Foot Placement Planning of a Hexapod Robot Moving Over Uneven Terrain

by

Andries Phillipus Lotriet

Thesis presented in partial fulfilment of the requirements for the degree of Master of Engineering (Electronic) in the Faculty of Engineering at Stellenbosch University

Supervisor: Prof. J.A.A. Engelbrecht

December 2024

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: 2024/09/15

Abstract

In recent times great strides have been made in the field of autonomous robotics, especially with regards to autonomous navigation of wheeled and aerial drones. Legged robotics however still face numerous problems before they can become practical to use, the most egregious of these problems being balancing of the robot, and optimal foot placement.

This thesis focuses on providing a solution to the latter problem of foot placement. This is achieved by using a depth camera to, in real time, construct a localised map of the environment, and subsequently analysing said map for optimal foot placement locations. The system is then tested using a hexapod robot, both in simulation and on a physical robot.

Uittreksel

In onlangse tye is groot vordering gemaak in die gebied van outonome robotika, veral met betrekking tot outonome navigasie van hommeltuie. Bebeende-robotika het egter steeds probleme om op te los voordat dit prakties gebruik kan word, die mees ernstige van hierdie probleme is balansering van die robot, en optimale voetplasing.

Hierdie tesis fokus daarop om 'n oplossing vir die laasgenoemde probleem van voetplasing voor te stel. Dit word bereik deur 'n dieptekamera te gebruik om 'n gelokaliseerde kaart van die omgewing te konstrueer, en daarna die kaart te ontleed vir optimale voetplasings areas. Die stelsel word dan getoets met behulp van 'n seskantige-robot, beide in simulasie en op 'n fisiese robot.

Acknowledgments

I would like to thank my supervisor, Prof. J.A.A Engelbrecht for his invaluable support and guidance throughout the course of this project. Additionally, I would like to thank my family and friends for their support and taking a keen interest in my work.

Table of contents

List of figures

List of tables

List of symbols

Constants

α_x	Focal length in x-direction	[302px]
α_y	Focal length in y-direction	[302px]
u_0	Principal point x-coordinate	[313px]
v_0	Principal point y-coordinate	[253px]
γ	Skew coefficient	[0]
$L1$	Hip length	[65cm]
$L2$	Femur lengt	[97cm]
$L3$	Tibula length	[145cm]
$\boldsymbol{T}^{\mathcal{B}}$	Resting feet positions	[cm]
$\boldsymbol{Q}_{\boldsymbol{i}}^{\mathcal{B}}$	Base hip rotation quaternions	[ ]
$\boldsymbol{K}$	Weighting kernel	[ ]
C_p	Proximity score weight	[1]
C_g	Slope score weight	[1]

Variables

$\boldsymbol{p}^{\mathcal{C}}$	Depth buffer projected to 3D camera space	[cm]
$\boldsymbol{p}^{\mathcal{W}}$	Depth buffer projected 3D camera spac	[cm]
$\boldsymbol{q}_{\mathcal{CW}}$	Quaternion rotation of the world frame relative to the camera	[ ]
$\boldsymbol{p}_{C0}^{\mathcal{W}}$	Position of the camera origin	[cm]
$\boldsymbol{p}_r^{\mathcal{M}}$	Robot position in map space	[px]
Δ	Heightmap scaling factor	[ ]
N	Heightmap size	[px]
$\boldsymbol{t}^{\mathcal{L}_i}$	Target foot position in leg space	[cm]

Θ_1	Required hip yaw angle	[rad]
Θ_2	Required hip pitch angle	[rad]
Θ_3	Required knee angle	[rad]
α	Inner hip pitch angle	[rad]
β	Inner knee pitch angle	[rad]
c	Hip to foot direct distance	[cm]
d	Hip to foot horizontal distance	[cm]
$\boldsymbol{p}_f^{\mathcal{L}_i}$	Current foot position in leg space	[cm]
θ_1	Current hip yaw angle	[rad]
θ_2	Current hip pitch angle	[rad]
θ_3	Current knee angle	[rad]
$\boldsymbol{m}^{\mathcal{B}}$	Required move vectors for swinging feet	[cm]
l_{str}	Nominal stride lenght	[cm]
$\boldsymbol{w}_{\text{dir}}^{\mathcal{B}}$	Desired walking direction	[ ]
$\boldsymbol{t}_{\text{nom}}^{\mathcal{B}}$	Nominal feet targets in body space	[cm]
$\boldsymbol{t}_{\text{opt}}^{\mathcal{B}}$	Optimised feet targets in body space	[cm]
$\boldsymbol{t}_{\text{prev}}^{\mathcal{B}}$	Previous, optimised, feet targets in body space	[cm]
$\boldsymbol{t}_{\text{nom}}^{\mathcal{M}}$	Nominal feet targets in map space	[cm]
C_h	Step height, as a function of stride length	[ ]
q	Vertical rise of foot above target position	[mm]
$\boldsymbol{h}_{i,j}$	Heightmap	[cm]

Subscripts

i	Index
nom	Nominal
opt	Optimised
prev	Previous

Superscripts

$\mathcal{W}$	Indicates the world reference frame.[1]
$\mathcal{B}$	Indicates the robot body reference frame.[1]
$\mathcal{M}$	Indicates the map reference frame.[1]
$\mathcal{C}$	Indicates the camera reference frame.[1]
$\mathcal{L}_i$	Indicates the reference frame of leg i.[1]

[1] See appendix A for reference frame definitions.

Notation

$x^{\mathcal{A}}$	Indicates that x is in reference frame $\mathcal{A}$.
$T_{\mathcal{AB}}(x)$	Transforms $\mathcal{A}$ to reference frame $\mathcal{B}$.[2]
$[\![a_0, \ldots, a_n]\!]$	Defines a integer sequence from a_0 to a_n, with a increment of 1.
$\langle \boldsymbol{a}, \boldsymbol{b} \rangle_F$	Takes the Frobenius inner product of matrix $\boldsymbol{a}$ and $\boldsymbol{b}$.

Abbreviations

MuJoCo	Multi-Joint dynamics with Contact
GUI	Graphical User Interface
ROS	Robot Operating System
LiDAR	Light Detection and Ranging
RGB-D	Red Green Blue Depth
SLAM	Simultaneous Localisation and Mapping
IMU	Inertial Measuring Unit
RL	Reinforcement Learing
ANN	Artificial Neural Network
GPS	Global Positioning System
MCU	Microcontroller Unit
GPU	Graphical Processing Unit
CPU	Central Processing Unit
LPS	Local Positioning System
CPG	Central Pattern Generator
VDP	Van der Pool
VIORB	Visual Inertial ORB-SLAM

[2] See appendix A for transform definitions.

Chapter 1

Introduction

1.1 Background

There are many applications where vehicles are required to traverse uneven terrain, such as in mines, rescue operations, agriculture, construction, etc. In many of these use cases uneven terrain makes the use of wheeled, or even tracked, vehicles difficult or impractical.

Compared to wheeled robots, legged robots could perform better in many of these environments, allowing navigation over terrain that would be impossible for wheeled or tracked vehicles to navigate. While legged robots possess extreme degrees of potential terrain traversability, advanced control and sensory systems are required to realise this potential.

1.2 Research Goal

The overarching goal of this project is to design and implement a sensory, and control system that will allow a hexapod robot to autonomously walk over uneven terrain.

This goal of the project is broken up into the following sub objectives:

1. Mathematically model the robot, its actuators, and its sensors.
2. Create a simulation model of the robot in a suitable simulation environment for development and testing.
3. Implement and test a baseline motion control system that enables the robot to perform a normal walking motion over flat, featureless terrain.
4. Develop a real-time vision-based dense mapping system that enables the robot to sense and create a three-dimensional dense map of the surrounding terrain in real time using images obtained from its onboard RBG-D camera.

5. Implement a vision-based Simultaneous Localisation and Mapping (SLAM) system that enables the robot to localise itself relative to the terrain while simultaneously creating the dense map of the terrain.

6. Develop a foot placement planning method that analyses the dense terrain map and determines suitable positions for the hexapod to place its feet while walking over uneven terrain.

7. Extend the baseline motion control system to enable the robot to perform a modified walking motion over uneven terrain.

8. Implement and test the system in simulation and on the physical hexapod robot.

1.3 Methodology

When deciding how to determine optimal foot placement, various sensing methods were considered, such as using a Red Green Blue Depth (RGB-D) camera to view the environment, placing force sensors on the robot's feet or measuring servo torques to determine when the feet are in contact with a surface. Since a previous paper by Erasmus *et al.* (2023) used a RGB-D camera by storing past image snapshots to adjust the feet to the correct height, it was decided that the primary sensing method for this thesis would also be a RGB-D camera. Instead of storing previous images, an onboard heightmap would be generated of the local environment. This would allow for more advanced methods of selecting foot target positions.

The first step in realising this system was to construct a representative simulation of the hexapod. The primary simulation packages that were considered are Gazebo, PyBullet, and MuJoCo. Gazebo was an appealing choice due to its easy integration with Robot Operating System (ROS). However it was decided to use MuJoCo since it was found to have superior contact physics simulation (Erez *et al.*, 2015).

Once the hexapod was adequately modelled in MuJoCo, a baseline motion control system for the hexapod to walk on flat terrain. The baseline control system included a tripod gait state machine, a foot position and trajectory planner, and leg motion controllers based on the inverse kinematics. The control interface allowed a user to command the hexapod to walk at a commanded speed in a commanded direction.

Next, the the system to generate the heightmap from the images taken by the RGB-D camera was implemented. This entailed using the depth information in the RGB-D images to update the occupancy information in the cells of the height map.Once the height map was implemented, the system to perform

foot placement planning on uneven terrain was developed. The foot placement planning system operates by using parallel image processing to generate a walkability score map from the heightmap, which takes into account a cell's steepness and a cell's proximity to steep terrain. The nominal foot positions generated by the baseline motion system are then shifted to a location on the score map with a acceptable walkability score. This shifted position is used as the new, optimised foot target.

The integrated system was first implemented and tested on the simulation model of the hexapod robot in MuJuCo. The mapping component of the system was then implemented and practically tested on the physical hexapod robot.

1.4 Scope and Limitations

The purpose of the system is to perform foot planning to allow the hexapod to move over uneven terrain. The system is not expected to provide obstacle avoidance or high-level path planning to avoid hazardous terrain.

The user interface to the system is a velocity command to the hexapod which specifies the speed and direction in which the hexapod must walk. The hexapod is to walk in a straight line at the velocity commanded by the user, while adjusting its foot placements to compensate for the terrain. The scope of the project does not include a waypoint navigation system. However, if the hexapod is able to follow velocity commands while adjusting its feet for the terrain, the system could easily be extended to also perform waypoint navigation.

The feet must be placed on suitable surfaces that can support the hexapod. Foot placements positions are considered to be suitable if the foot placement area is relatively flat and not too steeply inclined, and the foot will not be placed in close proximity to a steep edge, and the height of the terrain at the intended foot placement position is known. The last requirements is intended to prevent the hexapod from stepping on surfaces it has not seen yet, or from stepping into holes in the terrain.

The hexapod is not expected to identify unstable terrain or hazardous terrain types, such as loose earth or pools of water. This could be the topic of future projects.

The hexapod assumes that the uneven terrain is navigable, and trusts that the user will not steer it to navigate terrain that does not contain any suitable foot placement locations. If the system encounters terrain that the system cannot find valid foot placement solutions for, the robot will simply freeze in place and await an updated velocity command from the operator.

The primary focus of the project is the development of the mapping, terrain scoring, and foot placement planning functions. A secondary focus is

the development of the hexapod simulation model with representative contact physics. For the practical testing, the physical hexapod robot developed by Erasmus *et al.* (2023) was used, and the software that implements the mapping, scoring, and foot placement planning was added. The SLAM function was implemented by using the ORB-SLAM3 software library, which was developed by Campos *et al.* (2021). The ORB-SLAM3 library only performed the sparse mapping necessary for localisation.

It is assumed that the terrain sensing is limited to the images obtained from the RGB-D camera. Other terrain sensing methods, such as contact sensors on the feet, or torque sensors on the servo motors, are not available. If a leg were to collide with terrain due to inaccuracies in the terrain map, the robot would not adjust the leg trajectory.

It is assumed that the hexapod robot is not equipped with an Inertial Measuring Unit (IMU). The stability of the robot on the terrain therefore depends on the accuracy of the heightmap, and the hexapod's pose estimate depends entirely on the visual SLAM system.

1.5 Thesis Outline

Chapter 2 provides a literature review on the methods of control, sensing and simulation used for hexapod robots.

Chapter 3 provides an overview of the physical hexapod robot that was used for the project, and the simulation model that was created to support the development and testing of the system and the modelling thereof. The overview includes the robot's mechanical hardware, onboard computer and sensors, and the simulation environment that was used.

Chapter 4 describes the mapping system that was developed, including the dense height map to represent the local terrain about the hexapod, and the sparse SLAM system to localise the hexapod within the terrain

Chapter 5 describes the baseline motion control system that enables the hexapod to walk over flat, featureless terrain.

Chapter 6 describes the foot placement planning system that was developed to allow the hexapod to walk over uneven terrain. The system includes a terrain scoring system to analyse the terrain and an optimal search algorithm to find suitable foot placement positions

Chapter 7 describes the hardware and software implementation of the system on the physical hexapod robot.

Chapter 8 describes the final simulation tests that were performed on the system and presents the test results.

Chapter 9 provides the conclusion of the research and recommendations for future work.

Chapter 2

Literature review

This chapter provides an overview of past research done regarding the control of hexapod movement and of various sensing methods used. First a brief history of hexapods is presented after which various terrain sensing and adaptation methods are presented.

2.1 Hexapod History

Hexapoda, Greek for "six legs" refers the group of arthropods possessing three pairs of legs. As an example see a flesh-fly in Figure 2.1.

Figure 2.1: A Flesh-fly

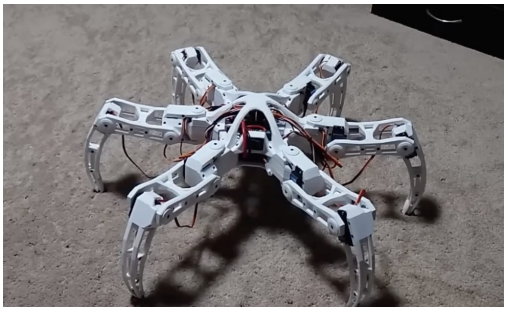


Figure 2.2: A circular hexapod

In the context of robotics, "hexapod" is used to refer to any robot with six legs. The most common configuration of Hexapods are either a rectangular layout with three legs on either side mimicking biological hexapoda, or a circular design with radially symmetrical leg spacing, as seen in Figure 2.2

The hexapod possess the minimum number of legs to allow a naturally stable platform since while taking a step there can be upwards of three anchor points around the center of mass at all times. This makes the hexapod hexa-

pods an ideal platform to navigate complex terrain while maintain stability, without requiring advanced balancing control systems.

For a hexapod to walk, it must swing some of its legs while supporting with others. The number of swinging to supporting legs, and how each is moved, is referred to as the walking "gait". The chosen gait influences the speed and stability of the hexapod. The tripod gait is considered to be the most well rounded, having good speed and stability. In the tripod gait, three legs support, while the remaining three swing. A example of a more stable gait would be the One by One gait, where only one leg is moved at a time.

It is also possible to create a system where there is no predetermined gait, but rather the system determines the optimal legs to support and swing depending on the current walking environment.

2.2 Hexapod Control

Walking over rough terrain requires a control system to correctly actuate the hexapods legs. Various types of control schemes exist. The primary schemes are traditional controllers, bio-inspired controllers and RL. These three schemes are discussed below. Control trends can be seen in the figure below.

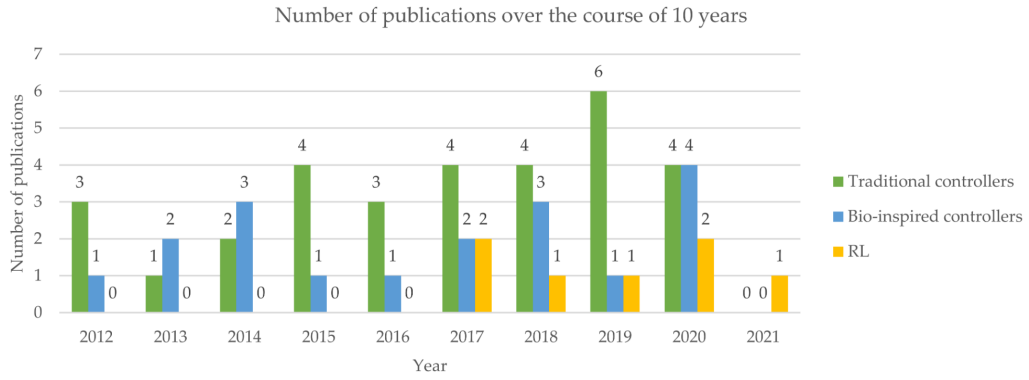


Figure 2.3: Trends of hexapod control schemes (Coelho *et al.*, 2021)

As can be seen from Figure 2.3, traditional and bio-inspired controllers are the most widely used type of controllers, with traditional controllers being the most used. This is mostly due to the well developed theory and knowledge base that exists for traditional controllers. This makes them relatively easy to design, and they result in predictable outcomes. With the recent advancement of machine learning and neural networks, RL control has become much more prevalent. However this technology is still quite new and can be a difficult and

length process to design properly. Predicting exactly what the controller will do is also very difficult.

2.2.1 Traditional

Traditional controllers rely on an exact mathematical model of the robot and inverse kinematics to calculate angular commands for all leg joints. This method of control is purely kinematic and does not take into account external forces applied to the robot, thus it does not inherently adjust to the environment. Isvara *et al.* (2014) used kinematic control with contact sensors to allow the walking gait to be adjusted to uneven terrain. Similarly, Irawan and Nonami (2012) used kinematic control with contact sensors on the robot's feet. The force reading from the contact sensors were also used to control the leg actuation to minimise impact shocks.

Instead of a purely kinematic model, a dynamic model can also be used. When using a dynamic model, the forces acting on the robots legs are taken into account. The forces are usually acquired through torque measurements from servos. By taking applied torque into account, dynamic model controllers will intrinsically detect a deviation when an external force is applied to the robot or its legs, and compensate appropriately. Khudher *et al.* (2017) used this type of control for a hexapod robot designed to aid in humanitarian demining.

2.2.2 Bio-Inspired

Bio-inspired controllers attempt to mimic the neural structure of animals to achieve the same locomotion methods that they use. In mammals, locomotion commands pass through the spinal cord to the Central Pattern Generator (CPG) of each leg, this causes rhythmic excitations of the muscles. In robotics, this is implemented by having a high level controller, as the brain, and a lower level Artificial Neural Network (ANN), as the CPG (Guo *et al.*, 2019). If implemented successfully a bio-inspired controller can be highly adaptable to the surrounding environment and is even able to adapt to damaged or missing legs. Yu *et al.* (2013) used this structure, with a CPG network that excites six Van der Pool (VDP) oscillators. A lower level converter was used to translate the oscillators' output into angle commands for the servos. Other types of oscillators could also be used, such as the Hopf oscillator (Chen *et al.*, 2012), which is a simpler type of oscillator. Additionally, using spiking neurons instead of oscillators is also an option, (Gutierrez-Galan *et al.*, 2020) did this to improve computational efficiency.

2.2.3 Reinforcement Learing (RL)

Reinforcement Learing (RL) controllers are created through using trial and error to construct a neural network that minimises a cost function for a specific goal. This theoretically allows RL controllers to adapt to any circumstances given enough time, allowing a very high level of autonomy, as no prior direction is required. RL controllers are notoriously difficult to train properly though, especially as the number of sensors and control outputs grow large, increasing the feature space. Additionally, even the most well-trained RL agent still has the possibility to exhibit unexpected behaviour. Liu *et al.* (2019) used the Monte Carlo method to optimise the gait strategy for walking over uneven terrain. While Verma *et al.* (2019) proposed a method to use a supervised learning neural network to continue operation while damaged. The system would self analyse damages, and then a gait policy would be found to work with the damages.

2.3 Terrain Sensing

No matter the control scheme used, to know where to place its feet the robot requires sensors to sense its environment in some way. This could be achieved through simple sensors, such as servo torque or touch. Zha *et al.* (2019) used contact sensors on the hexapod's feet to detect when a foot was placed on the ground. If a leg was unable to make contact with the ground, it would use the leg like a insect's feelers, searching in an area until the ground was found.

More advanced methods, such as machine vision or Light Detection and Ranging (LiDAR), are also used. Homberger *et al.* (2017) used stereoscopic vision to adjust end effector height and to classify surface materials. Mastalli *et al.* (2020) used vision based sensors to map and classify which parts of the terrain could be safely stepped on based on its shape.

2.4 Localisation

Depending on the terrain navigation system it might be required to localise the robot in 3D space. For this it is possible to use external sensors such as a type of beacon (RF, Reflective, Ultrasonic) or Global Positioning System (GPS). Internal sensor could also be used, such as cameras and IMUs combined with a SLAM system.

Substantial work has been done regarding SLAM systems. Some of the recent algorithms include ORB-SLAM3, VIORB and RGBDSLAMv2. RGBD-SLAMv2 is a popular RGB-D SLAM system. It uses the RANSAC algorithm to estimate translations from the extracted features. The ICP algorithm is then used to perform a pose estimation (Macario Barros *et al.*, 2022).

Visual Inertial ORB-SLAM (VIORB) is a algorithm that is based on the older ORB-SLAM algorithm (Mur-Artal and Tardós, 2017). Similar to ORB-SLAM, VIORB has three main threads: tracking, local mapping and loop closing. The tracking thread estimates pose, velocity and IMU biases. The loop closing thread is used to identify keyframes that have previously been observed.

ORB-SLAM 3 is a combination of the ORB-SLAM and VIORB algorithms, it additionally maintains a sparse map, called the atlas. The atlas contains active and dormant features. The active feature map is used by the tracking thread, while the dormant feature map is used for relocalisation (Macario Barros *et al.*, 2022).

2.5 Motion Over Uneven Terrain

As hexapods are inherently stable, they are good candidates for use in uneven terrain, and this tends to be where much hexapod research is focused. Such as in Homberger *et al.* (2017), which uses a stereoscopic camera to extract features and classify terrain by factors such as material type, slope and granularity. These parameters are then used to classify various gait parameters, such as step height and dampening, to improve stability while walking.

This is similar to Xu *et al.* (2023), which aims to classify terrain based on material properties, however, Xu *et al.* (2023) focuses on characterising terrain through touch sensors instead of visual sensors. This is achieved through the use of machine learning.

Moving away from gait optimisations and local terrain classification, Prágr *et al.* (2019) aims to characterize large areas of terrain in terms of navigational efficiency through the use of gaussian processes and a incrementally constructed spatial map. An example of this exploratory traversability classification can be seen in figure 2.4.

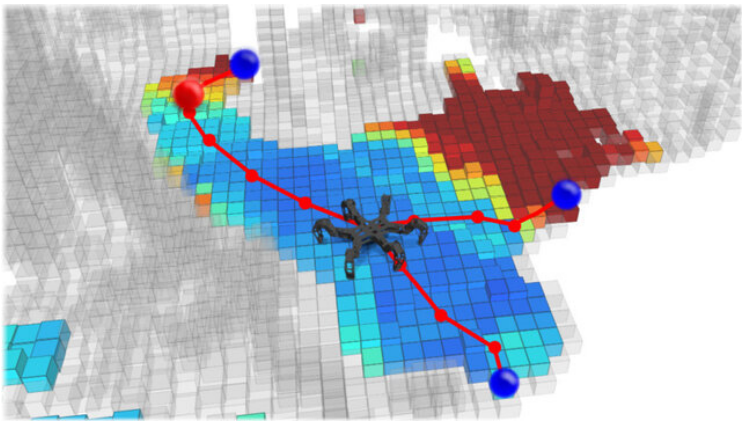


Figure 2.4: Exploratory terrain traversability classification from Prágr *et al.* (2019)

While this paper is on hexapods, it should be noted that much work has also been done on quadrupeds. For example Mastalli *et al.* (2020) uses various sensors to assign a cost to surrounding terrain, thus constructing a cost map, this cost map is then used to select optimal foot end positions for the next step. This can be seen in figure 2.5. Of course, being a quadruped, in addition

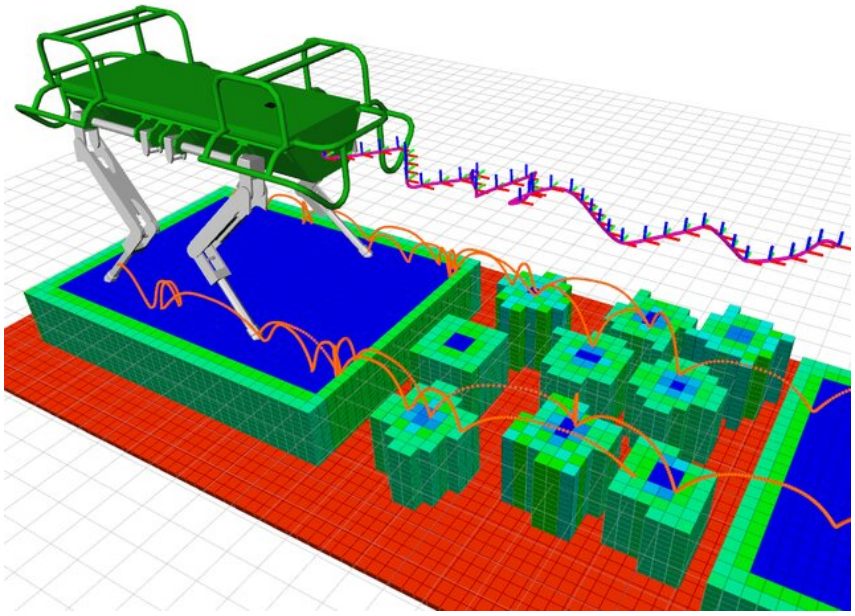


Figure 2.5: Quadruped terrain scoring and navigation from Mastalli *et al.* (2020)

to finding optimal foot end positions, significant consideration had to be given to maintaining the stability of the robot, something that is much less of a concern with hexapods.

2.6 Simulation Environment

The most popular physics simulators for robotics in recent times are Gazebo, MuJoCo, and CoppeliaSim (previously V-REP) (Collins *et al.*, 2021). Gazebo and CoppeliaSim both have easy to use Graphical User Interface (GUI) interfaces and easy integration with ROS. MuJoCo on the other hand does not have a full GUI interface, only a simulation viewer, and does not have native ROS integration. Having said this, MuJoCo was found to be the most accurate and fastest simulator when considering the use case of robotics (Erez *et al.*, 2015).

2.7 Research Decisions

Since this project focuses on using vision-based mapping, terrain classification and the optimisation of foot end positions, it was decided that traditional kinematic control will be used, as it is the simplest and most predictable method of control. This allows the focus to remain on terrain classification and foot placement optimisation.

The camera that will be used is the Intel Realsense D435i RGB-D camera, incorporating stereoscopic cameras and a LiDAR sensor. This camera also has a extensive existing codebase and support that ensures accurate depth readings are attained.

A system to localise the robot within its environment is required. As the primary sensor used is an RGB-D camera, various visual SLAM systems were considered. ORB-SLAM 3 is a relatively modern SLAM algorithm that incorporates many past SLAM algorithms into its design. Thus, it was decided to use ORB-SLAM3.

Finally, a simulation environment was chosen. MuJoCo has exceptional contact physics simulation and the only relevant disadvantages are its lack of native ROS integration and its lack of a comprehensive GUI. However, seeing as MuJoCo has good python bindings this could be seen as a advantage. MuJuCo was therefore chosen as the simulation environment.

Chapter 3

System Overview

This chapter provides a high level overview of the hexapod, starting with the hardware, then the software, and finally the simulation environment.

3.1 Hardware

The physical hexapod is primarily the same hexapod described in Erasmus *et al.* (2023), this robot is shown in figure 3.1. For computation, a JetsonNano and Teensy2.0 Microcontroller Unit (MCU) is used. Actuation of the six, three degrees of freedom, legs is handled by 24 Dynamixel RX-64 servos. Sensing is handled by a Realsense D435i RGB-D camera, which includes a IMU (However the IMU is not utilised in this project.) Two Marvelmind ultrasonic Local Positioning System (LPS) beacons are also present on the robot, although these are not used either.

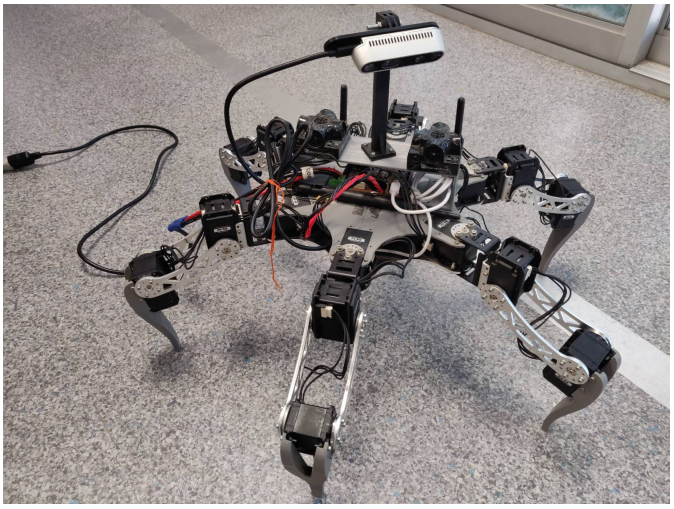


Figure 3.1: Physical Hexapod

The only alterations made from the version describe in Erasmus *et al.* (2023) is the thickening of the wires powering the JetsonNano, to prevent a voltage drop, and the replacement of the 3D printed legs with laser-cut aluminium legs, to prevent leg flexure.

A power cable is shown running to the left and going off image. All tests were conducted using 14.8V from a bench power supply plugged into the battery port. This was done for convenience and can easily be swapped for a battery.

3.2 Software

The most basic software flow for a robot walking over flat terrain is shown in figure 3.2. This system does not sense its environment in any way and simply moves its feet along predetermined paths.

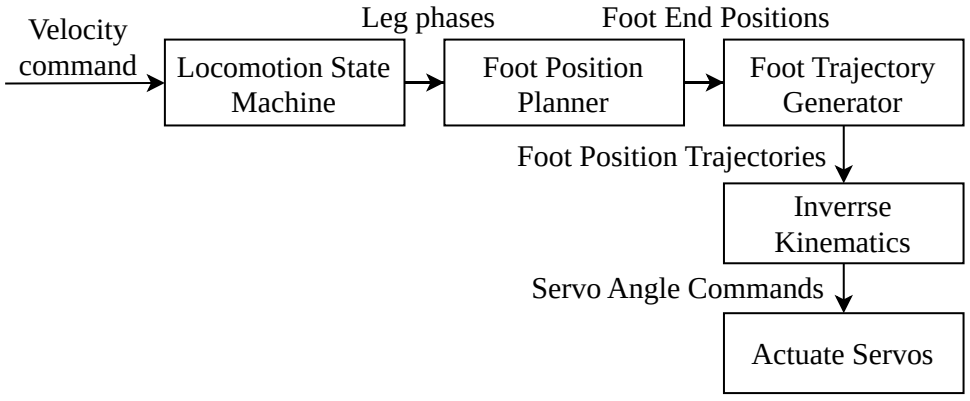


Figure 3.2: Basic motion system operation.

This basic system will work well enough for walking over flat terrain, but will struggle once any deviation in terrain height is present. Thus, the proposed, more advanced system, operates with the flow shown in figure 3.3.

The advanced system uses similar components as the basic system, but with the foot end position planner modified to incorporate checks against a score map and a height map to validate, and if necessary, adjust the foot end positions to place the feet at suitable positions in the terrain. If no suitable adjusted foot positions and trajectories can be found given the terrain, then the hexapod does not execute the motion and freezes in place. The human operator or high-level guidance system must then change the overall path of the hexapod. However, this is outside the scope of the current project.

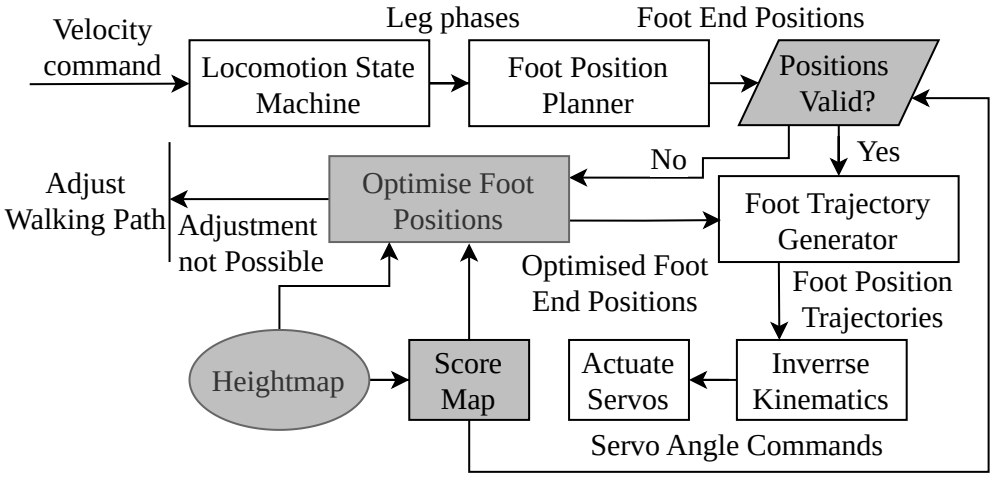


Figure 3.3: Advanced motion system operation.

The goal of maneuvering uneven terrain is achieved by a combination of 4 primary systems, namely a mapping, foot placement optimisation, motion control and a localisation system. A high-level overview of the system implementation can be seen in figure 3.4.

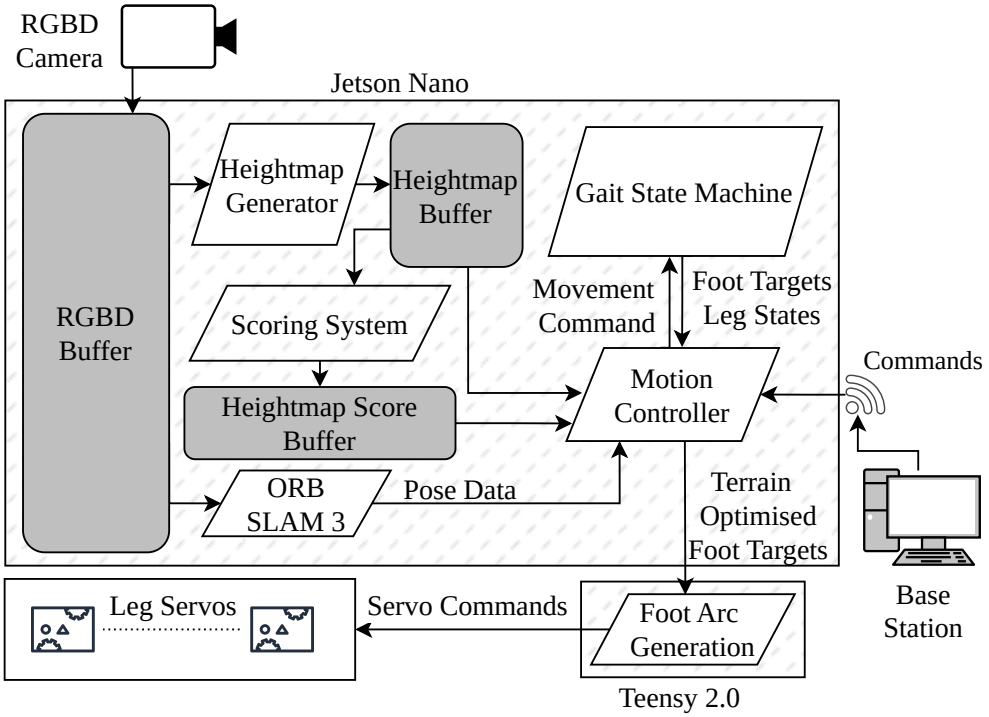


Figure 3.4: Physical system diagram

The mapping system utilises the RGB-D camera to construct a dense heightmap of the immediate surroundings of the robot. As the robot moves around, old data is erased to make way for new data. The size and resolution of the heightmap is adjustable to the available memory and computational power. The heightmap system is further covered in chapter 4.

The foot placement optimisation system takes the heightmap as input and produces another map of equal size to the heightmap. This new map is the score map and is found by assigning a score to each cell of the heightmap. The score is dependant on how stable a position the cell would be for the robot to place its feet. The score map can then be used to evaluate, and adjust if necessary, the initial foot placement proposed by the motion system. The foot placement optimisation system is further covered in chapter 6.

All the movement of the robot is handled by the motion system, it is comprised of a gait state machine, a foot end position planner, a foot trajectory generator, and inverse kinematics. The gait state machine selects the swinging and supporting legs during for each step to achieve a tripod gait. The foot position planner calculates the nominal foot end positions based on the stepping parameters, namely stride length and step height, but without taking the terrain into account. Simple linear motion is not acceptable for the swinging feet, thus the foot arc generator produces a movement vector based on the remaining distance to a foot's destination, which if followed, results in an arc-like motion to the destination. Finally to execute any movements, positions must be converted to servo angle commands, and the foot velocities must be converted to servo angular rates, using the inverse kinematics equations. The motion system is covered in more detail in chapter 5.

3.3 Simulation

As said in section 2.6, various simulation environments were considered, but finally MuJoCo was chosen due to its excellent contact physics simulation. The simulation of the hexapod includes the 24 servos, simulated as high gain, high damped angle controlled motors. The RGB-D camera is also simulated as a direct OpenGL rendering of the simulation environment. As the camera is a OpenGL rendering, all standard buffers used for rendering is generated, this includes the depth buffer. A SLAM system does not run in the simulation, rather simulated estimation noise, based on Macario Barros *et al.* (2022), is added to the true position and orientation directly taken from the simulation.

The software running on the simulation is largely equivalent to that running on the physical system, with only slight modification to integrate with the simulation instead of the hardware. Figure 3.5 shows a screenshot of the simulation environment

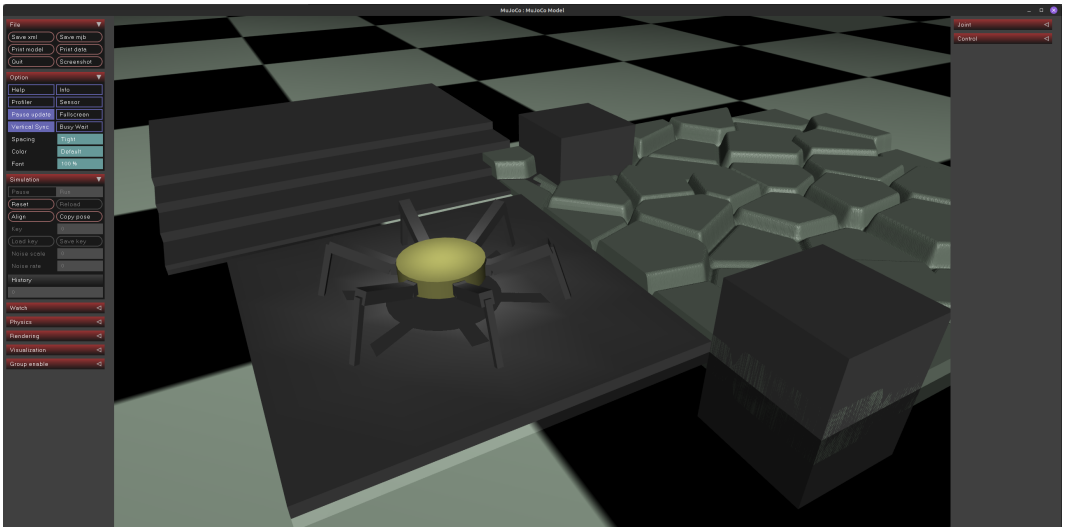


Figure 3.5: The MuJoCo simulation environment

Chapter 4

Mapping

This chapter covers the development of the mapping system, starting with the process of projecting the depth image from the RGB-D camera into 3D space. After this, placing said 3D points into the heightmap buffer is covered. Finally, the Graphical Processing Unit (GPU) implementation of the heightmap processing is described.

4.1 Overview

For accurate foot placement and localisation purposes, the robot makes use of two maps, a sparse map covering a large area, and a dense map covering a small area around the robot. The primary use of the sparse map is for localisation and extracting pose data, namely position, orientation, velocity, and angular rate, while the dense map is used to analyse the terrain and find an appropriate point to place the three swinging feet, which will become the supporting feet for the next half-cycle. While it is also possible to use the sparse map for autonomous path planning and navigation of the environment, this use case is outside the scope of this project.

The localisation, sparse mapping, and pose estimation is performed by ORB-SLAM3 as described in Campos *et al.* (2021). The ORB-SLAM3 system was implemented on the hexapod robot by building UZ-SLAMLab's ORB-SLAM3 library, which can be found on github. A brief, user perspective overview of ORB-SLAM is given in section 4.5.

The dense mapping is performed by downsampling and projecting the RGB-D camera's depth image into 3D space, and using the pose acquired from ORB-SLAM3 to shift and rotate the projected points into the heightmap's reference frame. Once properly aligned to the map each projected point's z value is written into its corresponding heightmap cell.

4.2 Projecting the Depth Image to Map Space

In order to generate a heightmap from a RGB-D image, the 2D depth image must first be projected into the 3D camera reference frame. The 3D points in the camera reference frame can then be converted to 3D points in the heightmap reference frame using the pose estimate obtained from ORB-SLAM3. Finally, the 3D points in the heightmap reference frame are used to updated the 2.5D heightmap.

The camera can be described by its intrinsic and extrinsic parameters. Extrinsic parameters characterise the cameras position in 3D space, and intrinsic parameters characterise the relationship between the image plane and 3D space, assuming that the camera reference frame is not displaced or rotated relative to the world reference frame (Hartley and Zisserman, 2003).

Refer to figure 4.1 as a visual aid regarding projection. Note that this figure is drawn from the perspective of projecting from the image plane into the world. If the objective was to project from the world onto the image plane, the projection center and image plane would swap places, causing the image to be inverted. Thus, this figure assumes that the image rotation has been corrected.

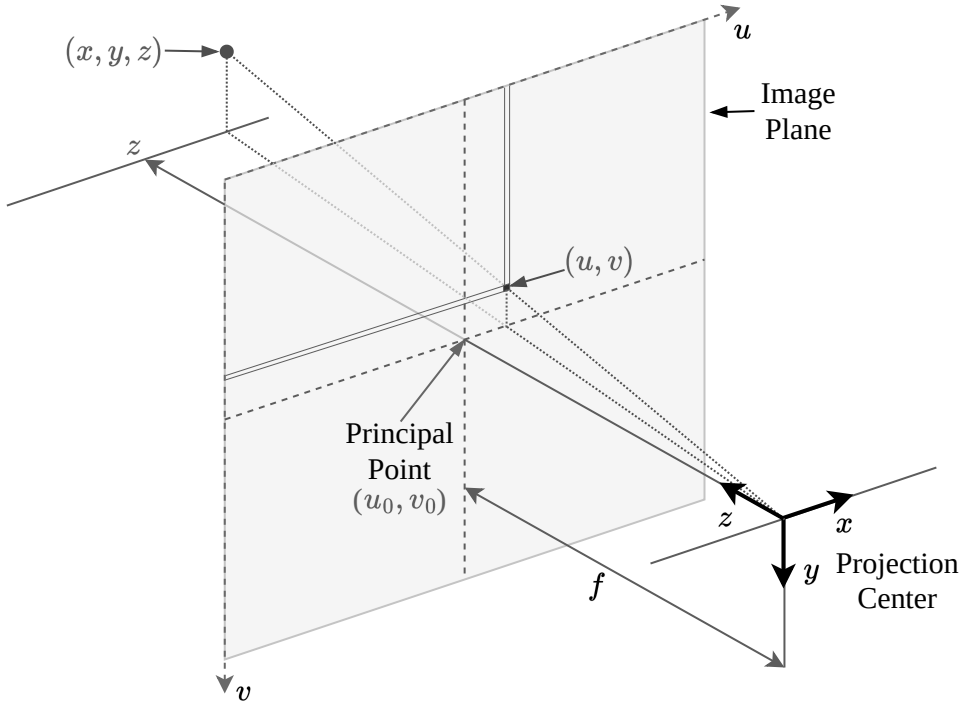


Figure 4.1: Camera Projection

Together the extrinsic and intrinsic matrices form the projection matrix, $\mathbf{P}$, which is defined as

$$\boldsymbol{P} = \boldsymbol{K} \begin{bmatrix} \boldsymbol{R} & \boldsymbol{T} \end{bmatrix} \tag{4.1}$$

where $\boldsymbol{K}$ is the intrinsic matrix and $\begin{bmatrix} \boldsymbol{R} & \boldsymbol{T} \end{bmatrix}$ is the extrinsic matrix. $\mathbf{R}$ and $\mathbf{T}$ represent the rotation and translation relative to the world. The project matrix can be used to project a point in the 3D world space into the 2D image plane, as follows:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \boldsymbol{P} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{4.2}$$

where u, v are the pixel coordinates on the image plane and x, y, z are the coordinates in world space.

The intrinsic matrix $\mathbf{K}$ is defined as,

$$\boldsymbol{K} = \begin{bmatrix} \alpha_x & \gamma & u_0 \\ 0 & \alpha_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.3}$$

with the focal lengths in pixels represented by,

$$\alpha_x = f \cdot m_y$$

$$\alpha_y = f \cdot m_x$$

where m_x and m_y are the inverse of the width and height of a image plane pixel, f is the focal length in millimeters, and (u_0, v_0) is the principal point, ideally located at the center of the image plane. The skew coefficient, γ, is often, and in this case, 0.

The extrinsic matrix is defined below

$$\begin{bmatrix} \boldsymbol{R} & \boldsymbol{T} \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}_{3\times3} & \boldsymbol{T}_{3\times1} \\ \boldsymbol{0}_{1\times3} & 1 \end{bmatrix} \tag{4.4}$$

where $\boldsymbol{R}$ characterises the camera's rotation relative to the world frame and $\boldsymbol{T}$ is the position of the world frame's origin expressed in the camera coordinate frame.

For ease of preprocessing, the 2D points in the camera's image plane are first projected into the 3D camera coordinate frame. In other words, the extrinsic matrix is omitted from equation 4.2. The projection from the 3D camera frame onto the 2D image plane is therefore given by

$$\begin{bmatrix} u \\ v \\ 1 \\ 1/z \end{bmatrix} = \frac{1}{z} \begin{bmatrix} \alpha_x & 0 & u_0 & 0 \\ 0 & \alpha_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{4.5}$$

Rearranging this equation, the corresponding 3D point in the world frame can be calculated from the 2D point in the image plane, given that the depth value for the 2D point is also available from the RGB-D image, as follows:

$$z = I_{u,v} \tag{4.6}$$

$$x = \frac{z(u - u_0)}{\alpha_x} \tag{4.7}$$

$$y = \frac{z(v - v_0)}{\alpha_y} \tag{4.8}$$

where $I_{u,v}$ is the depth image value at pixel coordinates u, v. For later use the 3D position $\boldsymbol{p}^{\mathcal{C}}$ in the camera frame is defined as

$$\boldsymbol{p}^{\mathcal{C}} = \begin{bmatrix} x & y & z \end{bmatrix}^T \tag{4.9}$$

The 3D positions in the camera frame can be converted to the corresponding 3D positions in the map frame, as follows:

$$\begin{aligned} \boldsymbol{p}^{\mathcal{W}} &= T_{\mathcal{CW}}(\boldsymbol{p}^{\mathcal{C}}) \\ &= \boldsymbol{q}_{\mathcal{CW}} \cdot \boldsymbol{p}^{\mathcal{C}} \cdot \boldsymbol{q}_{\mathcal{CW}} - 1 + \boldsymbol{p}^{\mathcal{W}}_{C0} \end{aligned} \tag{4.10}$$

where $\boldsymbol{p}^{\mathcal{W}}$ is the position in the world frame, $\boldsymbol{q}_{\mathcal{CW}}$ is the quaternion rotation of the world frame relative to the camera frame, and $\boldsymbol{p}^{\mathcal{W}}_{C0}$ is the position of the camera frame's origin in the world frame.

4.3 Updating the Heightmap

Once the depth image is projected into map space, their x and y coordinates are used as indices to write their z value into the heightmap. The heightmap is stored in a 2D circular buffer, this means that as the robot moves around, old map data is erased to make room for new data. This structure is illustrated in figure 4.2.

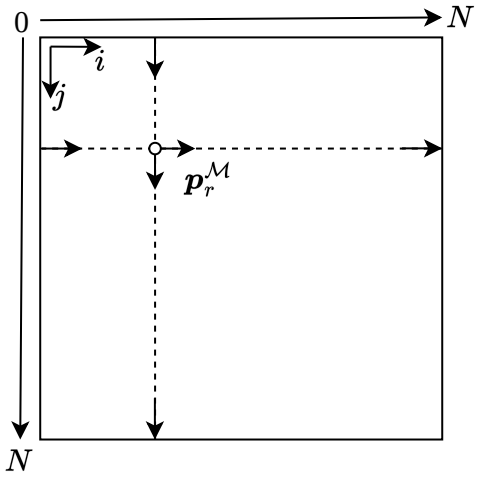


Figure 4.2: 2D Circular Buffer that stores Heightmap

The heightmap buffer, $\boldsymbol{M}$, is of size $N \times N$ and is indexed by $[i, j]$. $\boldsymbol{p}_r^{\mathcal{M}}$ represents the robot's current position in the heightmap, note how this position wraps around when reaching the bounds of the buffer. The indices i and j are calculated as follows:

$$\begin{aligned} i &= \lfloor x^{\mathcal{W}} \Delta \rfloor \mod N \\ j &= \lfloor y^{\mathcal{W}} \Delta \rfloor \mod N \end{aligned} \tag{4.11}$$

where $[x^{\mathcal{W}}, y^{\mathcal{W}}]$ are the horizontal coordinates in the world frame, Δ is the scaling factor of the heightmap, and N is the side length of the 2D array that stores the heightmap.

4.4 GPU Compute Pipeline

As the heightmap generation and heightmap scoring systems are essentially image manipulation processes, parallelisation of the algorithms is a very efficient way to increase computational speeds. Therefore, these algorithms are run in parallel on the Jetson nano's GPU using OpenGLcompute shaders. This section describes the compute pipeline used to build and score the heightmap. The compute pipeline can be seen in figure 4.3.

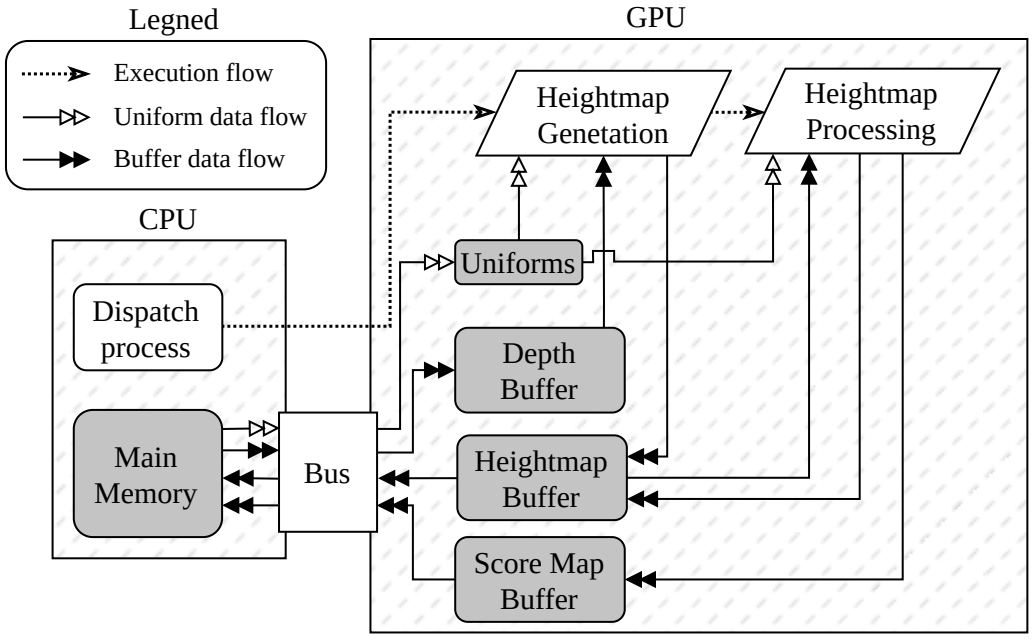


Figure 4.3: Compute pipeline.

As seen from figure 4.3, the GPU pipeline is relatively simple, being comprised of only two stages, the heightmap generation stage and the heightmap processing stage. The heightmap generation stage executes for each pixel on the depth image, constructing the heightmap as described in the previous section.

After the heightmap has been generated, the heightmap processing stage operates over the heightmap buffer. This stage has two tasks, to erase the old height data as the robot walks around, and to generate the score map, which will be described in chapter 6.

4.4.1 A note on GPU architecture

A process on a GPU operates fully in parallel and the GPU is highly optimised for parallelisation. Thus, there is a very specific execution structure that a GPU process must abide by to perform at maximum efficiency, or to even function at all. This execution structure is as follows: As per NVIDIA (2023), when writing compute code, a 3D size is specified, namely the localgroup size, $\boldsymbol{N}_l = [X_l, Y_l, Z_l]$. Next when the Central Processing Unit (CPU) dispatchesa compute task, the workgroup count, is fed as parameter, $\boldsymbol{n}_w = [x_w, y_w, z_w]$. The GPU then initialises $\boldsymbol{n}_w$ workgroups, and each workgroup initialises $\boldsymbol{N}_l$ threads. These threads are executed into warps, or waves, which, depending on the architectures processor count,can be either 32 or 64 threads. To ensure maximum efficiency it is important that $\boldsymbol{N}_l$ is divisible by the warp size. Originally, Nvidia GPUs utilised a warp size of 32 and AMD GPUs a warp size of 64, however with AMD's latest RDNA architecture, the warp size could be either 32 or 64. For the mapping system developed in this project, a warp size of 32 was used.

For the heightmap generation stage, a localgroup size of $\boldsymbol{N}_l = [32, 32, 0]$ was used for a total of 1024 threads per workgroup, or in other words, 32 warps per workgroup. As for the heightmap processing stage, a localgroup size of $\boldsymbol{N}_l = [32, 32, 32]$ was used, meaning 32768 threads, or 1024 warps per workgroup. As such, it is important that the camera images, the heightmap, and the score map are of a size divisible by 32.

While threads can directly communicate with each other within the same workgroup, direct communication across workgroups is impossible. If cross workgroup communication is desired, this must be performed using GPU buffers, which are slower to access. Therefore, the processes were designed to operate fully independently from each other. Of course, data transfer between the CPU and the GPU is orders of magnitudes slower than accessing local buffers. For this reason, all communication between the CPU and GPU is kept to a minimum.

Finally, it should be noted that, while it is possible to vary $\boldsymbol{n_w}$ with each execution cycle, $\boldsymbol{N_l}$ is a constant specified at compile time, and as such the number of threads per workgroup cannot be altered during operation.

4.5 Simultaneous Localisation and (Sparse) Mapping

This section provides a brief overview of the implementation and usage of ORB-SLAM3 from a user perspective.

ORB-SALM3 operates by detecting features in the image streams coming from the connected camera. These features are then used to build a sparse feature map of the environment, subsequent features are used to expand the map. When trying to determine the camera's pose, the current image frame(s) of the camera is compared against the generated feature map, if the features detected in the current images are successfully matched to that in the map, the pose, relative to the feature map's origin, is returned as the robot's (or any other use case) current pose estimate.

In this project ORB-SLAM3 is executed as a separate ROS node. This isolates ORB-SLAM3 from the rest of the system maximising modularity. The ROS node is subscribed to the color and depth streams originating from the RGB-D camera. The color and depth streams are used as inputs to the tracking function, which performs localisation. The tracking function then outputs the pose estimate, which is immediately published, with the same timestamp as the depth image attached. This is to ensure that the pose estimate and depth image used in generating the dense-map are synchronised.

4.6 Simulated Mapping Test

The mapping system decribed above was tested in the MuJoCo simulation environment. This mapping test aimed to show how the hexapod maps terrain while in motion. Figure 4.4 shows the terrain that was used in this test.

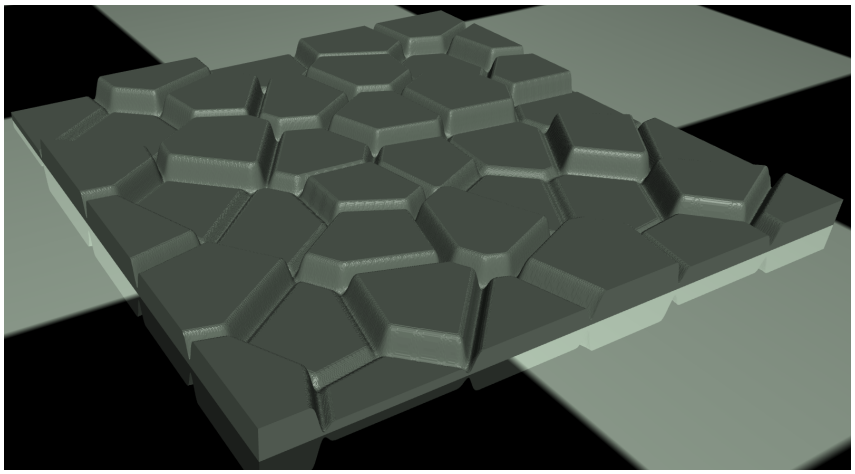


Figure 4.4: Simulated terrain used for the test.

The test consisted of the robot walking past a cobblestone-like terrain, while simultaneously generating a heightmap. Figures 4.5 to 4.7 show how the camera view and associated heightmap generated at the start, middle and end of the test. Please see this video of the simulated test.

As can be seen from figure 4.5 the map is initially empty, with only a small portion of the cobble visible. Also note that the hexapod legs are filtered out of the heightmap.

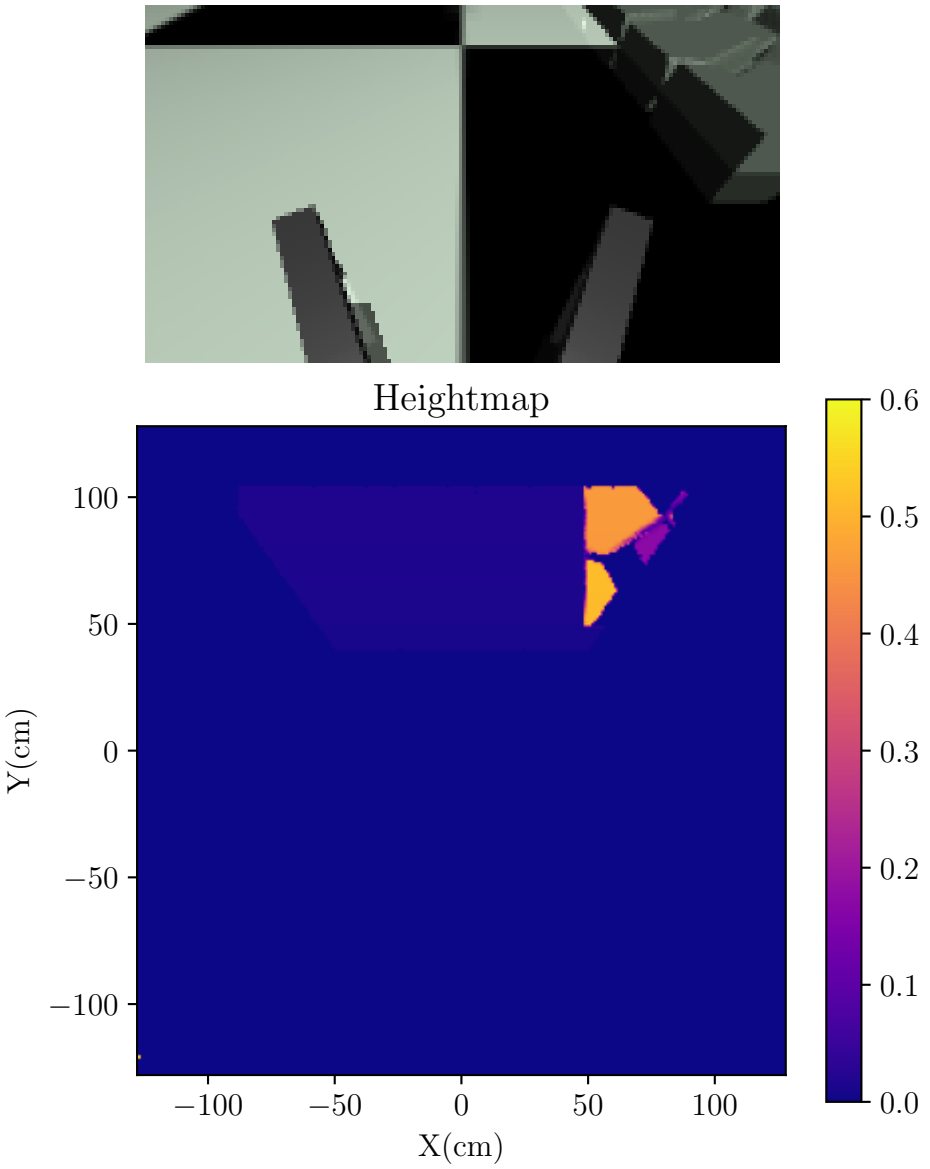


Figure 4.5: Simulated. Colour frame at the start of walking path (Top). Heightmap generated at the start of the walking path (Bottom).

Next, in figure 4.6, a larger part of the cobblestone is visible on the heightmap. The final piece of the cobble is still not visible however. Part of the cobblestone, most notably the area around $-100cm$ in the X axis, is not visible even though the camera is pointed there. This is due to the tall piece of terrain obscuring the camera's view. Also note how the heightmap has wrapped around to the opposite side of the buffer as the robot walked towards the right.

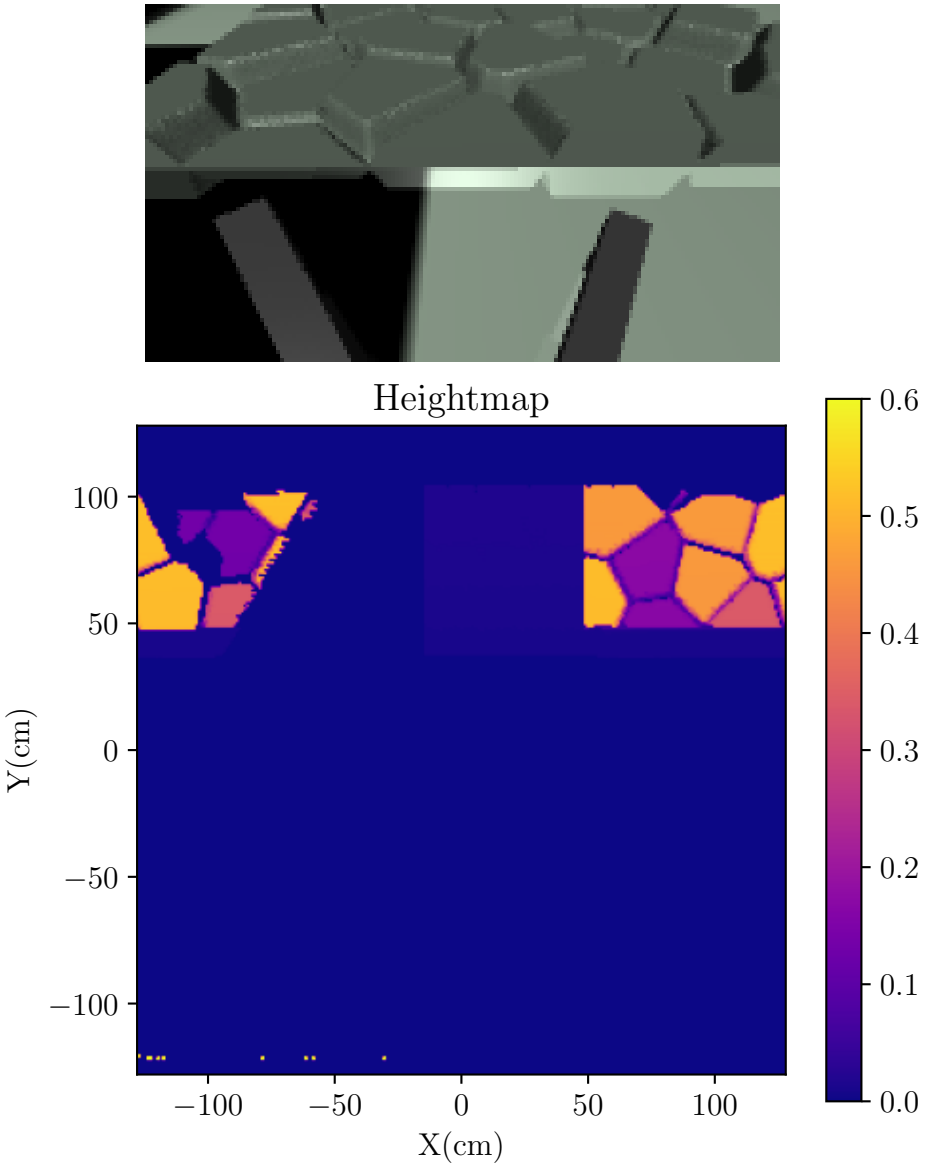


Figure 4.6: Simulated. Colour frame at the middle of walking path (Top). Heightmap generated at the middle of the walking path (Bottom).

Finally, in figure 4.7, the entire length of cobblestone surface is visible. The obscured part of the terrain from figure 4.6 was also filled in as the camera moved around the tall piece of terrain.

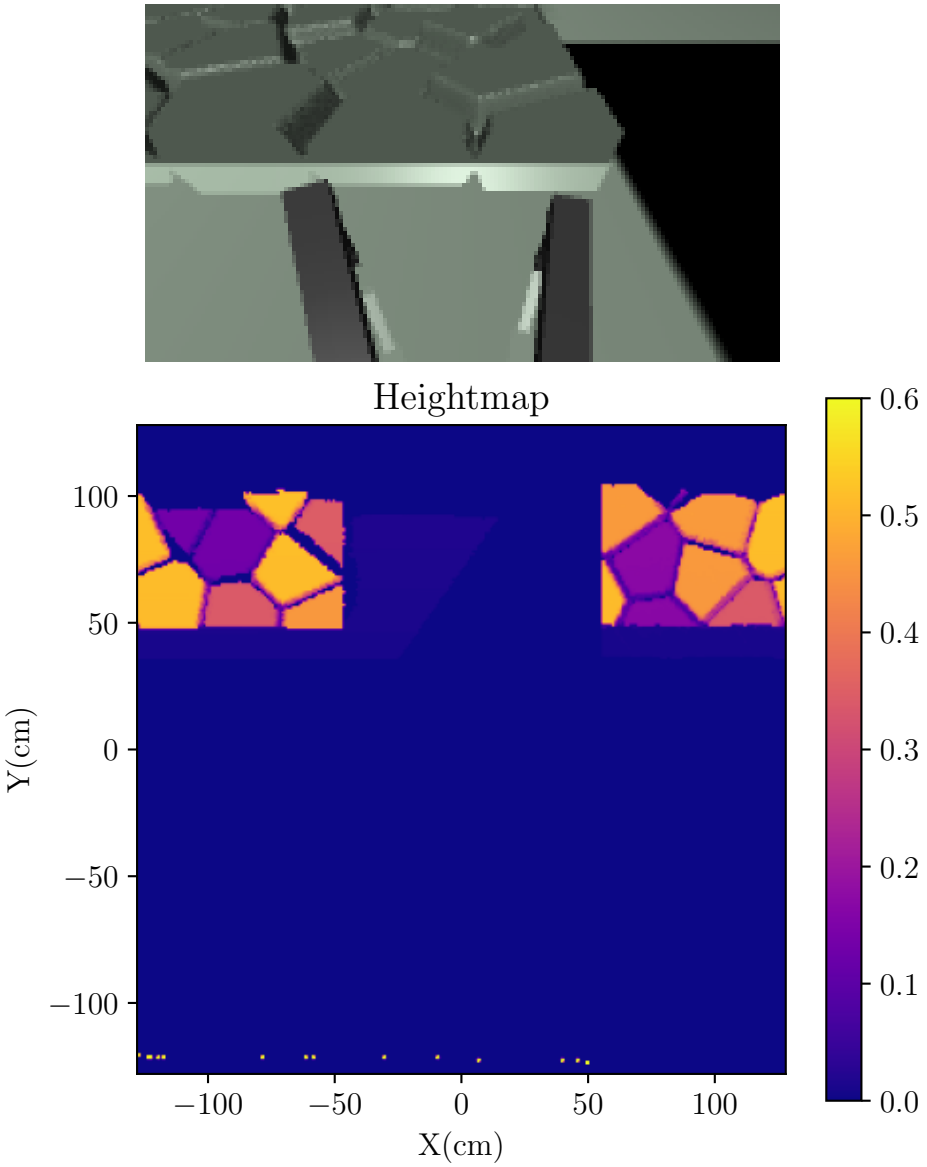


Figure 4.7: Simulated. Colour frame at the end of walking path (Top). Heightmap generated at the end of the walking path (Bottom).

4.7 Hardware Mapping Test

The mapping system was also tested on the physical robot. This mapping test aimed to show that the physical hexapod is also capable of mapping various objects while in motion. The test consisted of the robot walking past various obstacles, while simultaneously generating a heightmap.

Figure 4.8 shows a top-down diagram of the placed obstacles used in this test, with the path the robot walks.

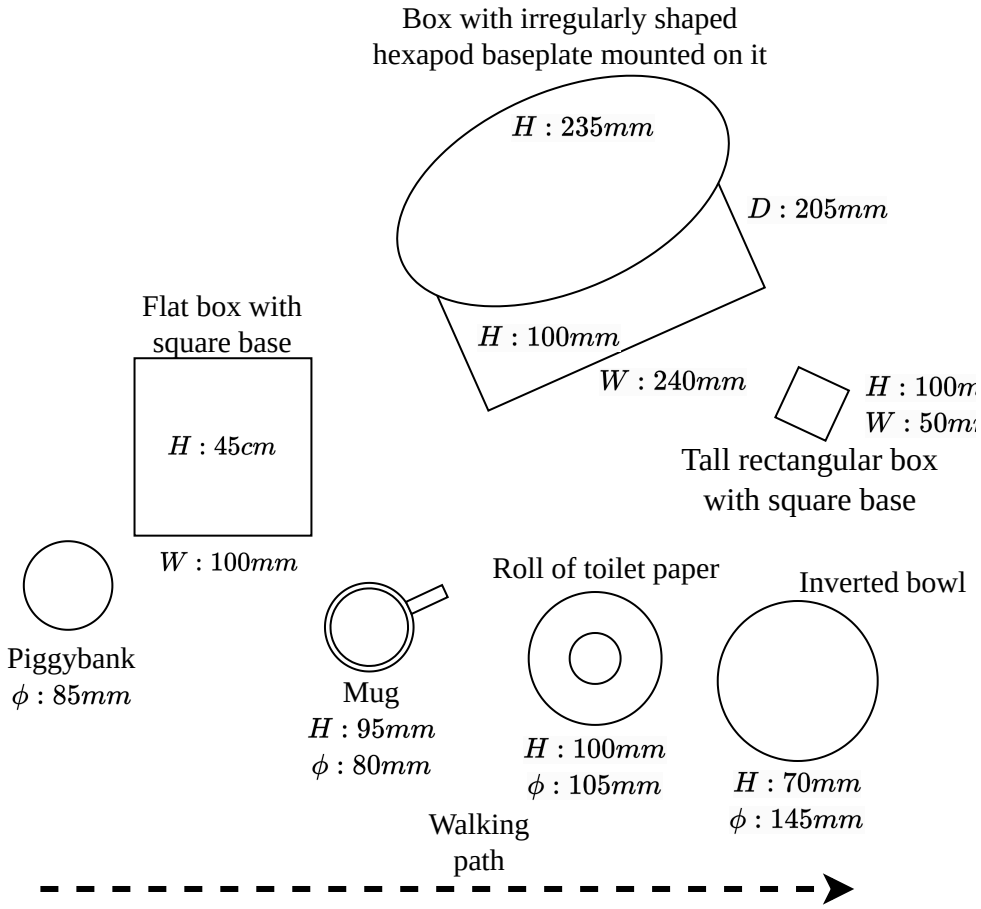


Figure 4.8: Top-down diagram of the test setup.

The test objects were selected to show a variety of obstacle types, with a variety of heights, that the robot might encounter, including sloped objects, spherical objects, objects with holes, and irregular objects. Figures 4.9, 4.10 and 4.11 show the heightmap, and its corresponding color frame, at the start middle and end of the robot's walking path. Please see this video of the test from the robot's perspective.

Figure 4.9 shows the heightmap at the start of the generation, at this point the heightmap is essentially a direct representation of what is currently seen by the camera. This is quite clear by the shadows cast by both the robot's leg (on the left) and by the obstacles (the piggybank and box). In this static case very little noise is also present.

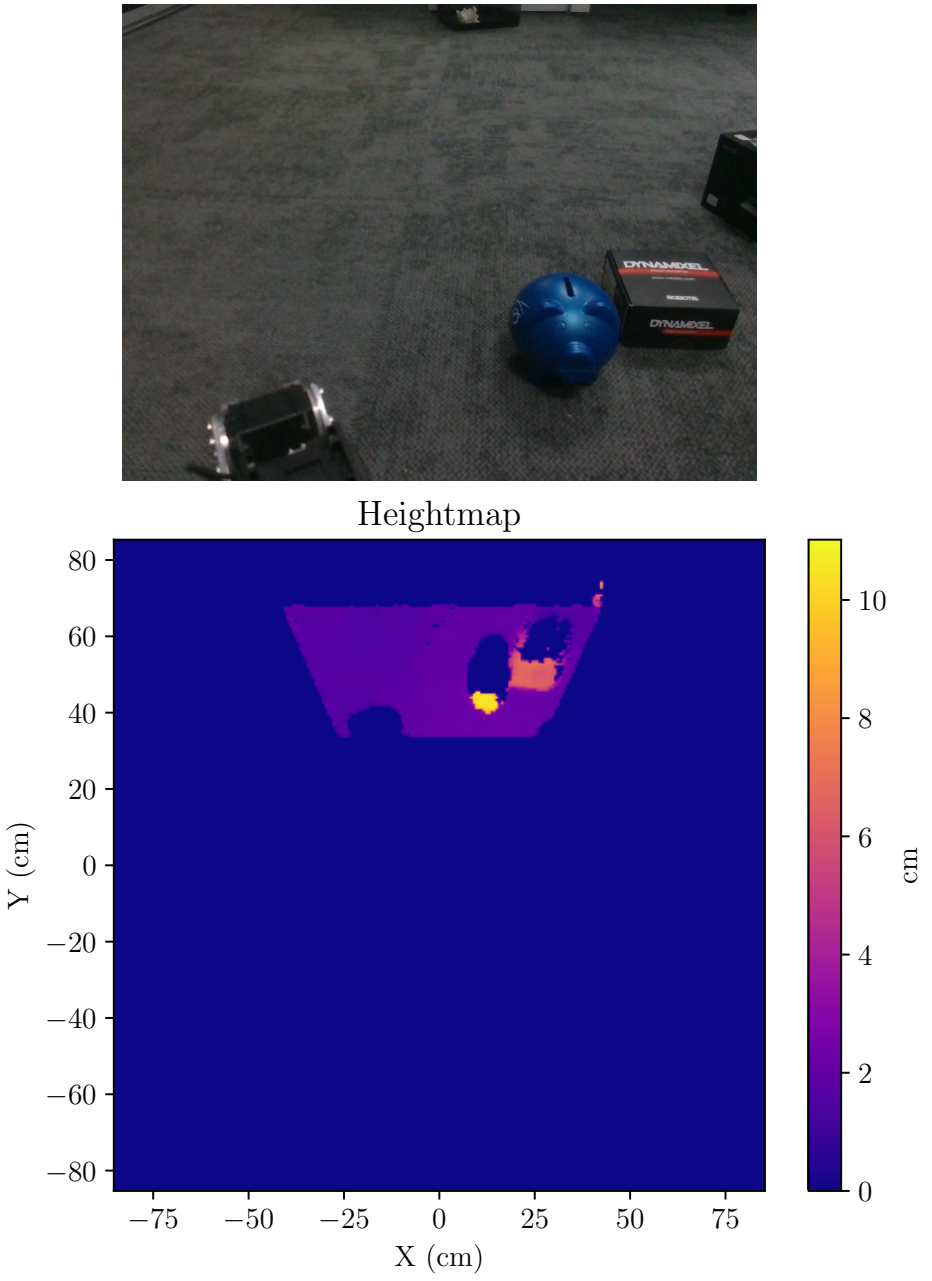


Figure 4.9: Hardware. Colour frame at the start of walking path (Top). Heightmap generated at the start of the walking path (Bottom).

Figure 4.10 shows the heightmap at a middle point of the path, here it can be seen how some of the shadows have been filled in by the movement of the camera. Particularly, the shadow caused by the robot leg is no longer present, and the shadows cast by the obstacles are reduced. The somewhat spherical pig object now also appears larger as part of the other side became visible.

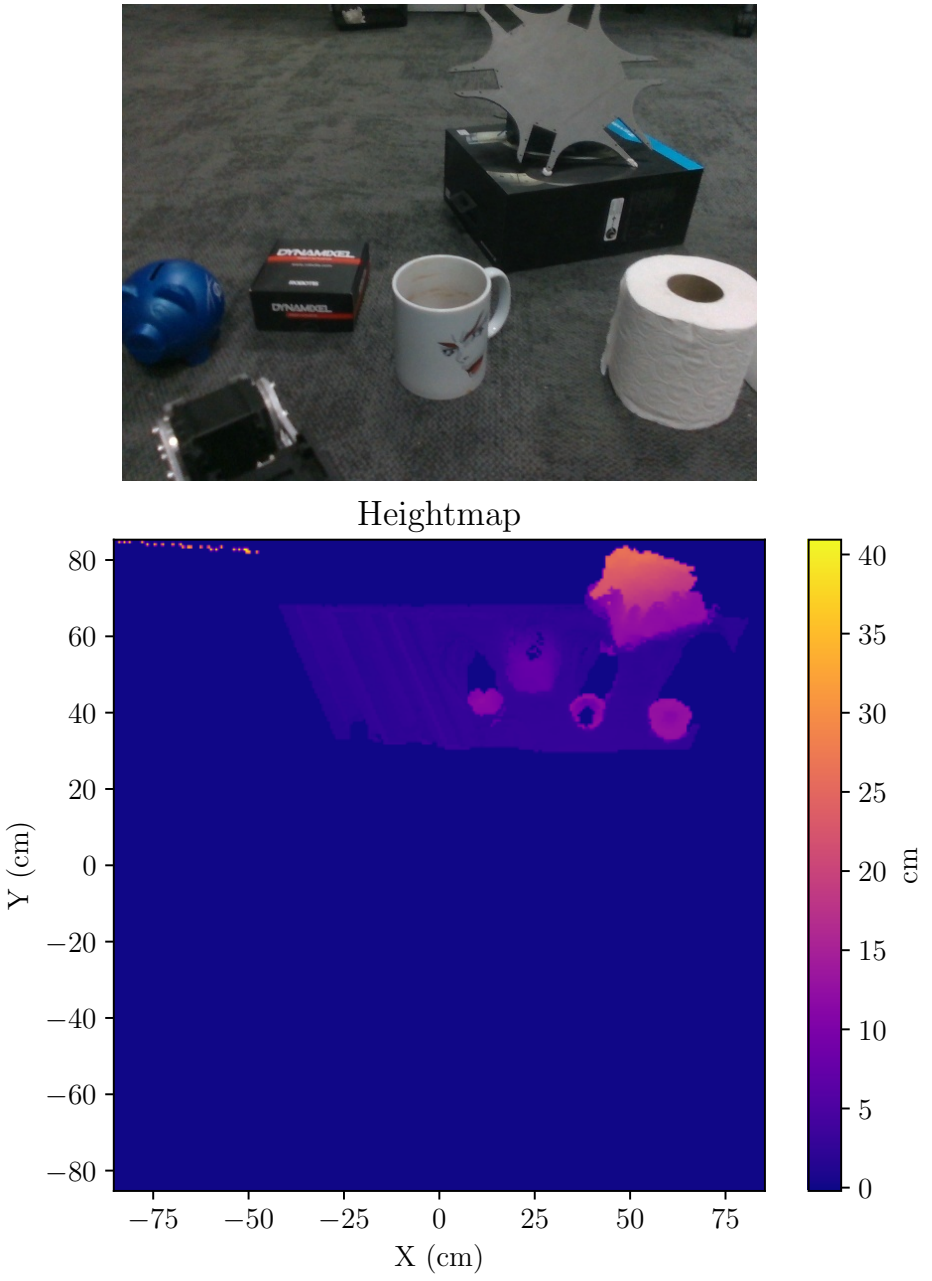


Figure 4.10: Hardware. Colour frame at the middle of walking path (Top). Heightmap generated at the middle of the walking path (Bottom).

In figure 4.10 however, noise is more prevalent, this is particularly visible when looking at the floor height, where distinct lines can be observed. These lines are the edge of the field of vision of the camera which is being moved to the right, and is caused by inaccuracies in the position estimation. As the

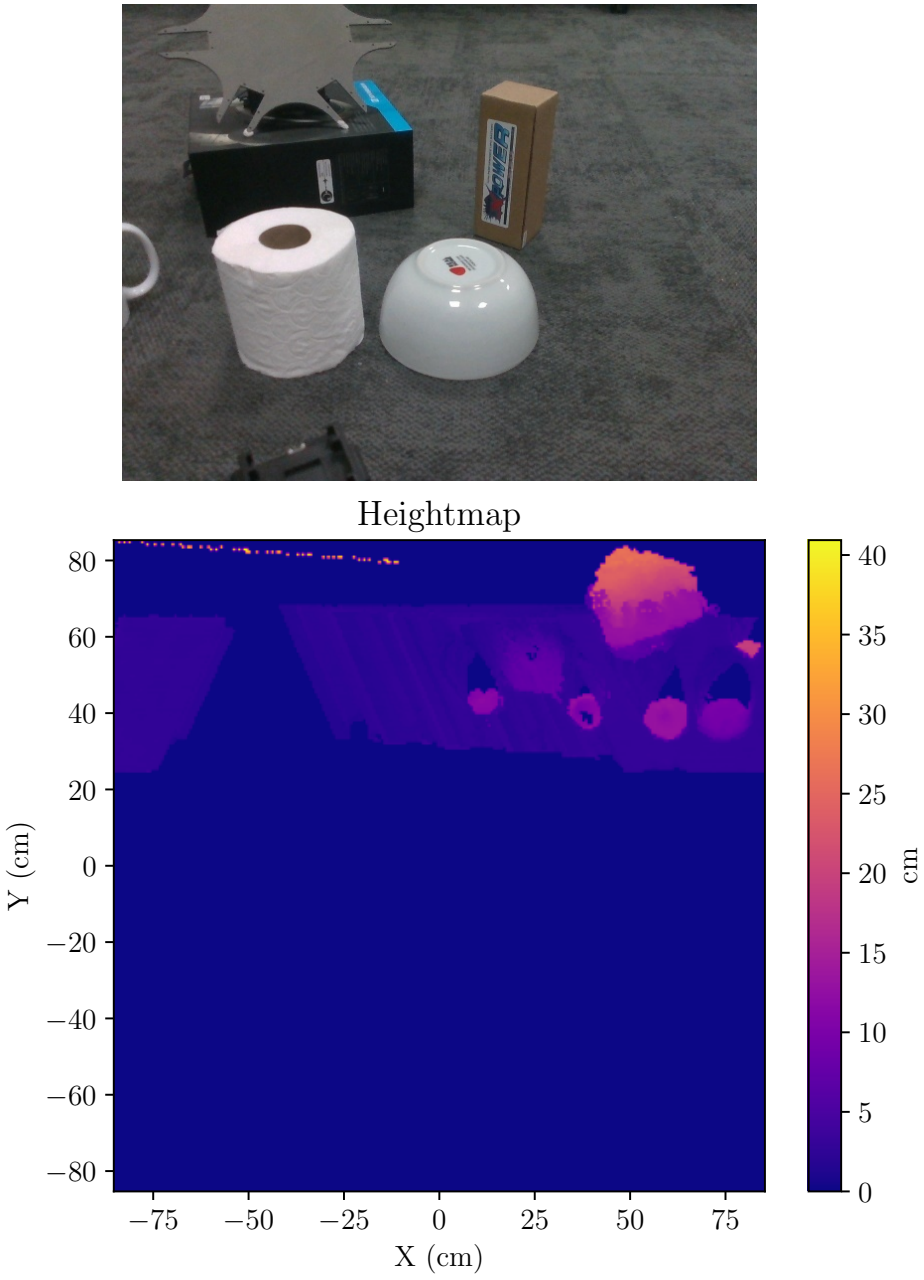


Figure 4.11: Hardware. Colour frame at the end of walking path (Top). Heightmap generated at the end of the walking path (Bottom).

previous, simulated, test has much more robust access to the robot's position, these types of artifacts were not present.

Finally, the heightmap generated at the end of the walking path can be seen in figure 4.11. In this heightmap the shadows cast have been shrunk even more, and some additional objects became visible. The original objects are also now out of vision, but they persist on the heightmap.

Figure 4.12 show the pose estimate provide by the ORB-SLAM3 system during the hardware test, with each dot being an estimate. The blue dot indicates the start of the test and the red dot indicates the end of the test.

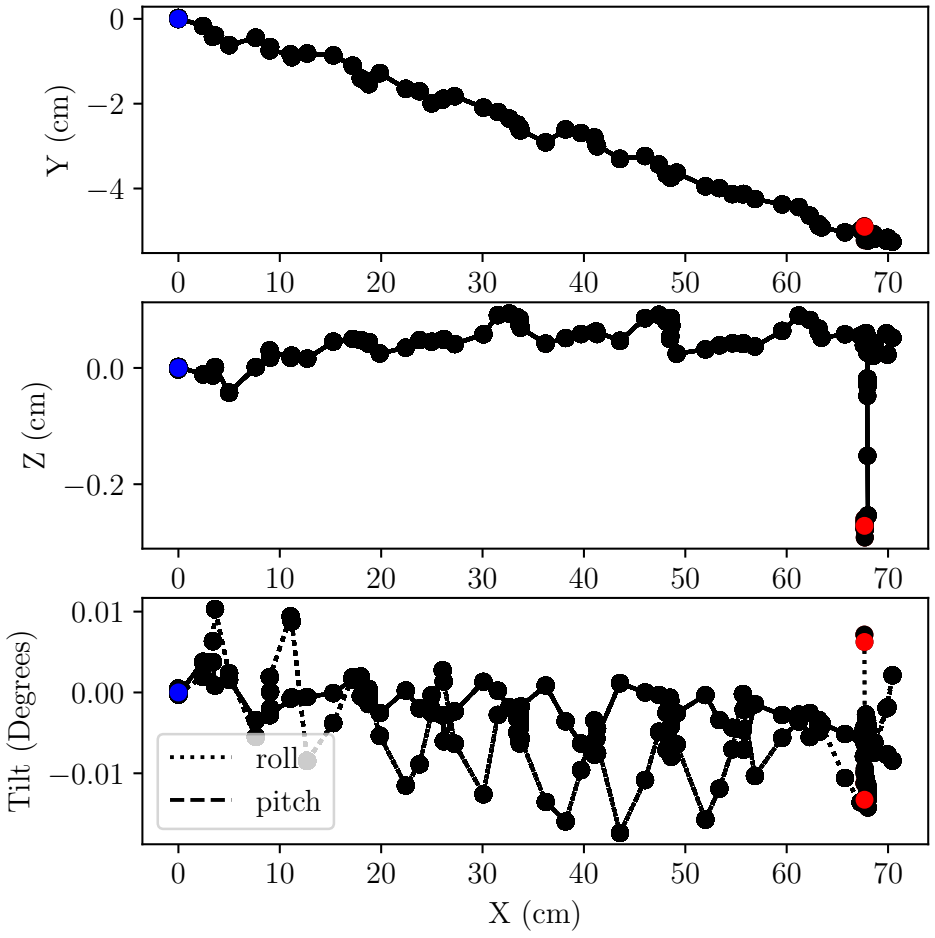


Figure 4.12: Pose estimate output from ORB-SLAM3

As can be seen, the frequency of pose estimates causes there to sometimes be large jumps between estimates. This is especially noticeable at the end where the robot quickly lowers itself. If the system was further optimised to increase the frequency and consistency of estimates, the accuracy of the heightmap would also increase.

Chapter 5

Baseline Motion System

This chapter covers the systems governing the baseline motion of the robot, meaning the motion over simple flat terrain. First the kinematics of the robot are modelled, Next the walking gait state machine is described. And finally, the equations to define a foot's path of motion is described. An overview of these systems can be seen in figure 5.1.

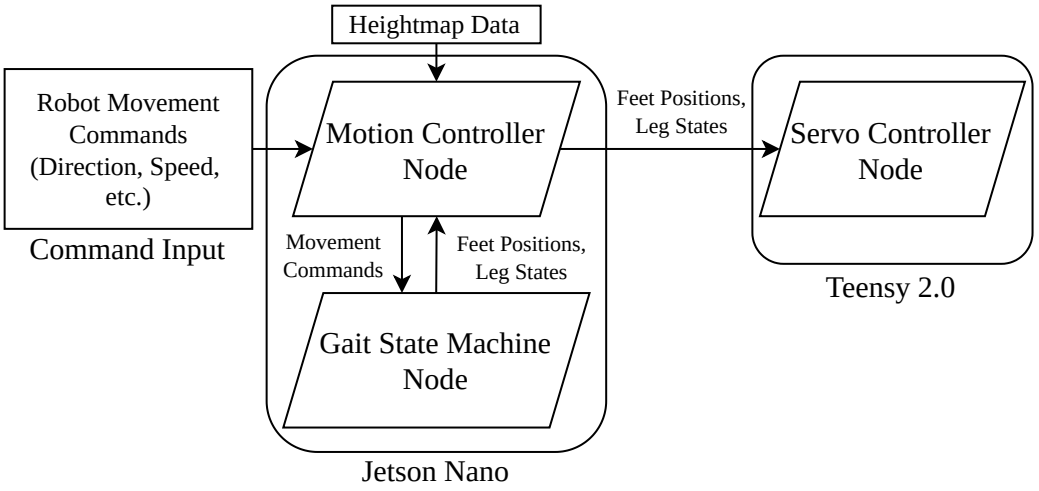


Figure 5.1: Motion System Overview

5.1 Overview

The basic operation of the motion system is as follows: First the robot is commanded to walk in a certain direction, at a certain speed, and with a certain body height. These commands are sent from the base station to the Jetson Nano on the robot. The motion controller node on the Jetson Nano then sends these commands to the gait state machine node, at a fixed frequency. The

gait state machine uses the received direction and stride length to generate leg states (swinging or supporting) and the ideal final position of each foot. These states and positions are sent back to the motion controller node where the positions are adjusted based on the heightmap data to ensure stable footing. The leg states and adjusted feet positions are then sent to the servo controller node. This node controls the servos to move the robots feet to their final positions, either in a arc or linearly, depending on their state (swinging or supporting).

5.2 Kinematics

When commanding a foot position, the servo controller requires a function to calculate servo angles. While the foot arc planner, which will be described in section 5.4, requires the current position of the feet to function. The inverse and forward kinematic functions described in this section provide this functionality.

5.2.1 Coordinate Frames

The coordinate frames relevant to the kinematics of the robot are the body coordinate frame, $\mathcal{B}$, and the leg frames, $\mathcal{L}_i$. All foot targets/positions are specified in the body coordinate frame, while the kinematic systems operate in the leg coordinate frames. Thus conversions from the body to leg frames are required. Figure 5.2 shows the world, body, and leg coordinate frames.

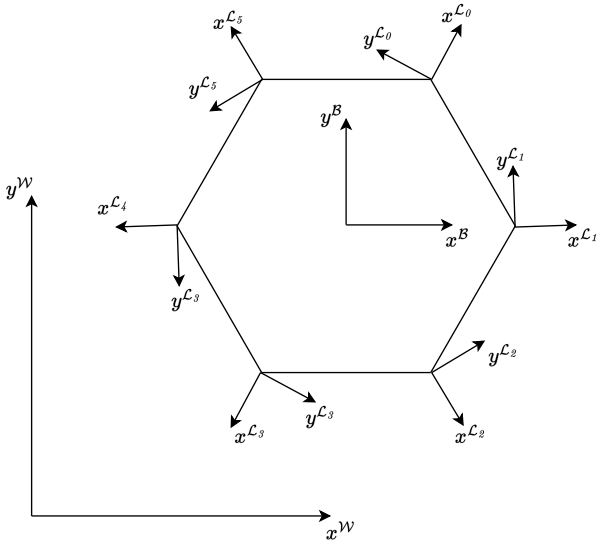


Figure 5.2: World, body and leg coordinate frames.

The leg coordinate systems are simply rotated and shifted from the body coordinate system. This transformation,$T_{\mathcal{BL}_i}(\boldsymbol{x}^{\mathcal{B}})$ is defined by equation 5.1, further explained in appendix A.

$$\begin{aligned} \boldsymbol{x}^{\mathcal{L}_i} &= T_{\mathcal{BL}_i}(\boldsymbol{x}^{\mathcal{B}}) \\ &= \boldsymbol{Q}_i^{\mathcal{B}} \cdot \boldsymbol{x}^{\mathcal{B}} \cdot \boldsymbol{Q}_i^{\mathcal{B}^{-1}} - \boldsymbol{R}_i \end{aligned} \tag{5.1}$$

where $\mathcal{L}_i$ is the coordinate frame of leg i, $\boldsymbol{Q}_i^{\mathcal{B}}$ is the rotation of said coordinate frame and $\boldsymbol{R}_i^{\mathcal{B}}$ is the root position of said leg coordinate frame in the body coordinate frame. For more detail on this transformation, such as the values of $\boldsymbol{Q}_i^{\mathcal{B}}$ and $\boldsymbol{R}_i^{\mathcal{B}}$, please see appendix A.

Now that positions can be transformed into the leg coordinate frames, the kinematic equations, as described in the following sections, can be applied. The kinematic equations are defined with reference to the variables in the leg frame as shown in figure 5.3.

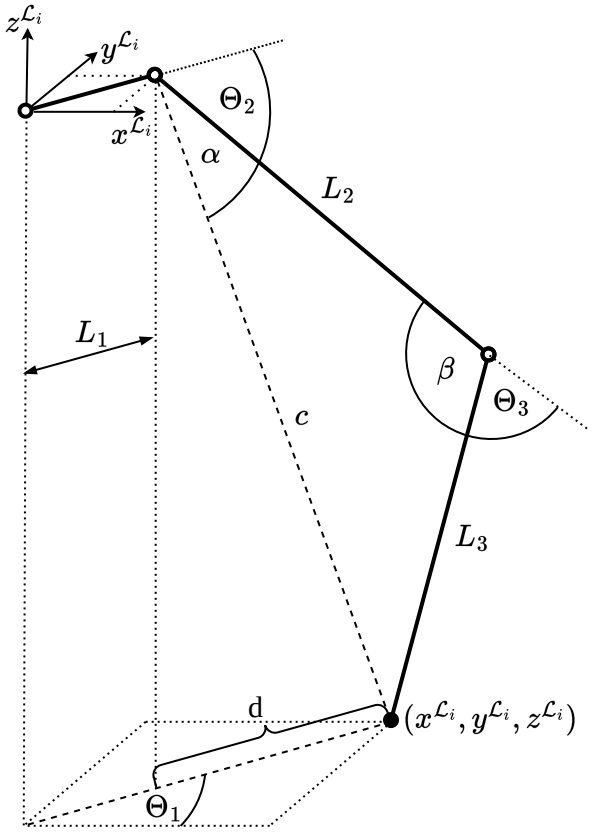


Figure 5.3: Leg coordinate frame with kinematic variables.

5.2.2 Inverse Kinematics

The inverse kinematic function calculates the leg servo angles, $\boldsymbol{\Theta}^{\mathcal{L}_i} = [\Theta_1, \Theta_2, \Theta_3]^T_,$ required for the foot to be at the given target position vector, $\boldsymbol{t}^{\mathcal{L}_i} = [x_t, y_t, z_t]^T_.$ The inverse kinematic equations are described by

$$\boldsymbol{\Theta}^{\mathcal{L}_i}(x_t, y_t, z_t) = \begin{bmatrix} \arctan\left(\dfrac{x_t}{y_t}\right) \\ \dfrac{\pi}{4} - \alpha - \arctan\left(\dfrac{y_t}{d - L_1}\right) \\ \dfrac{\pi}{2} - \beta \end{bmatrix} \tag{5.2}$$

where α, β, c and d are calculate as follows:

$$\alpha = \arcsin\left(\frac{L_3 \sin\beta}{c}\right) \tag{5.3}$$

$$\beta = \arccos\left(\frac{L_1^2 + L_2^2 - c^2}{2L_1L_2}\right) \tag{5.4}$$

$$c = \sqrt{(d - L_1)^2 + z_t^2} \tag{5.5}$$

$$d = \sqrt{x_t^2 + y_t^2} \tag{5.6}$$

5.2.3 Forward Kinematics

The forward kinematics function calculates the position vector of a foot, $\boldsymbol{p}_f^{\mathcal{L}_i} = [x_f, y_f, z_f]^T$, given the current angles of the leg servos, $\boldsymbol{\theta}^{\mathcal{L}_i} = [\theta_1, \theta_2, \theta_3]^T$.

$$\boldsymbol{p}_f^{\mathcal{L}_i}(\theta_1, \theta_2, \theta_3) = \begin{bmatrix} d\cos\theta_1 \\ d\sin\theta_1 \\ L_2\sin\theta_2 + L_3\sin(\theta_2 + \theta_3) \end{bmatrix} \tag{5.7}$$

where d is calculated as follows:

$$d = L1 + L_2\sin\theta_2 + L_3\sin(\theta_2 + \theta_3) \tag{5.8}$$

5.2.4 Angular Rate

To move a foot on a desired path it is important to not only know the absolute angle of the three leg servos, but also the angular rates of all three servos. If the servos are all moved at the same rate, the shape of the path that the foot follows will not be linear, but rather dependant on the current foot position. This is undesirable, thus equations 5.9 defines the derivative of the inverse kinematics equations (5.2), i.e. the angular rate, given the target movement speeds of a foot, $\dot{x}$, $\dot{y}$ and $\dot{z}$.

$$\boldsymbol{\omega}(\dot{x},\dot{y},\dot{z}) = \begin{bmatrix} \dfrac{-x\dot{y}+y\dot{x}}{x^2+y^2} \\ \dfrac{\left[(L_1-d)\dot{z}+z\dot{d}\right]\alpha+\left[(L_1-d)^2+z^2\right]\arctan\left(\dfrac{L_1-d}{z}\right)\dot{\alpha}}{(L_1-d)^2+z^2} \\ -\dot{\beta} \end{bmatrix} \tag{5.9}$$

where $\dot{\alpha}$, $\dot{\beta}$, $\dot{c}$ and $\dot{d}$ as shown in equations 5.10 to 5.13.

$$\dot{\alpha} = \frac{L_3\left[c\cos(\beta)\dot{\beta}-\sin(\beta)\dot{c}\right]}{c^2\sqrt{-\dfrac{L_3^2\sin^2(\beta)}{c^2}+1}} \tag{5.10}$$

$$\dot{\beta} = \frac{2c\dot{c}}{L_2L_3\sqrt{4-\dfrac{(L_2^2+L_3^2-c^2)^2}{L_2^2L_3^2}}} \tag{5.11}$$

$$\dot{c} = \frac{-(L_1-d)\dot{d}+z\dot{z}}{\sqrt{(L_1-d)^2+z^2}} \tag{5.12}$$

$$\dot{d} = \frac{x\dot{x}+y\dot{y}}{\sqrt{x^2+y^2}} \tag{5.13}$$

5.3 Walking Gait

To move, the hexapod must support its body with some of its legs while the remaining legs swing towards their new targets, at which point the swinging legs become the new supporting legs. The sequence in which the legs support and swing is called the walking gait. Figure 5.4 shows three different gait patterns that can be used with a hexapod, namely the wave, ripple, and tripod gaits (Darbha, 2017). A dark cell represents a swinging leg and a light cell a supporting leg.

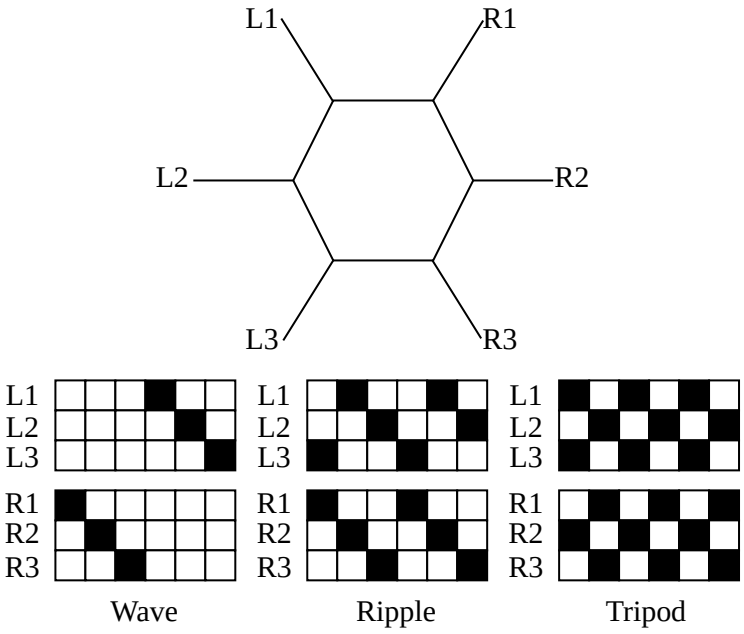


Figure 5.4: Three hexapod gait patterns.

The wave gait moves one leg at a time while supporting with the remaining 5, the ripple gait moves two legs at a time, and the tripod moves three legs at a time.

The speed of the hexapod is based on the parameters of the gait, specifically as described in equation 5.14,

$$v = \frac{S}{D\tau} \tag{5.14}$$

where S is the stride length, τ is the gait period and D is the duty factor. D is defined as the time a leg is in the support phase relative to its swing phase. The wave, ripple and tripod gaits have duty factors of $\frac{5}{6}$, $\frac{2}{3}$ and $\frac{1}{2}$ respectively.

From this it is clear that the wave gait is the slowest while the tripod gait is the fastest, and the ripple gait is in-between. It should however be noted the gait's stability is inverse to their speed.

The gait that will be used in this system is the tripod gait, which is the most common gait for hexapods as it supports with three legs, while maximising speed. Even though this is less stable than the wave and ripple gaits, it does maintain natural stability with three contact points, which is adequate for most circumstances.

5.3.1 Stride Reference Frame

When describing the stride of the robot it is important to note which reference frame is being used. Figure 5.5 shows the stride of a tripod gait in the body reference frame, $\mathcal{B}$.

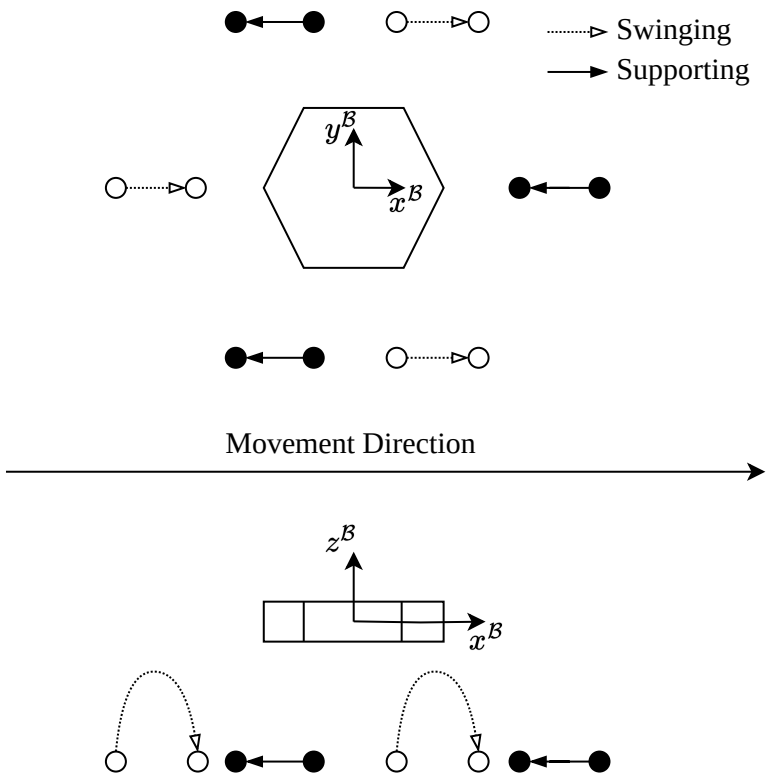


Figure 5.5: Hexapod stride relative to body coordinates.

As can be seen, the swinging legs move in the direction of movement, following an arced path, while the supporting legs move in the opposite direction of movement following a linear path.

However, when looking at the same stride in the world reference frame, $\mathcal{W}$, as can be seen in figure 5.6, the supporting legs appear to stay stationary, while the swinging legs move double the distance relative to that in the body reference frame.

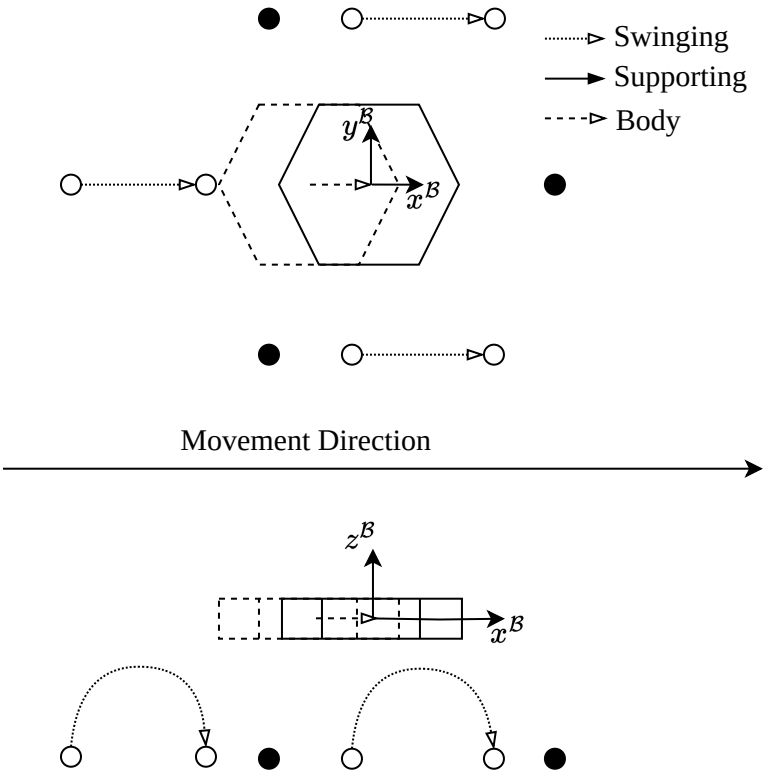


Figure 5.6: Hexapod stride relative to world coordinates.

This is important to note because, as further discussed in section 5.3.4, the nominal foot positions are chosen in the body reference frame, but must target a position in the world reference frame.

5.3.2 State Machine

The state machine used to realise the tripod gait used in the robot is quite simple, comprised of only two states, namely stepping and resting, as can be seen from figure 5.7. Table 5.1 defines the actions that should be taken during each state.

The primary computation done by this state machine is calculating which legs are supporting and which are swinging, which occurs on entering the "Stepping" state. This function is described in section 5.3.3.

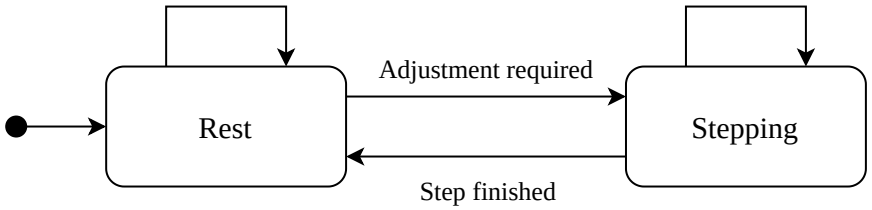


Figure 5.7: Gait State Machine

Rest State Definition	
Enter Condition	Has all feet reached their targets?
On Entering	Set all leg states as supporting.
While Active	Do nothing

Stepping State Definition	
Enter Condition	Is there a mismatch between feet targets and current position?
On Entering	Calculate and set the leg states based on walking direction, see section 5.3.3. Choose and optimise nominal targets for the current step, see chapter 6
While Active	Adjust feet targets based on direction, stride length and robot height, see chapter 6

Table 5.1: State Definitions

5.3.3 Choosing the Supporting and Swinging Legs

The robot body is divided up into sextants, centered around the nominal leg positions. When calculating the swinging legs it is first determined in which sextant the movement direction vector falls, this is called the active sextant. The leg related with the active sextant, and the two opposite, are then chosen as swinging, with the remaining three legs chosen as supporting. The states of the legs stored in a boolean array. Figure 5.8 illustrates and example with sextant 1 being active. Thus legs 1, 3 and 5 are swinging, while legs 0, 2 and 4 are supporting.

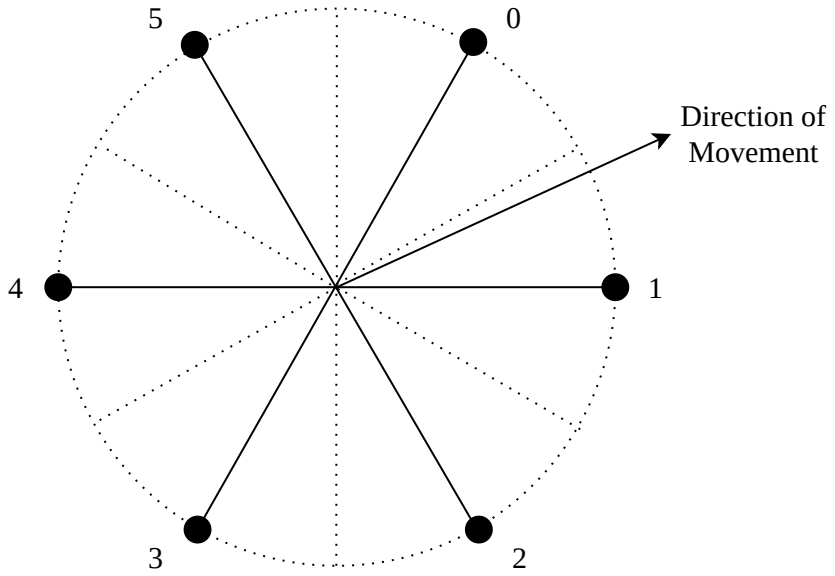


Figure 5.8: Leg sextants, with sextant 1 currently being active.

This of course would not be sufficient to define a walking gait, as at the end of each step the boolean array containing the leg states needs to be inverted. Thus, at the end of each step, if the currently swinging legs are the same as those in the previous step, then the boolean array is inverted.

5.3.4 Choosing nominal foot positions

Once the active and inactive legs have been selected, it is required to choose nominal targets for all the feet. Sections 5.3.4.1 and 5.3.4.2 describe the process for finding the support and swinging leg target positions. It should be noted that target matrices, denoted by $\boldsymbol{t}$, are common between these two sections, but are computed differently depending on whether the leg is swinging or supporting.

5.3.4.1 For Supporting Legs

The supporting leg nominal targets are chosen by first calculating the move vector $\boldsymbol{m}^{\mathcal{B}}$, and then adding the move vector to the foot's target position in it's swinging phase, which should be equal to the foot's current position. The move vector is calculated as the vector from the foot's nominal target, from the foot's swinging phase, to the movement direction, $\boldsymbol{w}^{\mathcal{B}}_{\text{dir}}$, inverted, multiplied by the stride length, and added to the foot's constant resting position, $\boldsymbol{T}^{\mathcal{B}}$. $\boldsymbol{T}^{\mathcal{B}}$ is a constant that contains the positions of the feet when the robot is standing still, on flat terrain. Equations 5.15 and 5.16 describe this process, with reference to figure 5.9.

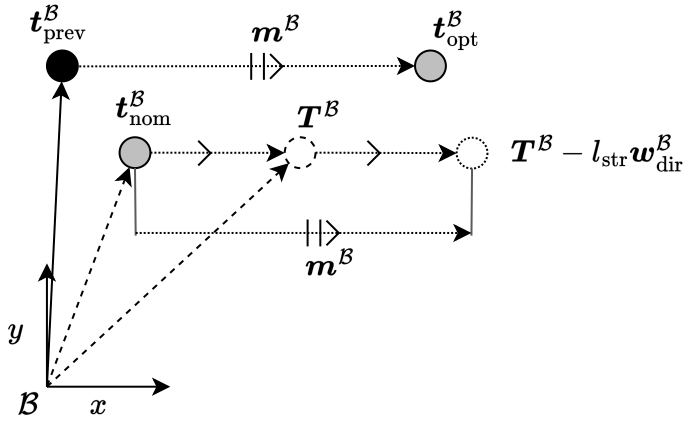


Figure 5.9: Supporting target choosing diagram in the body frame.

First the required move vectors, $\boldsymbol{m}^{\mathcal{B}}$, are calculated in equation 5.15,

$$\underset{3\times1}{\boldsymbol{m}^{\mathcal{B}}} = (\boldsymbol{T}^{\mathcal{B}} - l_{\text{str}}\boldsymbol{w}^{\mathcal{B}}_{\text{dir}}) - \boldsymbol{t}^{\mathcal{B}}_{\text{nom}} \tag{5.15}$$

where $\boldsymbol{T}^{\mathcal{B}}$ contains the constant resting positions of the feet, l_{str} is the nominal stride length, $\boldsymbol{w}^{\mathcal{B}}_{\text{dir}}$ contains the desired walk directions, and $\boldsymbol{t}^{\mathcal{B}}_{\text{nom}}$ contains the nominal targets as calculated in equation 5.18.

Then the optimised targets, $\boldsymbol{t}^{\mathcal{B}}_{\text{opt}}$, are calculated as the addition of the previous targets and the move vector in equation 5.15,

$$\underset{3\times6}{\boldsymbol{t}^{\mathcal{B}}_{\text{opt}}} = \boldsymbol{t}^{\mathcal{B}}_{\text{prv}} + \boldsymbol{m}^{\mathcal{B}} \tag{5.16}$$

where $\boldsymbol{t}^{\mathcal{B}}_{\text{prv}}$ contains the previous, optimised, targets. Note that equation 5.16 does not include the optimisation function, described in chapter 6. This is because the supporting feet will not move relative to the terrain, and thus do not need their targets optimised.

5.3.4.2 For Swinging Legs

The swinging leg nominal targets are chosen by adding the movement, direction, $\boldsymbol{w}^{\mathcal{B}}_{\text{dir}}$, multiplied by the stride length, to the constant feet resting positions, $\boldsymbol{T}^{\mathcal{B}}$, and the movement vector of the supporting legs, $\boldsymbol{m}^{\mathcal{B}}$.

A diagram of this process is shown in figure 5.10. Please note that $\boldsymbol{t}^{\mathcal{M}}_{\text{nom}}$ is in map space, thus $\boldsymbol{t}^{\mathcal{M}}_{\text{nom}}$ is represented as $T_{\mathcal{MB}}(\boldsymbol{t}^{\mathcal{M}}_{\text{nom}})$ in the figure, because the figure is in body space. This map to body transform is purely for illustrative purposes and is not used in the actual system, as it simply reverses the body to map transform in equation 5.17.

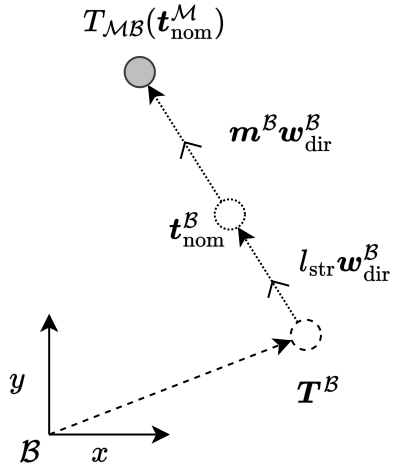


Figure 5.10: Swinging target choosing diagram in the body frame.

The swinging nominal targets, in map space, $\boldsymbol{t}^{\mathcal{M}}_{\text{nom}}$, are calculated as follows,

$$\underset{3\times 6}{\boldsymbol{t}^{\mathcal{M}}_{\text{nom}}} = T_{\mathcal{BM}}(\boldsymbol{T}^{\mathcal{B}} + \left(l_{\text{str}} + \boldsymbol{m}^{\mathcal{B}}\right) \boldsymbol{w}^{\mathcal{B}}_{dir}) \tag{5.17}$$

where $\boldsymbol{T}^{\mathcal{B}}$ contains the constant rest positions the feet, l_{str} is the nominal stride length, $\boldsymbol{w}^{\mathcal{B}}_{\text{dir}}$ contains the desired walk directions. $\boldsymbol{m}^{\mathcal{B}}$ is the average move vector.

For use in equation 5.15, the swinging legs nominal targets, $\boldsymbol{t}^{\mathcal{B}}_{\text{nom}}$, are calculated as in equation 5.18,

$$\underset{3\times 6}{\boldsymbol{t}^{\mathcal{B}}_{\text{nom}}} = \boldsymbol{T}^{\mathcal{B}} + l_{str}\boldsymbol{w}^{\mathcal{B}}_{\text{dir}} \tag{5.18}$$

Note that $T_{\mathcal{MB}}(\boldsymbol{t}^{\mathcal{M}}_{\text{nom}})$ is quite similar to $\boldsymbol{t}^{\mathcal{B}}_{\text{nom}}$, the difference is that $\boldsymbol{t}^{\mathcal{M}}_{\text{nom}}$ is set such that it will align with $\boldsymbol{t}^{\mathcal{B}}_{\text{nom}}$ at the end of the step. Explaining the omission of the support phase move vector, $\boldsymbol{m}^{\mathcal{B}}$, in calculating $\boldsymbol{t}^{\mathcal{B}}_{\text{nom}}$ in equation 5.18. Meaning that until the end of the step, $\boldsymbol{t}^{\mathcal{M}}_{\text{nom}}$ and $\boldsymbol{t}^{\mathcal{B}}_{\text{nom}}$ will point to different positions in world space. This is due to needing to account for body movement in map space, but not in local space.

5.4 Trajectory Generation

When taking a step, the foot can not simply be moved to its destination in a straight line, as doing so will cause the foot to be dragged on the terrain, impeding the movement of the robot. Thus it is required to move the foot in an arc-like trajectory to clear any obstacles that might be in its path.

A new method of generating trajectories is described in section 5.4.2, as opposed to the existing system by Erasmus *et al.* (2023), which is described in section 5.4.1.

5.4.1 Existing Trajectory System

The existing system will, at the start of each step, compute an arc for each foot to follow, this arc is then sent to the servo controller to be executed. The arc is computed as a polynomial passing through three points, the initial pointq_i, the midpoint q_m, and the final point q_f. Figure 5.11 shows the variables used to calculate this arc.

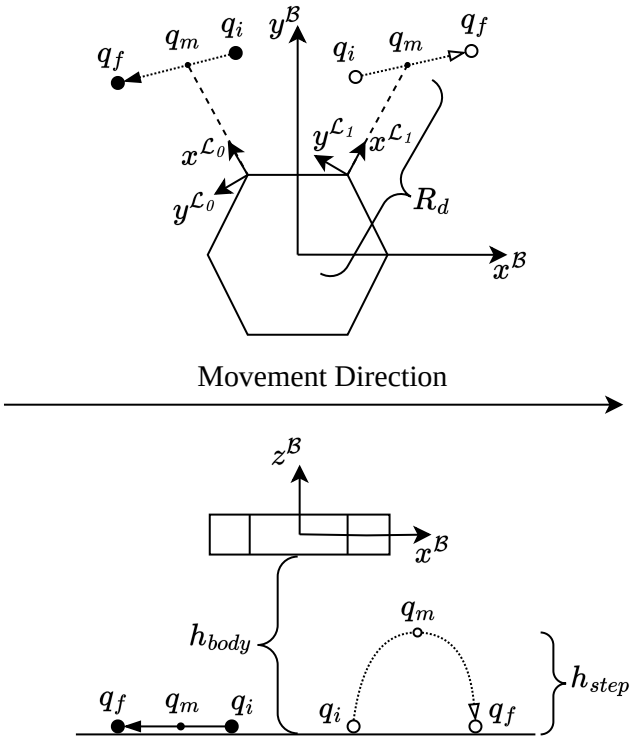


Figure 5.11: Variables used for calculating the arc.

The trajectory is generated by a single sixth order polynomials as stated in equation 5.19 by Erasmus *et al.* (2023).

$$
\begin{aligned}
q(t) &= a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5 + a_6 t^6 \\
\dot{q}(t) &= a_1 + 2a_2 t^1 + 3a_3 t^2 + 4a_4 t^3 + 5a_5 t^4 + 6a_6 t^5 \\
\ddot{q}(t) &= 2a_2 + 6a_3 t^1 + 12a_4 t^2 + 20a_5 t^3 + 30a_6 t^4
\end{aligned} \tag{5.19}
$$

where a_0 to a_6 are coefficients, these coefficients are defined below.

$$
\begin{aligned}
a_0 &= q_i \\
a_1 &= a_2 = 0 \\
a_3 &= \frac{2}{t_f^3}[32(q_1 - q_i) - 11(q_f - q_i)] \\
a_4 &= -\frac{3}{t_f^4}[64(q_1 - q_i) - 27(q_f - q_i)] \\
a_5 &= \frac{3}{t_f^5}[64(q_1 - q_i) - 30(q_f - q_i)] \\
a_6 &= -\frac{32}{t_f^6}[2(q_1 - q_i) - (q_f - q_i)]
\end{aligned} \tag{5.20}
$$

where t_f is the time at the end point and time at the start point, t_i, is 0.

While this is efficient and effective in ideal conditions, this method of defining the arc has poor performance when considering external influences. If for example the robot has to adjust the final target of its feet mid step, such as when the heightmap is updated, this arc would have to be recomputed in its entirety, thus leading to possible performance concerns.

In addition to this, the current system is designed with the assumption that the starting position of the foot is grounded, thus if the arc is recomputed mid-step the resulting new arc would be undesirable, as it will rise with the desired step height for a second time. This is illustrated in Figure 5.12.

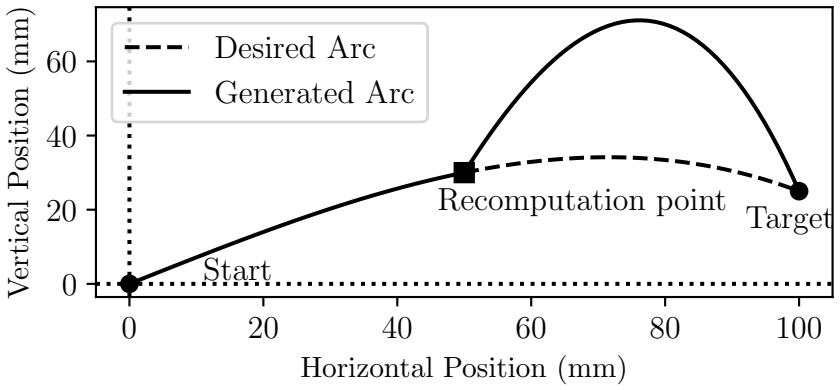


Figure 5.12: Existing arc recomputation problem

5.4.2 The Improved Trajectory System

The improved system solves this problem by utilising a flow function. During a step, this function will continuously calculate the direction that the foot must move to reach its destination. Thus this system is resilient to external disturbances and is capable of adjusting to varying destination and step height requirements.

The flow field is designed to first move the foot vertically upwards until horizontal coplanar with the destination, and then to follow a arc to the destination with a defined step height, this can be adjusted to make the arc start before or after coplanar. The step height can be adjusted at any point in time and the flow field will adjust accordingly. Figure 5.13 illustrates the field function and is described in section 5.4.2.1.

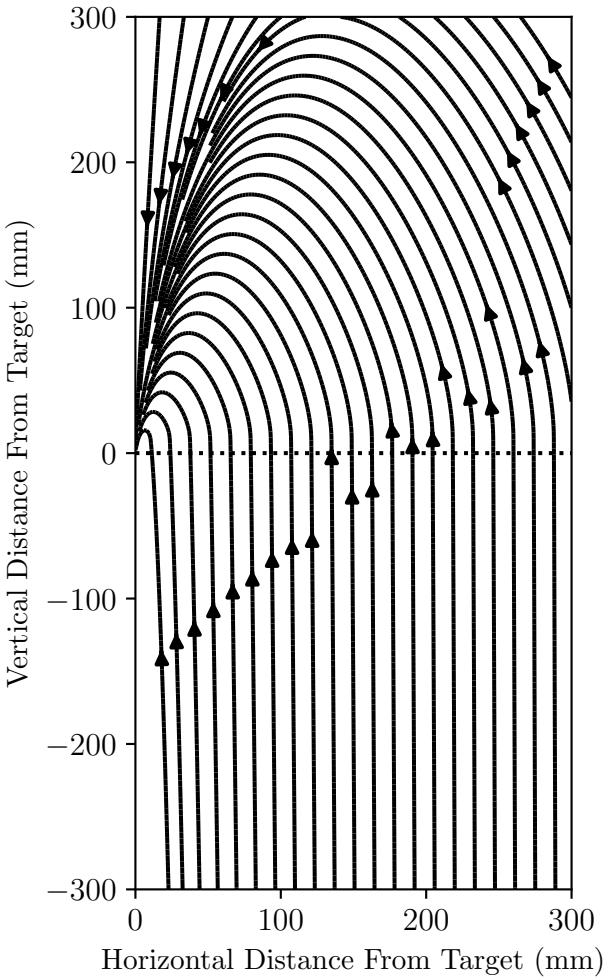


Figure 5.13: End effector movement path

5.4.2.1 Flow Function Description

The flow function, $\rho(x, y)$, uses the gradient function of a parabola passing through the point $[0, 0]$ and $[x, y]$ as a basis, where point $[x, y]$ is the current point that is being evaluated and x is the horizontal distance between the destination and the current point and y the vertical distance. The final function is described by equations 5.21 to 5.24.

$$\begin{aligned} \rho(x,y) = \frac{\delta}{\delta x \delta y} \quad & f_a(x,y)x^2 + f_b(x,y)x + C \\ = \quad & 2f'_a(x,y)x + f'_b(x,y) \end{aligned} \tag{5.21}$$

where,

$$f'_a(x,y) = -\left|\frac{C_h}{x}\right| - |S(y)| \tag{5.22}$$

$$f'_b(x,y) = \frac{y}{x} - f_a(x,y) \tag{5.23}$$

where C_h determines step height and $S(y)$ is a sigmoid-like function responsible for the initial vertical rise. $S(y)$ is defined in defined as

$$S(y) = \frac{0.515(y-q)}{1 + |y-q| - 0.515} \tag{5.24}$$

where q is the variable that determines at which vertical displacement the leg path transitions from primarily a vertical motion to an arc motion. Figure 5.14 illustrates the sigmoid-like function for different values of q.

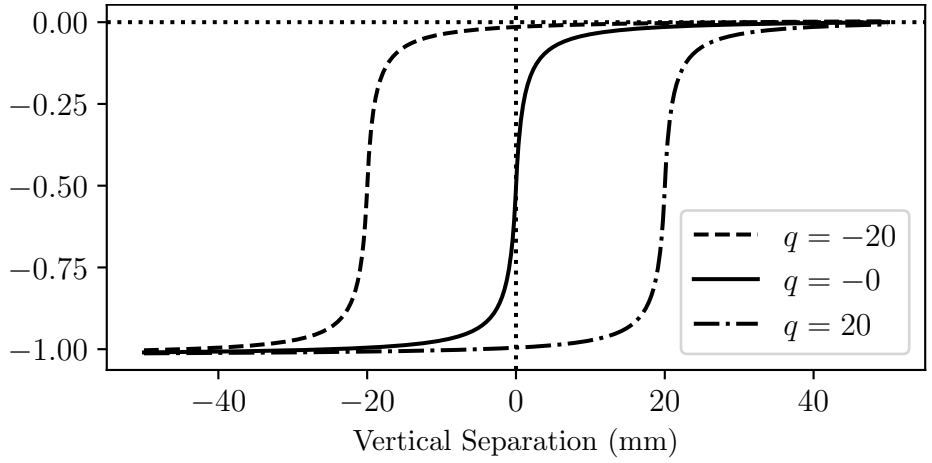


Figure 5.14: Sigmoid Like

Note that the 0.515 values in equation 5.24 are set to make its output range from roughly -1 to 0 over the active range.

5.4.2.2 Generated Trajectory Examples

To demonstrate the trajectories generated by the flow function, three different example are provided: One with the target on the same vertical level as the starting point, one with the target above the starting point, and one with the target below the starting point. Figures 5.15 to 5.17 show these three examples. Additionally, the parameters C_h and q are varied to show their effect. Please see this video for a simulated demonstration of the baseline walking system.

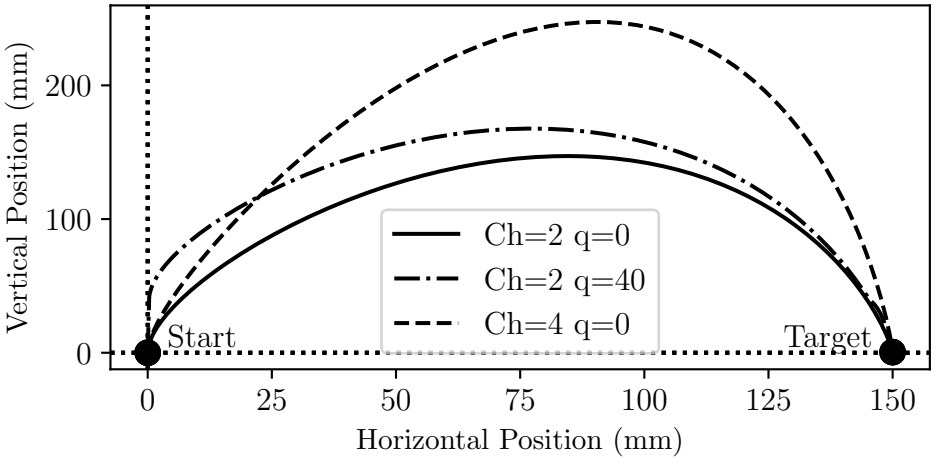


Figure 5.15: Equal start and target example trajectories.

As can be seen from figure 5.15 the parameter q specifies the initial vertical rise above the target position. When q is zero there is no additional vertical rise, similarly if q were to be made negative, the foot would start moving horizontally before the target's height is reached. While q does affect the stride height a little bit, the primary stride height parameter is C_h. As can be seen from the figure, adjusting C_h drastically affects the stride height while leaving vertical rise above the target unaffected.

The following example, shown in figure 5.16, shows the trajectory for when the target is placed above the starting point. Here the initial vertical rise can be seen clearly. Note how increasing q lengthens the vertical rise. Importantly it can be seen that once the vertical rise is completed, the remaining arc is essentially equivalent to that in figure 5.15. This is the intended benefit of using a policy such as the flow function.

The next example is shown in figure 5.17, this example has the target below the starting point. Here it can be seen that the generated trajectory has a much lower stride height compared to the previous examples. It should also be noted that in this case q has almost no effect on the resulting trajectory,

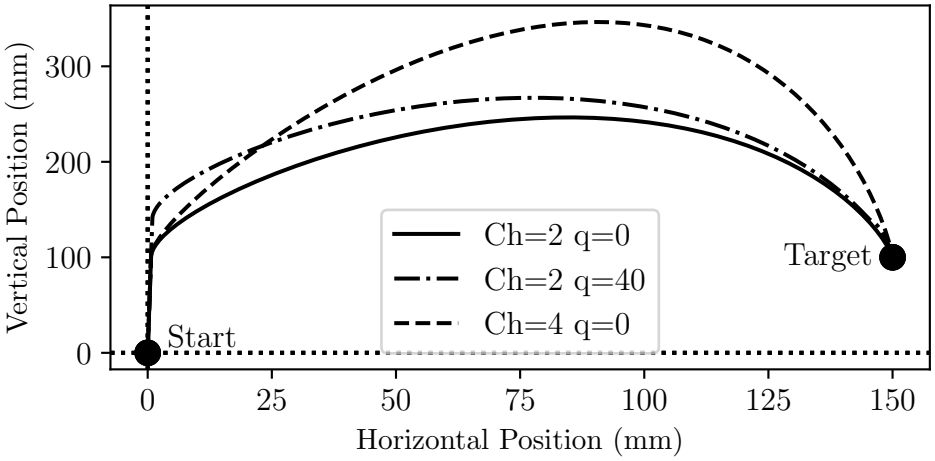


Figure 5.16: Elevated target example trajectories.

this is because the starting point is already above the target, and thus the sigmoid governing the vertical rise is close to negligible.

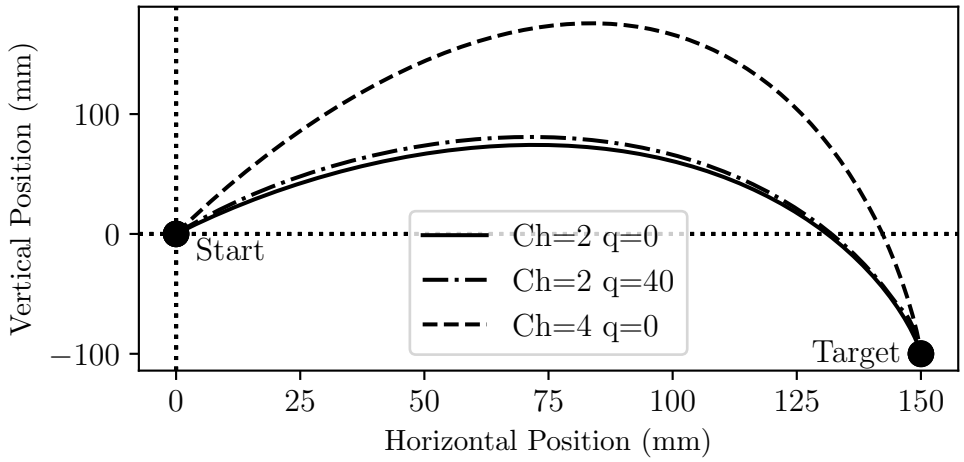


Figure 5.17: Lowered target example trajectories.

Next, two examples illustrating the recomputation capabilities provided by the flow function are shown. Figure 5.18 shows a similar initial setup to that in figure 5.18, where the star and initial end positions are on the same vertical level. However, partway through the trajectory the target position is changed to be further away and below the starting point, indicated by the square marker. As can be seen, the flow function competently adjusts the stride length to reach the new target. Importantly, the overall stride height is essentially unchanged by this operation.

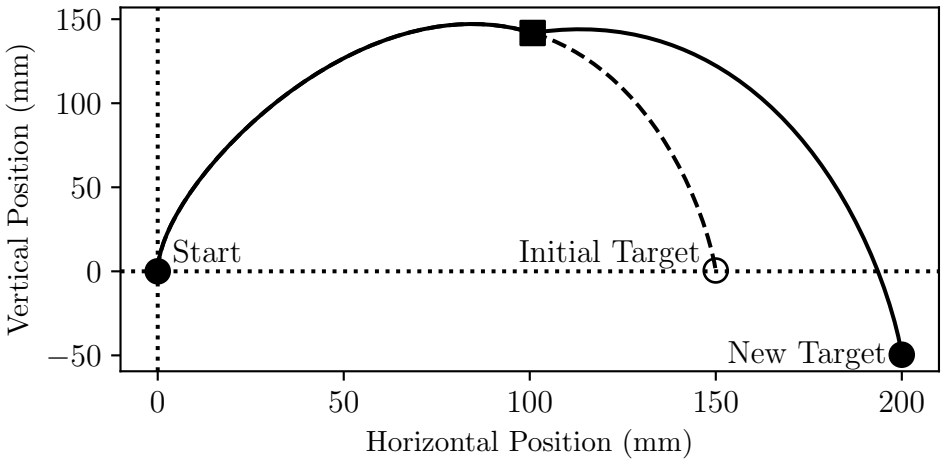


Figure 5.18: Recomputation example 1.

In the second example, shown in figure 5.19, the initial setup is equivalent to that in figure 5.17, with the initial target below the start point. The target is then adjusted to be above and closer to the start. Similarly to the previous example, a competent new trajectory is generated. The stride length is shortened and the stride height remains mostly unchanged. It should be noted that there is no vertical rise as there was in figure 5.16, this is because at the time the new target was provided, the foot was already above the new target.

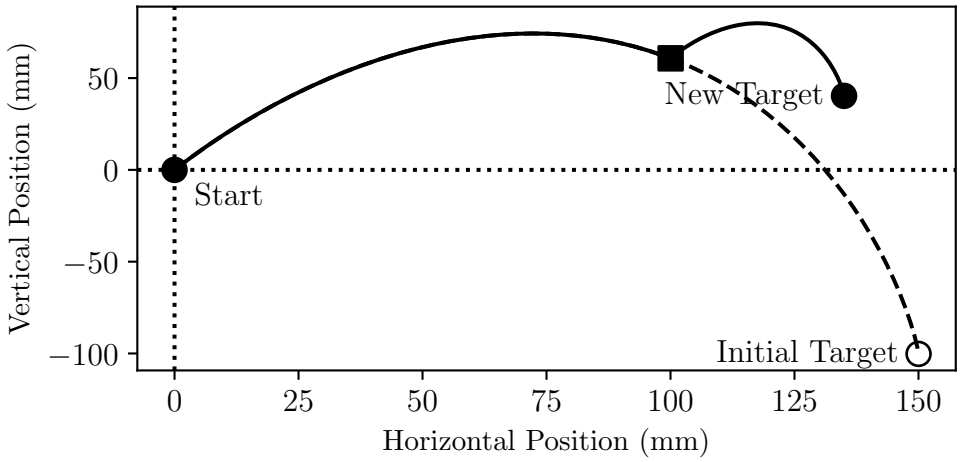


Figure 5.19: Recomputation example 2.

Chapter 6

Foot Placement Optimisation

This chapter describes the processes necessary for optimising the foot end positions, starting by describing the method of scoring the heightmap generated in chapter 4. Next, a semi radial search algorithm for locating the position with the best score is described. Finally, the process of choosing a reference floor height based on the terrain is described.

6.1 Overview

The best possible foot placement points for the supporting feet, given an initial point from the walking gait state machine, must be found based on the heightmap. This is achieved by assigning a walkability score to each cell in the heightmap based on the cell's slope and its proximity to edges. To find the first, best foot end position relative to the nominal foot end position, a semi radial search algorithm is used. Finally, since the robot will be walking over terrain, it is required to adjust its reference for the floor height.

6.2 Scoring

The scores considered are the slope and proximity scores. The slope score aims to reject points with high slopes while the proximity score rejects points close to other parts of the terrain with steep inclines, that is, to reject points inside holes or close to ledges. Note that cells with low scores are desirable placement positions and cells with higher values are less desirable.

6.2.1 Slope Score

The slope score is simply taken as the slope of the terrain at the current point. The aim of this score is to prevent the robot from slipping due to selecting anchor points with too steep of a slope.

As the heightmap slope is not defined by a known function, the slope is calculated using the Sobel operator (Sobel, 2014), a combination of a central finite difference and a smoothing operator.

Equations 6.1 and 6.2 describe the two separable x and y direction kernels, G_x and G_y, of the Sobel operator. These kernels are a combination of central finite difference and smoothing operator. Equation 6.3 combines G_x and G_y to produce the slope score, S_g. Note that these equations represent a single scalar result of the Sobel operator using the Frobenius inner product (Horn and Johnson, 2012). If evaluated over the whole heightmap, this is equivalent to convolution.

$$G_x(x,y) = \left\langle \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix}, \boldsymbol{h}_{i,j} \right\rangle_F \tag{6.1}$$

$$G_y(x,y) = \left\langle \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}, \boldsymbol{h}_{i,j} \right\rangle_F \tag{6.2}$$

$$S_g(x,y) = C_g \sqrt{G_x(x,y)^2 + G_y(x,y)^2} \tag{6.3}$$

where,

$$i = [\![x-1, x+1]\!]$$
$$j = [\![y-1, y+1]\!]$$

with C_g being the weighting constant associated with the slope score, S_g.

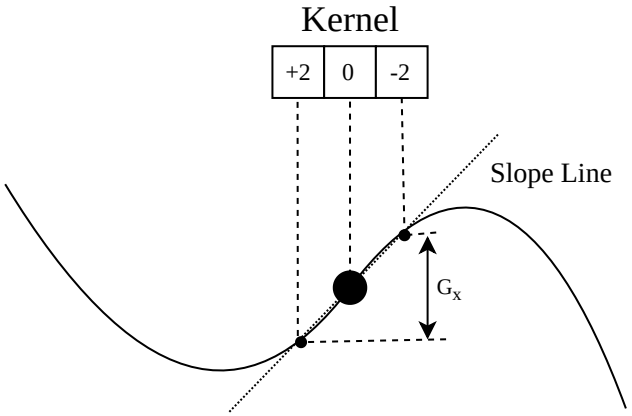


Figure 6.1: Slope Score

6.2.2 Edge Proximity Score

The proximity score aims to bias the selected anchor point away from points near steep inclines in the terrain. This score is defined as the average of the height difference of the current point weighted by the gaussian kernel $\boldsymbol{K}$, of size n by n. The distance around inclines that is rejected depends on the the chosen size of the kernel $\boldsymbol{K}$, the standard deviation of $\boldsymbol{K}$ and the height differences. This score is described in Equation 6.4.

$$S_p(x,y) = C_p \left| \frac{\sum \langle \boldsymbol{K}, (\boldsymbol{h}_{i,j} - \boldsymbol{h}_{x,y}) \rangle_F}{n^2} \right| \tag{6.4}$$

where,

$$i = [\![x - \lfloor \tfrac{1}{2}n \rfloor, x + \lfloor \tfrac{1}{2}n \rfloor]\!]$$
$$j = [\![y - \lfloor \tfrac{1}{2}n \rfloor, y + \lfloor \tfrac{1}{2}n \rfloor]\!]$$

with x and y being the indices of the cell whose score is currently being evaluated, $\boldsymbol{h}$, the heightmap and C_p the score's weighting constant.

A diagrammatic, sliced, representation of the proximity score can be seen in Figure 6.2. The slice is taken along the x axis of the heightmap.

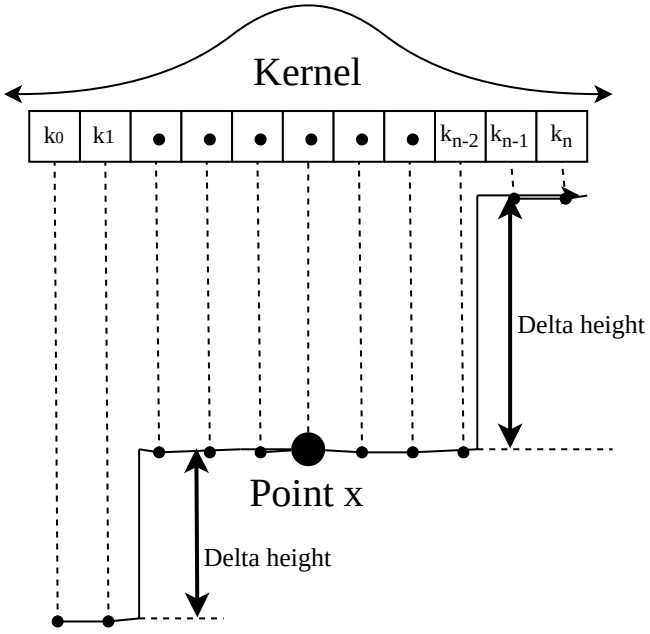


Figure 6.2: Proximity Score Diagrammatic Representation

6.2.3 Constraints

It is important to constrain the possible anchor points to conform to the stability triangle, meaning that the centre of mass of the robot must be inside the triangle formed by the three anchor points. Additionally, it is important that the points selected are not too far away, both in the horizontal and vertical direction for the robot to reach.

6.3 Score Search Algorithm

Once the heightmap has been processed into the score map, which is done by adding the slope score and terrain proximity score, the point with the best score must be found for every initial anchor point. The resolution of the heightmap is not very high and the adjusted anchor point is not allowed to deviate too far from the initial anchor point. Additionally, due to the parallel nature of the heightmap generation, it is possible to score the entire heightmap with minimal cost. Thus, it was decided to not employ an optimisation algorithm, such as gradient decent or Bayesian, but rather to use a radial search algorithm. This algorithm progressively expands its searching radius over the square score grid until a valid score is found, at which point it terminates, thus ensuring that the closest valid point to the initial anchor point is selected. See Figure 6.3 for a diagram representing the search pattern for a 5 grid squares search area. Note that this search pattern will become inaccurate for large search areas, as the pattern steps in a square manner. This however is not much of a concern for smaller search areas.

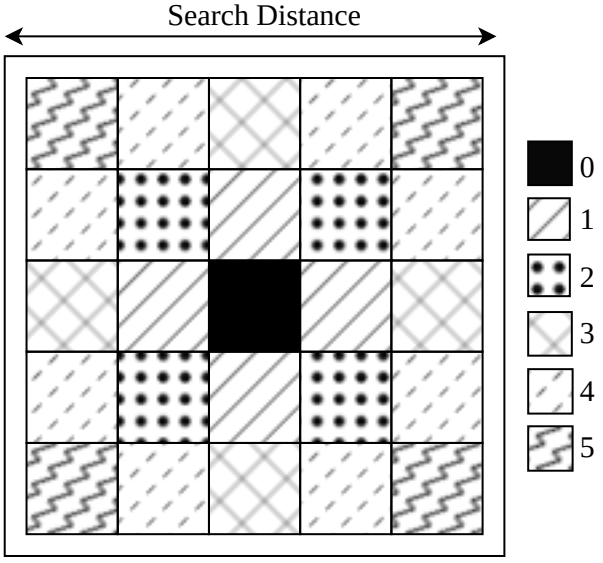


Figure 6.3: Radial search, shown for a 5 cell search diameter.

6.4 Floor Height Adjustment

After the horizontal position of the feet have been optimised, the height of the foot is simply set to the height of the terrain at the target horizontal coordinates. This ensures that the robots body remains level and at a constant global height.

It is however important to adjust the reference floor height that the robot uses to adapt to a varying average terrain height. If this is not done, then the robot will be incapable of surmounting terrain higher than its commanded standing height, as the robot will simply maintain said height and walk into the tall terrain. There are various methods to choosing a floor height, from using a time of flight sensor underneath the robots main body to using height data from the heightmap.

The method that was implemented in this project uses the average of the three highest target foot placement positions out of all the robots legs. This allows the robot to pre-emptively increase or decrease its floor height as it targets to step onto higher or lower terrain.

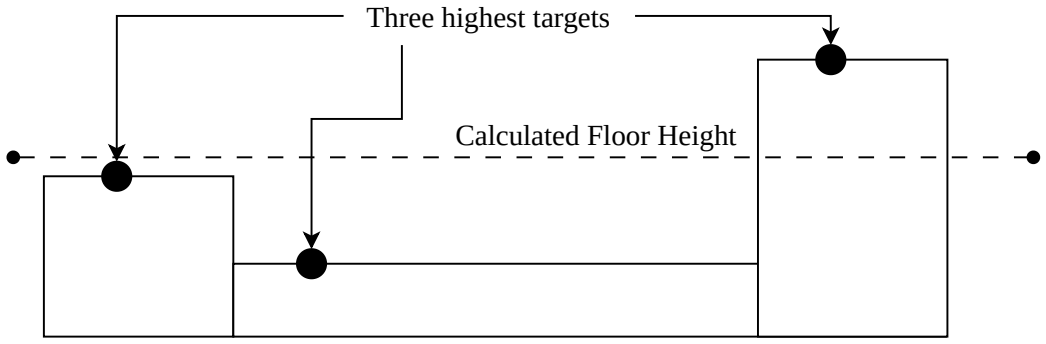
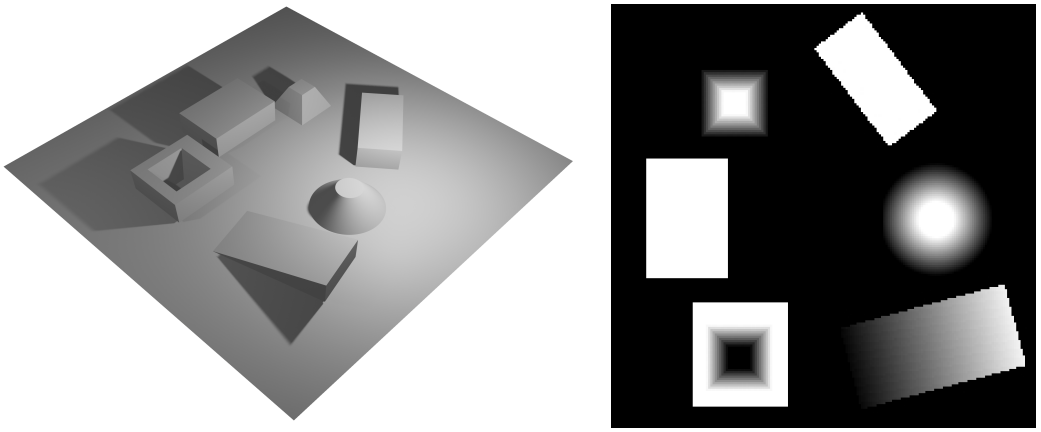


Figure 6.4: Floor height diagram.

While this method works well to pre-emptively adjust height and to optimise the body height relative to the foot height, it does present one problem: If there is a tall piece of terrain in the path of the robot's body without any feet positioned on it, then the robot will not adjust its body height to clear this piece of terrain. Thus, the bounding box of the robot's body, excluding the legs and feet, is checked against the heightmap. If any point on the heightmap, that is inside the bounding box, is higher than the previously set floor height, the floor height is updated to this increased height.

6.5 Scoring Test on Simulated Terrain

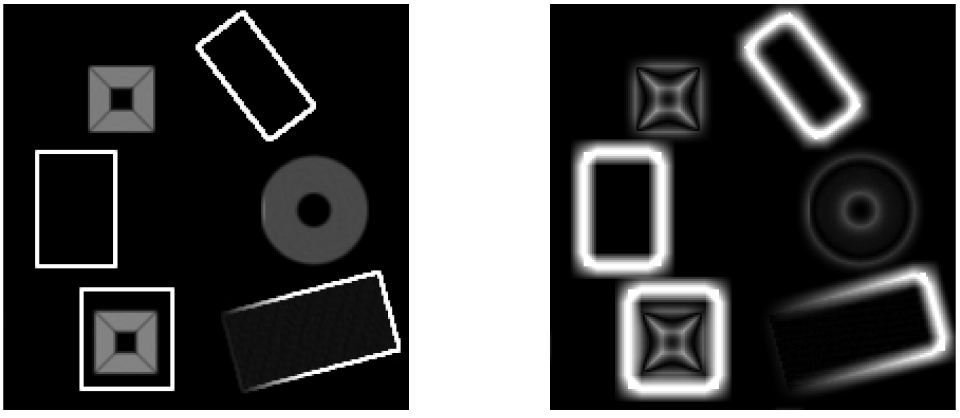
For testing the walkability scores described in section 6.2, a piece of terrain that demonstrates various types of obstacles that the robot might encounter was simulated. This terrain and its heightmap is shown in figure 6.5.



(a) 3D Terrain (b) Heightmap of 3D Terrain

Figure 6.5: 3D Model of Terrain and its heightmap.

The heightmap in figure 6.5b is used to test the various walkability scores. First, in order to show individual results, the slope and edge proximity scores are applied to the heightmap separately. This is shown in figure 6.6. Next the combined score is shown in figure 6.7. This is the score that is used to optimise foot end positions. The score images show a gradient from black to



(a) Slope score test. (b) Edge proximity score test.

Figure 6.6: Slope score and edge proximity score tests.

white, darker colours indicate more desirable foot placement positions, while light colors indicate undesirable locations.

The slope score can be seen in figure 6.6a. From this it can be seen that the slope score successfully indicates sloped areas as less suitable foot end positions. It is also clear that vertical inclines are strongly discouraged, and while not the primary purpose of the slope score, this is expected and does not have a negative impact on scoring.

Figure 6.6b shows that edge proximity score successfully discourages foot placement in areas with large height deviations, while allowing placement in areas with a locally similar slope, such as the ramp in the bottom-right. As can be seen from the larger, and more intense, discourage areas around the boxes in the top-right, bottom-right, left and bottom-left, it is clear that the magnitude of the height differential will increase the size and strength of the rejection area. Next, in figure 6.7 the combined slope and edge proximity scores can be seen. This is simply the sum of the two scores.

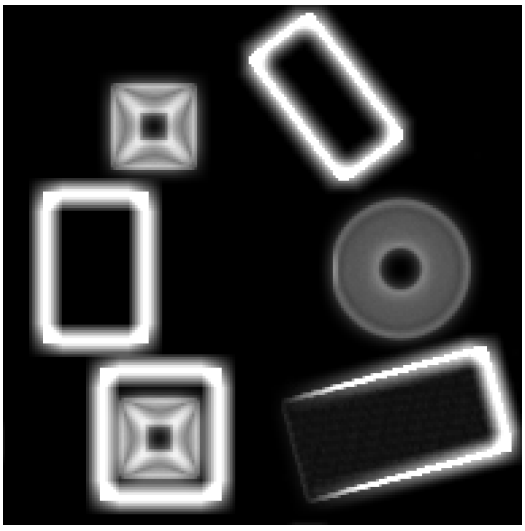


Figure 6.7: Full walkability score.

Finally, the radial search algorithm described in section 6.3 is used to optimise various nominal foot end positions. This is shown in figure 6.8. In this figure the blue box indicates the search area of the radial search algorithm from section 6.3. The red marker indicates the nominal foot position that would be received from the baseline motion system, and the green marker is the walkability score optimised foot position. If there is only a green marker this means that the nominal position was already valid, and no adjustment is necessary.

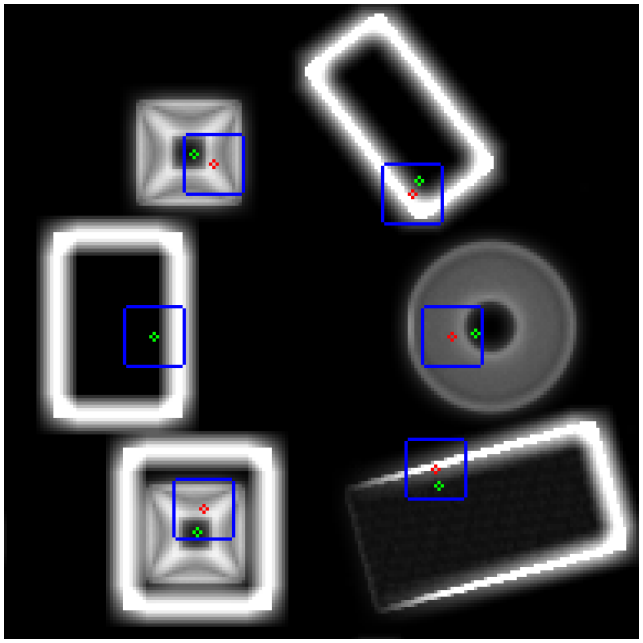


Figure 6.8: Walkability score optimisation test.

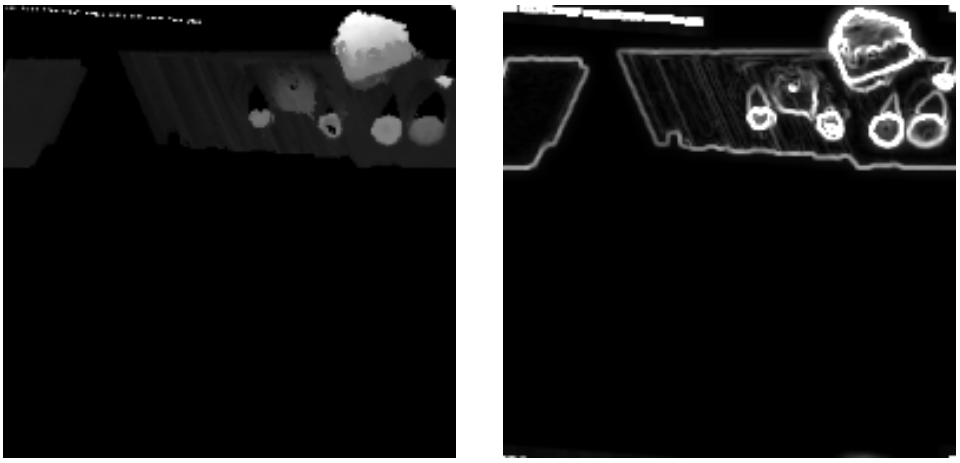
If the nominal foot position has an acceptable score, the position will not be adjusted. This can be seen with the middle-left nominal foot placement position in figure 6.8 where the green marker coincides with the red marker. If however the nominal placement position is too close to an edge, such as with the top-right example, the nominal position is adjusted to the closest acceptable position. The adjusted position does not need to have a perfect score, as illustrated by the bottom-right example. This example is similar to the top-right example, with the nominal foot placement located too close to an edge. However, it adjusts the foot placement onto the ramp, which has a non-zero, but acceptable, walkability score. Next, the middle-right example shows how the nominal placement position is placed on a piece of terrain that is too steep. Thus, it is adjusted onto the flat surface on top of the round pedestal.

Finally, the top-left and bottom-left examples show the cases of nominal foot placement positions placed on a sloped pillar and a sloped hole, respectively. In both cases the nominal placements are adjusted to the flat surface at the top of the pillar and at the bottom of the hole respectively. (The foot could be placed at the bottom of the hole, because the height there is known.).

The flat surfaces on the pillar an in the hole are large enough not to be rejected by the edge proximity score. However, if the slopes were steeper or the flat surfaces were smaller, these nominal positions could be unsolvable, as there would be no suitable foot placement positions inside the search area. In this case the robot would need to make a course adjustment.

6.6 Scoring Test on Physical Terrain

The walkability scoring system was also tested on the practical heightmap that was generated from real RGB-D images recorded by the physical hexapod. Using the same colour scheme as above, the practical heightmap found in section 4.7 is again shown for reference in figure 6.9a. The scoring system was then applied to this heightmap and the resulting walkability score map is shown in figure 6.9b.



(a) Hardware heightmap (b) Score applied to hardware heightmap.

Figure 6.9: Heightmap generated from hardware and the corresponding score map.

As can be seen from figure 6.9 the scoring system applied to the hardware-generated heightmap functions similarly to when it is applied to a simulated heightmap. The primary difference that should be noted is the score artifacts created around the mapped areas edge, seen as an outline around the mapped area. Light score lines can also be seen as diagonal stripes within the mapped area. These lines are most clear on the portion of flat terrain. The lines are created by the moving field of vision and inaccuracies in the pose estimate obtained from ORB-SLAM3, mostly inaccuracies in the height estimates, as explained previously in section 4.7.

It should also be noted that, while the score for the sloped plane at the top of the heightmap is not as even as it was in the ideal heightmap, even with these undulations present it is still sufficiently accurate for the correct foot position adjustment to be made. This can be seen in figure 6.10 which shows how different hypothetical nominal foot placement positions would be adjusted based on the hardware-generated score map.

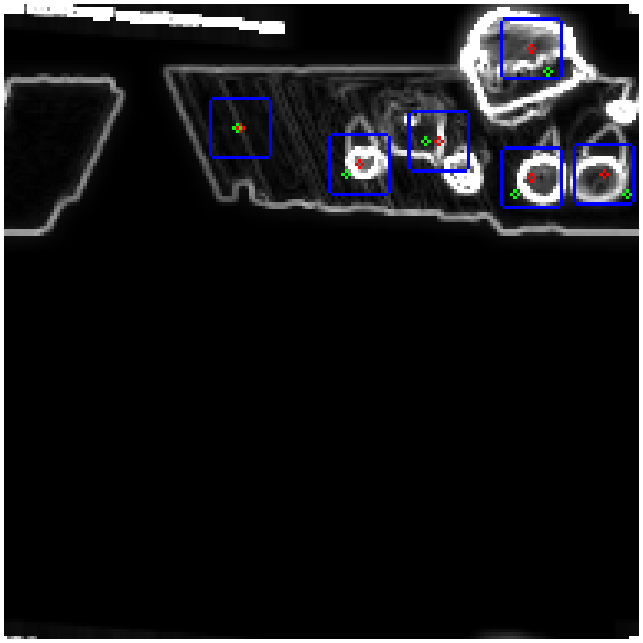


Figure 6.10: Test points on hardware score map.

The results show that all of the nominal foot placements are adjusted to suitable new placement positions, where necessary. Note the point applied to the sloped plane at the top of the map is adjusted appropriately even with the slightly undulating score map across the sloped area.

Additionally, it should be noted that while some of the outlining artifacts do grow large enough to affect the final foot positions, as can be seen in the left most test point, these effects are generally not large enough to be of concern.

Chapter 7

Hardware Implementation

This chapter describes the practical implementation of the system on the physical hexapod robot. Section 7.1 provides an overview of the implementation. Section 7.2 describes the ROS in the system, their functions and the data they communicate. Appendix **??** describes the details of the ROS messages sent by the nodes and their data types.

7.1 Overview

The entire system was implemented in the Robot Operating System (ROS) environment, ROS is a system which, among other things, greatly simplifies the task of communication between devices using various different communication protocols. This is achieved by sectioning all code into different ROS nodes, where every node is treated as a component in a large network. All ros nodes are joined together by the core node, which is hosted at a set IP address on the network.

7.2 ROS Architecture

ROS operates by the concept of nodes, publishers and subscribers. A ROS node is a piece of code executed in the ROS network, publishers and subscribers are initialised by a node. A publisher will send a message into the ROS network, while a subscriber will listen for a certain message on the network. When a subscriber receives a message it will execute a user defined callback function to use the received data.

The various ROS nodes are distributed between the base station on the desktop computer, and the Jetson Nano and the Teensy MCU on the hexapod robot. Figure 7.1 provides an overview of the nodes and how they communicate with each other.

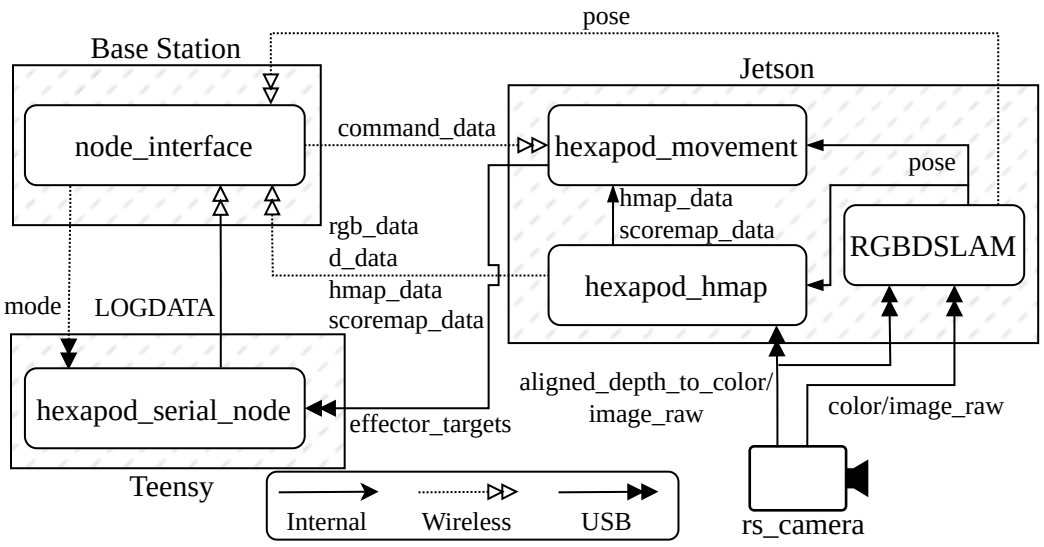


Figure 7.1: ROS nodes and communication.

Note that the data transfer arrows between the base station and the teensy are a combination of the wireless and USB arrows, this implies that the data passes to the Jetson as a intermediate step. Transfers between the base station and the Jetson are wireless, while between the Jetson and the Teensy are through USB. Section 7.2.1 and 7.2.2 provide details on the node architecture shown in figure 7.1.

7.2.1 Base Station

The base station consists of a single ROS node, this node is responsible for transmitting all commands to and receiving data from the robot. The base station publishes two messages, the mode message and the command data message. As the names imply the mode message contains the commanded mode that the robot should switch to, while the command data message contains the remaining commands that the operator could issue. The mode is separated from the other commands because it is only sent to the Teensy, while the other commands are sent to the Jetson.

The base station subscribes to the pose message from the perception node, and the rgb data, d data and hmap data from the hexapod map node, and also the LOGDATA message from the hexapod serial node. The pose message contains the pose estimate produced from ORB-SLAM3, while the rgb data, d data and hmap data message contain the color, depth and heightmap images respectively. The LOGDATA message contains generic string log messages from the Teensy.

The base station is what the operator uses to observe and issue commands

to the robot, as such there is wireless communication link between the robot and the base station. Among the commands the operator can send the robot is the operating mode, the robot currently only has two operating modes, torque cutoff mode and walking mode. The torque cutoff mode is the initial mode the robot is in, while in this mode the leg servos disable all torque control, thus entering a relaxed state. In walking mode the robot walks in the commanded direction whilst optimising its foot positions according to the terrain. If the robot encounters a piece of terrain for which no optimisation can be found the human controller will have to adjust the walking direction from the base station.

7.2.2 On Board

The hexapod has two computational units on board, first the Jetson Nano, which handles all high level operations, including heightmap generation and scoring, foot optimisation, maintain a walking gait and localisation using ORB-SLAM3. Secondly a Teensy2.0 MCU handles low level operations, including interpolating feet movement paths and servo control.

The Jetson hosts four different nodes, the hexapod hmap, hexapod movement, RGBDSLAM and rs camera nodes. The rs camera node publishes the camera stream consisting of the color and aligned depth to color messages. As could be deduced from the names, the depth message is aligned to the color. These messages are transmitted at a rate of 30Hz.

The RGBDSLAM module runs the ORB-SLAM3 algorithm and subscribes to the depth and color streams from the rs camera node as inputs. As ORB-SLAM3 uses both the color and depth streams for pose estimation these streams are synchronised in time. Once the pose is estimated it is published in the pose message.

The hexapod hmap node is responsible for generating the heightmap and score maps. It subscribes to the pose, depth and color messages, the depth and color messages are downsampled before being used to generate the heightmap. The entire generation process is ran on the Jetson Nano's GPU as described in chapter 4. Once the heightmap and scoremap are generated they are published in the hmap data and scoremap data messages. Additionally the downsampled colour and depth messages are also published for transmission to the base station.

Next, the hexapod movement node subscribes to the command, pose, hmap and scoremap messages to generate and optimise the feet targets, as described in chapter 5 and 6. The feet targets are published as the effector targets message.

Lastly, the Teensy MCU also consists of a single ROS node, the hexapod serial node. The functionality of this node is simply to facilitate communication over USB. The node is subscribed to the mode and effector targets

messages, these messages are then used to calculate the feet trajectories as described in chapter 5. The trajectories are then translated into servo angles and angular rates, also described in chapter 5. Whereafter the servos are actuated. The only publisher present on the Teensy is the LOGDATA publisher, which simply publishes general string log messages for use by the operator.

Chapter 8

Final Walking Tests

The final walking tests are comprised of three different terrains: a simple flat plane as a baseline, then a staircase to demonstrate simple foot and body height adjustment, and finally a uneven organic like surface.

Please note that all data is plotted with the world axis aligned to the robot's starting orientation in the world. This is done for graphing clarity.

8.1 Flat Terrain

The flat plane test aims to quantify nominal body height oscillations and to provide a baseline to compare subsequent test to. Figure 8.1 and 8.2 shows screenshots of the simulation test being performed in MuJoCo. A video of the simulation can be seen here.

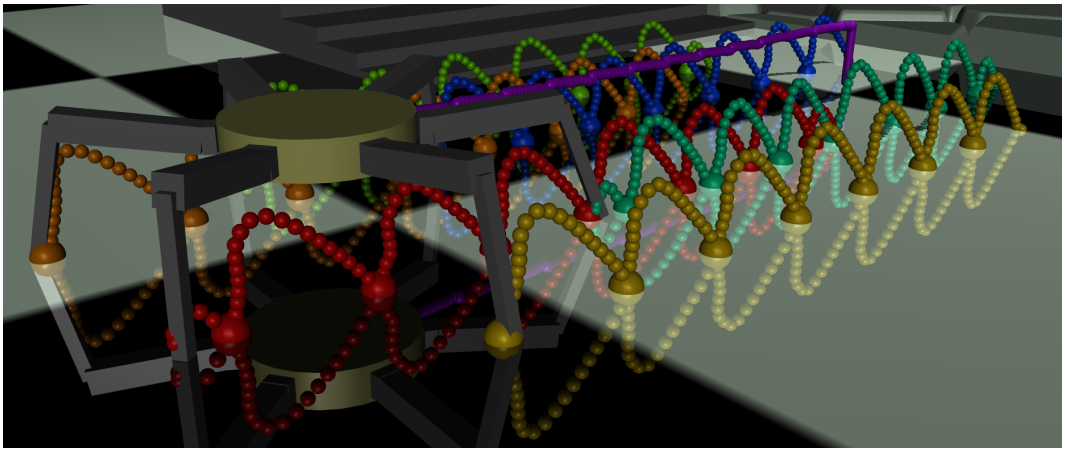


Figure 8.1: Flat terrain MuJoCo view.

The test was performed with a nominal stride length of 15cm, while the flow function parameters Ch and q were set to 2.0 and 14.0 respectively. This

equates to a desired step height roughly equal to true stride length, which is dependant on the terrain. The target body height was set to 13.5cm above the virtual floor height (see section 6.4). The robot was commanded to walk forwards at a constant speed.

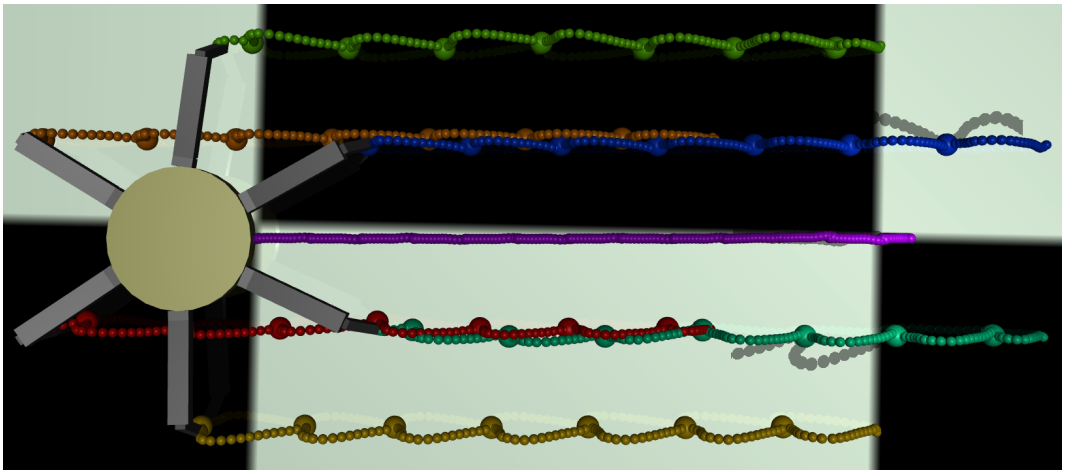


Figure 8.2: Flat terrain MuJoCo top view.

The simulation results for the body feet trajectories are show in Figure 8.3,

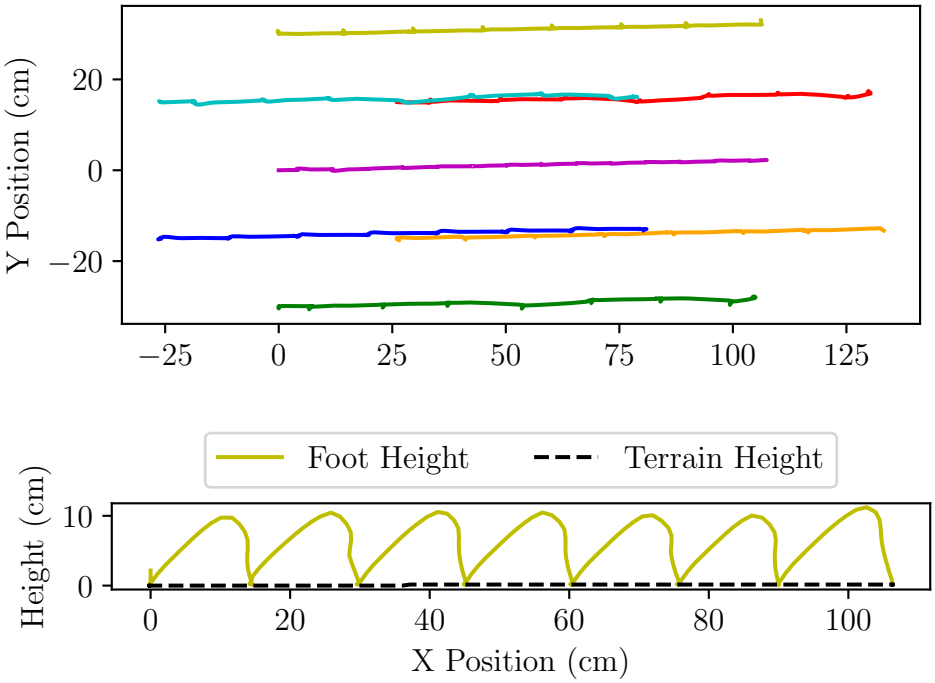


Figure 8.3: Flat terrain. Feet top view (Top). One foot side view (Bottom).

where the color codes match that of the above screenshots. It can be seen that the stride length is equal to the set stride of $15cm$. The robot also maintains a linear trajectory throughout the test. Note that the step height is about $10cm$ which is a little lower than expected. The reason for this is that the swinging feet are being "pushed" through the flow function faster by the supporting legs faster than expected. Thus, resulting in a lower arc.

Figure 8.4 shows the body height and rotation. Here it can be seen that there is a constant error of about 1cm in the body height of the robot. The reason for this is that there is no body height feedback control, as such, errors in the servos accumulate to result in the body height error. This can be easily solved by adding a constant offset or by adding feedback control for the body height. It is also clear that as the robot walks there are some oscillations

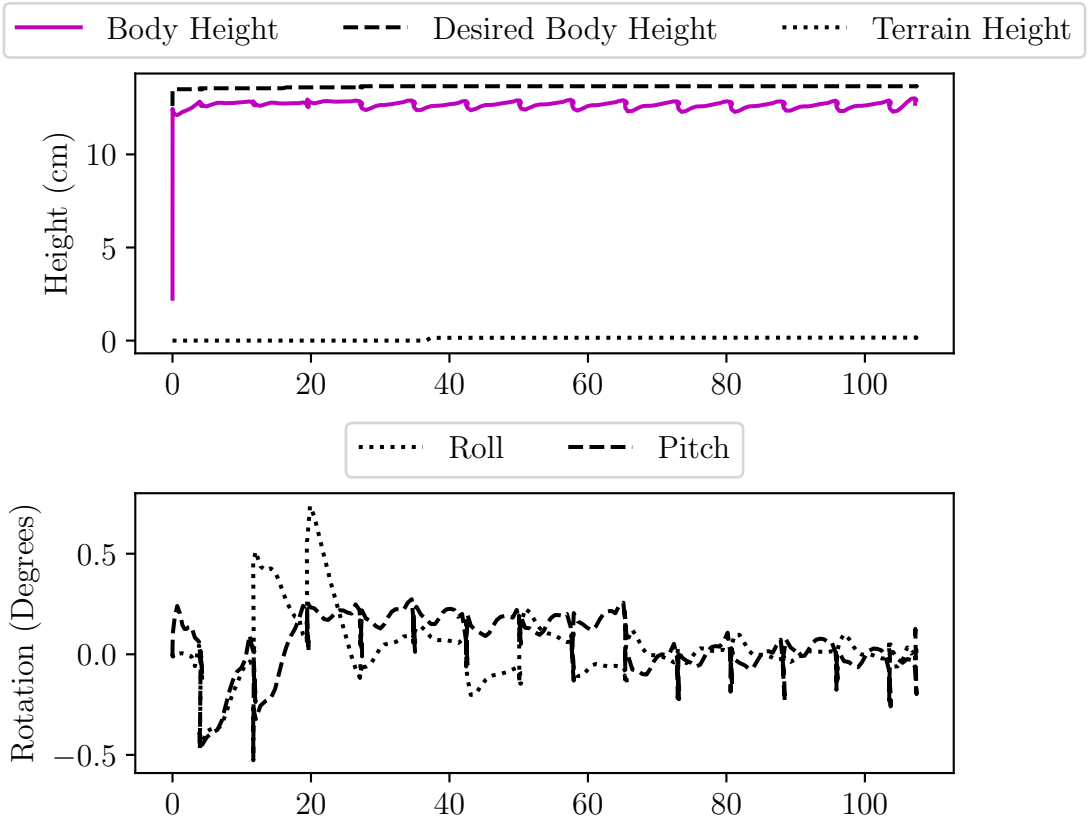


Figure 8.4: Flat terrain. Body height (Top). Body tilt (Bottom).

in the body height. This is to be expected as the servos are not modeled as having infinite torque, thus, the robot will sag when three legs are lifted off the ground, and rebound once those legs are placed on the floor again. Similarly to the body height error, adding feedback control on the body height would

reduce these oscillations. The amplitude of the oscillations are lower than the 10% of the desired body height, which is acceptable. Next, it can be seen that the tilt of the body also exhibit small oscillations, in both roll and pitch. However, the amplitudes of these oscillations remain below 0.5 degrees.

8.2 Staircase

A staircase test was performed where the hexapod was commanded to move onto a set of steps. This tested the system's ability to choose foot placements that are not close to edges, to maintain a level body, and to automatically adjust its body height relative to the local terrain height. Figure 8.5 show screenshots of the test simulation in MuJoCo. A video of the simulation can be found here.

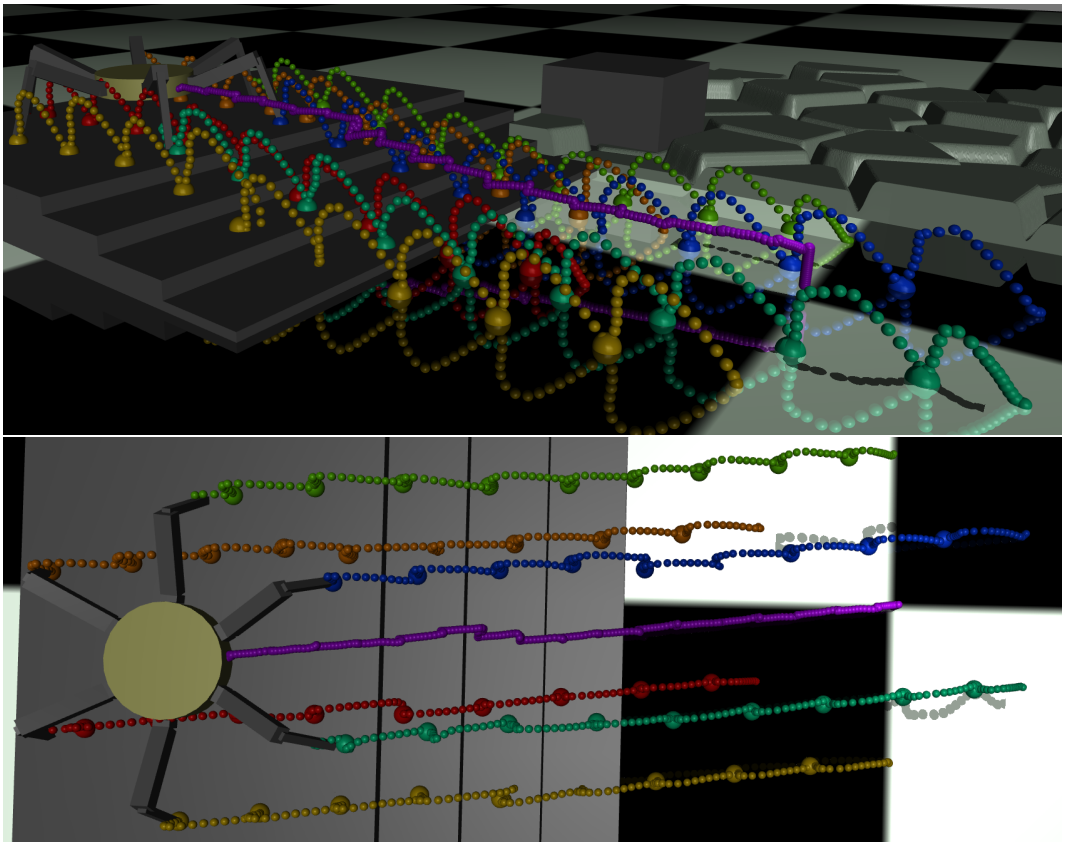


Figure 8.5: Stairs MuJoCo view.

The test was performed with a nominal 15cm stride length, the flow parameters Ch and q were set to 2.0 and 14.0 respectively. This results in a step height roughly equal to the true stride length. The target body height is set to 9cm

target height above the virtual floor height (see section 6.4). The robot was commanded to walk forwards at a constant velocity.

The body and feet trajectories are show in figure 8.6. The robot maintains a mostly linear trajectory up the staris, with a small deviation around 80cm down the X axis. The stride lenght is mostly equal to the nominal 15cm, however, the step starting at 70cm has a visibly longer stride length, this is to avoid stepping on the edge of the step. Also note that some steps start with a small loop, this occurs when the robot bumps itself backwards with the front feet. This happens when the front feet push into the ground during a longer stride, due to the slight error in the body height. Similarly to the flat terrain test, the constant body height error is still present.

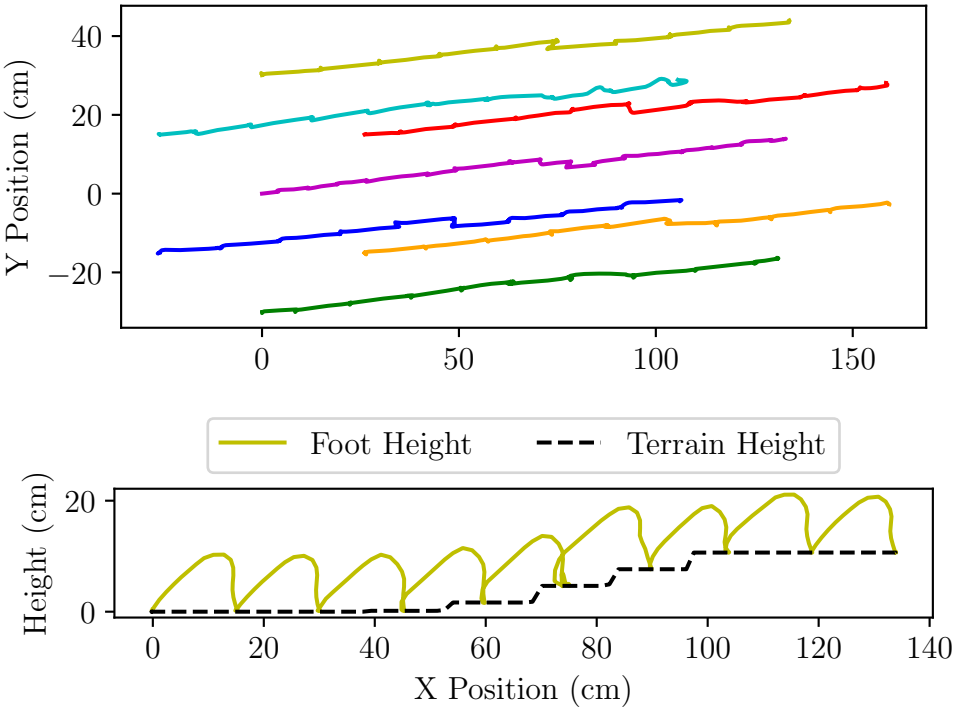


Figure 8.6: Stairs. Feet top view (Top). One foot side view (Bottom).

Figure 8.7 shows the body height and rotation. From this it can be seen that the robot does increase its height to maintain a constant height above the terrain. It is clear that the body height is not increased in the four discrete steps as the stairs do, rather the body height is increased more smoothly using many smaller steps. This is thanks to the system described in section 6.4. The oscillations in the body tilt angles are more prominent in the staircase test than in the flat terrain test, but still mostly stay below 1 degrees, with some intermittent jumps approaching 2 degrees.

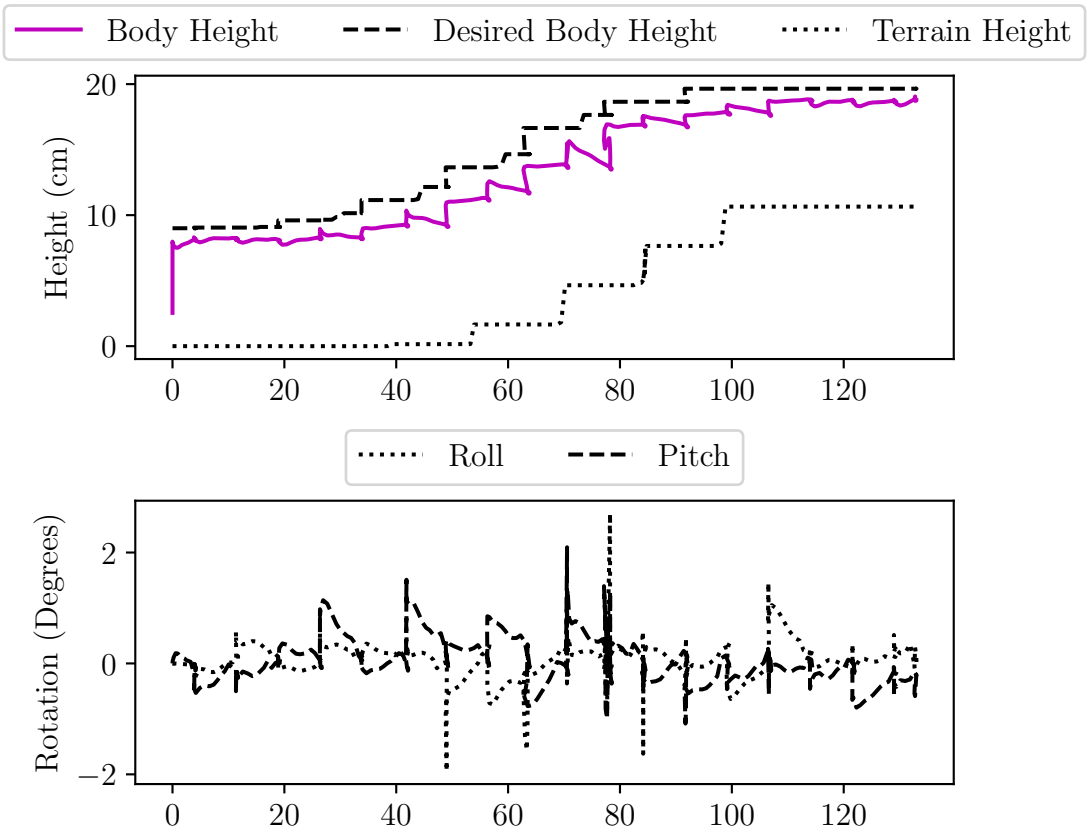


Figure 8.7: Stairs. Feet top view (Top). One foot side view (Bottom).

8.3 Cobblestone terrain

An organic terrain test was performed where the hexapod was commanded to move over uneven terrain consisting of "cobblestones" with grooves between them and with varying slopes and heights. Figure 8.8 and 8.9 show screenshots

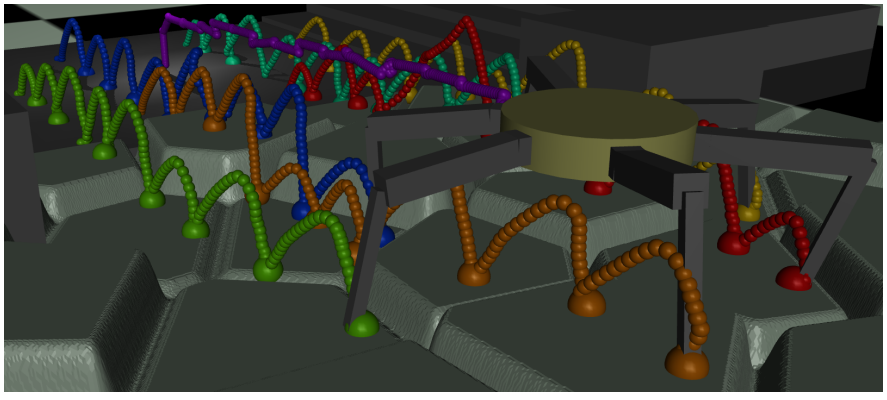


Figure 8.8: Cobblestone test MuJoCo view.

of the cobblestone test in MuJoCo. A video of the simulated test can be seen here. The cobblestones presented islands of suitable foot placement locations, while the grooves represented areas that are not suitable for foot placement. This tested the system's ability to choose suitable foot placements, to maintain a level body and a desired body height, and to traverse uneven terrain.

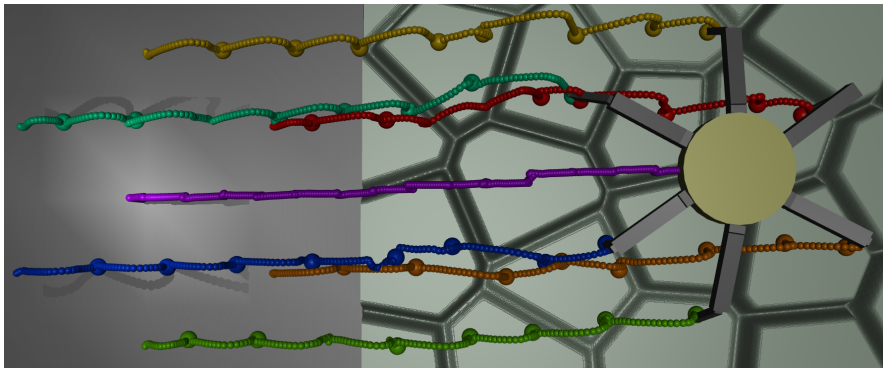


Figure 8.9: Cobblestone test MuJoCo top view.

Similar to past tests, the robot was commanded to walk over the terrain in a straight line at a constant velocity. The nominal stride length was set to $15cm$ and the flow parameters Ch and q were set to 2.0 and 14.0 respectively. Figure 8.10 shows the body and leg trajectories. The robot still maintains a straight

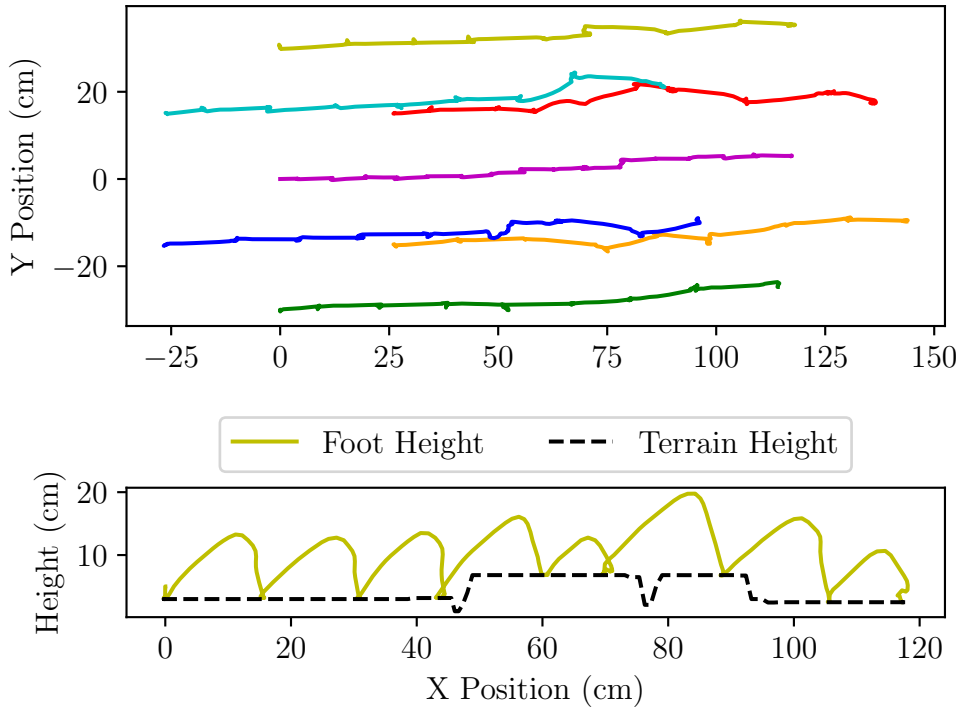


Figure 8.10: Cobblestones. Feet top view (Top). One foot side view (Bottom).

trajectory over the terrain. This test clearly demonstrates how the robot adjusts its feet targets in the horizontal plane to place them on the cobbles. Similarly, stride length is also greatly adjusted, as can be seen from figure 8.10.

Figure 8.11 shows the body heigh and rotation during the cobblestone terrain test. From this it can be seen that the body height error seen in the

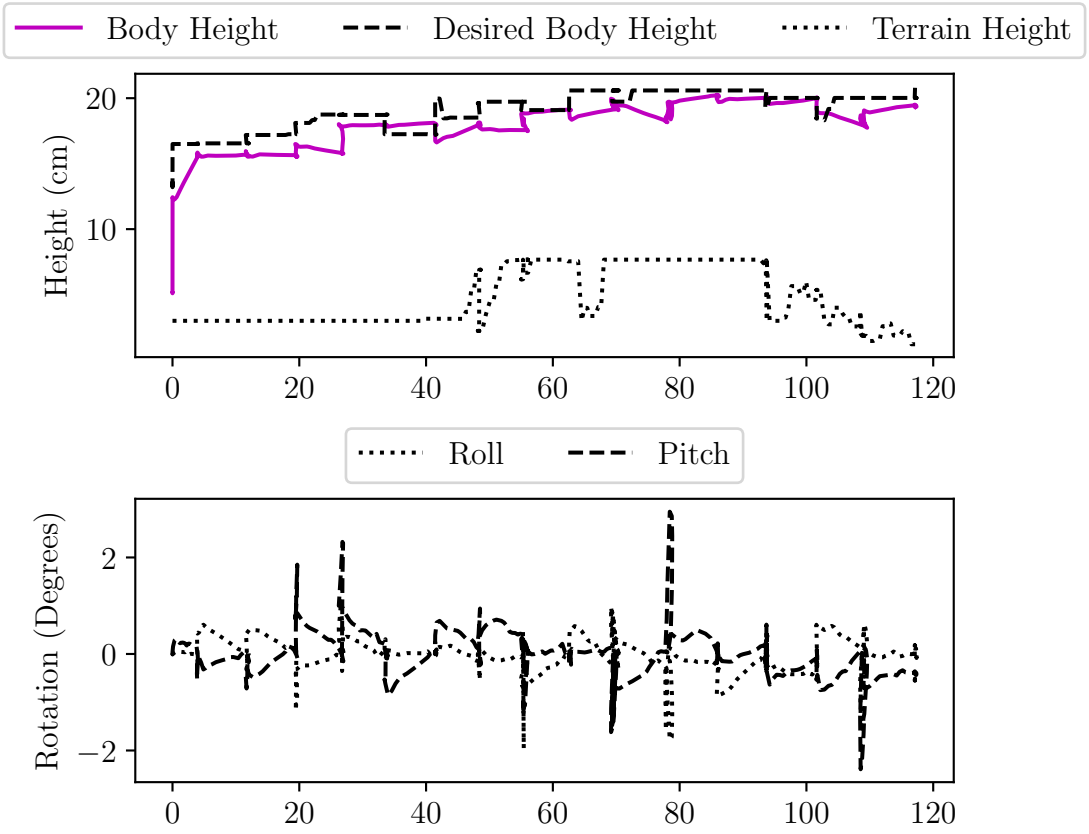


Figure 8.11: Cobblestones. Body height (Top). Body tilt (Bottom).

flat plane test is still present, and more severe. However, the body height still stays within a reasonable margin from the desired height.

For the most part, the oscillations in the body rotation is similar to that of the stairs test. However, they are more frequent, and more sever spikes are present. This is due, in part, to the more erratic mismatch between the body height and the requested body height. The horizontal adjustments made to the foot placement positions cause the swinging feet to end their steps at different times. This is the primary cause of the tilt spikes, as when one foot is placed on the floor before the other two, the robot is tilted slightly. This, however, would not occur if there was no, or little, error in the body height control.

Chapter 9

Conclusions

9.1 Summary and Conclusions

This thesis presented the development of a foot placement planning system for a hexapod robot to enable it to traverse uneven terrain containing height changes, sloped surfaces, and edges. A simulation model of the existing physical hexapod robot, and its environment, was created in MuJoCo, an advanced physics engine with superior contact physics modelling. A real time local dense-mapping system was developed by using depth images obtained from an RGB-D camera mounted on the robot. The mapping system also utilised pose estimations from ORB-SLAM3, a stereoscopic visual SLAM algorithm. ORB-SLAM3 was implemented using a third party library. A walkability scoring system was developed to further process the heightmap generated by the mapping system, resulting in a walkability map. The walkability map indicates where on the terrain the robot can and can not safely step. Next, a baseline motion control system that enables the robot to walk on flat terrain was developed. The baseline system was then further modified with a algorithm that adjusts the nominal foot positions from the baseline motion control system based on the score map. The resulting new foot positions are then adjusted both in the horizontal and vertical plane to a position on the terrain that is safe to step on, all while the robot maintains a level body. The robot's floor height reference was also updated based on the terrain map, this allowed the robot to adjust its body height appropriately to the average terrain height. Furthermore, a new foot trajectory generation system based on a flow function was developed to produce a trajectory based on the horizontal and vertical distance to the foot's destination.

The mapping, scoring, and optimal foot placement systems were tested in simulation and on practical RGB-D images recorded by the physical hexapod robot. Simulation tests were performed to demonstrate that the integrated system can successfully control the hexapod robot to walk on flat terrain,

staircase terrain, and uneven terrain consisting of "cobblestones" with grooves between them. The simulation results show that the hexapod is able to traverse both flat and uneven terrain, is able to adjust both its horizontal and vertical foot placements in response to the terrain.

9.2 Future Work

Below is listed possible future tests, improvements, and additions to the system.

1. Although the mapping and scoring was implemented and practically tested on the physical hexapod, the motion control system was only tested in simulation on uneven terrain. Future work should include practical tests with the physical hexapod traversing uneven terrain such as steps and cobblestones.

2. The variations in the height estimate received from the ORB-SLAM3 system caused artifacts in the heightmap and the score map. Methods to improve the height estimate should be investigated.

3. The foot arcs are currently only adjusted for the new foot placements, but are not adjusted to avoid obstacles in the terrain. Future work could look at detected obstacles close to the legs and adjusting he arcs to avoid collisions.

4. The hexapod currently "freezes" if it cannot find suitable alternate foot placement positions for the given terrain, and the human operator must then manually change the commanded direction of motion. A short-term path planner should be investigated that can autonomously change the hexapod's planned path to circumvent the terrain that cannot be crossed.

5. The mapping and scoring system currently does not identify hazardous terrain such as pools of water or loose earth. Terrain classification and image segmentation algorithms could be investigated to add a "terrain type" score to to the walkability score.

Appendix A

Reference Frame Transforms

Reference frame transforms are defined by the shorthand $T_{AB}(\boldsymbol{x}^{\mathcal{A}})$, meaning vector $\boldsymbol{x}^{\mathcal{A}}$ is transformed from reference frame $\mathcal{A}$ to $\mathcal{B}$. Reference frames used are the camera frame, $\mathcal{C}$, the map frame, $\mathcal{M}$, the body frame, $\mathcal{B}$, the leg reference frames, $\mathcal{L}_i$, and the world frame, $\mathcal{W}$. Transforms used are descried in this appendix.

A.1 Camera to World

Transforming a vector from camera to map space requires rotating the vector by the camera quaternion and adding the camera's global position to the vector. As shown in equation A.1.

$$\begin{aligned} \boldsymbol{p}^{\mathcal{W}} &= T_{\mathcal{CW}}(\boldsymbol{p}^{\mathcal{C}}) \\ &= \boldsymbol{q}_{\mathcal{CW}} \cdot \boldsymbol{p}^{\mathcal{C}} \cdot \boldsymbol{q}_{\mathcal{CW}} - 1 + \boldsymbol{p}^{\mathcal{W}}_{C0} \end{aligned} \tag{A.1}$$

where $\boldsymbol{p}^{\mathcal{W}}$ is the position in the world frame, $\boldsymbol{q}_{\mathcal{CW}}$ is the quaternion rotation of the world frame relative to the camera frame, and $\boldsymbol{p}^{\mathcal{W}}_{C0}$ is the position of the camera frame's origin in the world frame.

A.2 World to Map

The world to map transform is similar to camera to map tansform, except that the rotation is not required, as the map and world frames are already aligned. This is shown in equation A.2.

$$\begin{aligned} \boldsymbol{x}^{\mathcal{M}} &= T_{\mathcal{WM}}(\boldsymbol{x}^{\mathcal{W}}) \\ &= \boldsymbol{x}^{\mathcal{W}}\Delta \mod N \end{aligned} \tag{A.2}$$

where Δ is the scaling factor used to relate heightmap cells to cm, and the heightmap buffer is of size $N \times N$.

A.3 Local to Map

Local to map space is very similar to equation A.2, with the exception of using the body quaternion instead of the camera quaterion. This can be seen in equation A.3.

$$\boldsymbol{x}^{\mathcal{M}} = (\boldsymbol{q}_{\text{bod}} \cdot \boldsymbol{x}^{\mathcal{B}} \cdot \boldsymbol{q}_{\text{bod}}^{-1}\Delta + \boldsymbol{p}_{\text{rob}}^{\mathcal{M}}) \mod N \tag{A.3}$$

where $\boldsymbol{x}^{\mathcal{B}}$ is the vector in local space, $\boldsymbol{q}_{\text{bod}}$ is the robot body's quaternion that rotates $\boldsymbol{x}^{\mathcal{B}}$ into map space, with its z axis pointing upwards, $\boldsymbol{p}_{\text{rob}}^{\mathcal{M}}$ is the coordinate vector of the robot in map space, and Δ is the scaling factor used to relate heightmap cells to cm.

A.4 Map to Local

Due to the circular nature of the map buffer, translating from map to local space is more challenging. First an intermediate position, $\boldsymbol{x}_{\text{temp}}$, is defined as a helper,

$$\boldsymbol{x}_{\text{temp}} = \boldsymbol{p}_{\text{rob}}^{\mathcal{M}} \frac{1}{\Delta} \tag{A.4}$$

where $\boldsymbol{x}_{\text{temp}}$ is the map coordinates with the inverse map scaling applied. Now $\boldsymbol{x}_{\text{temp}}$ must be checked to see if either of its coordinates exceed half the map extents, if so, then it must be adjusted such that both coordinates fall within $\frac{1}{2}E$. This is expressed in equation A.5

$$\boldsymbol{x}_{\text{temp}} \Leftarrow \begin{cases} \boldsymbol{x}_{\text{temp}} & \boldsymbol{x}_{\text{temp}} \leq \frac{1}{2}E \\ \boldsymbol{x}_{\text{temp}} - E(\operatorname{sgn} \boldsymbol{x}_{\text{temp}}) & \boldsymbol{x}_{\text{temp}} > \frac{1}{2}E \end{cases} \tag{A.5}$$

where sgn is the signum function, which return either 1 or -1 depending on the sign of the operand. **Note** that equation A.5 is applied to both coordinates in the vector $\boldsymbol{x}_{\text{temp}}$ separately, in pursuit of readability this is not explicitly stated.

Now that $\boldsymbol{x}_{\text{temp}}$ has been corrected for the circular nature of the map buffer, only the rotating into the local space remains. As show in equation A.6.

$$\begin{aligned} \boldsymbol{x}^{\mathcal{B}} &= T_{\mathcal{MB}}(\boldsymbol{x}^{\mathcal{M}}) \\ &= \boldsymbol{q}_{\text{bod}}^{-1} \cdot \boldsymbol{x}_{\text{temp}} \cdot \boldsymbol{q}_{\text{bod}} \end{aligned} \tag{A.6}$$

where $\boldsymbol{q}_{\text{bod}}$ is the robot body's quaternion.

A.5 Body to Leg

The body to leg transformation is a very simple transformation, consisting of only a rotation. This is because the body and leg frames are aligned to the same z axis. As there are six distinct leg coordinate frames the leg frame is annotated as, $\mathcal{L}_i$, with the subscript i indicating the frame's associated leg number. Equation A.7 shows this transformation.

$$\begin{aligned} \boldsymbol{x}^{\mathcal{L}_i} &= T_{\mathcal{BL}_i}(\boldsymbol{x}^{\mathcal{B}}) \\ &= \boldsymbol{Q}_i^{\mathcal{B}} \cdot \boldsymbol{x}^{\mathcal{B}} \cdot \boldsymbol{Q}_i^{\mathcal{B}^{-1}} - \boldsymbol{R}_i \end{aligned} \tag{A.7}$$

where $\boldsymbol{Q}_i^{\mathcal{B}}$ is the rotation quaternion of leg i.

Note that this rotation need not be represented as a quaternion, as it is a simple single axis rotation, but as it is entirety possible to have a leg axis system with a multi axis rotation, and quaternions are used throughout this paper, it was decided to use quaternions.

Appendix B

ROS Messages

This Appendix provides a detailed account of the ROS messages used in the ROS nodes described in chapter 7. Table B.1 and B.2 describe the ROS publishers and subscribers present on the base station.

Base Station Publishers			
Name	**Data Type**	**Description**	**Frequency**
command_data	HexapodCommands	Various robot command parameters.	On change
mode	Int32	Specifies the oporationg mode of the robot.	On change.

Table B.1: Base station publishers

Base Station Subscribers		
Name	**Data Type**	**Description**
rgb_data	Image	The processed color image from the robot.
d_data	Image	The processed color depth from the robot.
hmap_data	Image	The heightmap generated on the robot.
scoremap_data	Image	The scoremap generated on the robot.
LOGDATA	String	General logs from the robot.

Table B.2: Base station subscribers

Table B.3 and B.4 describe the ROS publishers and subscribers present on the Jetson Nano.

Jetson Publishers			
Name	**Data Type**	**Description**	**Frequency**
effector_targets	EffectorTargets	Data indicating which feet to move where, and what type of interpolation to use.	On change
rgb_data	Image	The processed color image from the RGB-D camera.	15Hz.
d_data	Image	The processed depth image from the RGB-D camera.	15Hz.
hmap_data	Image	The heightmap generated on the robot.	15Hz.
scoremap_data	Image	The scoremap generated on the robot.	15Hz.
position	Vector3	The localised position of the robot	15Hz
rotation	Quat	The localised rotation of the robot	15Hz

Table B.3: Jetson publishers

Jetson Subscribers		
Name	**Data Type**	**Description**
command_data	HexapodCommands	Operator commands.
color/image_raw	Image	Color image from the camera.
aligned_depth_to_color/image_raw	Image	Depth image from the camera.

Table B.4: Jetson subscribers

Table B.5 and B.6 describe the puvblishers and subscribers present on the Teensy MCU.

Teensy Publishers			
Name	**Data Type**	**Description**	**Frequency**
LOGDATA	String	General logs.	10Hz
effector_current_position	Eigen::Vector3d	Current feet positions.	10Hz

Table B.5: Teensy publishers

Teensy subscribers		
Name	**Data Type**	**Description**
command_data	HexapodCommands	The color image from the RGB-D camera.
effector_targets	EffectorTargets	Data indicating which feet to move where, and what type of interpolation to use.
mode	Int32	Receives mode data from the base station

Table B.6: Teensy subscribers

Lastly, the custom ROS data types used are defined in table B.7.

Name	**Type Definition**	**Description**
Vector3	float[3] data.	A vector in 3D space.
EffectorTargets	Vector3[6] targets bool[6] swinging.	Data describing the targets of the robot's feet and which feet are swinging.
HexapodCommands	float32[2] walk_dir float32 speed float32 height	Data packet containing various command parameters for the robot.

Table B.7: ROS data type descriptions

List of references

Campos, C., Elvira, R., Rodríguez, J.J.G., Montiel, J.M. and Tardós, J.D. (2021). Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam. *IEEE Transactions on Robotics*, vol. 37, no. 6, pp. 1874–1890.

Chen, W., Ren, G., Zhang, J. and Wang, J. (2012 09). Smooth transition between different gaits of a hexapod robot via a central pattern generators algorithm. *Journal of Intelligent & Robotic Systems*, vol. 67.

Coelho, J., Ribeiro, F., Dias, B., Lopes, G. and Flores, P. (2021). Trends in the control of hexapod robots: a survey. *Robotics*, vol. 10, no. 3, p. 100.

Collins, J., Chand, S., Vanderkop, A. and Howard, D. (2021). A review of physics simulators for robotic applications. *IEEE Access*, vol. 9, pp. 51416–51431.

Darbha, N.H. (2017). An optimization strategy for hexapod gait transition. Available at: `https://api.semanticscholar.org/CorpusID:196004349`

Erasmus, S. *et al.* (2023). Guidance, control, and motion planning for a hexapod robot moving over uneven terrain.

Erez, T., Tassa, Y. and Todorov, E. (2015). Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4397–4404.

Guo, D., Fu, L. and Wang, L. (2019). Robots solving the urgent problems by themselves: A review. In: *2019 IEEE International Conference on Consumer Electronics - Taiwan (ICCE-TW)*, pp. 1–2.

Gutierrez-Galan, D., Dominguez-Morales, J.P., Perez-Peña, F., Jimenez-Fernandez, A. and Linares-Barranco, A. (2020). Neuropod: A real-time neuromorphic spiking cpg applied to robotics. *Neurocomputing*, vol. 381, pp. 10–19. ISSN 0925-2312. Available at: `https://www.sciencedirect.com/science/article/pii/S0925231219315644`

Hartley, R. and Zisserman, A. (2003). *Multiple view geometry in computer vision.* Cambridge university press.

Homberger, T., Bjelonic, M., Kottege, N. and Borges, P.V. (2017). Terrain-dependant control of hexapod robots using vision. In: *2016 International Symposium on Experimental Robotics*, pp. 92–102. Springer.

Horn, R. and Johnson, C. (2012). *Matrix Analysis*. Cambridge University Press. ISBN 9780521839402.
Available at: `https://books.google.co.za/books?id=NAffwAEACAAJ`

Irawan, A. and Nonami, K. (2012). Force threshold-based omni-directional movement for hexapod robot walking on uneven terrain. In: *2012 Fourth International Conference on Computational Intelligence, Modelling and Simulation*, pp. 127–132.

Isvara, Y., Rachmatullah, S., Mutijarsa, K., Prabakti, D.E. and Pragitatama, W. (2014). Terrain adaptation gait algorithm in a hexapod walking robot. In: *2014 13th International Conference on Control Automation Robotics and Vision (ICARCV)*, pp. 1735–1739.

Khudher, D., Powell, R. and Abbod, M. (2017). Operational space control in hexapod robot for humanitarian demining applications. In: *2017 3rd International Conference on Control, Automation & Robotics (ICCAR)*, pp. 212–216.

Liu, C., Li, Z., Zhang, C., Yan, Y. and Zhang, R. (2019). Gait planning and control for a hexapod robot on uneven terrain based on markov decision process. In: *2019 14th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, pp. 583–586.

Macario Barros, A., Michel, M., Moline, Y., Corre, G. and Carrel, F. (2022). A comprehensive survey of visual slam algorithms. *Robotics*, vol. 11, no. 1, p. 24.

Mastalli, C., Havoutis, I., Focchi, M., Caldwell, D. and Semini, C. (2020 06). Motion planning for quadrupedal locomotion: Coupled planning, terrain mapping and whole-body control. *IEEE Transactions on Robotics*, vol. 0.

Mur-Artal, R. and Tardós, J.D. (2017). Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE transactions on robotics*, vol. 33, no. 5, pp. 1255–1262.

NVIDIA (2023 02). *GPU performance background user's guide*.

Prágr, M., Čížek, P., Bayer, J. and Faigl, J. (2019 06). Online incremental learning of the terrain traversal cost in autonomous exploration.

Sobel, I. (2014 02). An isotropic 3x3 image gradient operator. *Presentation at Stanford A.I. Project 1968*.

Verma, S., Nair, H.S., Agarwal, G., Dhar, J. and Shukla, A. (2019). Deep reinforcement learning for single-shot diagnosis and adaptation in damaged robots. *Proceedings of the 7th ACM IKDD CoDS and 25th COMAD*.
Available at: `https://api.semanticscholar.org/CorpusID:203642140`

Xu, P., Li, Z., Yang, H., Wang, Z., Gao, H., Zhou, R., Su, Y., Deng, Z. and Huang, Y. (2023 02). Learning physical characteristics like animals for legged robots. *National Science Review*, vol. 10.

Yu, H., Guo, W., Deng, J., Li, M. and Cai, H. (2013). A cpg-based locomotion control architecture for hexapod robot. In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5615–5621.

Zha, F., Chen, C., Guo, W., Zheng, P. and Shi, J. (2019 03). A free gait controller designed for a heavy load hexapod robot. *Advances in Mechanical Engineering*, vol. 11, p. 168781401983836.