hardware Upgrades

By enabling untethered power for the external compute unit and sensors, we expanded the robot's ability to perform perceptive locomotion across a greater variety of test scenarios and for longer durations.

To achieve this, we connected a 24V to 19V voltage regulator to the Go1's external power port, supplying power to both the NUC and LiDAR.

However, during testing, we observed that the power supply frequently experienced brownouts due to the high power draw from the motors. To address this issue, we replaced the power source with an externally mounted 24V 5Ah LiPo battery, which provides power to the external compute unit and sensors for a duration comparable to the onboard Go1 battery. These upgrades allowed us to test the perceptive locomotion pipeline in new environments beyond the lab.

3.3.3 Hardware Mounting Solutions

Throughout the process of our project, we focused primarily on two different sensing solutions using the LiDAR. Our initial design used an adjustable mount as shown in Figure 3.2. This design would allow us to adjust the angle at which the LiDAR is stationed without any major hardware adjustments. This could allow for tuning as we conducted experiments.

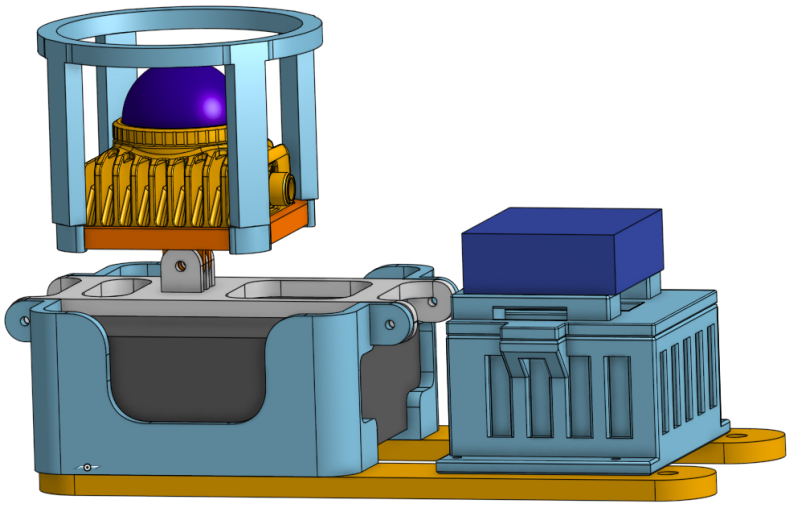


Figure 3.2: Adjustable LiDAR mount solution

Our other configuration uses just the LiDAR in a stable mount, at a slant of 22 degrees. This allows the LiDAR to see more terrain directly in front of the robot. This configuration is shown in Figure 3.3. This design features a more stable solution than the first design. Since the mount does not feature any joints that are likely to loosen over time, it is more prepared to endure vibrations and the general shaking that occurs when a quadruped is in motion. For these reasons, we ultimately decided to go with the stable mount

solution which can be seen in Figure 3.4.

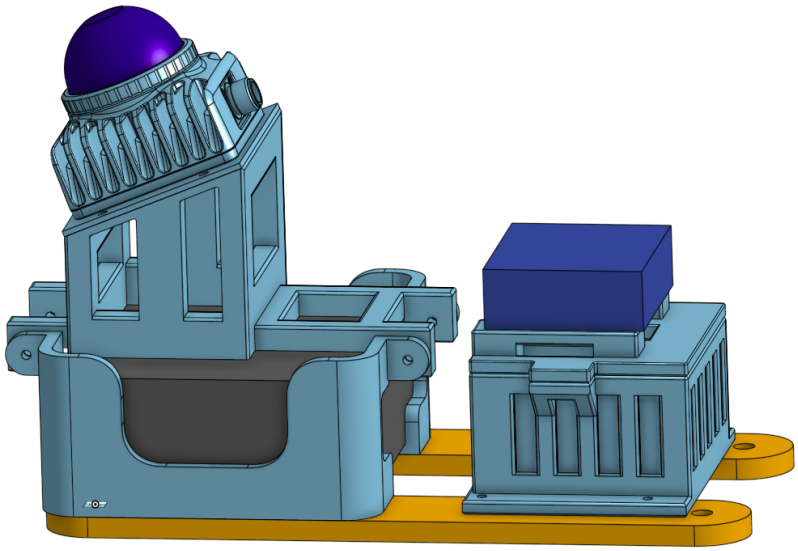


Figure 3.3: Stable LiDAR solution at slant of 22 degrees

3.3.4 Elevation Mapping Fusion and Sensor Integration

To integrate the LiDAR point cloud into the elevation mapping module, we implemented a cropping filter that only keeps relevant points in-front of the robot. Following this, we conducted extensive simulation trials to fine-tune the elevation mapping fusion parameters. The fusion process uses a Bayesian update framework, as described in 2.1.1.1.

3.3.5 Improving Odometry Using LiDAR

We implemented LiDAR-Inertial Odometry (LIO) before elevation mapping to reduce drift in state estimation caused by inaccuracies in odometry. As described in Section 2.1.2, incorporating additional state estimation methods, such as LiDAR and depth camera fusion, could improve the accuracy of the estimated robot state before it is fed into the elevation mapping node. By minimizing odometry drift, the refresh rate of the elevation map may become less critical to the overall performance of the perception pipeline. We implemented a pipeline as shown in Figure 3.5. Furthermore, we utilized FAST-LIO2 [22], an accelerated LiDAR-Inertial Odometry (LIO) algorithm with integration for Livox sensors. This package consumes LiDAR and IMU data to estimate motion and generate new odometry. This odometry output is then fused with additional odometry sources from joint encoders and contact estimation in an Extended

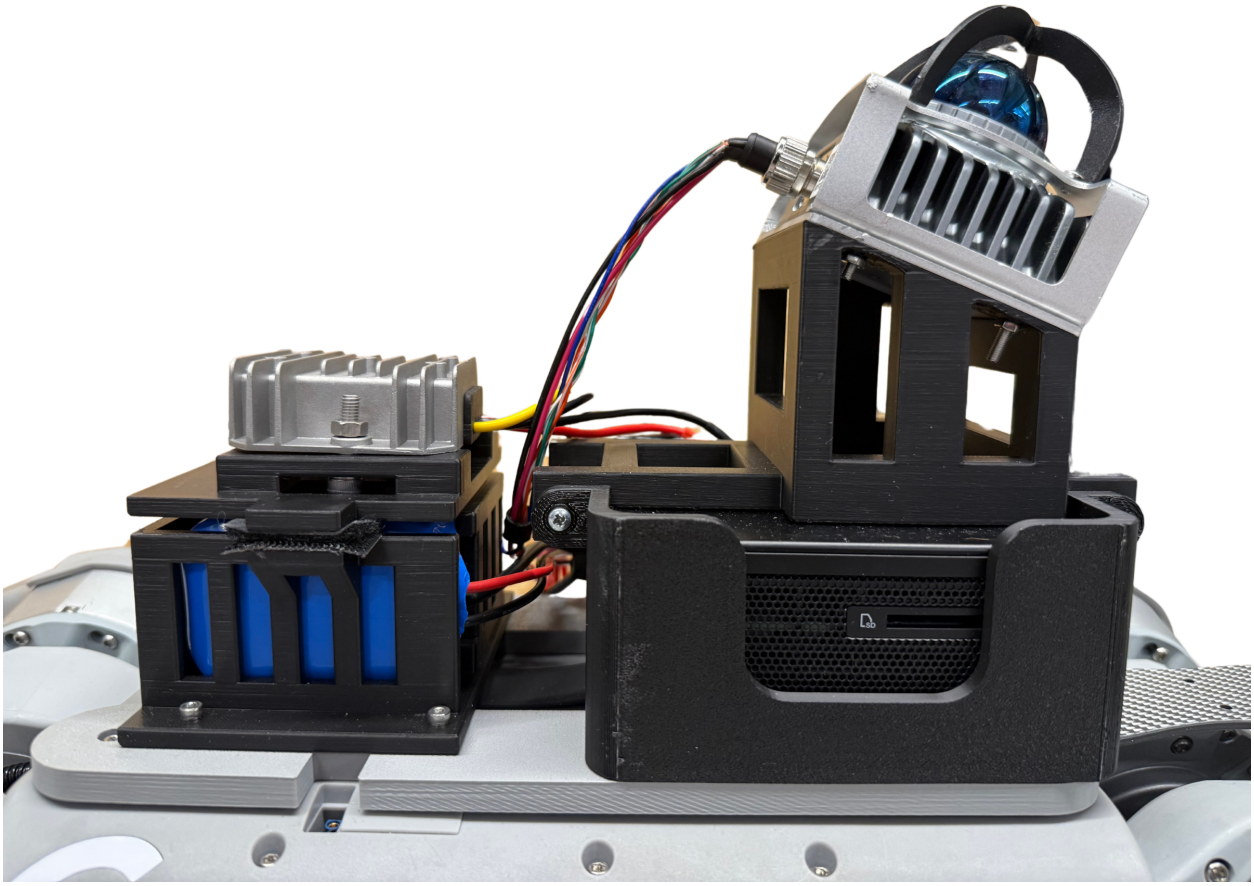


Figure 3.4: Final LiDAR mount using stable 22 degree slant

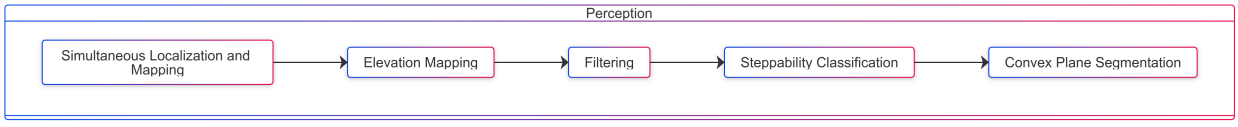


Figure 3.5: Proposed Updated Perception Pipeline

Kalman Filter, ultimately improving the state estimate used for elevation mapping. This extended Kalman Filter integration is similar to the method described by ETH Zürich [4].

To integrate the odometry from FAST-LIO2 into our robot's state estimation, we incorporate it into the Legged Control [23] Kalman filter-based estimation framework, which fuses multiple sensor modalities, including joint encoders, contact detection, and IMU measurements, to produce a robust state estimate. Specifically, the subscribed FAST-LIO2 odometry updates the position and orientation of the base frame within the Kalman filter. The measured position from FAST-LIO2 is treated as an observation in the Kalman filter update step, where the innovation term helps correct drift in state estimation. Additionally, FAST-LIO2 orientation updates are fused using a separate small-angle correction to improve orientation estimation. By integrating this odometry stream with kinematic state estimates from the legged system, we achieve a more drift-resilient and accurate localization solution, which is ultimately used to improve elevation mapping and overall terrain perception.

3.3.6 Exposed Region Data for MPC Integration

To enhance the robot's ability to interact with terrain through model predictive control (MPC), we exposed additional information from the perception module's planar decomposition. This includes height confidence values that quantify the certainty of surface estimation.

The elevation map generated by sensor readings contains many numerical values about each cell in the map, such as elevation, h, representing the estimated height of the cell, as well as variance, σ_m^2, referencing the uncertainty of the height at that cell. This uncertainty value, σ_m^2, is updated based on its previous value, as well as an estimate for the variance from a sensor noise model (See Section 3.3.4). We devised a method to generate a value of 'region variance', σ_r^2, which takes information about the elevations and individual variances of each cell within a defined region and returns a new composite variance for that region.

One proposed implementation is as follows. $\sigma_{m_{\text{avg}}}^2$ is calculated by taking the average value of the variances for each cell in the region. Then the statistical variance of the heights, σ_h^2, is calculated by summing the square differences between each elevation and the mean elevation of the region, and dividing by the number of cells in the region.

A typical σ_m^2 value is around 0.001, with an max value capped at 0.05. We convert this to a multiplier, m, to scale up our height variance σ_h^2 based on the uncertainty we have within each measurement.

$$\sigma^2_{m_{\text{avg}}} = \frac{1}{N}\sum_{i=1}^{N}(\sigma^2_{m_i})$$
$$\sigma^2_h = \frac{1}{N}\sum_{i=1}^{N}(h_i - \bar{h})^2$$
$$m = (1 + 100 * \sigma^2_{m_{\text{avg}}}) \in [1, 6]$$
$$\sigma^2_r = m * \sigma^2_h \tag{3.1}$$

This new region variance σ^2_r is exposed within the contact estimation pipeline, to provide a more data-driven approximation for how uneven the terrain is in the region where the robot is stepping.

This value, derived from preprocessing statistics within the elevation mapping pipeline, dynamically influences footstep planning in regions of lower confidence, thereby improving robustness in challenging environments. The approach mitigates errors resulting from uncertain terrain data, enhancing the robot's stability and performance during tasks such as stair climbing.

We also propose alternative, more computationally expensive but perhaps more accurate, methods for calculating the combined variance of the forward kinematics and the map. One such alternative method proposes a weighted average of the grid map cells around the end effector. This weighted averages functions similar to the Gaussian blur kernel of image processing. That is, the cell immediately below the foot receives the highest weighting while the cells around the foot are weighted according to their distance from the foot relative to the standard deviation of the foot measurements σ_j. This weighted average can be calculated as

$$\sigma^2_{m_{\text{avg}}} = \frac{1}{\sum_{i=1}^{W}\sum_{j=1}^{H}\frac{\sigma_j}{r\sqrt{i^2+j^2}}}\sum_{i=1}^{W}\sum_{j=1}^{H}\frac{\sigma_j}{r\sqrt{i^2+j^2}}(\sigma^2_{m_i}) \tag{3.2}$$

where r is the resolution of the grid map, denoting the length and width of one cell in meters. In kernel form, this can look like

$$\sigma^2_{m_{\text{avg}}} = \frac{r\sqrt{10}}{6\sigma_j\sqrt{5} + 6\sigma_j\sqrt{10} + 8\sigma_j\sqrt{2} + r\sqrt{10}}\begin{bmatrix} \frac{\sigma_j}{2r\sqrt{2}} & \frac{\sigma_j}{r\sqrt{5}} & \frac{\sigma_j}{2r} & \frac{\sigma_j}{r\sqrt{5}} & \frac{\sigma_j}{2r\sqrt{2}} \\ \frac{\sigma_j}{r\sqrt{5}} & \frac{\sigma_j}{r\sqrt{2}} & \frac{\sigma_j}{r} & \frac{\sigma_j}{r\sqrt{2}} & \frac{\sigma_j}{r\sqrt{5}} \\ \frac{\sigma_j}{2r} & \frac{\sigma_j}{r} & 1 & \frac{\sigma_j}{r} & \frac{\sigma_j}{2r} \\ \frac{\sigma_j}{r\sqrt{5}} & \frac{\sigma_j}{r\sqrt{2}} & \frac{\sigma_j}{r} & \frac{\sigma_j}{r\sqrt{2}} & \frac{\sigma_j}{r\sqrt{5}} \\ \frac{\sigma_j}{2r\sqrt{2}} & \frac{\sigma_j}{r\sqrt{5}} & \frac{\sigma_j}{2r} & \frac{\sigma_j}{r\sqrt{5}} & \frac{\sigma_j}{2r\sqrt{2}} \end{bmatrix}(\sigma^2_{m_i}) \tag{3.3}$$

for a 5×5 kernel. The downside of this approach is it does not take changes in elevation into account when weighting the variance. Including the elevation map in the variance calculation may be helpful in determining the effect that the variance has on the robot and, by extension, the level of trust that can be placed in the probability of contact given height.

Therefore, we propose a Kalman filter and Gaussian-based method of calculating an elevation-informed and distance-discounting variance estimate of the map beneath the foot. The algorithm first relies on the Kalman prediction equations shown in Equation 3.30. The values of the matrices in the Kalman prediction are

$$A_k = 0_N \quad , \quad B_k = \mathbf{I}_N \quad , \quad u_k = \begin{Bmatrix} \sigma^2_{m_1} \\ \vdots \\ \sigma^2_{m_N} \end{Bmatrix}_k \quad \text{, and} \quad \Sigma_{w_k} = \begin{bmatrix} \sigma^2_{m_1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma^2_{m_n} \end{bmatrix}_k \tag{3.4}$$

Using the Kalman correction equations in Section 3.4.2.3, we can change the correction variances to have a weight that increases if the elevation is significantly different from the elevation direction beneath the foot.

We therefore define the matrices as

$$\tilde{z}_{i,j,k} = \begin{Bmatrix} |1 + \frac{h_{i,j,1} - h_1}{\sigma_j}| \\ \vdots \\ |1 + \frac{h_{i,j,N} - h_N}{\sigma_j}| \end{Bmatrix}_k \tag{3.5}$$

$$\tilde{z}_k = \begin{bmatrix} \tilde{z}_{1,1,k} \\ \vdots \\ \tilde{z}_{1,N,k} \\ \tilde{z}_{2,1,k} \\ \vdots \\ \tilde{z}_{N,N,k} \end{bmatrix} \tag{3.6}$$

$$\Sigma_{v_{i,j,k}} = \begin{bmatrix} \text{cdf}(\frac{r\sqrt{i^2+j^2}}{\sigma_j})\sigma^2_{m_{i,j,1}} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \text{cdf}(\frac{r\sqrt{i^2+j^2}}{\sigma_j})\sigma^2_{m_{i,j,N}} \end{bmatrix}_k \tag{3.7}$$

$$\Sigma_{v_k} = \begin{bmatrix} \Sigma_{v_{1,1,k}} & 0_N & \dots & \dots & \dots & 0_N \\ 0_N & \Sigma_{v_{1,2,k}} & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \Sigma_{v_{2,1,k}} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0_N & \dots & \dots & \dots & \dots & \Sigma_{v_{N,N,k}} \end{bmatrix} \tag{3.8}$$

$$\mathbf{H}_k = \begin{bmatrix} \mathbf{I}_N \\ \vdots \\ \mathbf{I}_N \end{bmatrix} \tag{3.9}$$

where $h_{i,j,N}$ is the height of the grid cell i, j relative leg N, h_N is the height of the grid cell directly beneath foot N, $\text{cdf}(\frac{r\sqrt{i^2+j^2}}{\sigma_j})$ is the output of the two-tailed cumulative distribution function of the normal distribution defined by $\mathcal{N}(h_N, \sigma_j^2)$ representing the probability of making contact at that location or further, and $\sigma^2_{m_{i,j}}$ is the variance of the map at cell i, j. The CDF $\text{cdf}(x)$ of a two-tailed normal distribution $\mathcal{N}(\mu, \sigma)$ can otherwise be written as $\frac{1}{2}(1 + \text{erf}(\frac{x-\mu}{\sigma\sqrt{2}}))$. Using these as the Kalman update equations, therefore, turns the problem into an instantaneous weighted measurement of the map variances, where the overall variance is increased if there are large elevation changes nearby. Additionally, these corrected variances are the map variance multiplied by the z-score of the difference in height between the cell and the foot relative to the

foot variance. The corrected variances are then weighted by the Kalman gain according to the Gaussian probability of making contact at or past that location. This ensures the entire Kalman filter is consistent under a normal distribution and weights the variance according to both the effect that a change in variance would have on the robot and the probability that the change in variance will occur. The results of these variance estimation algorithms are shown in Section 4.2.2 with their variances on different terrains, its effect on the contact estimation, and the calculation time reported accordingly.

3.4 Control

3.4.1 Footstep Plan Optimizer for Walking Robots (FPOWR)

Model predictive control (MPC) relies on solving trajectory optimization problems continually as a control policy. State-of-the-art trajectory optimization libraries typically operate on fixed footstep contact sequences [34]. To achieve motion without prior perception, the robot needs to be able to plan footsteps dynamically as new terrain data is received. As such, we require a dynamic footstep planner that can map surfaces to foot placements in real-time.

Our proposed method is called FPOWR, which stands for Footstep Plan Optimizer for Walking Robots. FPOWR is built on top of the trajectory optimization framework TOWR (Trajectory Optimizer for Walking Robots) as discussed in Section 2.2.3. We obtain a footstep plan by extracting the footsteps from the trajectory generated by TOWR and mapping them onto convex steppable regions (Figure 3.6). To implement this, there were a number of challenges—namely, correctly representing data for TOWR, reconstructing the planes in the output of TOWR, and integrating it in the ROS environment. This section describes solutions to these challenges.

3.4.1.1 Heightmap Representation

In TOWR, the world is represented using a function $f : \mathbb{R}^2 \to \mathbb{R}$ which retrieves the Z height for a corresponding X, Y coordinate. This imposes an issue—what is the best way to represent the data from the perception pipeline in this system? The perception pipeline outputs a list of convex planes and a number of processed and raw heightmaps. For this use case, and for the sake of isolating this module from in the perception pipeline, we choose to use the raw heightmap data.

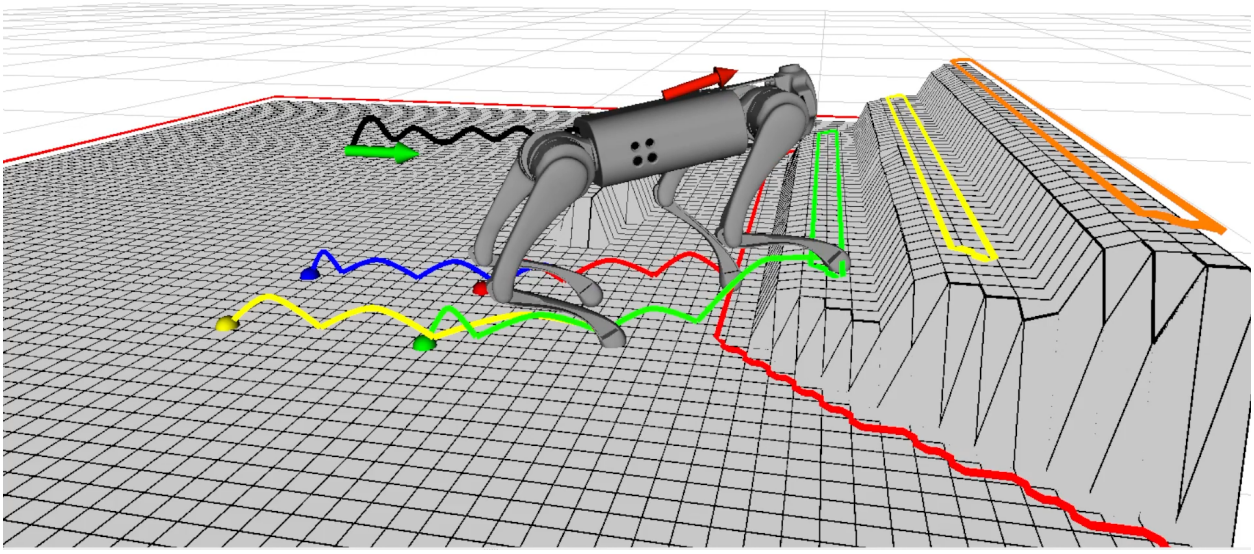


Figure 3.6: FPOWR Example Trajectory and Footstep Plan

3.4.1.2 Output Plane Reconstruction

The output from TOWR is a trajectory for the robot model. This trajectory contains information about the trunk ($SE(3)$) and the four end effectors ($\mathbb{R}^3$). We had to extract the footstep planes ourselves. To do this, we started by processing the trajectory to find the times where every foot made or broke contact with the ground. At each of these times the nearest plane to each end effector is found.

Mapping positions onto their nearest plane is rather trivial, just requiring significant data manipulation. We use `boost::geometry::distance` to do this after fitting the planar perception data into the appropriate boost data structures. This approach takes $O(N \cdot M)$ time, where N is the number of footsteps and M is the number of planes to check. Checking for distances between the footstep point and every plane is an expensive computation, but our testing found it not to be an issue.

This process of mapping footsteps onto nearest planes could introduce issues down the line. There are circumstances where "rounding" the position of footsteps in could lead to kinematic infeasibility when Galileo is trying to solve the trajectory. It would be prohibitively challenging to force TOWR to only step in the prescribed planes without modifying the internal code. Despite this theoretical issue, the system performs well when used in the real world for stair climbing.

3.4.1.3 ROS Integration

We wanted to make FPOWR easy to use for others. For this reason, it was designed as a ROS action. This allows FPOWR to integrate easily with other ROS systems. Additionally, it allows for the footstep plan to be generated in the background and only retrieved when needed. ROS actions can thought of as a client/server connection as shown in Figure 3.7. In the context of the proposed future system control diagram (Figure 6.1), the client connecting to FPOWR will connect to Galileo to inform it of the contact sequence, convex planes, and provide initial guesses.

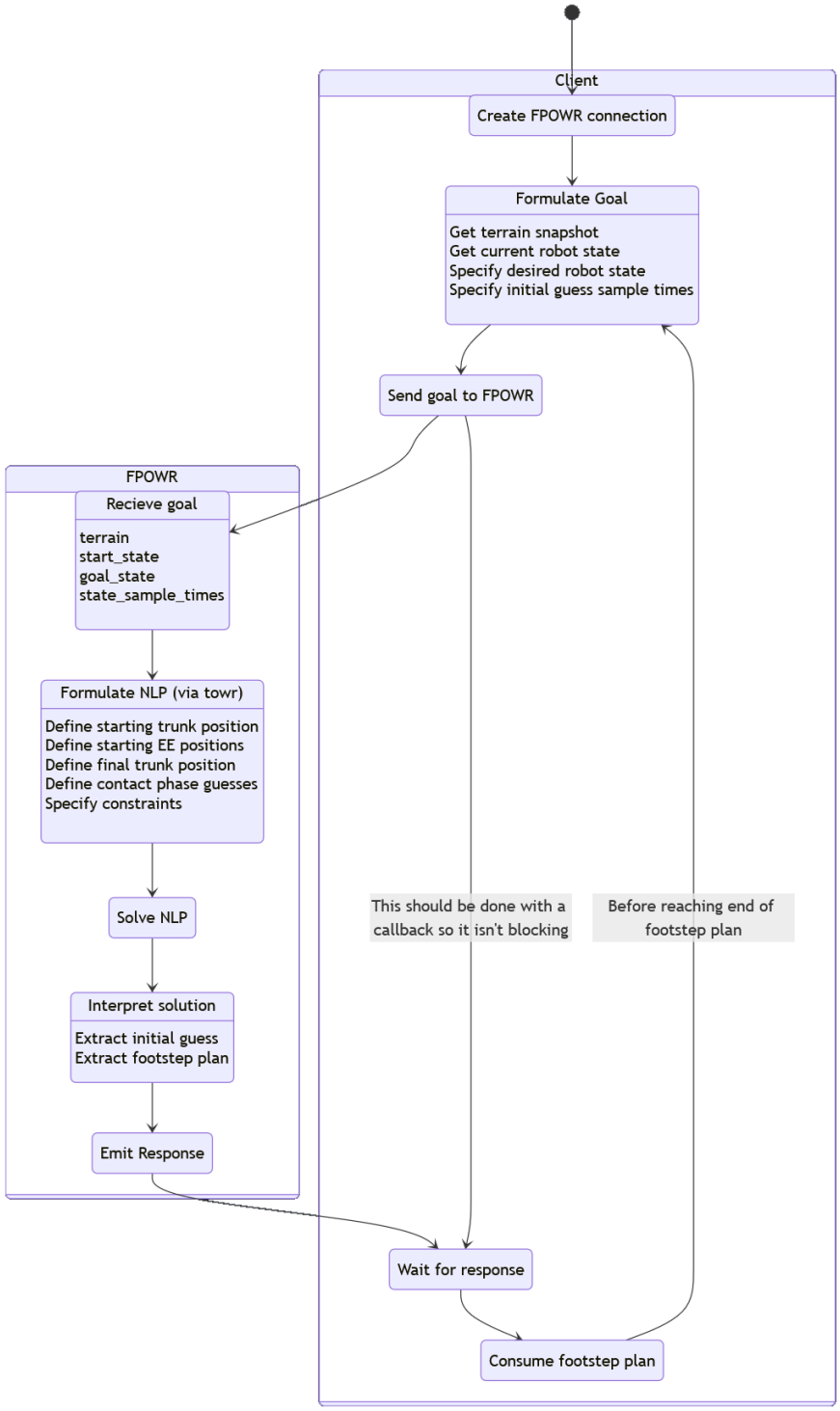


Figure 3.7: Example Usage of FPOWR

3.4.1.4 Using Initial Guesses for MPC Speedup

Another advantage of FPOWR is that the trajectory it uses to generate the footstep plan can also be fed into the MPC's trajectory optimizer as an initial guess, allowing for faster computation. As mentioned in Section 2.2.3, TOWR uses a single rigid-body dynamic model, which assumes that the legs have negligible inertia (i.e., light legs). An MPC using a more complicated dynamic model can benefit by using the trajectory generated from the simpler rigid-body dynamic model as an initial guess. This speeds up computation and allows for a faster MPC control loop.

3.4.2 Contact Estimation

One of the major limitations of the robot, as noted by the previous BiQu team, is issues of accurately representing the terrain and understanding its location in the world. While the perception team is taking steps to address these issues, there is a place for the Whole Body Control (WBC) framework to include these disturbances as well. Some controllers, for instance, can recognize a lack of contact where contact should have been made and can lower the legs, assuming a small gap between location and goal. The controls team will implement contact estimation in Galileo, specifically leveraging confidence intervals in the position of the planar decompositions provided by the perception pipeline.

3.4.2.1 Force Observation

The standard robot equations of motion can be written as

$$\mathbf{M\ddot{q} + C\dot{q} + g = S^\top \tau + J^\top f}$$

where $q = [h_{com}^T, q_b^T, q_j^T]^T \in R^{n_d}$, $h_{com} \in R^6$ represents the collection of the normalized centroidal momentum, $[q_b^T, q_j^T]^T \in R^{n_j}$ is the collection of joint angles and n_j is the number of actuated joints, $M \in R^{n_d \times n_d}$ the mass matrix, $C\dot{q} \in R^{n_d}$ the generalized Coriolis force, $g \in R^{n_d}$ the generalized gravity force, $S \in R^{n_j \times n_d}$ an actuated joint selector matrix, $J \in R^{3n_l \times n_d}$ the Jacobian for all the contacts to the inertial fram vertically concatenated, and $f \in R^{3n_l}$ the force of all the contacts represented in the form $f = [f_{0,x}, f_{0,y}, f_{0,z}, ..., f_{n_l,x}, f_{n_l,y}, f_{n_l,z}]^T$. These standard equations of motion therefore map joint torques, accelerations, and velocities into their effects on the robot's center of mass coordinate system and its acceleration.

The external or disturbance forces can naturally be solved for

$$\tau_{\mathbf{obs}} = \mathbf{J}^\top \mathbf{f} = \mathbf{M}\ddot{\mathbf{q}} + \mathbf{C}\dot{\mathbf{q}} + \mathbf{g} - \mathbf{S}^\top \tau \tag{3.10}$$

where $\tau_{\mathbf{obs}}$ serves as an observer of the current effects on the contact forces as a result of the joint torques, accelerations, velocities, and the center of mass's accelerations. There is still an issue with this method as it is prone to noise and momentary fluctuations in joint readings. As such, a low-pass filter is placed on the system to eliminate noise that would not be present in previous readings. Normally, a low-pass filter would be implemented in Laplace space and discretized, but [47] recognized that the discretization introduced errors and led to a less accurate estimation than the discrete-time frequency domain, or z-domain. Although there does exist an exact z-transform function, equations written in the z-domain can typically be expressed as

$$\mathbf{H}(\mathbf{z}) = \frac{\mathbf{Y}(\mathbf{z})}{\mathbf{X}(\mathbf{z})} = \frac{\mathbf{A}_0 + \mathbf{A}_1 z^{-1} + ... + \mathbf{A}_{h-1} z^{-h}}{1 - \mathbf{B}_1 z^{-1} - ... - \mathbf{B}_{h-1} z^{-h}} \tag{3.11}$$

$h <= n$ is the order of the system or the number of previous samples included in the current calculation, and where z^0 represents a reading 0 steps back in time, z^{-1} represents a reading from 1 step back in time, and so on. This equation can then be rewritten in the discrete-time space as

$$\mathbf{Y}[n] - \mathbf{B}_1\mathbf{Y}[n-1] - ... - \mathbf{B}_h\mathbf{Y}[n-h] = \mathbf{A}_0\mathbf{X}[n] + \mathbf{A}_1\mathbf{X}[n-1] + ... + \mathbf{A}_h\mathbf{X}[n-h] \tag{3.12}$$

This can then be rewritten as

$$\mathbf{Y}[n] = \sum_{i=0}^{k} \mathbf{A_i}\mathbf{X}[n-i] - \sum_{i=1}^{k} \mathbf{B_i}\mathbf{Y}[n-i] \tag{3.13}$$

Using this representation, we introduce a first-order low-pass filter on the observance vector such that

$$\hat{\tau}_{obs} = \frac{1-\gamma}{1-\gamma z^{-1}}(\mathbf{M}\ddot{\mathbf{q}} + \mathbf{C}\dot{\mathbf{q}} + \mathbf{g} - \mathbf{S}^\top \tau) \tag{3.14}$$

where γ is the z-domain of the cutoff frequency. Using the z-transform, we find that $\gamma = e^{-\lambda \Delta t}$ where λ is the cutoff frequency in Hz and Δt is the sampling time. Notably, this equation still has joint accelerations, which we do not have access to, so they must be removed. Rewriting the filtered accelerations such that

$$\hat{\mathbf{M}}\ddot{\mathbf{q}} = \frac{1-\gamma}{1-\gamma z^{-1}}\mathbf{M}\ddot{\mathbf{q}} \tag{3.15}$$

We can use the previous expression of the z-domain in discrete time to find that

$$\hat{\mathbf{M}}\ddot{\mathbf{q}}[n] = \gamma\hat{\mathbf{M}}\ddot{\mathbf{q}}[n-1] + (1-\gamma)\mathbf{M}\ddot{\mathbf{q}}[n] \tag{3.16}$$

starting from $n = 1$ and assuming $\hat{\mathbf{M}}[-1] = 0$, we find that

$$\hat{\mathbf{M}}\ddot{\mathbf{q}}[0] = (1-\gamma)\mathbf{M}\ddot{\mathbf{q}}[0] \tag{3.17}$$

$$\hat{\mathbf{M}}\ddot{\mathbf{q}}[1] = (1-\gamma)\mathbf{M}\ddot{\mathbf{q}}[1] + \gamma\hat{\mathbf{M}}\ddot{\mathbf{q}}[0] = (1-\gamma)\mathbf{M}\ddot{\mathbf{q}}[1] + \gamma(1-\gamma)\mathbf{M}\ddot{\mathbf{q}}[0] \tag{3.18}$$

$$\hat{\mathbf{M}}\ddot{\mathbf{q}}[2] = (1-\gamma)\mathbf{M}\ddot{\mathbf{q}}[2] + \gamma\hat{\mathbf{M}}\ddot{\mathbf{q}}[1] = (1-\gamma)\mathbf{M}\ddot{\mathbf{q}}[2] + \gamma((1-\gamma)\mathbf{M}\ddot{\mathbf{q}}[1] + \gamma(1-\gamma)\mathbf{M}\ddot{\mathbf{q}}[0]) \tag{3.19}$$

$$\hat{\mathbf{M}}\ddot{\mathbf{q}}[n] = (1-\gamma)\sum_{k=0}^{n}\gamma^{n-k}\mathbf{M}\ddot{\mathbf{q}}[k] \tag{3.20}$$

The equation can now be evaluated using summation by parts. The general equation for summation by parts is

$$\sum_{k=0}^{N} f_k(g_{k+1} - g_k) = f_{n+1}g_{n+1} - f_0 g_0 - \sum_{k=0}^{N} g_{k+1}(f_{k+1} - f_k)$$

By selecting

$$f_k = \gamma^{n-k}\mathbf{M}[k] \quad \text{and} \tag{3.21}$$

$$g_k = \dot{\mathbf{q}}[k]/\Delta t \tag{3.22}$$

we can solve for

$$(1-\gamma)\sum_{k=0}^{n}\gamma^{n-k}\mathbf{M}\ddot{\mathbf{q}}[k] = (1-\gamma)\sum_{k=0}^{n}\gamma^{n-k}\mathbf{M}[k](\dot{\mathbf{q}}[k] + \dot{\mathbf{q}}[k+1])/\Delta t =$$
$$(1-\gamma)(\gamma^{-1}/\Delta t\mathbf{M}[n+1]\dot{\mathbf{q}}[n+1] + \gamma^{n}/\Delta t\mathbf{M}[0]\dot{\mathbf{q}}[0] - \sum_{k=0}^{n}\dot{\mathbf{q}}[k+1]/\Delta t(\gamma^{n-k-1}\mathbf{M}[k+1] - \gamma^{n-k}\mathbf{M}[k])) \tag{3.23}$$

assuming $\dot{q} = 0$ and knowing that $\lim_{n\to\infty}\gamma^n = 0$, we can simplify to

$$\hat{\mathbf{M}}\ddot{\mathbf{q}}[n] = (1-\gamma)(\gamma^{-1}/\Delta t\mathbf{M}[n+1]\dot{\mathbf{q}}[n+1] - \sum_{k=0}^{n}\gamma^{n-k}\dot{\mathbf{q}}[k+1]/\Delta t(\gamma^{-1}\mathbf{M}[k+1] - \mathbf{M}[k])) =$$
$$(1-\gamma)(\gamma^{-1}\mathbf{M}[n+1]\dot{\mathbf{q}}[n+1] - \sum_{k=0}^{n}\gamma^{n-k}\dot{\mathbf{q}}[k+1]/\Delta t(\gamma^{-1}\mathbf{M}[k+1] - \mathbf{M}[k+1] + \mathbf{M}[k+1] - \mathbf{M}[k])) \tag{3.24}$$

Grouping terms yields

$$\hat{\mathbf{M}}\ddot{\mathbf{q}}[n] = \frac{1-\gamma}{\gamma^{-1}\Delta t}\mathbf{M}[n+1]\dot{\mathbf{q}}[n+1] - (1-\gamma)\sum_{k=0}^{n}\gamma^{n-k}\dot{\mathbf{q}}[k+1]/\Delta t(\frac{1-\gamma}{\gamma}\mathbf{M}[k+1] + \mathbf{M}[k+1] - \mathbf{M}[k])) =$$
$$\frac{1-\gamma}{\gamma^{-1}}\mathbf{M}[n+1]\dot{\mathbf{q}}[n+1] - (1-\gamma)\sum_{k=0}^{n}\gamma^{n-k}(\frac{1-\gamma}{\gamma\Delta t}\mathbf{M}[k+1]\dot{\mathbf{q}}[k+1] + \dot{\mathbf{M}}[k+1]\dot{\mathbf{q}}[k+1]) \quad (3.25)$$

Assuming $\dot{M} = C + C^T$, $\beta = \frac{1-\gamma}{\gamma\Delta t}$, and the generalized momentum $p = M\dot{q}$ we can rewrite as

$$\hat{\mathbf{M}}\ddot{\mathbf{q}}[n] = \beta\mathbf{p}[n+1] - \sum_{k=0}^{n}(1-\gamma)\gamma^{n-k}(\beta\mathbf{p}[k+1] + C[k+1]\dot{\mathbf{q}}[k+1] + C^T[k+1]\dot{\mathbf{q}}[k+1]) \quad (3.26)$$

applying the inverse of Equation 3.20, we can rewrite the filter output to be equivalent to

$$\hat{\mathbf{M}}\ddot{\mathbf{q}} = \beta\mathbf{p} - \frac{1-\gamma}{1-\gamma z^{-1}}(\beta\mathbf{p} + \mathbf{C}\dot{\mathbf{q}} + \mathbf{C}^T\dot{\mathbf{q}}) \quad (3.27)$$

and finally rewrite the filtered disturbance observer without joint accelerations as

$$\hat{\tau}_d = \beta\mathbf{p} - \frac{(1-\gamma)}{1-\gamma z^{-1}}(\beta\mathbf{p} + \mathbf{S}^\top\tau + \mathbf{C}^\top\dot{\mathbf{q}} - \mathbf{g})$$

or

$$\hat{\tau}_{obs}[n] = \gamma\hat{\tau}_{obs}[n-1] + \beta\mathbf{p}[n] - \gamma\beta\mathbf{p}[n-1] - (1-\gamma)(\beta\mathbf{p} + \mathbf{S}^\top\tau + \mathbf{C}^\top\dot{\mathbf{q}} - \mathbf{g})[n] \quad (3.28)$$

These observed effects on the robot dynamics can then be converted back into contact forces by inverting Equation 3.11 with the selection matrix into

$$\hat{\mathbf{f}}_i = (\mathbf{S}_{\ell_i}\mathbf{J}_{\mathrm{i}}^T)^{-1}\mathbf{S}_{\ell_i}\hat{\tau}_{obs} \quad (3.29)$$

3.4.2.2 Kalman Prediction - Gait Timing

As in [47], we use a Kalman filter to combine probability estimations according to the foot height, foot force, and gait timing. The Kalman prediction equations

$$\hat{x}_{k|k-1} = A_k\hat{x}_{k-1} + B_k u_k \quad (3.30)$$

$$\Sigma_{k|k-1} = A_k\Sigma_{k-1}A_k^T + \Sigma_{w_k} \quad (3.31)$$

are calculated using

$$A_k = 0_N \quad \text{and} \quad B_k = \mathbf{I}_N \quad \text{and} \quad u_k = \begin{Bmatrix} P_1(c|s_\phi\phi) \\ \vdots \\ P_N(c|s_\phi,\phi) \end{Bmatrix}_k \quad \text{and} \quad \Sigma_{w_k} = \begin{bmatrix} \sigma^2_{\phi,1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma^2_{\phi,N} \end{bmatrix}_k \tag{3.32}$$

where $P(c|s_\phi\phi$ represents the probability of contact given the scheduled contact $s_\phi = \{0, 1\}$ and the current progress through the planned motion $\phi = \frac{t-t_0}{t_{end}-t_0}$ where t is the current time, t_0 is the expected starting time of the motion t_{end} is the expected ending time of the motion. This probability can be calculated by

$$P(c|s_\phi,\phi) = \frac{1}{2}\,(\, s_\phi \left[\operatorname{erf}\left(\frac{\phi-\mu_{c_0}}{\sigma_{c_0}\sqrt{2}}\right) + \operatorname{erf}\left(\frac{\mu_{c_1}-\phi}{\sigma_{c_1}\sqrt{2}}\right)\right] + \tag{3.33}$$

$$\overline{s}_\phi \left[2 + \operatorname{erf}\left(\frac{\mu_{\overline{c}_0}-\phi}{\sigma_{\overline{c}_0}\sqrt{2}}\right) + \operatorname{erf}\left(\frac{\phi-\mu_{\overline{c}_1}}{\sigma_{\overline{c}_1}\sqrt{2}}\right)\right]\,) \tag{3.34}$$

where μ_{c_n} represents the mean expected motion progression (ϕ) value at which the motion will switch to the next contact type and σ_{c_n} represents the expected variance of that timing. Because of the values of the A and B matrices, the Kalman prediction can be simplified to static likelihood maximization with Bayes law [47] where

$$\hat{x}_{k|k-1} = u_k \tag{3.35}$$

$$\Sigma_{k|k-1} = \Sigma_{w_k} \tag{3.36}$$

3.4.2.3 Kalman Update - Foot Height and Force

The Kalman update equations

$$K_k = \Sigma_{k|k-1} H_k^T (H_k \Sigma_{k|k-1} H_k^T + \Sigma_{v_k})^{-1} \tag{3.37}$$

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k(\tilde{z}_k - H_k \hat{x}_{k|k-1}) \tag{3.38}$$

$$\sum\nolimits_{k|k} = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \sum\nolimits_{k|k-1} \tag{3.39}$$

are used to update the probability estimates from the prediction equations. As in [47], we use the probability of contact given the foot height and estimated foot force according to

$$\tilde{z}_{1,k} = \begin{Bmatrix} P_1(c|p_{z,1}) \\ \vdots \\ P_N(c|p_{z,N}) \end{Bmatrix}_k \quad \tilde{z}_{2,k} = \begin{Bmatrix} P_1(c|f_{z,1}) \\ \vdots \\ P_N(c|f_{z,N}) \end{Bmatrix}_k \qquad \tilde{z}_k = \begin{bmatrix} \tilde{z}_{1,k} \\ \tilde{z}_{2,k} \end{bmatrix} \tag{3.40}$$

$$\Sigma_{v_{1,k}} = \begin{bmatrix} \sigma^2_{p_{z,1}} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma^2_{p_{z,N}} \end{bmatrix}_k \quad \Sigma_{v_{2,k}} = \begin{bmatrix} \sigma^2_{f_{z,1}} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma^2_{f_{z,N}} \end{bmatrix}_k \qquad \Sigma_{v_k} = \begin{bmatrix} \Sigma_{v_{1,k}} & 0_N \\ 0_N & \Sigma_{v_{2,k}} \end{bmatrix} \tag{3.41}$$

$$\mathbf{H}_k = \begin{bmatrix} \mathbf{I}_N \\ \mathbf{I}_N \end{bmatrix} \tag{3.42}$$

where $\sigma_{p_{z,i}}$ is the foot height variance for foot i and $\sigma_{f_{z,i}}$ is the foot force variance for foot i. The probability $P_i(c|p_z)$ is the probability of contact at foot i given the foot height p_z from joint data according to

$$P(c|p_z) = \frac{1}{2}\left[1 + \text{erf}\left(\frac{\mu_{z_g} - p_z}{\sigma_{z_g}\sqrt{2}}\right)\right] \tag{3.43}$$

where u_{z_g} is the average footstep height according to vision data and historical footsteps. The probability of contact given foot force in the downward direction f_z as measured by the external force observer is

$$P(c|f_z) = \frac{1}{2}\left[1 + \text{erf}\left(\frac{f_z - \mu_{f_c}}{\sigma_{f_c}\sqrt{2}}\right)\right]$$

where μ_{f_c} is the average force to determine contact. By taking the updated probability of contact from the Kalman correction equations, we can set a threshold probability to determine contact as a binary variable for each leg. Additionally, digital hysteresis can be used to ensure that slight noisiness in sensor readings won't effect the estimation of contact once it has already been detected.

3.4.2.4 Kalman Update - Foot Sensor Force

The schema introduced in Section 3.4.2.3 can be easily modified to include more readings. The Unitree Go1 has a force sensor in each foot for detecting the forces exerted by the terrain on the leg. These force sensor readings can be incorporated in the Kalman Correction equations much like the estimated forces were. By extending the correction measurement noise Σ_{v_k}, measurement conversion matrix H_k, and

measurement vector $\tilde{z}_k$ to

$$\Sigma_{v_k} = diag\begin{bmatrix}\Sigma_{v_{1,k}} & \dots & \Sigma_{v_{r,k}}\end{bmatrix} \qquad \mathbf{H}_k = \begin{bmatrix}\mathbf{I}_N \\ \vdots \\ \mathbf{I}_N\end{bmatrix} \qquad \tilde{z}_k = \begin{bmatrix}\tilde{z}_{1,k} \\ \vdots \\ \tilde{z}_{r,k}\end{bmatrix} \tag{3.44}$$

the Kalman correction equations can be extended to any r sources of measurement, provided the readings are provided as a probability of contact for each leg with some variance. To add the force sensors, the measurement vector z_3 must also be calculated as

$$\tilde{z}_{3,k} = \begin{Bmatrix} P_1(c|f_{z,1,s}) \\ \vdots \\ P_N(c|f_{z,N,s}) \end{Bmatrix}_k \tag{3.45}$$

where $f_{z,N,s}$ is the force measured from the force sensor at leg N.

3.4.3 Perception-Aware Contact Estimation (PACE)

The contact estimation introduced so far is notably perception-less. The contact estimation relied on foot force, contact timing, and foot height, but the foot height was notably based on some μ_{z_g} mean foot height for contact. With a constant μ_{z_g} a robot cannot climb stairs, the goal of our project. In the world frame, the contact location is changing according to the ascent or descent of the stairs. An increase in contact height relative to a constant μ_{z_g} naturally results in a lower probability of contact whereas a decrease in contact height results in a higher probability of contact, neither of which is representative of reality and both of which can cause divergent actions in the WBC leading to a fall. In the robot frame, this divergence in the WBC holds true with a constant μ_{z_g}. When the robot is ascending, the torso naturally adjusts to the slope of the stairs with the center of mass lying closer vertically to the front contacts than the back contacts. Because of this vertical difference between the front contacts and the back contacts relative to the center of mass, the same issue occurs where the back legs detect contact early and the front legs detect contact late or not at all. The reverse is true when descending. This then reveals a need for perception-informed contact estimation that incorporates the ground height as measured by the LiDAR into the probability of contact given foot height.

In Section 3.3.6, we introduced methods for extracting the terrain height and confidence from the grid map developed as part of the LiDAR and depth camera perception pipelines. In this section, we

introduce a method for implementing these for perception-aware contact estimation (PACE). The perception-blind method for calculating the probability of contact given foot height is

$$P(c|p_z) = \frac{1}{2}\left[1 + \text{erf}\left(\frac{\mu_{z_g} - p_z}{\sigma_{z_g}\sqrt{2}}\right)\right] \tag{3.46}$$

In perception-aware contact estimation, μ_{z_g} is no longer a constant. Instead, we can calculate the mean foot height for contact using

$$\mu_{z_g,l} = h(p_{x,y,l}) + o \tag{3.47}$$

where $h(p_{x,y})$ is the output of the methods in Section 3.3.6 representing the height of the terrain beneath foot l as derived by its 2D position in the world frame $p_{x,y}$ and o is a constant offset from the map to the foot to ensure the desired output of the probability of contact. The variance of the contact can then be calculated as

$$\sigma_{z_g,l} = \sigma(p_{x,y,l}) + \sigma_j \tag{3.48}$$

where $\sigma(p_{x,y}, l)$ is the variance of the map as derived by the methods in Section 3.3.6 and σ_j represents the constant variance of the forward kinematics of the end effector in the z direction. Combining these with Equation 3.46, we can find the perception-aware probability of contact given foot height as

$$P(c|p_z) = \frac{1}{2}\left[1 + \text{erf}\left(\frac{h(p_{x,y,l}) + o - p_z}{(\sigma(p_{x,y,l}) + \sigma_j)\sqrt{2}}\right)\right] \tag{3.49}$$

3.5 Containerization

The team spent a significant amount of time creating Docker containers to run the project code in. Initially, this was done out of necessity as many project contributors were using Windows or Mac without access to standard Linux build tools. This allowed all team members to share the same development environment regardless of their system architecture. In the future, it would be possible to deploy these Docker containers onto the Go1. This would significantly streamline the step from sim to real hardware, but at this point the team has not yet attempted this. These should significantly reduce the amount of setup required for future BiQu teams. In the end, we created Docker containers for the following code bases listed below, which can all be found on the WPI ALMaS GitHub at https://github.com/ALMaSWPI:

- FPOWR
- Legged Control

- Legged Perception
- Galileo
- DIAL-MPC

Chapter 4

Results and Discussion

4.1 Perception Pipeline

4.1.1 Drift Reduction

4.1.1.1 Experimental Setup

We evaluated the impact of integrating FAST-LIO2 on odometry drift reduction through a series of trials involving both single-platform and multi-stair real-world configurations. In these experiments, the Livox Mid-360 served as the sole data source for both FAST-LIO2 and the elevation mapping pipeline. The Livox Mid-360 was mounted above the robot, angled forward as seen in Figure 4.1. For each trial, the robot was placed approximately 2 meters from the elevated platform or stairs and navigated toward the obstacle with a consistent velocity.

To use the LiDAR data effectively as the sole input source, we applied a simple CropBox filter to retain only the points in front of the robot for elevation mapping. Meanwhile, the full 360° point cloud was used as input to FAST-LIO2, allowing it to capture environmental features in all directions. This front-facing filter helped exclude nearby moving objects—such as team members—from affecting the elevation map. It also reduced erroneous high-elevation readings from objects like the undersides of desks and tables, which occasionally distorted the torso orientation map due to the large smoothing kernel used in its generation. The impact of this filtering process is shown in Figure 4.2.

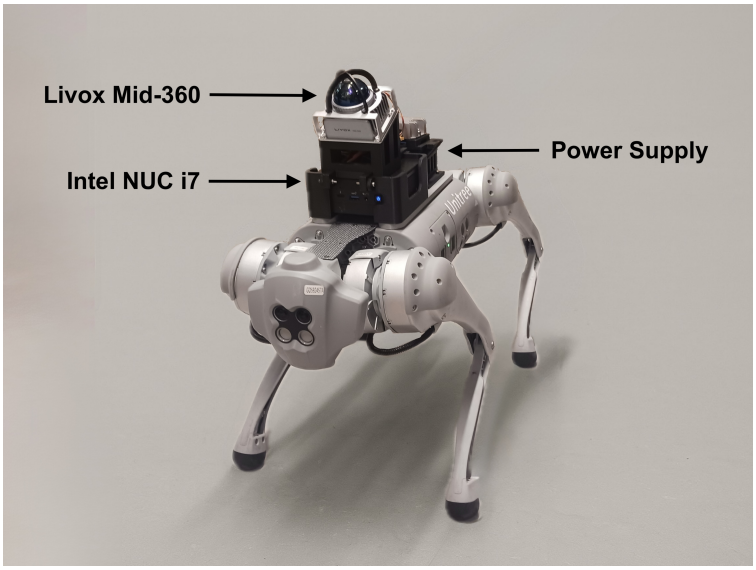


Figure 4.1: Experimental mounting setup of Livox Mid-360, Intel NUC i7 11th gen, and power supply on the Unitree Go1

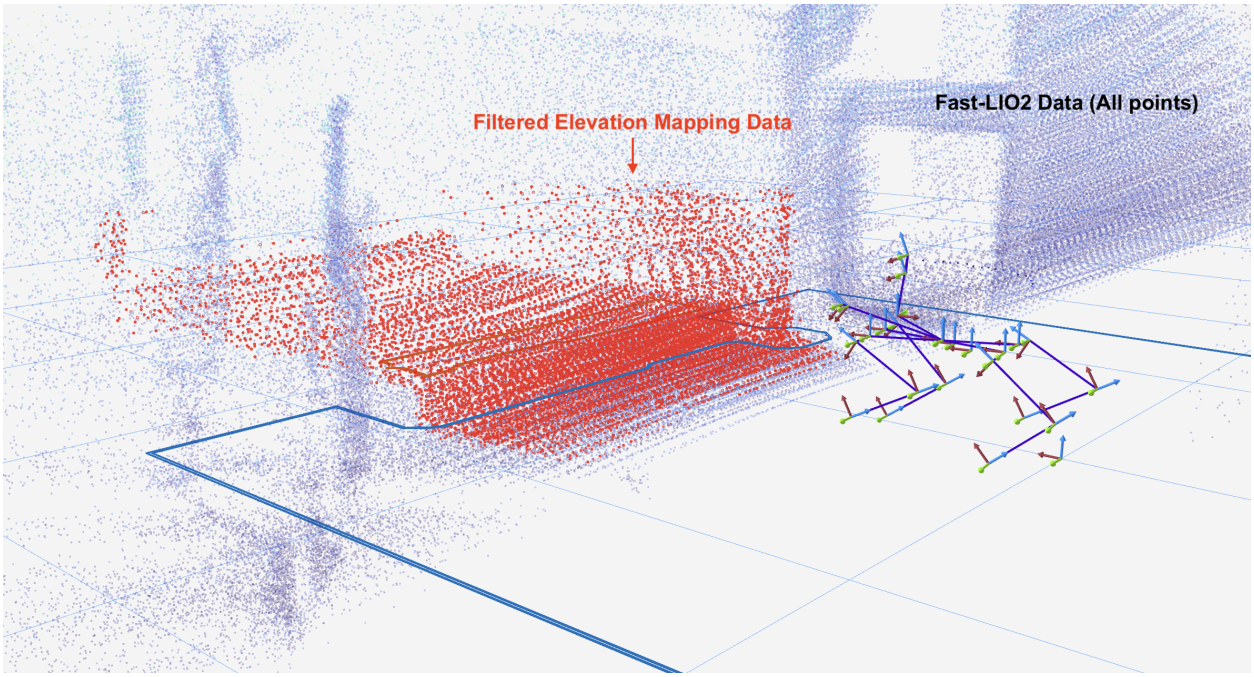


Figure 4.2: Comparison between the filtered front-facing point cloud used for elevation mapping and the full 360° point cloud used as input to FAST-LIO2

4.1.1.2 Results and Discussion

In the absence of FAST-LIO2, the robot's odometry (and consequently the elevation map) exhibited significant drift laterally as the robot approached the elevated plane. This drift consistently led the robot to misstep while attempting to ascend the platform, resulting in catastrophic failure.

Conversely, when FAST-LIO2 odometry was used in the state estimator, the odometry and elevation map closely aligned with the robot's actual position and orientation. As a result, the robot reliably placed its first step correctly onto the obstacle. Additionally, the robot was able to maintain a correct representation of the elevated plane while ontop of the obstacle, allowing it to descend the obstacle using only historical data.

An important finding in this testing is that the IMU data had a lot of drift and inconsistencies, and the fact that SLAM corrected it during fast movements (for example, climbing up a flight of stairs) resulted in the state-estimate "jumping" and throwing off the planned trajectories. As seen in FIGURE, the estimated base frame of the robot is significantly altered during upward movement, which then is corrected by the FAST-LIO2 odometry measurement. Due to this inconsistency during fast vertical movements, we had many failures possibly due to this issue during staircase climbing, but not during single platform or multi-platform trials. We explore these results more in Section 4.3.2.

These findings demonstrate that incorporating FAST-LIO2 was crucial for maintaining accurate odometry and improving the success rate of platform and stair climbing in our trials. FAST-LIO2's tightly-coupled fusion of IMU and LiDAR data at high frequency significantly mitigated the drift issues encountered with legged and inertial odometry, particularly in the lateral (X, Y) directions. This fusion removed long-term drift from state estimation and maintained spatial alignment between the robot's perceived environment and the physical terrain.

The importance of exteroceptive odometry is further amplified in our case, where the Livox Mid-360 LiDAR serves as the only sensor input for both localization and mapping. Due to the forward-facing and limited field of view configuration, the elevation map immediately around the robot is predominantly based on historical LiDAR measurements. Without accurate odometry, this historical data cannot be correctly integrated as the robot advances, leading to misalignment between the robot's assumed position and the true location of the elevated surfaces. This mismatch ultimately results in catastrophic failures, as observed in the trials without FAST-LIO2.

Currently, FAST-LIO2 operates somewhat as a detached system, providing pose estimates at ap-

proximately 10 Hz, which we integrate into the robot's high-frequency (400 Hz) Extended Kalman Filter (EKF). These pose updates are incorporated as asynchronous position and orientation measurements upon arrival, complementing the EKF's continuous propagation from internal IMU and leg odometry sources. While this integration significantly reduces drift, it is not fully time-synchronized or tightly coupled with the filter's prediction cycle. Future work could explore tighter fusion strategies, potentially integrating FAST-LIO2 outputs at a higher frequency or leveraging more predictive interpolation to bridge the gaps between updates and further enhance real-time state estimation.

These experiments exemplify the critical role of precise odometry when using sparse or forward-focused perception systems. Additionally, they suggest that further improvements could be achieved by expanding the field of view or incorporating additional exteroceptive sensors such as rear or side-mounted LiDARs, or other visual odometry systems. Such enhancements could provide better situational awareness, especially when navigating complex terrains with abrupt elevation changes.

4.1.2 Sensor Configurations

4.1.2.1 Experimental Setup

We 3D printed and assembled the two LiDAR mount modules described in Section Section 3.3.3. For each of the configurations, we ran the legged perceptive pipeline on the robot while standing on flat ground with our demo stair about one meter in front of it. We simultaneously visualized the elevation map in RViz for observational analysis.

4.1.2.2 Results and Discussion

Our test with the adjustable mount quickly revealed some major issues. The elevation map faced a large amount of noise, and most of the cells had a high height variance. Flat surfaces were not recognized as flat, but instead as bumpy surfaces with spikes and holes. We observed that the physical LiDAR mount visibly shakes during use, and the screw that is used for re-adjustment needed to be re-tightened frequently. Rapid spinning of the LiDAR causes constant vibrations, and the adjustable mount we designed was clearly not strong enough to hold.

Our test with the angled mount resulted in a much more stable elevation map, with few bumps on surfaces that should be flat. The mount itself was not shaking in spite of the LiDAR's vibrations. The range of the map was sufficient to observe the entire stair as well. We decided to select the stable angled mount

shown in figure Figure 4.1 because of these reasons

4.1.3 Usage of GPU Acceleration

4.1.3.1 Experimental Setup

To test the effectiveness of GPU acceleration for the full pipeline, we ran legged perception with the D435i in simulation on the Intel NUC and Jetson Orin NX described in section Section 3.3.1. We used the RealSense D435i because the corresponding sensor simulation software requires fewer resources to run than the Livox Mid-360 LiDAR. We could then have a more clear picture of how the program would run on the physical robot. The pipeline that we run on the Jetson is the same as the one on the NUC, except it uses the GPU-accelerated version of elevation mapping described by [4]. To conduct this experiment, we simply have the simulated robot trot forward about one meter in an empty world.

4.1.3.2 Results and Discussion

The Jetson produced an astoundingly low map refresh rate of about 2 hertz. The NUC outputs an elevation map at about 10 hertz. Both the perceptive and control pipelines are CPU-intensive processes. The major decrease in refresh rate is probably due to differences in the processors of the NUC and the Jetson. In addition, we planned to integrate SLAM into the perception pipeline, which would increase the computational load put on the CPU. GPU acceleration is most effective when processing larger point clouds, as in [4], and we use a crop box to filter out irrelevant points before performing elevation mapping processing, which means that in our live pipeline, we would not be gaining the full advantage of the feature anyway. These factors combined led to our choice to use the Intel NUC with no GPU acceleration in our final pipeline.

4.2 Controls

4.2.1 Footstep Planning

4.2.1.1 Experimental Setup

The experiments for FPOWR were done in simulation. Computational efficiency is critical to ensure that footsteps can be generated faster than in real-time for an MPC. Because of this, we focus on improving the overall solve time of the FPOWR and improving the quality of the generated trajectories.

We chose to test on terrain data that most closely represents what the robot will encounter in the real world. We created a ROS bag file containing real perception data of the robot in front of a staircase. Using this data allowed our tests to be consistent and exactly represent what FPOWR would see when deployed to the robot.

To evaluate FPOWR, we tested different combinations of terrain (flat and staircases) with four different solvers: SNOPT, IPOPT with Mumps, IPOPT with HSL-MA57, and IPOPT with HSL-MA97 (See Section 2.2.4 for more details). We compare their solve times and describe the quality of their generated trajectories. It is important to note that the most important product of the FPOWR is the contact sequence, but that is much harder to analyze, and it is directly influenced by the robot trajectory.

4.2.1.2 Results and Discussion

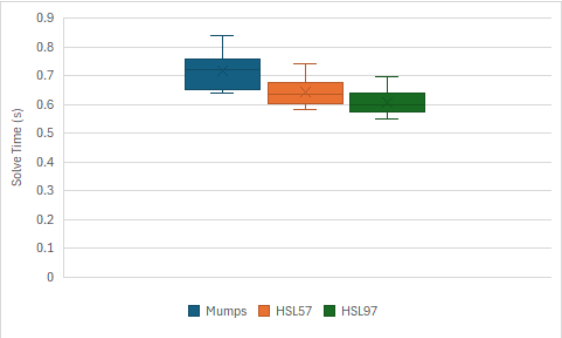
All tests were run on a computer with an Intel i7-6700K CPU (4 core, 8 thread, 4.0 GHz). The solve times listed for different combinations of terrain and gait optimization settings are shown in Figure 4.3. All of the solvers converged to visually identical solutions for each problem, so only one example trajectory is shown for each of the test environments. The sample trajectory for walking on flat ground can be seen in Figure 4.4, and stair climbing can be seen in Figure 4.5. Both of these figures have corresponding videos linked. Example images and videos were not provided for the other solvers because they all converge to nearly identical results.

These results show a stark difference in solve times between the tested solvers. We have chosen to move forward with IPOPT with HSL-MA97 for this project. This is because of its superior solve time in more complicated problems, and its solution quality is on par with the others. Despite this, FPOWR remains usable with any compatible solver, so for users without a license to HSL-MA97, Mumps can be used instead.

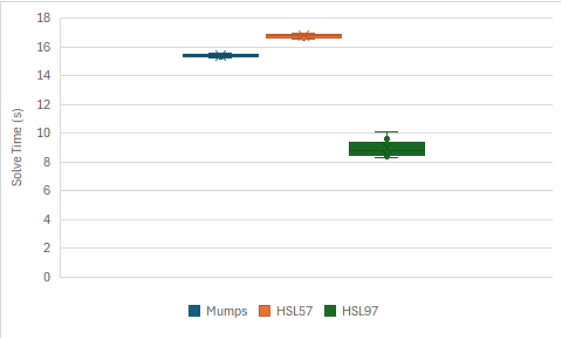
We believe these results are promising, but the solution time still has room for improvement. The solve time for the footstep planner module has to be faster than the actions that the robot will perform—in order to stay ahead of the real world. Ideally, this module would run as quickly as possible to not interrupt other functions on the Intel NUC. The CPU this module was tested on is slightly worse performing than the NUC, but it was also not running any other background tasks at the same time.

4.2.2 Contact Estimation

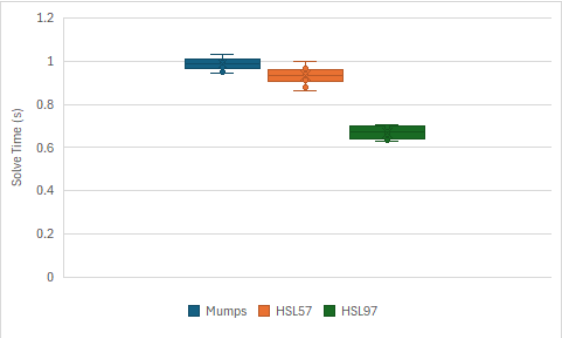
The experiments for contact estimation were conducted both in simulation and in real scenarios.



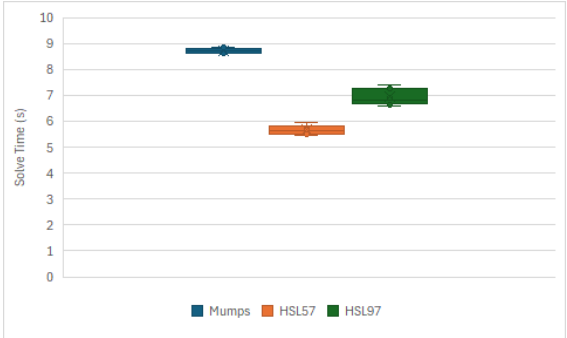
(a) Solve times for 1m of straight movement over flat ground in 2s without gait optimization.



(b) Solve times for 1m of straight movement over flat ground in 2s with gait optimization.



(c) Solve times for climbing 1, 7" step in 2s without gait optimization.



(d) Solve times for climbing 1, 7" step in 2s with gait optimization.

Figure 4.3: Solve times for Mumps, HSL57, and HSL97 on flat ground and stairs, and with and without gait optimization

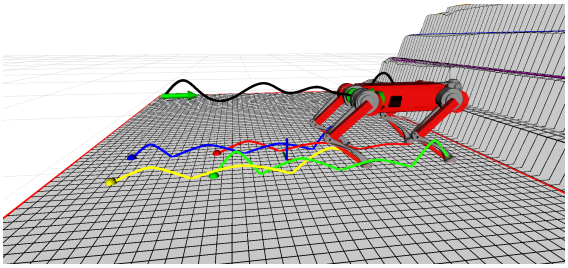


Figure 4.4: Resultant trajectory from HSL-MA57 after commanding the robot forwards 1m on flat ground in 2s with gait optimization. A video can be seen here.

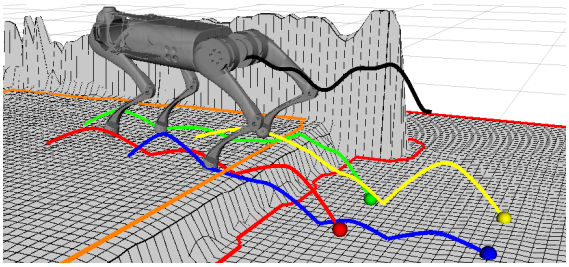


Figure 4.5: Resultant trajectory from HSL-MA57 after commanding the robot to climb 1, 7" step in 2s with gait optimization. A video can be seen here.

4.2.2.1 Simulation Experiments

For experiments in simulation, a controller with perfect odometry was used to confirm the accuracy of the contact estimation. Initially, various cutoff frequencies were compared against the unfiltered generalized observer to determine the best cutoff frequency. The chosen cutoff frequencies were 5, 15, 100, and 500 Hz, corresponding to z-domain poles of 0.995, 0.985, 0.904, and 0.606, respectively.

The contact estimation was compared against a perfect contact sensor as a control. The contact estimation was tested on multiple terrains with varying levels of flatness. Both perception-aware and perception-less contact estimation were employed and compared using the ground truth from the simulated contact sensors through rqt_plot. The results are reported on flat ground and while climbing stairs for a Go1 robot with a trotting gait switching every 0.3 seconds with a swing height of 0.08 meters.

The result of various cutoff frequencies was measured for the generalized momentum observer and compared against the unfiltered input. The forces measured from one contact are reported in Figure 4.6, Figure 4.7, Figure 4.8, and Figure 4.9. The results appear to suggest that lower frequencies most closely follow the raw measurement and higher frequencies filter the data more, although this is not always ideal because in areas of high measurement change, the filter does not always conform closely to the average value. This can likely be changed with higher orders of low-pass filter, but this comes with an associated delay in contact detection, which can somewhat be seen in the compared measurements and becomes more pronounced with increased order of the low-pass filter. This delay is not ideal for instantaneous contact detection and therefore, the higher-order low-pass filter was not implemented. Notably, the difference in cutoff frequency did not seem to significantly change the timing of the detection contact or contact loss.

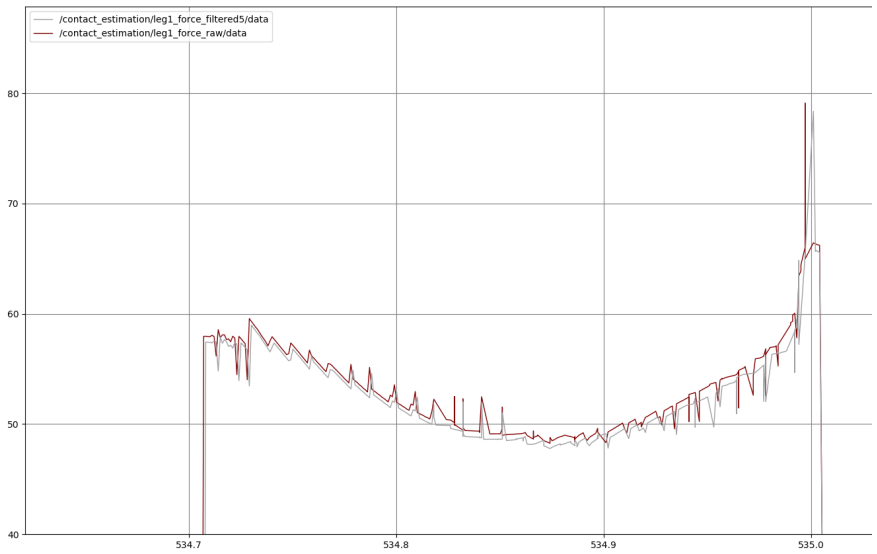


Figure 4.6: Raw input vs. low-pass filtered input with 5 Hz cutoff frequency.

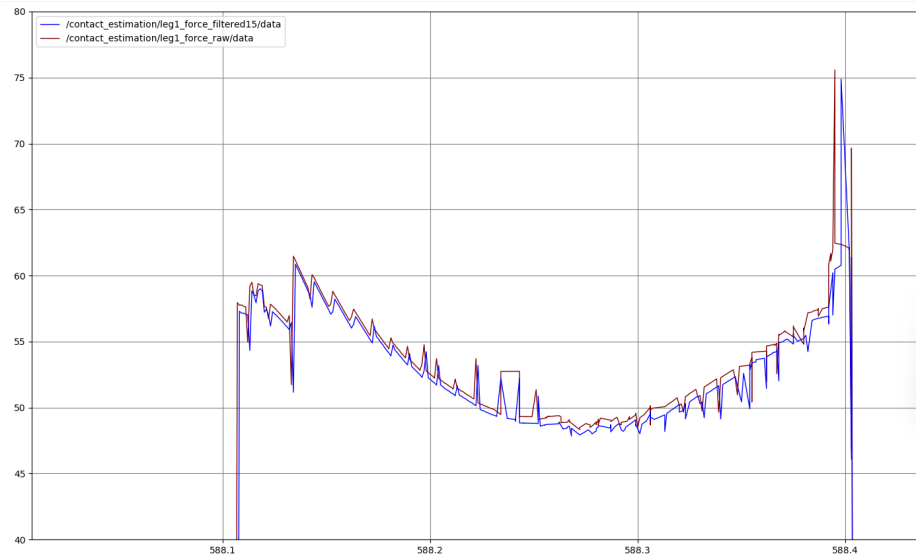


Figure 4.7: Raw input vs. low-pass filtered input with 15 Hz cutoff frequency.

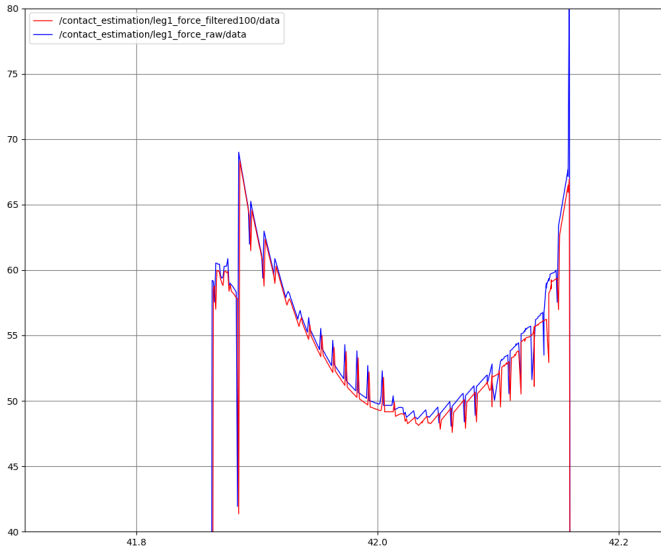


Figure 4.8: Raw input vs. low-pass filtered input with 100 Hz cutoff frequency.

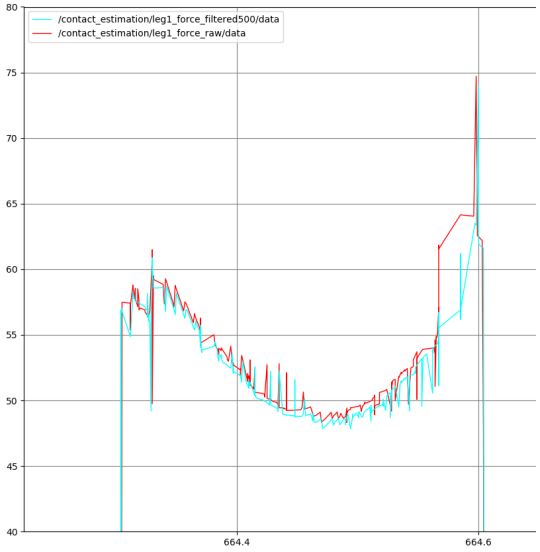


Figure 4.9: Raw input vs. low-pass filtered input with 500 Hz cutoff frequency.

The contact detection was initially compared in simulation between the contact sensor and the generalized momentum observer. The difference between the ground truth contact sensor and the output of the generalized momentum observer is reported in Figure 4.10. The generalized momentum observer appears to detect contact and loss of contact slightly before the ground truth. Although we could alter this to exactly fit the ground truth in sim by changing the hysteresis values, we opted to avoid potentially overfitting the hysteresis because the difference between the measurement and ground truth was on average 10 milliseconds. Furthermore, this never seemed to affect the robot's movement in simulations or in reality.

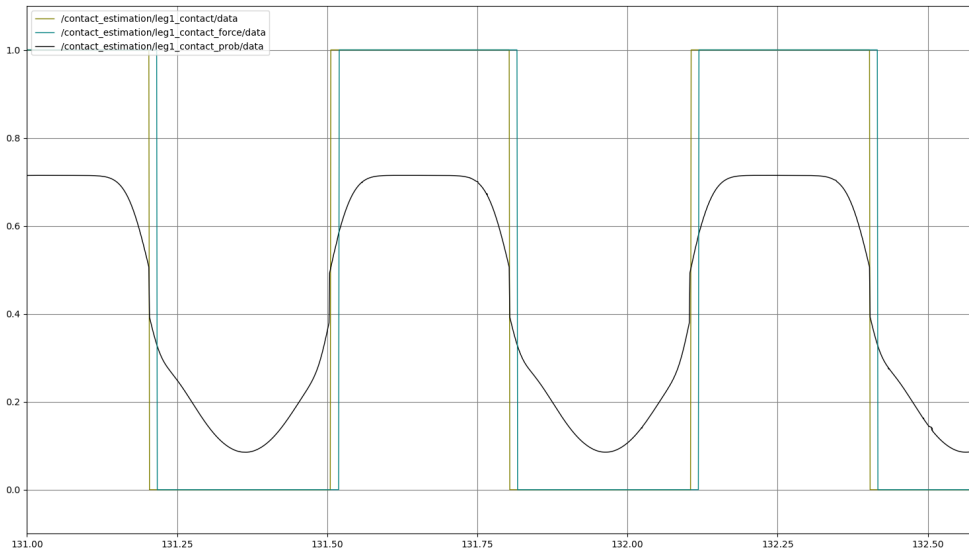


Figure 4.10: Contact estimation from the generalized momentum observer vs ground truth

The contact detection on a flat plane is composed of the probability of contact given the foot height, gait timing, and force measurements. The breakdown of each probability and its contribution to the overall probability of contact is plotted in Figure 4.11 for a series of contacts with variables $\mu_{c_0}, \mu_{\bar{c}_0} = 0$, $\mu_{c_0}, \mu_{\bar{c}_0} = 1$, $\sigma_{c_0}, \sigma_{\bar{c}_0}, \sigma_{c_1}, \sigma_{\bar{c}_1} = 0.05$, $\mu_{z_g} = 0.02$, $\sigma_{z_g} = 0.075$, $\mu_{f_c} = 30$, and $\sigma_{f_c} = 15$. Notably, the probability of contact given gait timing rises and falls at nearly the same time as the probability of contact given foot force. With the two changing drastically at nearly the same time, they compose the largest contribution to whether contact has been made. The probability of contact given height, on the other hand, never truly reaches its maximum or minimum value on flat terrain. Instead, the probability of contact given reaches its lowest when the leg is at the highest point of the swing and reaches its highest when the leg has been placed down, but does not peak abruptly like the other two probabilities. The probability of contact given height tends to rise before the other two probabilities and persist high after they have fallen. This is because to the Bezier curve of the swing leg trajectory requires time to accelerate, leaving the swing leg vertically near the point of contact even though contact is no longer being made. With this understood, the probability of contact given height serves as more of a baseline that must be satisfied before contact is believed to have been made.

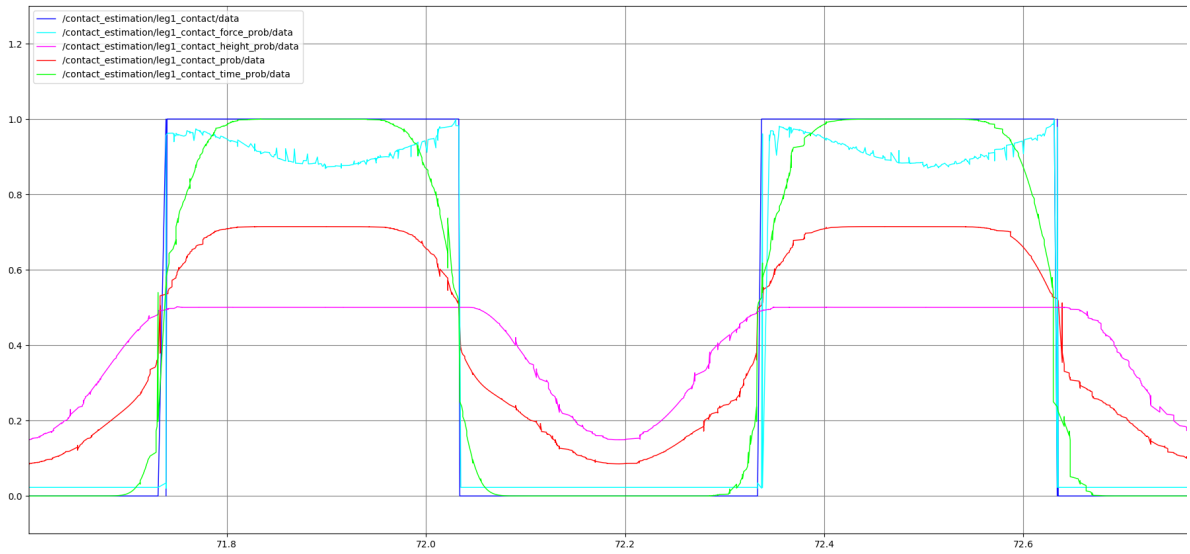


Figure 4.11: Contact estimation and its component parts

The baseline that is provided by the probability of contact given foot height explains why perception data is so important to detecting contact when the terrain is no longer flat. Figure 4.12 demonstrates the necessity of incorporating perception data into the height element of the contact estimation. When the robot walks up a stair with a height of 0.1 m, the probability of contact given height for a front leg can be seen decreasing between the ground plane and the stair. This leads to a lower overall probability of contact. With future steps, the decrease will become more pronounced. This decrease cannot be measured, however, because of the instability in the back leg under the same conditions. In Figure 4.13, the contact estimation of a back leg is reported. It can be seen that for this leg, the probability of contact given height increases after the contact is made and stays high well after the contact is lost. In this case, the probability of contact remaining high causes the contact to be detected past the ground truth, resulting in unstable applied forces, and an unstable center of mass. This instability eventually causes the robot to fall.

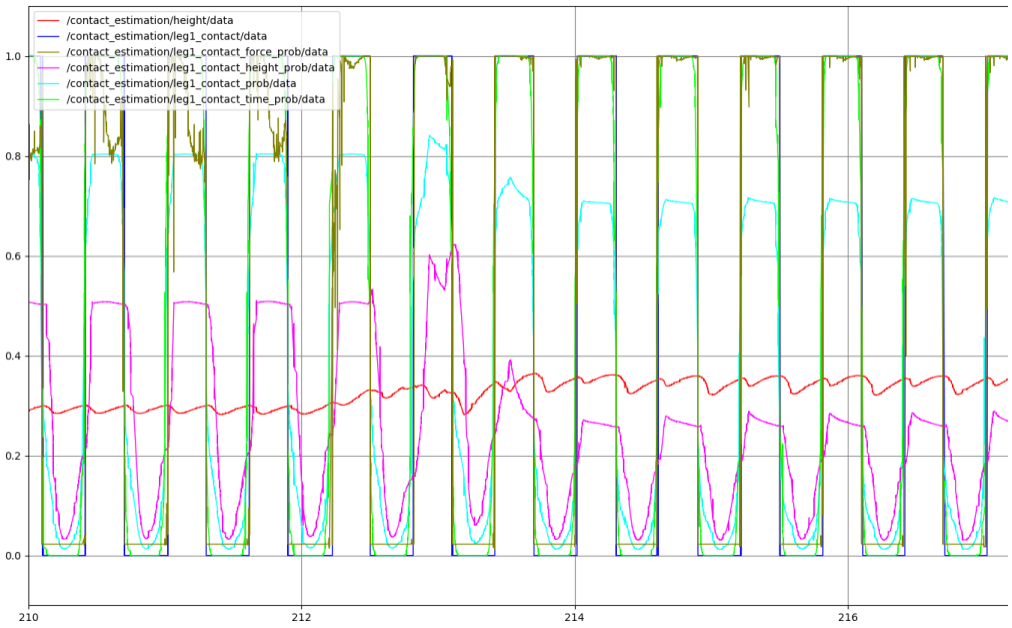


Figure 4.12: Contact estimation of a front leg when going up stairs without vision

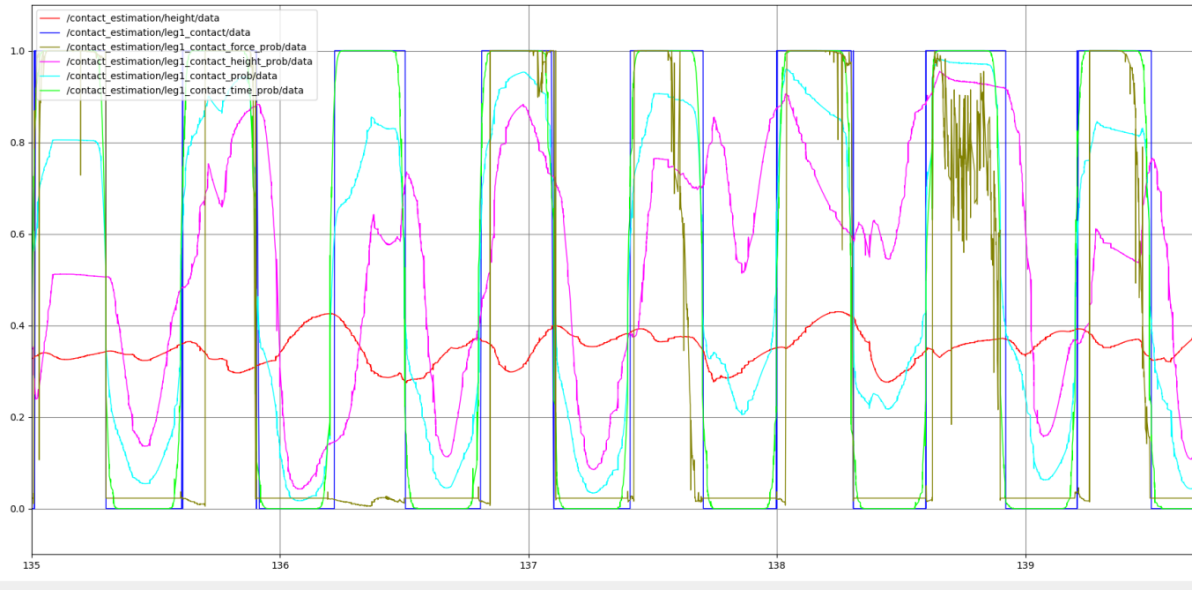


Figure 4.13: Contact estimation of a back leg when going up stairs without vision

Once the perception-based contact estimation has been integrated as in Equation 3.49 with parameters $\sigma_j = 0.075$ and $o = 0.02$, it can be observed that the probability of contact given height returns to a normal baseline with the probabilities of contact resembling the contact estimation on a plane shown in Figure 4.14. The vision instead incorporates the terrain data, which can be seen rising in the figure, allowing the contact estimation to adjust to the terrain underneath it. Using this adapted contact estimation, the

robot was able to completely scale the stairs.

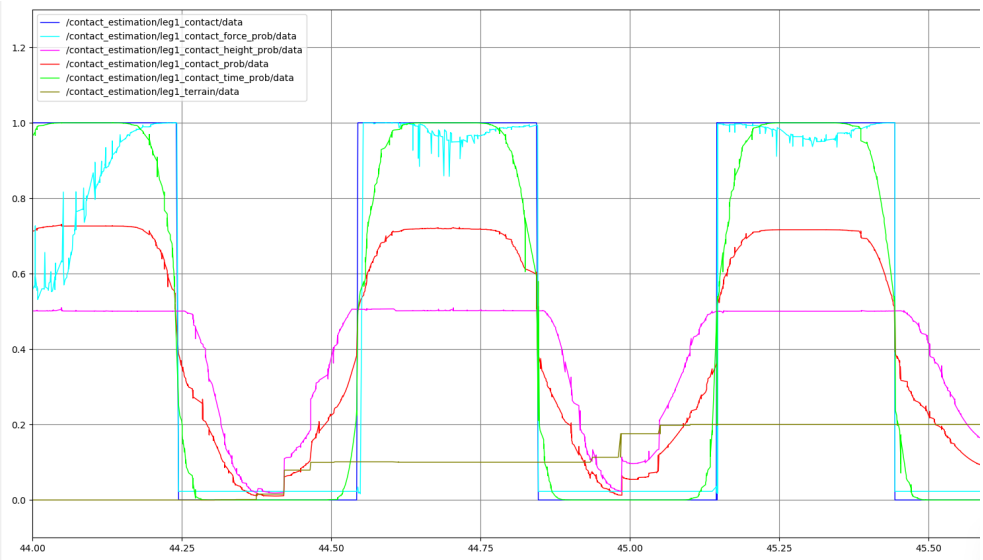


Figure 4.14: Contact estimation of a front leg going up stairs with vision

4.2.2.2 On-Robot Experiments

The on-robot experiments were conducted for both flat terrain and on stairs. The experiments were conducted for both perception-less and perception-aware contact estimation and compared against a ground-truth from high-speed camera footage. The time of contact from the camera footage was compared against the time of contact as estimated by the contact estimation and the results are reported in Section 4.2.2. The results are reported for a Go1 robot with a trotting gait switching every 0.3 seconds with a swing height of 0.08 meters.

On flat terrain while walking, the force readings were very consistent. The force, while in contact, often averaged around 50 newtons of applied force and appeared to rise and fall with the real-world contact.

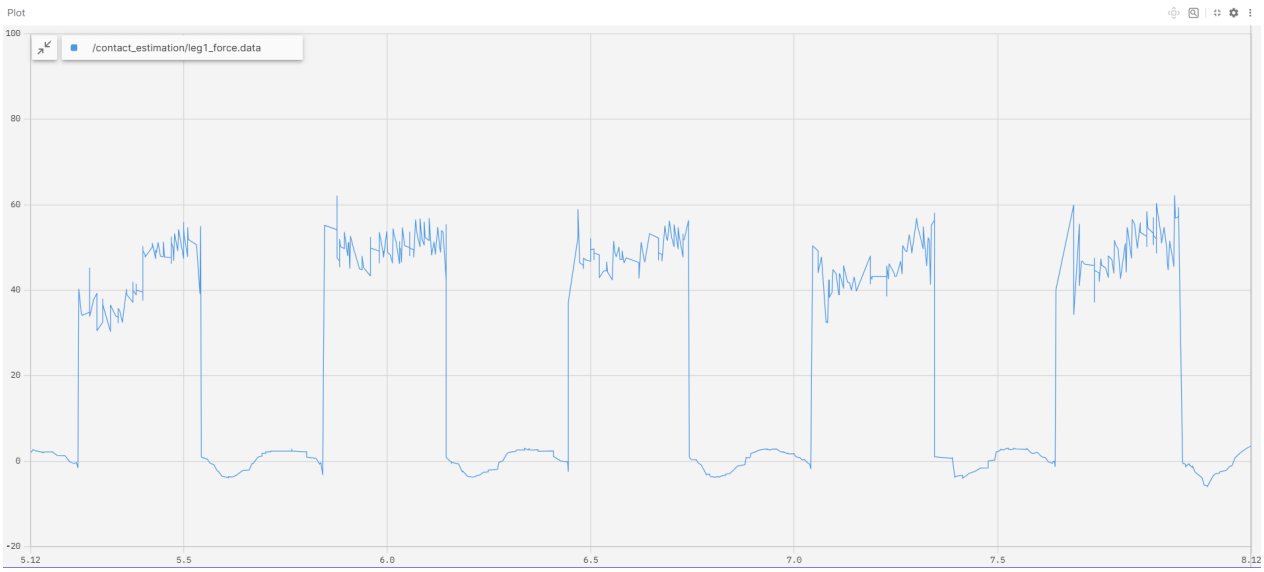


Figure 4.15: Force estimation real robot with 15 HZ cutoff frequency

The difference in contact between the real world and the robot's belief was assessed by taking high-speed video of the robot walking. The robot's onboard lights were used to display the contact belief in real-time. Top lights referred to the front leg, and the bottom lights referred to the back leg on each side. Red light meant no contact was detected, and green light meant contact was detected. The light color was then displayed against the real contact state. It was overall determined that there appeared to be no delay between the real world contact and the robot's contact belief.

Figure 4.16: Robot lights when there is no contact

Figure 4.17: Robot lights when contact is nearly made

Figure 4.18: Robot lights when both legs are in contact

Figure 4.19: Robot lights when contact has been lost

Overall, the robot was highly successful at detecting instantaneous contact. That considered,

improvements could be made when detecting slips. Ongoing experiments aim to improve the implementation of perception as in Section 3.3.6 for slip detection on the edge of stairs. To accomplish this, we compared the foot sensors to the proprioceptive force estimation to identify if the foot sensors could improve the contact estimation. The force sensors were ultimately of a much higher magnitude, more delayed, noisier, and less consistent than our proprioceptive force estimation. Therefore, the force estimation was used instead of the force sensors, rather than including the force sensors in the Kalman filter.

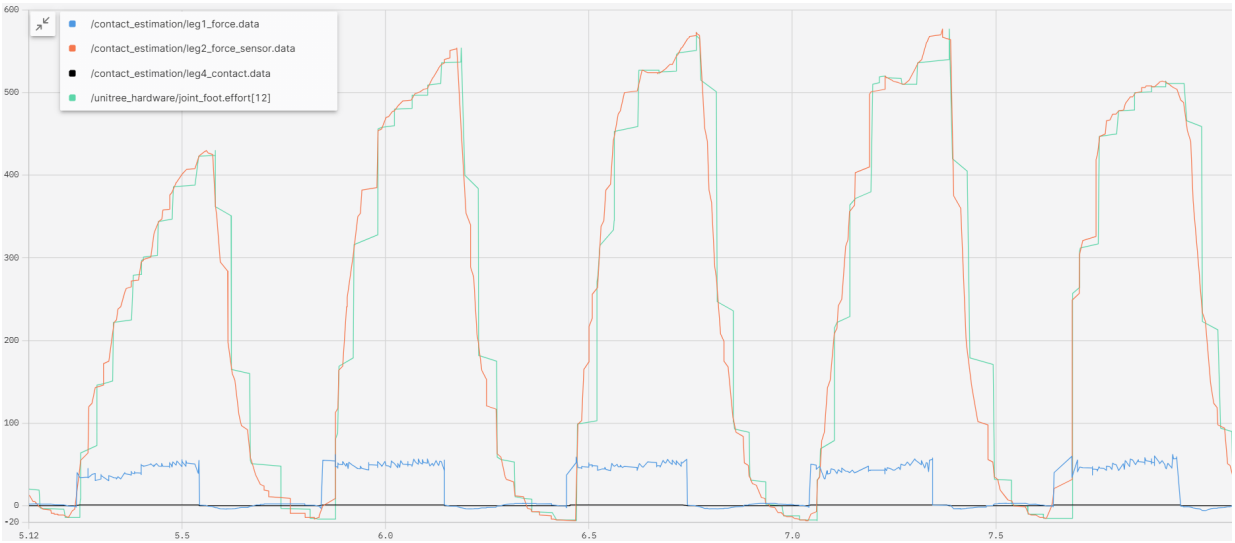


Figure 4.20: The foot force sensors compared against the proprioceptive force estimation

4.3 Stair Climbing

By integrating the improved perception pipeline and perception-aware contact estimation onto the robot, we were able to achieve partial success in climbing repeated inclined steps across various test setups.

4.3.1 Experimental Setup

To systematically evaluate the robot's stair-climbing capabilities, we defined a series of test cases with increasing difficulty. The final on-robot setup used the Livox Mid-360 LiDAR as input for both the elevation mapping pipeline and the FAST-LIO2 SLAM package. We implemented the Legged Control and Legged Perception frameworks [23], incorporating the perception-aware contact estimation detailed in Section 3.4.2.

All trials used the Livox Mid-360 exclusively for both elevation mapping and SLAM. The onboard NUC powered both the control and perception modules, running on the robot's internal battery. Each

trial began with the robot positioned approximately two meters from the first obstacle, ensuring the entire obstacle was captured by the forward-angled LiDAR.

The test environments varied in platform height, depth, and sequence complexity. Here, a "platform" is defined as a flat surface large enough for the robot to place all four legs before ascending further. In all trials, the robot was commanded to walk forward at a constant velocity of 0.4m/s. The setups are as follows (see Figure 4.21 for visual reference):

1. Single platform, half-stair height (3.5 in)
2. Single platform, full-stair height (7.5 in)
3. Multiple platforms, half-stair height (4.5 in)
4. Two-step ascension to a platform, half-stair height (4.5 in)
5. Two-step ascension to a platform, full-stair height (7.5 in)
6. Multi-step ascension, full-stair height (7.5 in)

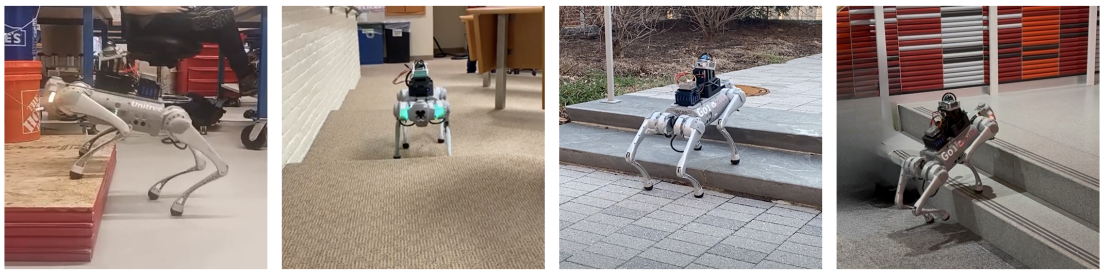


Figure 4.21: Comparison of final trial setups. From left to right: single full-stair platform, multiple half-stair platforms, two half-height sequential platforms, and two full-height steps.

4.3.2 Results and Discussion

The robot successfully completed the following test cases:

1. Single half-height platform (3.5 in)
2. Single full-height platform (7.5 in)
3. Multiple half-height platforms (4.5 in)
4. Two-step ascension to a platform, half-height (4.5 in)

The robot failed in the following test cases:

5. Two-step ascension to a platform, full height (7.5 in)
6. Multi-step ascension, full height (7.5 in)

In the failed cases, the robot was able to ascend the first full-height step but could not place either front foot on the next step. As a result, it attempted to step onto a nonexistent planar region, applying force where no foothold existed, which led to catastrophic failure.

Based on our analysis of the failed trials, we believe the issues encountered in the final two test cases stem primarily from limitations in the current planning architecture. Specifically, the static gait MPC framework appears to lack sufficient solution space when the robot attempts to ascend a second full-height stair in sequence. What tends to happen is the current footstep planner operates on a linear interpolation based on the input velocity and goal position. The linear interpolation has no concept of the boundary area and often places the footstep inside an unsteppable area. The trajectory optimizer then successfully maps the linear footstep to a convex plane but this leaves the robot prone to significant changes when replanning. The most common issue is therefore replanning the swing trajectory while late in climbing, forcing the planned contact onto the previous step. With so little time, the swing leg is then accelerated the joints quickly to reach the location, causing the leg to over-accelerate and trigger its own joint velocity shutdown criteria. Moreover, while the footstep planner always plans contacts on the detected terrain, the swing leg planner does not always seem to adequately satisfy this constraint in the z direction, although it gradually comes to satisfy it as part of its updates. Overall, this causes the robot to rapidly accelerate its joints toward a location hovering over the stairs, believing it can apply force there. Unable to do so, it falls.

We hypothesize that this failure arises due to the inability of the static planner to plan for the changing elevation and limited foothold space of multi-step obstacles. In contrast, a more dynamic planning framework, such as the FPOWR footstep planner described in Section 4.2.2, may be better suited to handle these conditions. FPOWR's ability to generate feasible contact sequences over complex terrain in real time could allow the robot to maintain balance and progress even under tighter foothold constraints.

Despite FPOWR's benefits, continuing to use a linear footstep planner may still be possible. If the swing leg planner is corrected to more strongly force the swing leg onto the z location of a plane we believe it could solve the falling issue. Moreover, the linear footstep planner could be constrained to only update when all the robot's legs are in contact. This would significantly limit the robot's ability to replan if there are large changes in the state estimation while climbing but with the contact estimation improvements, we believe these may not be very necessary.

Additionally, we believe that fine-tuning the resolution of the elevation mapping and the inset margins used in plane segmentation could increase the number of valid foothold candidates available to the planner. Recent experiments suggest that our odometry and perception is good enough that inset margins and even foot placement constraints may not be necessary and instead could be causing the footstep planning issues. Adjustments would expand the feasible solution space, potentially preventing the planner issues observed in sequential stair trials. Overall, we believe that improvements in both the perception and control sections are likely required for robust multi-stair climbing.

Chapter 5

Conclusions

In this work, we have demonstrated a comprehensive framework enabling a Unitree Go1 quadruped to perceive and traverse full-sized stairs in real time. Our key contributions include:

1. Enhanced Perception Pipeline

   By utilizing the Livox Mid-360 LiDAR for elevation mapping and FAST-LIO2 SLAM, we substantially reduced odometry drift and improved terrain mapping accuracy. These improvements enabled the robot to correctly register and ascend obstacles that previously caused missteps, and to have the ability to rely on historical map data if needed.

2. Dynamic Footstep Planning (FPOWR)

   We developed FPOWR, a dynamic footstep planner built on top of the TOWR trajectory optimizer. FPOWR generates kinematically feasible footholds on convex terrain regions, with the ability to provide initial guesses to the MPC for a faster control loop.

3. Perception-Aware Contact Estimation (PACE)

   By fusing generalized-momentum observer outputs with foot height, gait timing, and local map confidence values, our PACE module adapts the probability of contact to the measured terrain beneath each foot. This robust estimation improved foothold reliability, especially on irregular surfaces.

4. Experimental Validation

   Hardware trials verified the system's capability to climb:

   - Single half-height (3.5 in) and full-height (7.5 in) platforms

- Multiple sequential half-height platforms
- Two-step ascent to a half-height platform. However, the static gait MPC struggled when attempting two consecutive full-height steps, slipping on the second ascent due to limited solution space in the current planner

Overall, our framework unites perception, planning, and control to enable real-time stair climbing in complex environments. The results highlight both the efficacy of our pipeline and the remaining challenges, particularly in multi-step full-height ascension, that motivate future enhancements.

Chapter 6

Future Work

6.1 FPOWR Integration with Galileo

One of the reasons for developing FPOWR was to enable the use of custom trajectory optimization frameworks in an MPC context, such as Galileo. Galileo is an open-source trajectory optimization framework that utilizes pseudospectral collocation (See Section 2.2.1 for more details). Specifically, Galileo uses the Legendre-Gauss-Radau (LGR) pseudospectral collocation method [49]. It is a lightweight, extensible, and efficient C++ library developed by the previous year's BiQu team. There is also a updated version of Galileo currently being developed. Galileo is designed to be used in an MPC context and can achieve high-fidelity plans at 50 Hz for 2-second horizons. For these reasons, it would be an excellent candidate to replace OCS2 in the currently employed pipeline, as described in Section 3.2. Making this swap and adding in the additional lower-level controllers used by last year's team (whole body controller, joint controller, state estimator) would result in the system diagram proposed for future work, Figure 6.1. This would be beneficial because it would allow for greater control over the full system, leaving room to implement custom gait and footstep generation.

Another benefit of FPOWR, as mentioned in Section 3.4.1.4, is using the trajectory generated from TOWR as an initial guess for the MPC trajectory optimizer, allowing for faster computation. TOWR uses a single rigid-body dynamic model, which assumes that the legs have negligible inertia (Section 2.2.3). On the other hand, Galileo uses a more complex centroidal momentum dynamic model, which accounts for the inertia of the legs influencing the robot's center of mass. Note that some literature refers to the single rigid-body model as "centroidal dynamics," which is not to be confused with "centroidal *momentum* dynamics."

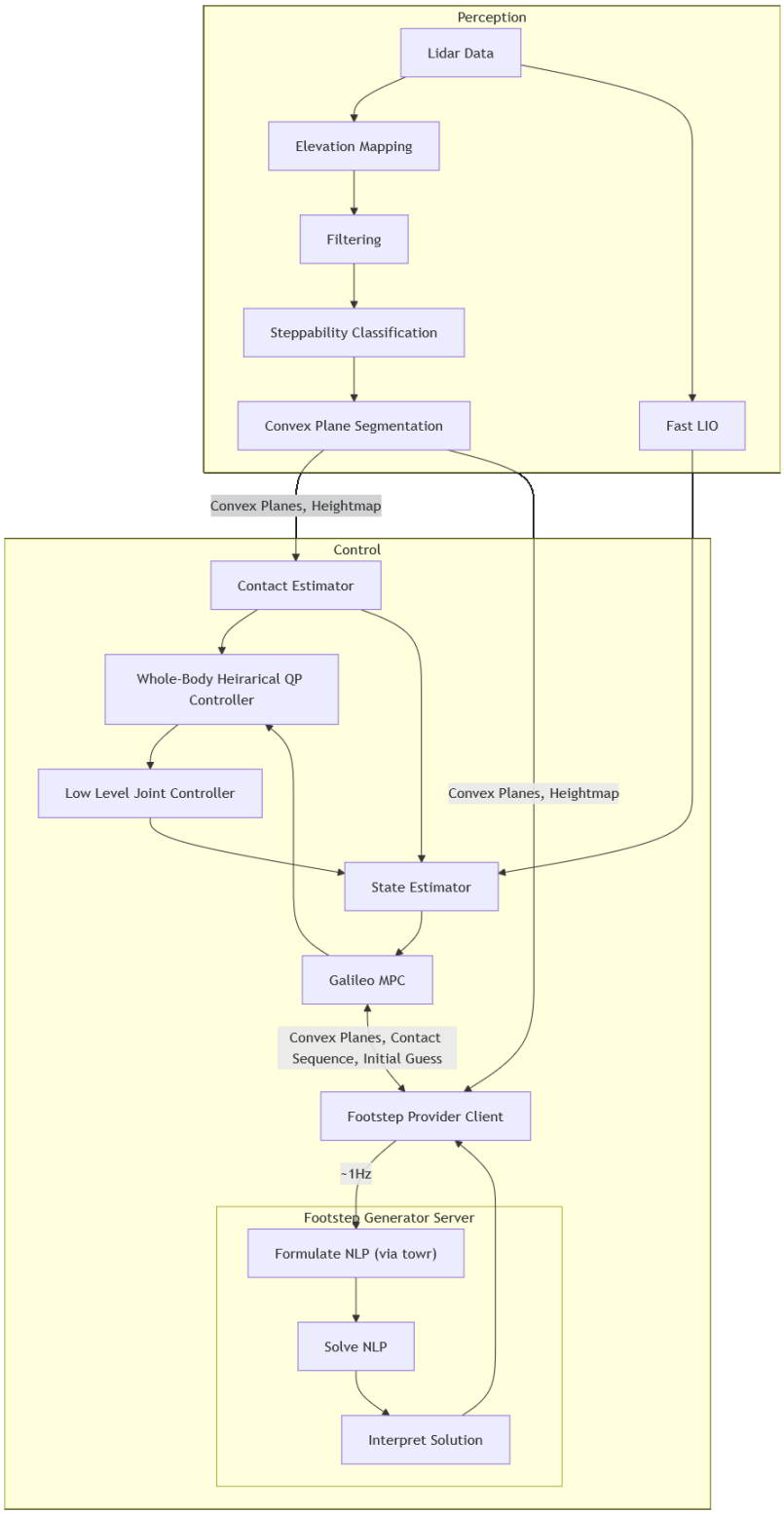


Figure 6.1: Proposed Control and Perception Hierarchy of the Go1

This means that trajectories from towr using the simpler single rigid-body model can be computed quickly and then fed into Galileo to be calculated at a higher fidelity.

6.2 FPOWR Improvements

FPOWR could be improved by adjusting the representation of end-effector bounding boxes. Internally, TOWR represents the region of possible end-effector positions by placing an axis-aligned bounding box (AABB) centered on each end-effector's position in the robot's nominal stance (Figure 6.2). AABBs allow for fast kinematic feasibility checks, but they only represent a subset of the true reachable end-effector positions. For normal use cases, these AABBs represent a large enough subset of the reachable positions to achieve natural motion. However, the AABBs introduce issues during steep stair climbing because they limit the end-effector positions more severely than what is truly kinematically feasible on the robot. Specifically, when attempting to climb staircases greater than 45°, the robot model is unable to place its hind legs far enough back to stabilize itself. This issue is only present within simulation because the robot's hip range of motion is not accurately captured with the AABBs. This issue can be fixed by modifying the mechanism TOWR uses to check for kinematic feasibility.

Additionally, FPOWR could also benefit from improved solver speeds. All footstep plan optimizers need to compromise between model fidelity and solve time. FPOWR has taken the design philosophy of high fidelity, optimizing the reduced order dynamics simultaneously with the footstep plan. Unfortunately this leads to very slow computation times. Internally, TOWR is also capable of optimizing the contact sequence, but it reduces the solve times to slower than real time, severely limiting its use. Speeding up TOWR to the point that contact sequences can also be optimized would provide immense benefit to both FPOWR and BiQu. Another viable option for the future of FPOWR would be to replace TOWR with a stochastic solver or a faster optimization library.

6.3 Retrospective SLAM Fusion with Time-Offset Compensation

Another promising direction for future work involves the retrospective alignment of SLAM odometry from FAST-LIO2. As discussed in Section 2.1.2, we observed that FAST-LIO2 occasionally produced pose estimates with timestamps up to 20 milliseconds in the past, due to its internal processing delay. This discrepancy can introduce a mismatch between the robot's most recent state estimate and the delayed SLAM output, particularly during fast movements. By implementing a retrospective fusion strategy [50],

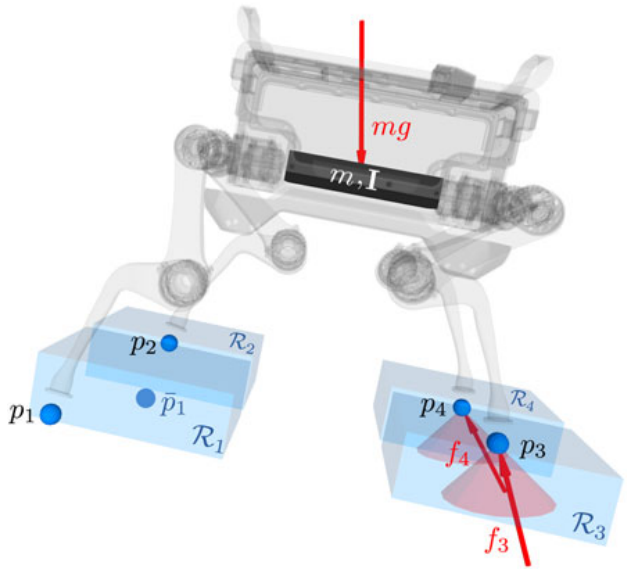


Figure 6.2: TOWR robot model. Shows single rigid body dynamics, foot friction cones, and range of motion constraints overlaid on the ANYmal model for example purposes. [7]

where past robot odometry updates are corrected using the newly available but delayed SLAM poses, we can more accurately align SLAM-derived localization with the robot's real-time motion. This could reduce lag in localization, smooth out discontinuities in the fused trajectory, and improve overall state estimation quality.

6.4 Usage of Internal Robot Components

The Unitree Go1 robot contains multiple resources that are not currently being fully utilized. There are several ways that we could take advantage of the onboard cameras. On startup, the internal SDK runs the robot's cameras and publishes their outputs on the local ROS Master. This output could be transferred to the NUC and integrated into our perception pipeline. Additionally, the SDK could be saved, duplicated, and manipulated to give the users control of whether cameras are off or on. This has already been accomplished with the Go1's face lights, but the camera SDK appears less documented but significantly more valuable. The limited documentation makes the task more difficult, but incredibly beneficial since it would pave the way for people to take advantage of other internal sensors on the Go1 and enable computer vision to develop a better understanding of the environment. Another important robot component that we do not use is the GPU on the robot's main Jetson. Due to the Jetson's software version, setting up a small subsection of the perception pipeline to run on it would not be possible. In order to take advantage of it, the robot would need to be disassembled, and then a user would have to flash the Jetson to a newer version with a compatible

Ubuntu and ROS version from their computer. Then, one could set the internal Jetson up to connect to the NUC automatically, and run the GPU-accelerated Elevation Mapping node. Since the Jetson would not be hindered by any CPU-intensive processes that would be run on the NUC, this could potentially increase the refresh rate of various elements of our pipeline. These potential methods could greatly improve the capabilities of our robot.

6.5 Perception-Aware Contact Estimation (PACE)

PACE needs more data to support its efficacy. While we have successfully demonstrated our ability to incorporate perception data into our probability of contact given height, our dynamic weight of Kalman variances has not been tested, and its ability to detect slips has not been confirmed. We plan to collect more data, develop an experimental method for inducing a slip on the edge of a stair, and tune the Kalman variances for the elements of detection. Additionally, future groups may be interested in using machine-learning to detect high-slip likelihood areas using perception data.

References

[1] Livox Technology Co., Ltd., *Livox Mid-360 User Manual*, 2024. Retrieved from https://www.livoxtech.com/mid-360/downloads.

[2] Intel RealSense, "Depth camera d435i," 2024. Accessed: 2024-12-12.

[3] P. Fankhauser, M. Bjelonic, C. Dario Bellicoso, T. Miki, and M. Hutter, "Robust Rough-Terrain Locomotion with a Quadrupedal Robot," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, (Brisbane, QLD), pp. 5761–5768, IEEE, May 2018.

[4] T. Miki, L. Wellhausen, R. Grandia, F. Jenelten, T. Homberger, and M. Hutter, "Elevation mapping for locomotion and navigation using GPU," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2273–2280. ISSN: 2153-0866.

[5] P. Fankhauser, M. Bloesch, C. Gehring, M. Hutter, and R. Siegwart, "Robot-centric elevation mapping with uncertainty estimates," in *International Conference on Climbing and Walking Robots (CLAWAR)*, 2014.

[6] R. Grandia, F. Jenelten, S. Yang, F. Farshidian, and M. Hutter, "Perceptive locomotion through nonlinear model-predictive control," vol. 39, no. 5, pp. 3402–3421.

[7] A. W. Winkler, C. D. Bellicoso, M. Hutter, and J. Buchli, "Gait and trajectory optimization for legged systems through phase-based end-effector parameterization," vol. 3, no. 3, pp. 1560–1567.

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

[9] DroneBlocks, "Unitree go1 stair climbing mode."

[10] M. Kalakrishnan, J. Buchli, P. Pastor, M. Mistry, and S. Schaal, "Fast, robust quadruped locomotion over challenging terrain," in *2010 IEEE International Conference on Robotics and Automation*, pp. 2665–2670, 2010.

[11] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Science Robotics*, vol. 5, Oct. 2020.

[12] F. Jenelten, T. Miki, A. E. Vijayan, M. Bjelonic, and M. Hutter, "Perceptive locomotion in rough terrain – online foothold optimization," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 5370–5376, 2020.

[13] Y. Cui, R. Chen, W. Chu, L. Chen, D. Tian, Y. Li, and D. Cao, "Deep learning for image and point cloud fusion in autonomous driving: A review," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 2, pp. 722–739, 2022.

[14] O. A. V. Magana, V. Barasuol, M. Camurri, L. Franceschi, M. Focchi, M. Pontil, D. G. Caldwell, and C. Semini, "Fast and continuous foothold adaptation for dynamic locomotion through cnns," *IEEE Robotics and Automation Letters*, vol. 4, p. 2140–2147, Apr. 2019.

[15] Z. Jian, Z. Yan, X. Lei, Z. Lu, B. Lan, X. Wang, and B. Liang, "Dynamic control barrier function-based model predictive control to safety-critical obstacle-avoidance of mobile robot," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3679–3685, IEEE.

[16] H. Xie, C. Cui, X. Zhong, X. Zhong, and Q. Liu, "Real-time support terrain mapping and terrain adaptive local planning for quadruped robots," vol. 9, no. 12, pp. 11018–11025.

[17] K. Ma, Z. Sun, C. Xiong, Q. Zhu, K. Wang, and L. Pei, "IMOST: Incremental memory mechanism with online self-supervision for continual traversability learning,"

[18] M. Liu, J. Xiao, and Z. Li, "Deployment of whole-body locomotion and manipulation algorithm based on NMPC onto unitree go2quadruped robot," in *2024 6th International Conference on Industrial Artificial Intelligence (IAI)*, pp. 1–6, IEEE.

[19] M. Labbé and F. Michaud, "RTAB-map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation," vol. 36, no. 2, pp. 416–446.

[20] I. Hroob, R. Polvara, S. Molina, G. Cielniak, and M. Hanheide, "Benchmark of visual and 3d lidar SLAM systems in simulation environment for vineyards," in *Towards Autonomous Robotic Systems* (C. Fox, J. Gao, A. Ghalamzan Esfahani, M. Saaj, M. Hanheide, and S. Parsons, eds.), vol. 13054, pp. 168–177, Springer International Publishing. Series Title: Lecture Notes in Computer Science.

[21] J. Jorge, T. Barros, C. Premebida, M. Aleksandrov, D. Goehring, and U. J. Nunes, "Impact of 3d LiDAR resolution in graph-based SLAM approaches: A comparative study."

[22] W. Xu, Y. Cai, D. He, J. Lin, and F. Zhang, "FAST-LIO2: Fast direct LiDAR-inertial odometry," vol. 38, no. 4, pp. 2053–2073.

[23] Q. Liao, B. Zhang, X. Huang, X. Huang, Z. Li, and K. Sreenath, "Berkeley humanoid: A research platform for learning-based control," 2024.

[24] M. Bjelonic, R. Grandia, M. Geilinger, O. Harley, V. S. Medeiros, V. Pajovic, E. Jelavic, S. Coros, and M. Hutter, "Offline motion libraries and online MPC for advanced mobility skills," vol. 41, no. 9, pp. 903–924.

[25] G. Bellegarda and K. Byl, "Trajectory optimization for a wheel-legged system for dynamic maneuvers that allow for wheel slip," in *2019 IEEE 58th Conference on Decision and Control (CDC)*, pp. 7776–7781, 2019.

[26] D. Kim, J. Di Carlo, B. Katz, G. Bledt, and S. Kim, "Highly dynamic quadruped locomotion via whole-body impulse control and model predictive control,"

[27] R. Deits and R. Tedrake, "Footstep planning on uneven terrain with mixed-integer convex optimization," in *2014 IEEE-RAS International Conference on Humanoid Robots*, pp. 279–286, IEEE.

[28] B. Aceituno-Cabezas, C. Mastalli, H. Dai, M. Focchi, A. Radulescu, D. G. Caldwell, J. Cappelletto, J. C. Grieco, G. Fernandez-Lopez, and C. Semini, "Simultaneous contact, gait and motion planning for robust multi-legged locomotion via mixed-integer convex optimization," pp. 1–1.

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

[30] P. E. Gill, W. Murray, and M. A. Saunders, "SNOPT: An SQP algorithm for large-scale constrained optimization," vol. 12, no. 4, pp. 979–1006. Publisher: Society for Industrial and Applied Mathematics.

[31] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," vol. 106, no. 1, pp. 25–57.

[32] J. T. Betts, "Survey of numerical methods for trajectory optimization," vol. 21, no. 2, pp. 193–207.

[33] D. Pardo, L. Moller, M. Neunert, A. W. Winkler, and J. Buchli, "Evaluating direct transcription and nonlinear optimization methods for robot motion planning," vol. 1, no. 2, pp. 946–953.

[34] C. Mastalli, R. Budhiraja, W. Merkt, G. Saurel, B. Hammoud, M. Naveau, J. Carpentier, L. Righetti, S. Vijayakumar, and N. Mansard, "Crocoddyl: An efficient and versatile framework for multi-contact optimal control," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2536–2542, IEEE.

[35] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2024.

[36] "GNU Linear Programming Kit."

[37] M. Berkelaar, K. Eikland, and P. Notebaert, "lp_solve: Open source (mixed-integer) linear programming system," 2004. Version 5.1.0.0 dated 1 May 2004. Multi-platform, pure ANSI C / POSIX source code, Lex/Yacc based parsing. Licensed under the GNU LGPL (Lesser General Public License).

[38] D. Song, P. Fernbach, T. Flayols, A. D. Prete, N. Mansard, S. Tonneau, and Y. J. Kim, "Solving footstep planning as a feasibility problem using l1-norm minimization (extended version)," vol. 6, no. 3, pp. 5961–5968.

[39] H. Xue, C. Pan, Z. Yi, G. Qu, and G. Shi, "Full-order sampling-based MPC for torque-level locomotion control via diffusion-style annealing."

[40] Robotic Systems Lab: Legged Robotics at ETH Zürich, "Tutorial: Gait and trajectory optimization for legged robots."

[41] A. W. Winkler, "Ifopt - A modern, light-weight, Eigen-based C++ interface to Nonlinear Programming solvers Ipopt and Snopt.," 2018.

[42] Science and Technology Facilities Council, "Coin-HSL."

[43] Unitree, *Go1Software Manual*. Unitree.

[44] J. Hwangbo, C. D. Bellicoso, P. Fankhauser, and M. Hutter, "Probabilistic foot contact estimation by fusing information from dynamics and differential/forward kinematics," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3872–3878, 2016.

[45] M. Camurri, M. Fallon, S. Bazeille, A. Radulescu, V. Barasuol, D. G. Caldwell, and C. Semini, "Probabilistic contact estimation and impact detection for state estimation of quadruped robots," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 1023–1030, 2017.

[46] M. Maravgakis, D.-E. Argiropoulos, S. Piperakis, and P. Trahanias, "Probabilistic Contact State Estimation for Legged Robots using Inertial Information," 2023. Version Number: 2.

[47] G. Bledt, P. M. Wensing, S. Ingersoll, and S. Kim, "Contact model fusion for event-based locomotion in unstructured terrains," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4399–4406, 2018.

[48] M. Bloesch, M. Hutter, M. A. Hoepflinger, S. Leutenegger, C. Gehring, C. D. Remy, and R. Siegwart, "State Estimation for Legged Robots: Consistent Fusion of Leg Kinematics and IMU," in *Robotics: Science and Systems VIII*, The MIT Press, July 2013. _eprint: https://direct.mit.edu/book/chapter-pdf/2266776/9780262315722_cac.pdf.

[49] D. Garg, M. Patterson, W. Hager, A. Rao, R. D. Benson, and G. T. Huntington, "An overview of three pseudospectral methods for the numerical solution of optimal control problems,"

[50] Y. Sun, F. Jing, and Z. Liang, "Iterated extended kalman filter for time-delay systems with multi-sample-rate measurements," in *Proceeding of the 11th World Congress on Intelligent Control and Automation*, pp. 4532–4536, 2014.