



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: 2023/02/10

Copyright © 2024 Stellenbosch University
All rights reserved.

Abstract

Foot Placement Planning of a Hexapod Robot Moving Over Uneven Terrain

A.P. Lotriet

*Department of Electrical and Electronic Engineering,
Stellenbosch University,
Private Bag X1, Matieland 7602, South Africa.*

Thesis: MEng (EE)

December 2024

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 an 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

Voet Plaasing Beplanning van 'n Sespotige Robot oor Onegalige asdasd

A.P. Lotriet

Tesis: MIng (Meg)

Desember 2024

Om 'n tand implement te vibreer is 'n effektiewe manier om die trekkrag, wat benodig word om dit deur die grond te trek, te verminder. Die graad van krag vermindering is afhanklik van die kombinasie van werks parameters en die grond toestand. Dus is dit nodig om die vibrerende implement te optimeer vir verskillende omstandighede.

Numeriese modulering is meer buigsaam en goedkoper as eksperimentele opstellings en analitiese modelle. Die Diskrete Element Metode (DEM) was spesifiek vir korrelrige materiaal, soos grond, ontwikkel en kan gebruik word vir die modellering van 'n vibrerende implement vir die ontwerp en optimering daarvan. Die doel was dus om die vermoë van DEM om 'n vibrerende skeurploeg the modelleer, te evalueer, en om die oorsaak van die krag verminderung te ondersoek.

Die DEM model was geïvalueer teen data ...

Acknowledgments

Dedication

Table of contents

List of figures	ix
List of tables	xi
List of symbols	xii
1 Introduction	1
1.1 Background	1
1.2 Research Goal	1
1.3 Methodology	2
1.4 Scope and Limitations	3
1.5 Thesis Outline	4
2 Literature review	5
2.1 Hexapod history	5
2.2 Control	6
2.2.1 Traditional	6
2.2.2 Bio Inspired	7
2.2.3 Reinforcement Learning	7
2.3 Sensing Methods	7
2.4 Simulation Environment	8
2.5 Uneven Terrain	8
2.6 Research Decisions	9
3 System Overview	11
3.1 Hardware	11
3.2 Software	12
3.3 Simulation	14
4 Mapping	16
4.1 Overview	16
4.2 Projecting the Depth Image to Map Space	17
4.3 Updating the Heightmap	20
4.4 GPU Compute Pipeline	21

4.4.1	A note on Graphical Processing Unit (GPU) architecture	22
4.5	Simultaneous Localisation and (Sparse) Mapping	23
4.6	Hardware Mapping Test	24
5	Baseline Motion System	29
5.1	Overview	29
5.2	Kinematics	30
5.2.1	Coordinate Frames	30
5.2.2	Inverse Kinematics (IK)	32
5.2.3	Forward Kinematics (FK)	32
5.2.4	Angular Rate	33
5.3	Walking Gait	34
5.3.1	Stride Reference Frame	35
5.3.2	State Machine	37
5.3.3	Choosing The Supporting And Swinging Legs	38
5.3.4	Choosing nominal foot positions	39
5.4	Foot Motion	41
5.4.1	Existing System	41
5.4.2	Improved System	43
6	Foot Placement Optimisation	45
6.1	Overview	45
6.2	Scoring	45
6.2.1	Slope Score	45
6.2.2	Edge Proximity Score	47
6.2.3	Constraints	48
6.3	Score Search Algorithm	48
6.4	Floor Height Adjustment	49
6.5	Scoring Test	50
6.5.1	Hardware Scoring Test	53
7	Hardware Implementation	55
7.1	Overview	55
7.2	ROS Architecture	56
7.2.1	Base Station	56
7.2.2	On Board	57
8	Final Walking Test	59
8.1	Flat Plane	59
8.2	Staircase	61
8.3	Organic	63
9	Conclusions	65

A Reference Frame Transforms	66
A.1 Camera to Map	66
A.2 World to Map	67
A.3 Local to Map	67
A.4 Map to Local	67
A.5 Body to Leg	68
B ROS Messages	69
List of references	73

List of figures

2.1	A Flesh-fly	5
2.2	A circular hexapod	5
2.3	Trends of hexapod control schemes (Coelho <i>et al.</i> , 2021)	6
2.4	Exploratory terrain traversability classification from Prágr <i>et al.</i> (2019)	8
2.5	Quadruped terrain scoring and navigation from Mastalli <i>et al.</i> (2020)	9
3.1	Physical Hexapod	11
3.2	Basic system operation	12
3.3	Advanced system operation	12
3.4	Physical system diagram	13
3.5	The MuJoCo simulation environment	15
4.1	Camera Projection	17
4.2	2D Circular Buffer that stores Heightmap	20
4.3	Compute pipeline.	21
4.4	Top-down diagram of the test setup.	24
4.5	Heightmap generated at the start of the walking path (Top). Color frame at the start of walking path (Bottom).	25
4.6	Heightmap generated at the middle of the walking path (Top). Color frame at the middle of walking path (Bottom).	26
4.7	Heightmap generated at the end of the walking path (Top). Color frame at the end of walking path (Bottom).	27
4.8	Position output from ORB-SLAM3	28
5.1	Motion System Overview	29
5.2	World, body and leg coordinate frames.	30
5.3	Leg coordinate frame with kinematic variables.	31
5.4	Three hexapod gait patterns.	34
5.5	Hexapod stride relative to body coordinates.	35
5.6	Hexapod stride relative to world coordinates.	36
5.7	Gait State Machine	37

5.8	Leg sextants, with sextant 1 being active.	38
5.9	Supporting target choosing diagram	39
5.10	Swinging target choosing diagram	40
5.11	Variables for calculating foor arc.	41
5.12	Existing arc recomputation problem	42
5.13	End effector movement path	43
5.14	Sigmoid Like	44
6.1	Slope Score	46
6.2	Proximity Score Diagrammatic Representation	47
6.3	Radial search, shown for a 5 cell search diameter.	48
6.4	Floor height diagram.	49
6.5	3D Model of Terrain and its heightmap.	50
6.6	Slope score and edge proximity score tests.	50
6.7	Full walkability score.	51
6.8	Walkability score optimisation test.	52
6.9	Heightmap generated from hardware and the corresponding score map.	53
6.10	Test points on hardware score map.	54
7.1	ROS nodes and communication.	56
8.1	Flat plane walk test	59
8.2	Body height (top). Body tilt (center). Position of one foot (bottom)	60
8.3	Stairs walk test	61
8.4	Body height (top). Body tilt (center). Position of one foot (bottom)	62
8.5	Organic walk test.	63
8.6	Body height (top). Body tilt (center). Position of one foot (bottom)	64

List of tables

5.1	State Definitions	37
B.1	Base station publishers	69
B.2	Base station subscribers	69
B.3	Jetson publishers	70
B.4	Jetson subscribers	71
B.5	Teensy publishers	71
B.6	Teensy subscribers	71
B.7	Robot Operating System (ROS) data type descriptions	72

List of symbols

Constants

$$L_0 \quad 300 \text{ mm}$$

Variables

Re_D	Reynolds number (diameter)	[]
x	Coordinate	[m]
\ddot{x}	Acceleration	[m/s ²]
θ	Rotation angle	[rad]
τ	Moment	[N·m]

Vectors and Tensors

Subscripts

a	Adiabatic
a	Coordinate

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

¹See appendix ?? 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
$\llbracket a_0, \dots, a_n \rrbracket$	Defines a integer sequence from a_0 to a_n , with a increment of 1.
$\langle \mathbf{a}, \mathbf{b} \rangle_F$	Takes the Frobenius inner product of matrix \mathbf{a} and \mathbf{b} .

Abbreviations

IK	Inverse Kinematics
FK	Forward Kinematics
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 Learning
ANN	Artificial Neural Network
GPS	Global Positioning System
MCU	Microcontroller Unit
GPU	Graphical Processing Unit
CPU	Central Processing Unit
LPS	local positioning system

²See 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 RGB-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, but instead of storing previous images, an onboard heightmap would be generated of the local environment. This would allow for more advanced methods of anchor point selection.

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 Multi-Joint dynamics with Contact (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 an 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, if the foot will not be placed in close proximity to a steep edge, and if the height of the terrain at the intended foot placement position is known. The last requirement 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 un-navigable terrain, 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 measurement 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 sensing methods. 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 lift some of its legs while bracing with others, the number of swinging to bracing 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 are bracing 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 brace and swing depending on the current walking environment.

2.2 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 Reinforcement Learning (RL). These three schemes are discussed below. Control trends can be seen in Figure 2.3

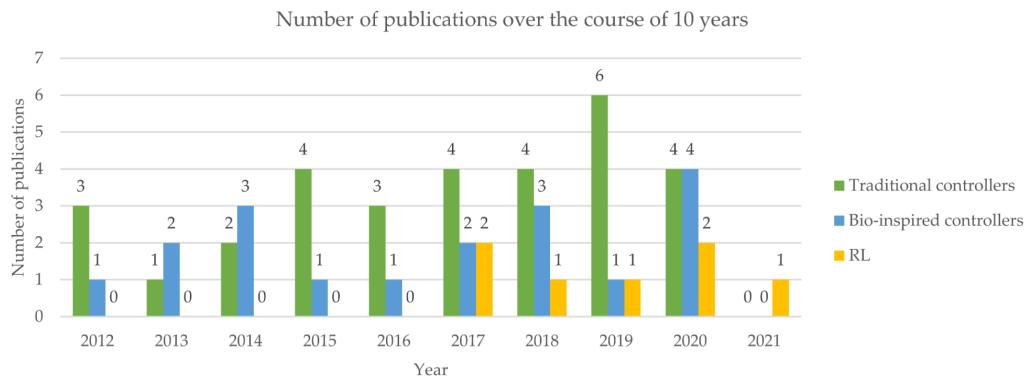


Figure 2.3: Trends of hexapod control schemes (Coelho *et al.*, 2021)

2.2.1 Traditional

Traditional controllers rely on an exact mathematical model of the robot and IK to calculate angular commands for all leg joints. This method of control is purely kinematic does not take into account external forces applied to the robot, thus it does not inherently adjust to the environment.

Instead of a purley kinematic model, a dynamic model can also be used. Using a dynamic model the forces acting on the robots legs are taken into account 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.

It should be noted that it is possible for a kinematic model controller to also adjust to external disturbances, but this is not intrinsic to the control model and requires additional control logic.

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. This is implemented through the use of a Artificial Neural Network (ANN) 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.

2.2.3 Reinforcement Learng

RL controllers are created through using trial and error to construct a neural net 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 hight level of autonomy, as no prior direction is required. RL controllers are though notoriously difficult to train properly, especially when the amount of sensors and control outputs grow large, increasing the feature space. And event he most well trained RL agent still has the possibility to exhibit inexplicable behaviour.

2.3 Sensing Methods

No matter the control scheme used, to know where to place its feet the robot requires sensor(s) to sense its environment in some way, this could be achieved through simple sensors such as servo torque or touch. More advanced methods such as vision or Light Detection and Ranging (LiDAR) are also used. Homberger *et al.* (2017) uses stereoscopic vision to adjust end effector height and to classify surface materials.

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), Global Positioning System (GPS) or, through the use of a SLAM system, internal sensors such as vision could be used.

2.4 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).

2.5 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.

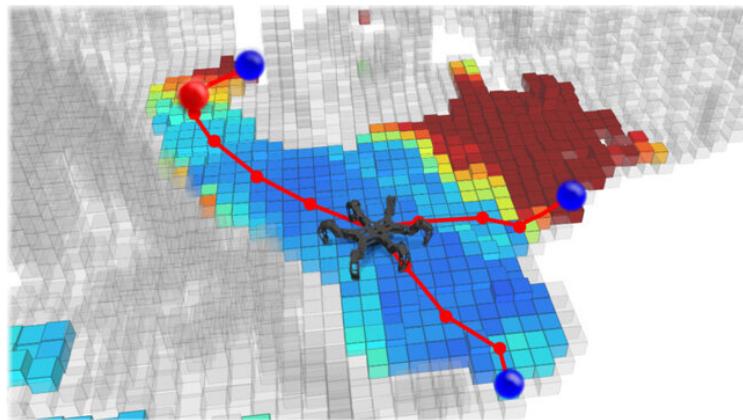


Figure 2.4: Exploratory terrain traversability classification from Prágr *et al.* (2019)

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

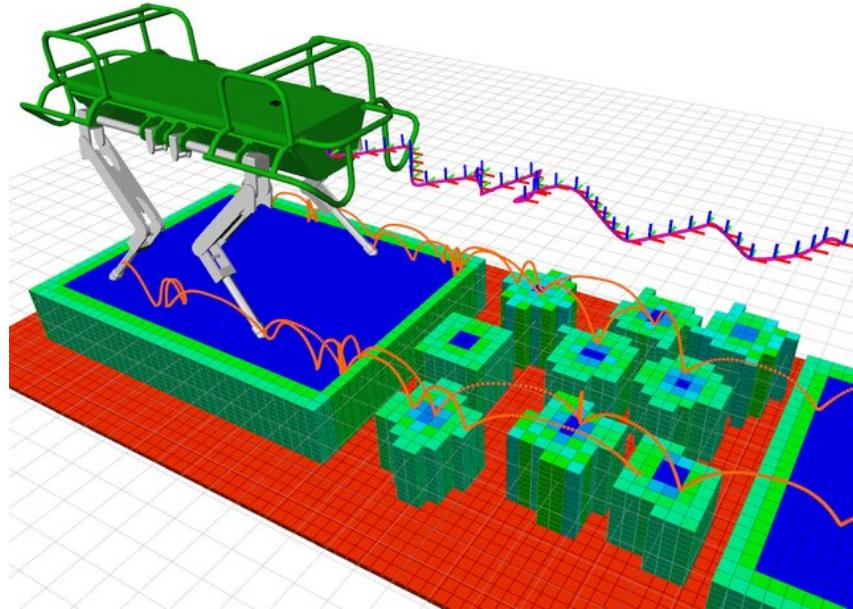


Figure 2.5: Quadruped terrain scoring and navigation from Mastalli *et al.* (2020)

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 Research Decisions

This research paper focuses on using vision based mapping, terrain classification and the optimisation of foot end positions, thus it was decided that traditional kinematic control is used as it is the simplest and most predictable method of control. Thus allowing the focus to remain on terrain classification and foot end position optimisation.

The camera that is 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, a optimisation-based, sparse map SLAM system was chosen to be used. ORB-SLAM 3 maintains a sparse map, an atlas, of both active and dormant features. This atlas is used to localise in the sparse map (Macario Barros *et al.*, 2022). The implementation of a dense map to be used for end effector placement is discussed in chapter 4. **Compare this more to others?**

Finally a simulation environment was chosen. MuJoCo has exceptional contact physics simulation and the only relevant downside is the lack of native ROS integration and the lack of a comprehensive GUI, which seeing as MuJoCo has good python bindings, could be seen as a advantage. Considering this, MuJoCo was chosen as the simulator.

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). 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 inertial measuring unit (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.

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. Figure 3.1 shows the hexapod in its current state.



Figure 3.1: Physical Hexapod

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.

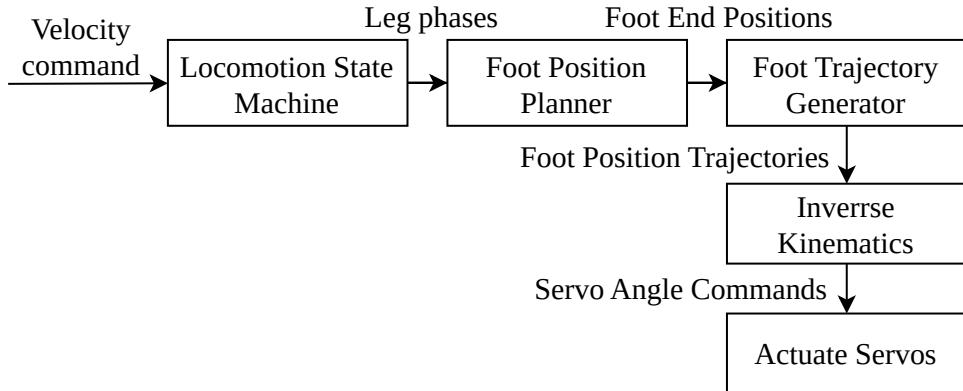


Figure 3.2: Basic system operation

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.

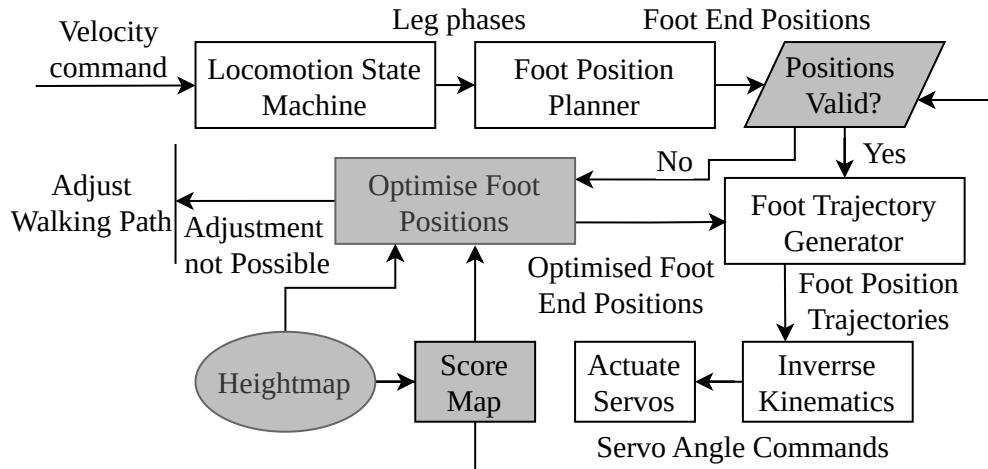


Figure 3.3: Advanced system operation

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.

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.

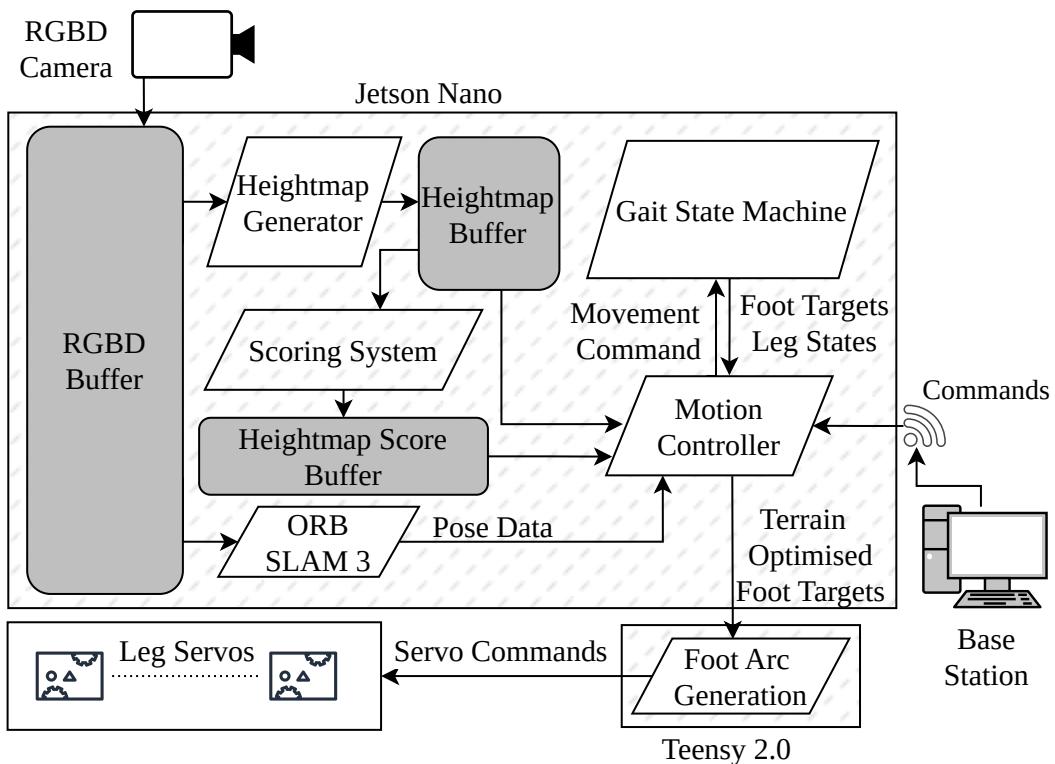


Figure 3.4: Physical system diagram

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 place 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 2.4, 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

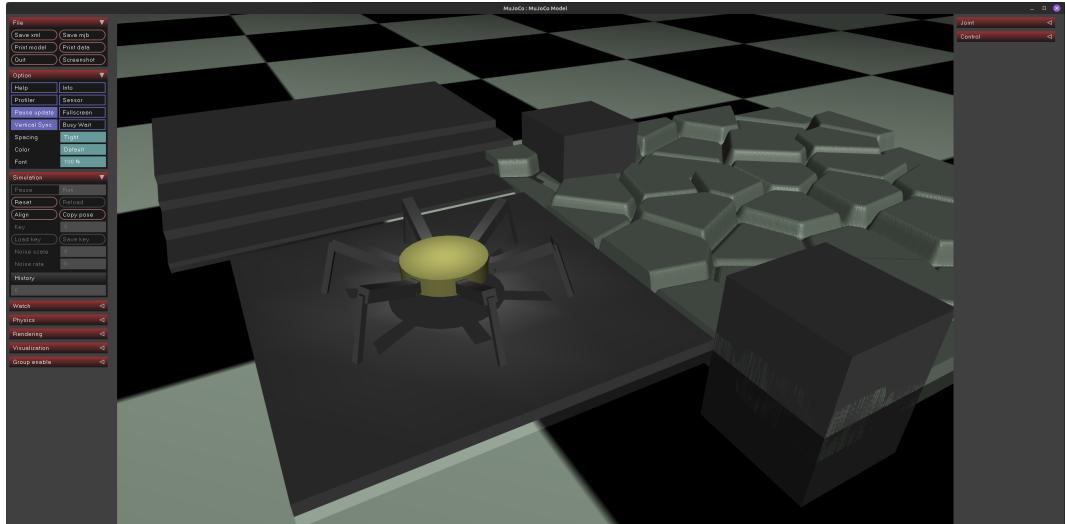


Figure 3.5: The MuJoCo simulation environment

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 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.

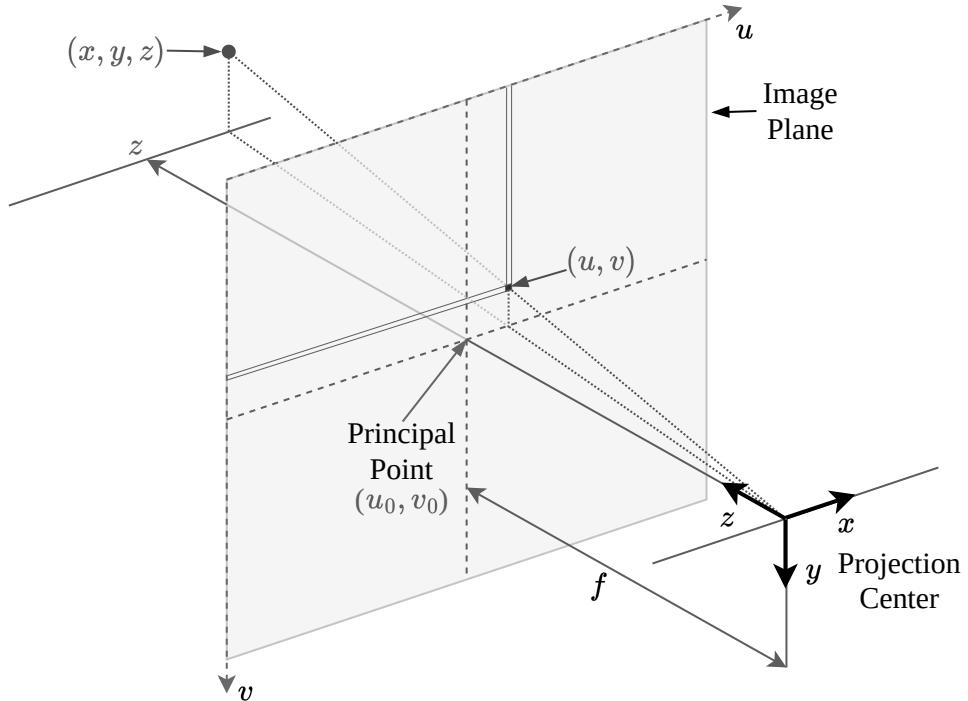


Figure 4.1: Camera Projection

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

$$\mathbf{P} = \mathbf{K} [\mathbf{R} \ \mathbf{T}] \quad (4.1)$$

where \mathbf{K} is the intrinsic matrix and $[\mathbf{R} \ \mathbf{T}]$ 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:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{P} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (4.2)$$

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,

$$\mathbf{K} = \begin{bmatrix} \alpha_x & \gamma & u_0 \\ 0 & \alpha_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \quad (4.3)$$

with the focal length 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, 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

$$[\mathbf{R} \ \mathbf{T}] = \begin{bmatrix} \mathbf{R}_{3 \times 3} & \mathbf{T}_{3 \times 1} \\ \mathbf{0}_{1 \times 3} & 1 \end{bmatrix} \quad (4.4)$$

where \mathbf{R} characterises the camera's rotation relative to the world frame and \mathbf{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

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

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:

$$z = I_{u,v} \quad (4.6)$$

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

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

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

$$\mathbf{p}^C = \begin{bmatrix} x & y & z \end{bmatrix}^T \quad (4.9)$$

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

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

where \mathbf{p}^M is the position in the world frame, \mathbf{q}_{CW} is the quaternion rotation of the world frame relative to the camera frame, and \mathbf{p}_{C0}^M 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.

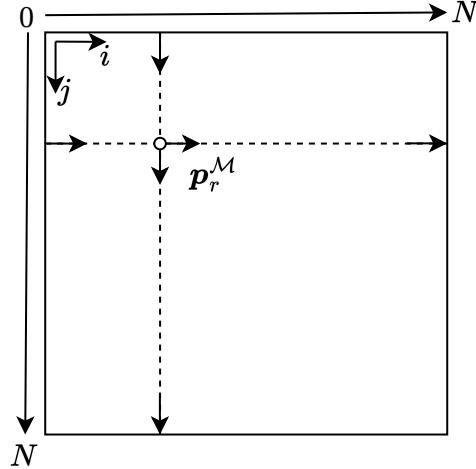


Figure 4.2: 2D Circular Buffer that stores Heightmap

The heightmap buffer, \mathbf{M} , is of size $N \times N$ and is indexed by $[i, j]$. \mathbf{p}_r^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:

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

where $[x^W, y^W]$ are the horizontal coordinates in the world frame, Δ is the horizontal resolution 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.

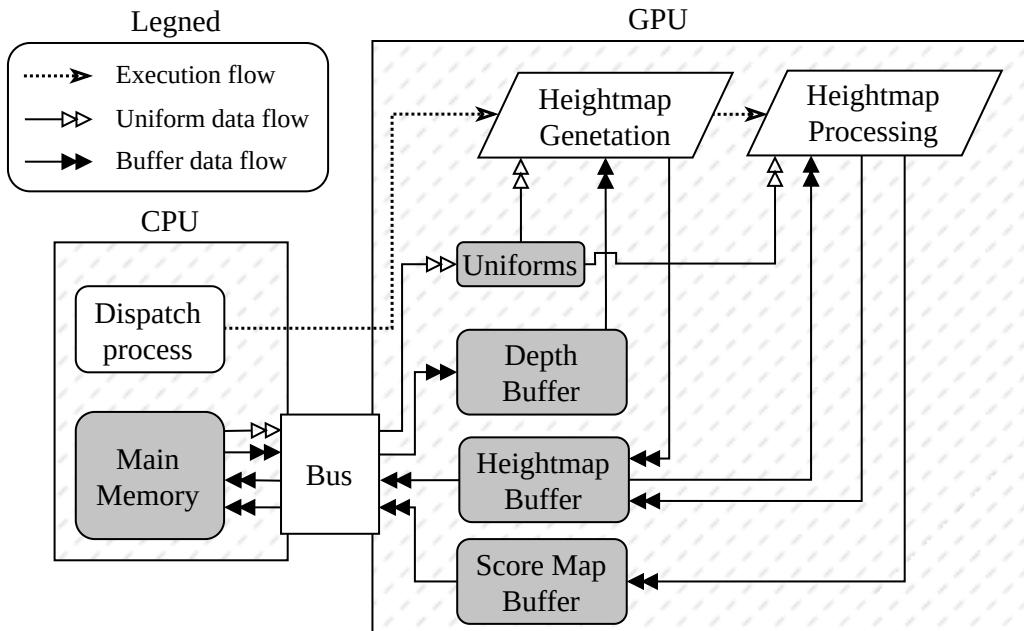


Figure 4.3: Compute pipeline.

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, $\mathbf{N}_l = [X_l, Y_l, Z_l]$. Next when the Central Processing Unit (CPU) dispatches a compute task, the workgroup count, is fed as parameter, $\mathbf{n}_w = [x_w, y_w, z_w]$. The GPU then initialises \mathbf{n}_w workgroups, and each workgroup initialises \mathbf{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 \mathbf{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 $\mathbf{N}_l = [32, 32, 0]$ 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 $\mathbf{N}_l = [32, 32, 32]$ 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 \mathbf{n}_w with each execution cycle, \mathbf{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-SLAM3 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 Hardware Mapping Test

The mapping system described above was tested on the physical robot, this mapping test aims to show how the physical hexapod maps various objects while in motion. The test consists of the robot walking past various obstacles, while simultaneously generating a heightmap.

Figure 4.4 shows a top-down diagram of the placed obstacles used in this test, with the path the robot walks.

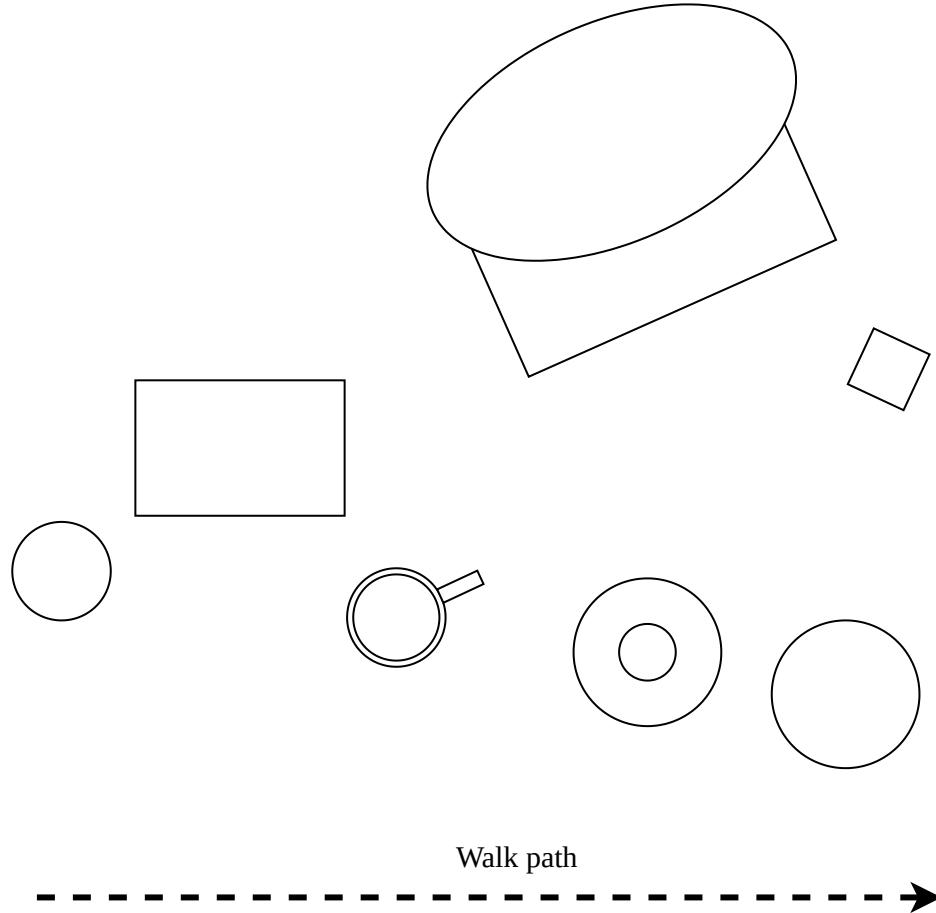


Figure 4.4: Top-down diagram of the test setup.

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 4.5, 4.6 and 4.7 show the heightmap, and its corresponding color frame, at the start middle and end of the robot's walking path.

Figure 4.5 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 very clear by the shadows cast by both the robot's leg (on the left) and by the obstacles (the pig and box). In this static case very little noise is also present.

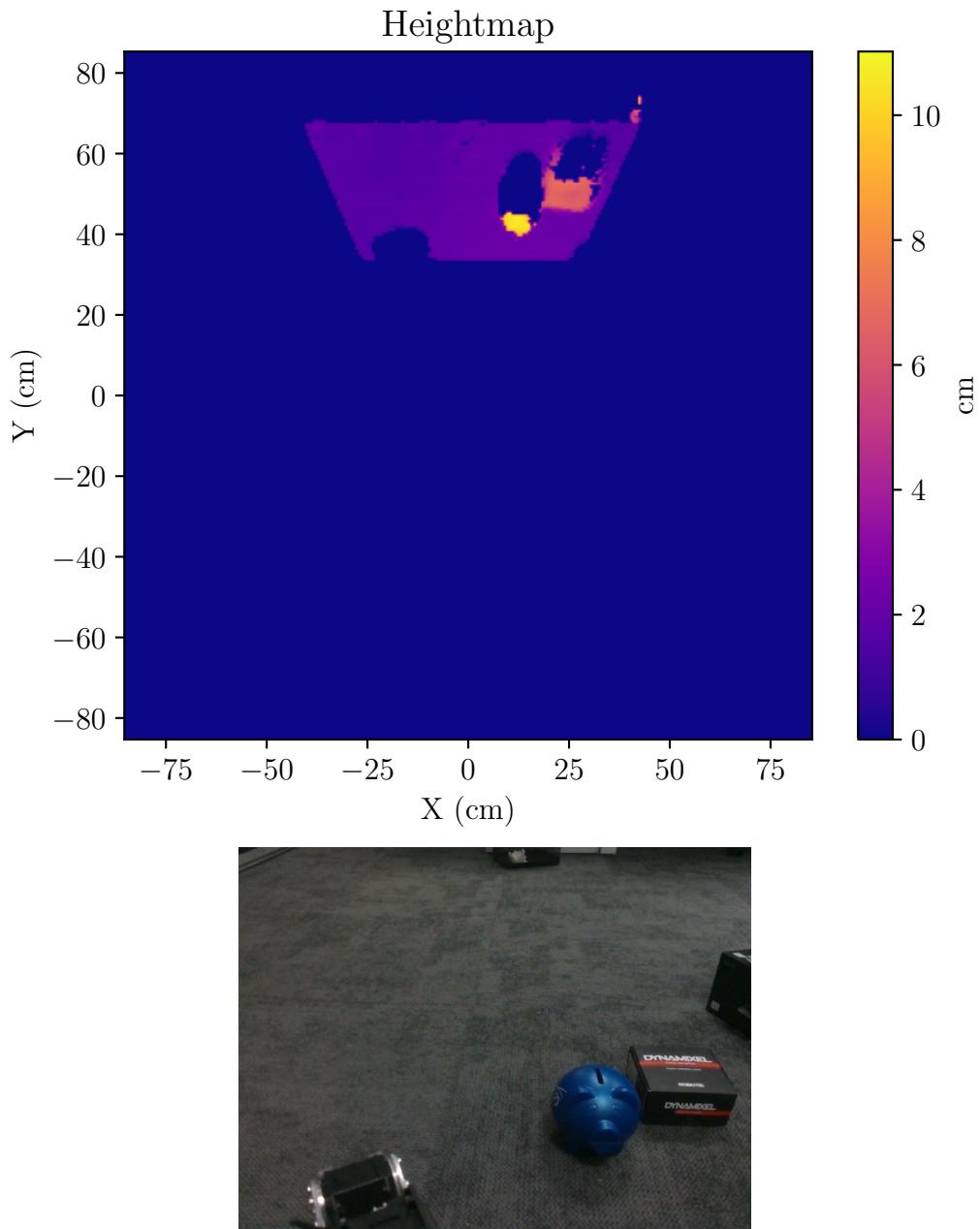


Figure 4.5: Heightmap generated at the start of the walking path (Top). Color frame at the start of walking path (Bottom).

Figure 4.6 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.

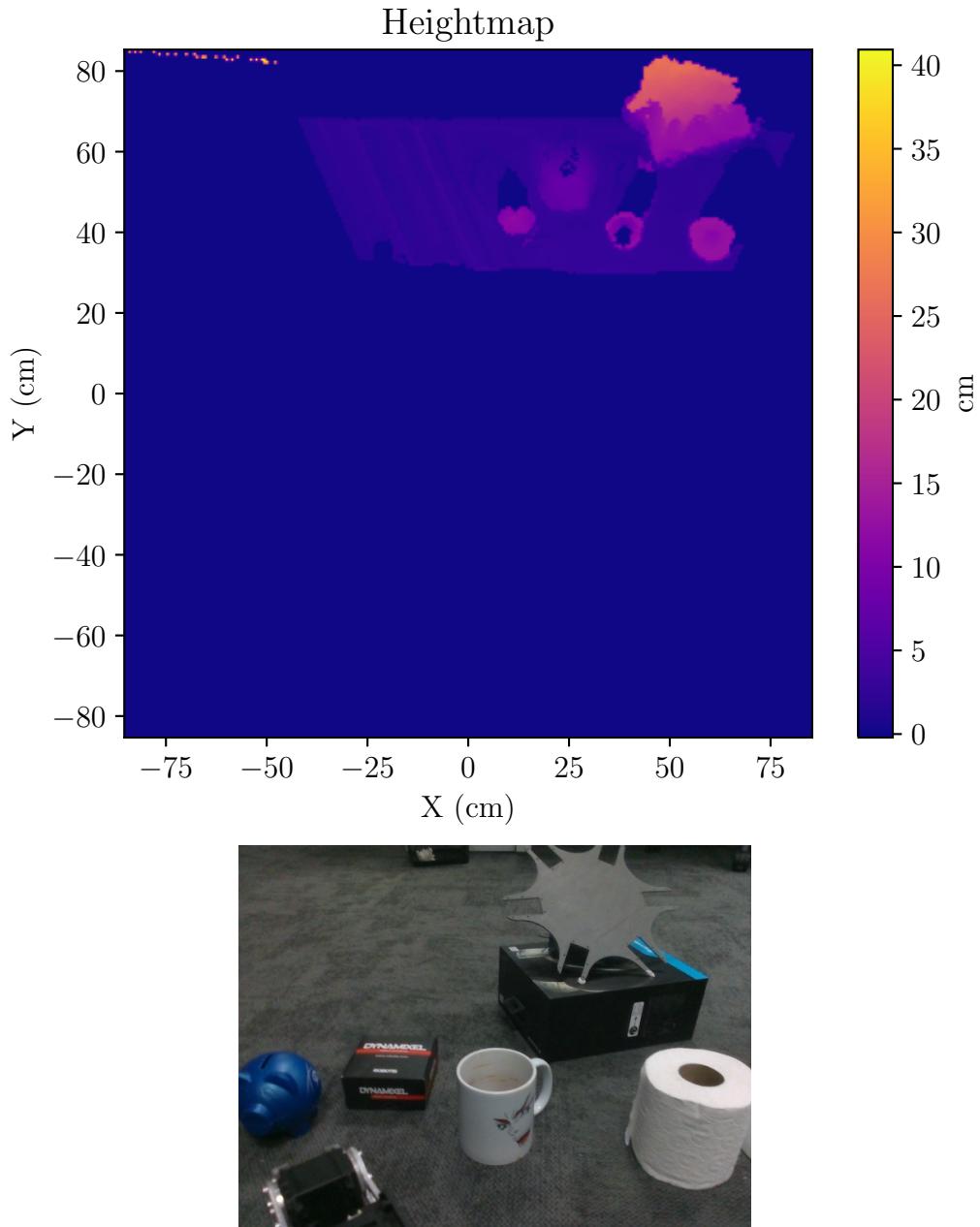


Figure 4.6: Heightmap generated at the middle of the walking path (Top). Color frame at the middle of walking path (Bottom).

In this heightmap 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.

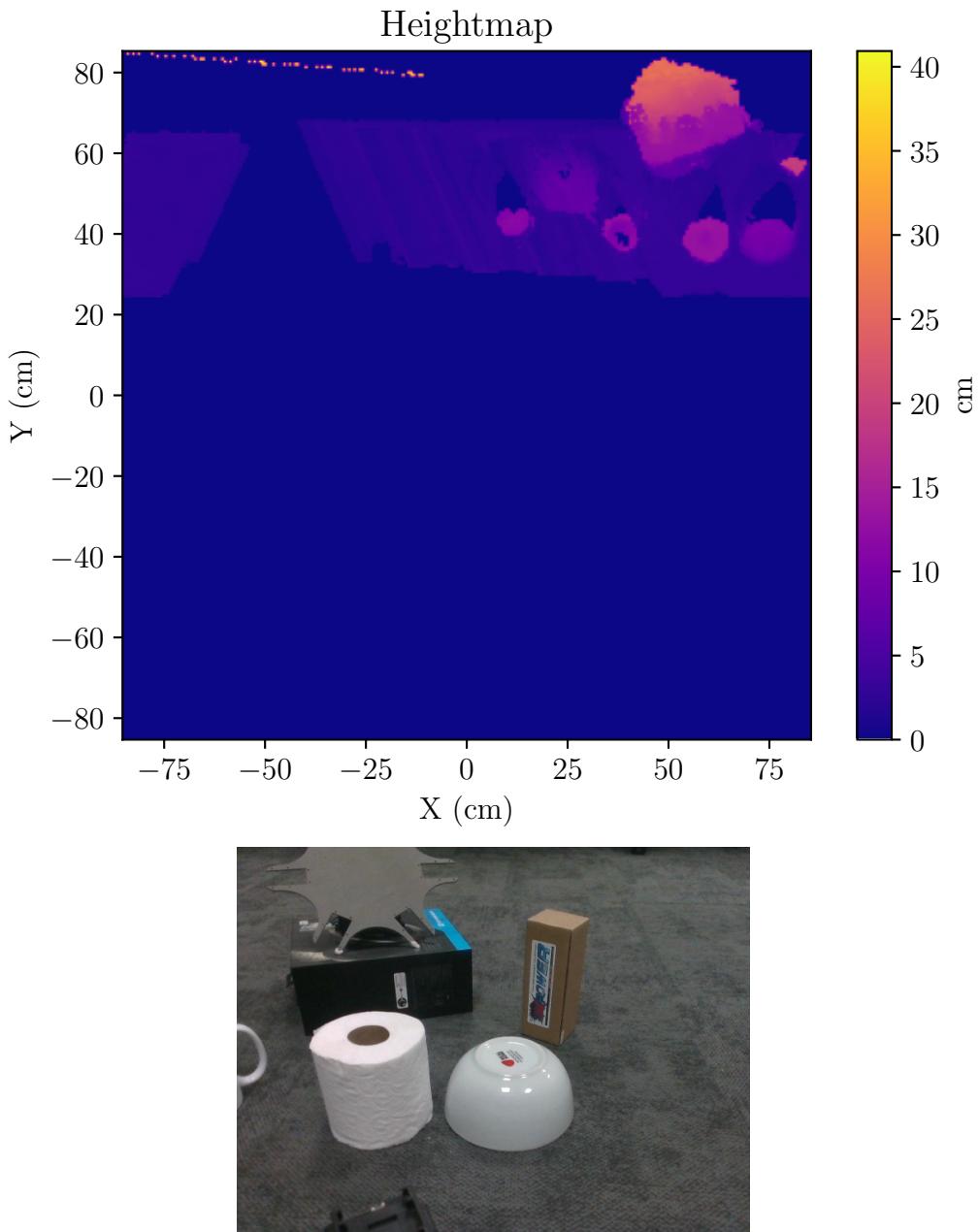


Figure 4.7: Heightmap generated at the end of the walking path (Top). Color frame at the end of walking path (Bottom).

Finally, the heightmap generated at the end of the walking path can be seen in figure 4.7. 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 4.8 shows the robot's estimated position, with each dot being a estimation. The blue dot is the start of the test and the red dot the end of the test.

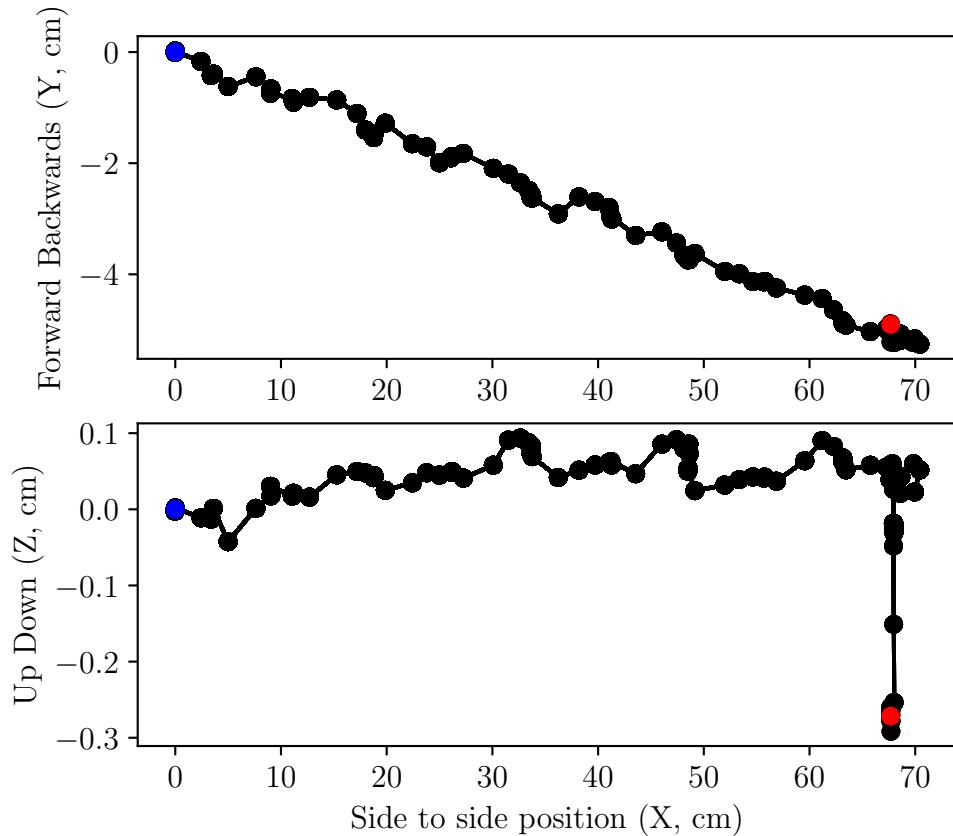


Figure 4.8: Position output from ORB-SLAM3

As can be seen, the frequency of pose estimation causes there to sometimes be large jumps between estimations. 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 estimations the accuracy of the heightmap would also increase.

Note that the line of motion is not aligned perfectly with the x-axis as stated in the test diagram in figure 4.4, this is due to slight user error while inputting the target velocity, but does not have any negative effects on the resultant heightmap.

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 defined, next the walking gait state machine is described, and finally the equations to defining a foots path of motion is described. An overview of these systems can be seen in figure 5.1.

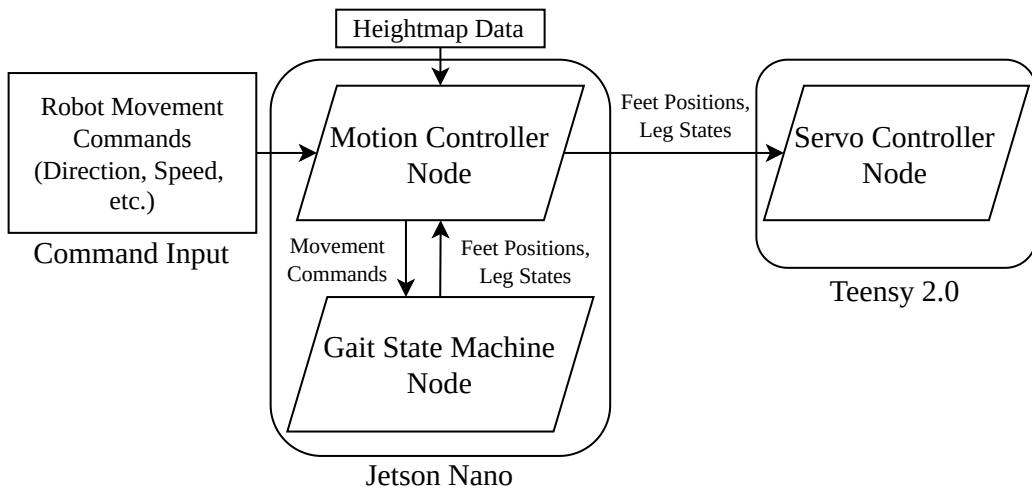


Figure 5.1: Motion System Overview

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 body height. These commands are sent from the base station to 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 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, see section 5.4, requires the current position of the feet to function. The IK and FK 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 frame, \mathcal{L}_i . All foot targets/positions are specified in the body coordinate frame, while the kinematic systems operate in the leg coordinate frame. Thus a conversion from the body to leg frames is required. Figure 5.2 show the world, body and multiple leg coordinate frames.

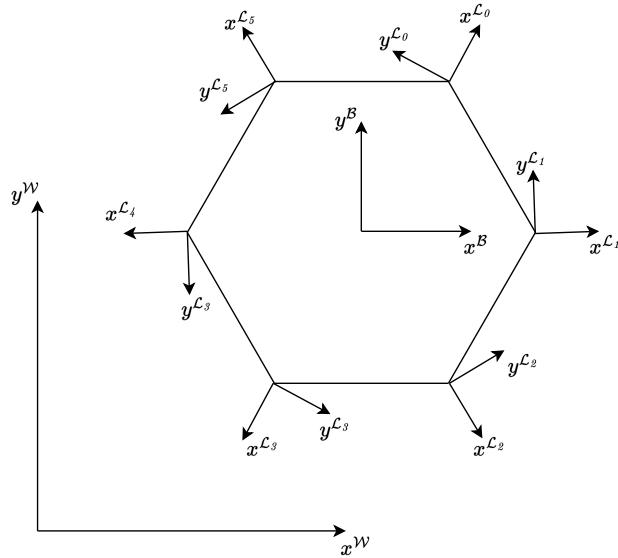


Figure 5.2: World, body and leg coordinate frames.

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

$$\begin{aligned}\mathbf{x}^{L_i} &= T_{BL_i}(\mathbf{x}^B) \\ &= \mathbf{Q}_i^B \cdot \mathbf{x}^B \cdot \mathbf{Q}_i^{B^{-1}} - \mathbf{R}_i\end{aligned}\quad (5.1)$$

where the L_i is the coordinate frame of leg i , \mathbf{Q}_i^B is the rotation of said coordinate frame and \mathbf{R}_i^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 \mathbf{Q}_i^B and \mathbf{R}_i^B , please see appendix A.

Now that positions can be transformed into the leg coordinate frame 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.

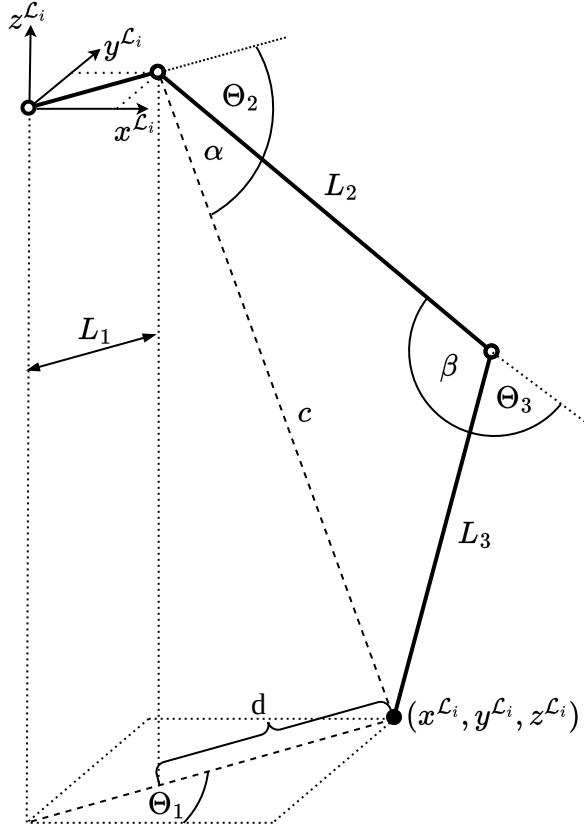


Figure 5.3: Leg coordinate frame with kinematic variables.

5.2.2 Inverse Kinematics (IK)

The IK function calculates the leg servo angles, $\Theta = [\Theta_1, \Theta_2, \Theta_3]^T$ required to move the foot to the given target position vector, $\mathbf{t} = [x_t, y_t, z_t]^T$. Equation 5.2 describes the IK function.

$$\Theta(x_t, y_t, z_t) = \begin{bmatrix} \arctan\left(\frac{x_t}{y_t}\right) \\ \frac{\pi}{4} - \alpha - \arctan\left(\frac{y_t}{d - L_1}\right) \\ \frac{\pi}{2} - \beta \end{bmatrix} \quad (5.2)$$

where α , β , c and d are calculate as shown in equations 5.3 to 5.6. For derivations of these variable please see ??.

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

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

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

$$d = \sqrt{x_t^2 + y_t^2} \quad (5.6)$$

5.2.3 Forward Kinematics (FK)

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

$$\mathbf{p}_f(\theta_1, \theta_2, \theta_3) = \begin{bmatrix} d \cos \theta_1 \\ d \sin \theta_1 \\ L_2 \sin \theta_2 + L_3 \sin (\theta_2 + \theta_3) \end{bmatrix} \quad (5.7)$$

where d is calculated as shown in in equation 5.8.

$$d = L_1 + L_2 \sin \theta_2 + L_3 \sin (\theta_2 + \theta_3) \quad (5.8)$$

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 define the derivative of the IK equations (5.2), i.e. the angular rate, given the target movement speeds of a foot, \dot{x} , \dot{y} and \dot{z} .

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

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

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

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

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

$$\dot{d} = \frac{x\dot{x} + y\dot{y}}{\sqrt{x^2 + y^2}} \quad (5.13)$$

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. 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, the wave ripple and tripod gait (Darbha, 2017). The dark cell represents a swinging leg and the light cell a supporting leg.

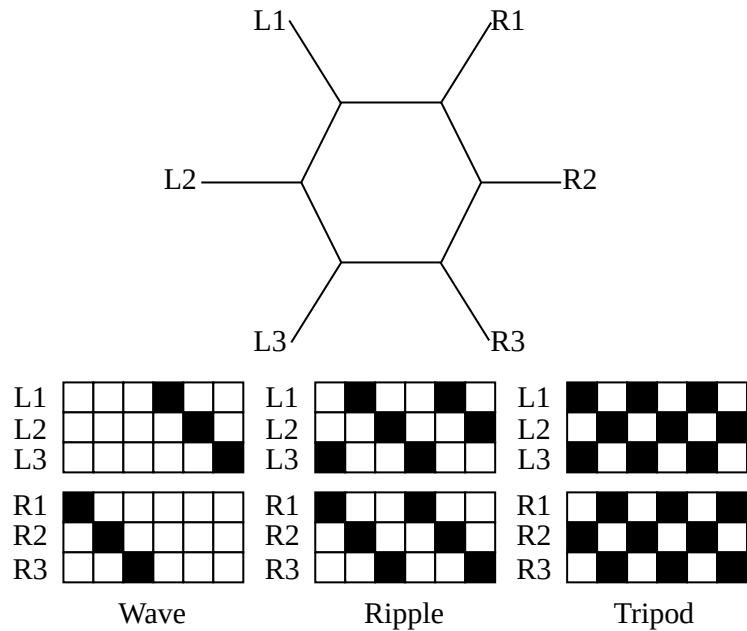


Figure 5.4: Three hexapod gait patterns.

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,

$$v = \frac{S}{D\tau} \quad (5.14)$$

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 a duty factor 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} .

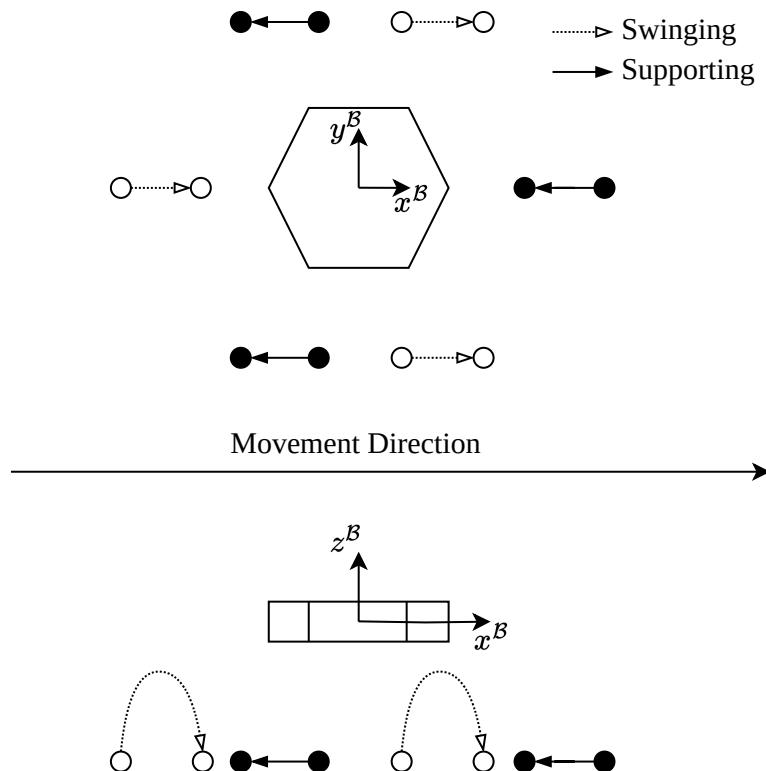


Figure 5.5: Hexapod stride relative to body coordinates.

As can be seen, the swinging legs move in the direction of movement, following a 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.

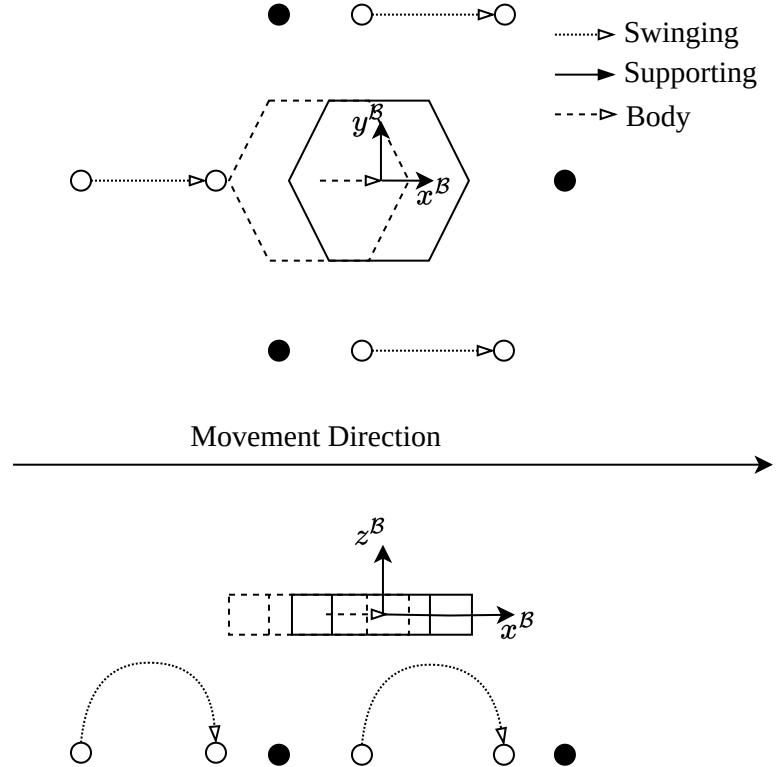


Figure 5.6: Hexapod stride relative to world coordinates.

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, 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.

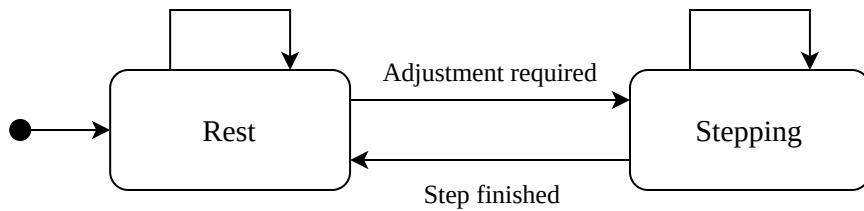


Figure 5.7: Gait State Machine

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 are encapsulated by the boolean array, \mathbf{S}_i , defined by equation 5.15, where a 1 indicates swinging and 0 supporting.

$$\mathbf{S}_{i-\xi} = [i \text{ is even}] \quad (5.15)$$

where $\xi \in i$ is the active sextant/leg number, and,

$$i = [\![0, \dots, 5]\!]$$

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.

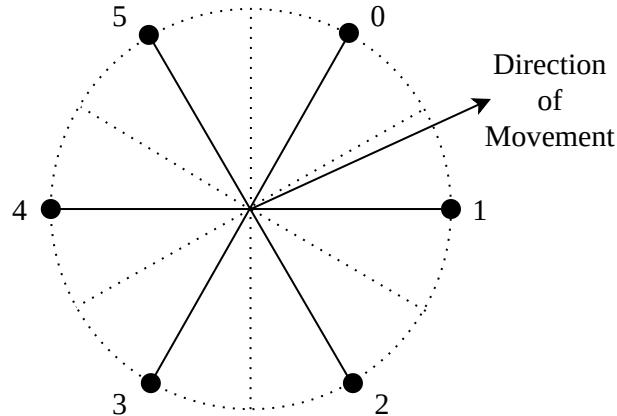


Figure 5.8: Leg sextants, with sextant 1 being active.

This of course would not be sufficient to define a walking gait, as at the end of each step \mathbf{S}_i does not invert. Thus an additional step after equation 5.15 is added. The current horizontal length of leg ξ , defined as l_ξ , is compared to its nominal horizontal length, L_ξ . If $l_\xi > L_\xi$, invert \mathbf{S}_i . As shown in equation 5.16.

$$\mathbf{S}_{i-\xi} = \begin{cases} \mathbf{i} \setminus \mathbf{S}_{i-\xi} & l_\xi > L_\xi \\ \mathbf{S}_{i-\xi} & l_\xi \leq L_\xi \end{cases} \quad (5.16)$$

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 targets; It should be noted that target matrices, denoted by \mathbf{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 with using equations 5.17 and 5.18, with reference to figure 5.9.

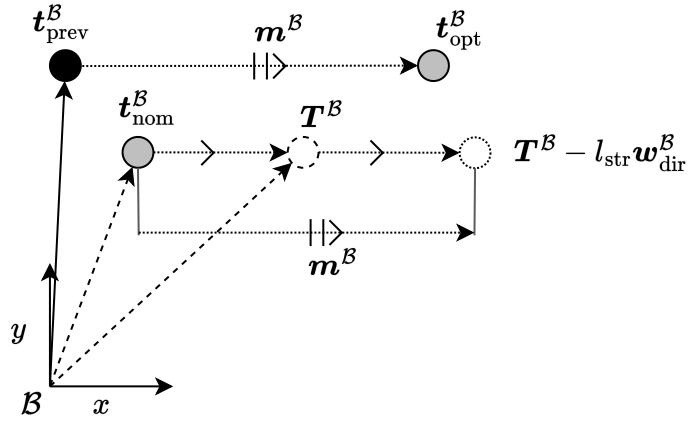


Figure 5.9: Supporting target choosing diagram

First the required move vectors, $\mathbf{m}^{\mathcal{B}}$, are calculated in equation 5.17,

$$\mathbf{m}^{\mathcal{B}} = (\mathbf{T}^{\mathcal{B}} - l_{\text{str}} \mathbf{w}_{\text{dir}}^{\mathcal{B}}) - \mathbf{t}_{\text{nom}}^{\mathcal{B}} \quad (5.17)$$

where $\mathbf{T}^{\mathcal{B}}$ contains the constant rest positions the feet, l_{str} is the desired stride length, $\mathbf{w}_{\text{dir}}^{\mathcal{B}}$ contains the desired walk directions, and $\mathbf{t}_{\text{nom}}^{\mathcal{B}}$ contains the nominal targets as calculated in equation 5.20.

Then the optimised targets, $\mathbf{t}_{\text{opt}}^{\mathcal{B}}$, are calculated as the addition of the previous targets and the move vector in equation 5.17,

$$\mathbf{t}_{\text{opt}}^{\mathcal{B}} = \mathbf{t}_{\text{prv}}^{\mathcal{B}} + \mathbf{m}^{\mathcal{B}} \quad (5.18)$$

where $\mathbf{t}_{\text{prv}}^{\mathcal{B}}$ contains the previous, optimised, targets. Note that equation 5.18 does not include the optimisation function **XXXX**, this is because the supporting feet will not move relative to the terrain, and thus do not need optimising.

5.3.4.2 For Swinging Legs

The swinging leg nominal targets are chosen with reference to figure 5.10.

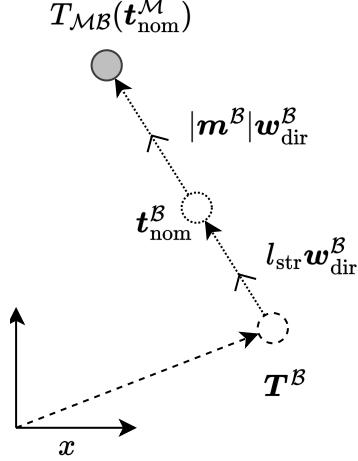


Figure 5.10: Swinging target choosing diagram

The nominal targets in map space, $\mathbf{t}_{\text{nom}}^{\mathcal{M}}$, are calculated as in equation 5.19,

$$\mathbf{t}_{\text{nom}}^{\mathcal{M}} = T_{\mathcal{B}\mathcal{M}}(\mathbf{T}^{\mathcal{B}} + (l_{\text{str}} + |m^{\mathcal{B}}|) \mathbf{w}_{\text{dir}}^{\mathcal{B}}) \quad (5.19)$$

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

For use in equation 5.17, the swinging legs nominal targets, $\mathbf{t}_{\text{nom}}^{\mathcal{B}}$, are calculated as in equation 5.20,

$$\mathbf{t}_{\text{nom}}^{\mathcal{B}} = \mathbf{T}^{\mathcal{B}} + l_{\text{str}} \mathbf{w}_{\text{dir}}^{\mathcal{B}} \quad (5.20)$$

Note that $T_{ML}(\mathbf{t}_{\text{nom}}^{\mathcal{M}})$ is quite similar to $\mathbf{t}_{\text{nom}}^{\mathcal{B}}$, the difference is that $\mathbf{t}_{\text{nom}}^{\mathcal{M}}$ is set such that it will align with $\mathbf{t}_{\text{nom}}^{\mathcal{B}}$ at the end of the step. Thus, the omission of the average supporting move vector, $\text{avg } m^{\mathcal{B}}$, in calculating $\mathbf{t}_{\text{nom}}^{\mathcal{B}}$ in equation 5.20. Meaning at until the end of the step $\mathbf{t}_{\text{nom}}^{\mathcal{M}}$ and $\mathbf{t}_{\text{nom}}^{\mathcal{B}}$ will point to different positions if world space. This is due to needing to account for body movement in world space, but not in local space.

5.4 Foot Motion

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 motion to clear any obstacles that might be in its path.

5.4.1 Existing 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 point q_i , the middle point q_m , and the final point q_f . Figure 5.11 shows the variables used to calculate this arc.

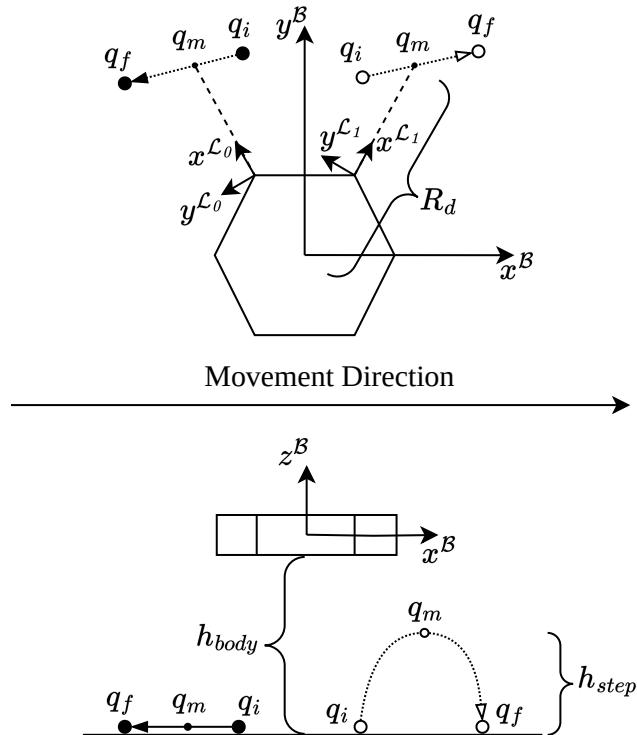


Figure 5.11: Variables for calculating foot arc.

While 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, this arc would have to be recomputed in its entirety, thus leading to possible performance concerns.

In addition to this Text, 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 arc will be undesirable, as it will rise with the desired step height for a second time. This is illustrated in Figure 5.12.

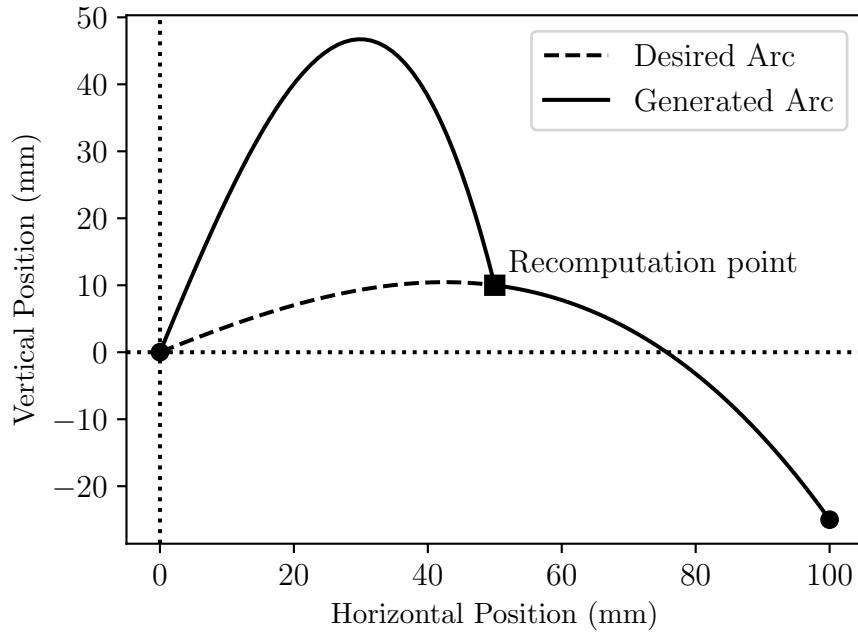


Figure 5.12: Existing arc recomputation problem

5.4.2 Improved 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.

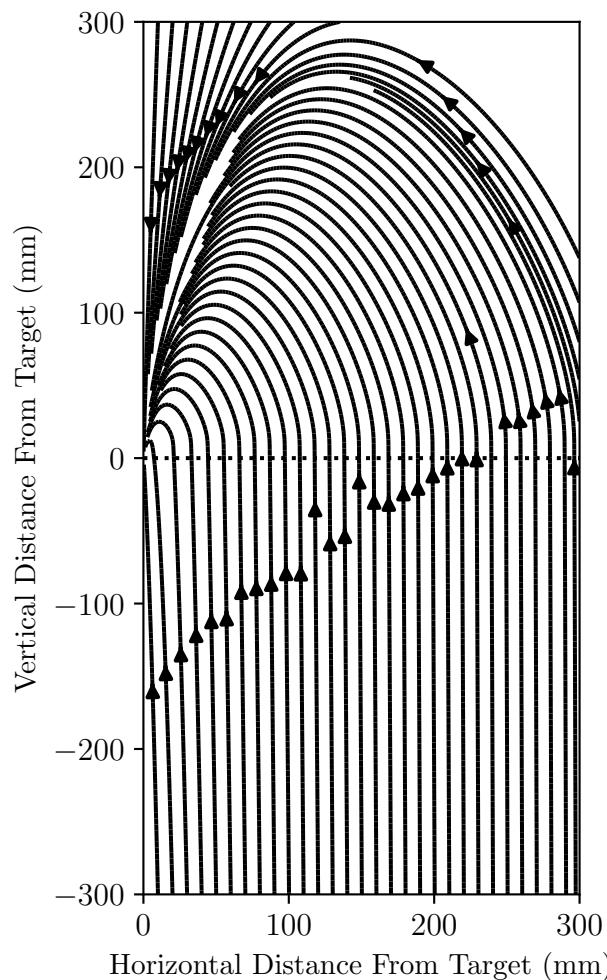


Figure 5.13: End effector movement path

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, for the process of designing the flow function please see appendix ??.

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

where,

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

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

with v_h being the variable describing the step height and $S(y)$ being a sigmoid like function responsible for the initial vertical rise. $S(y)$ is defined in Equation 5.24.

$$S(y) = \frac{0.515(y - q)}{1 + |y - q| - 0.505} \quad (5.24)$$

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

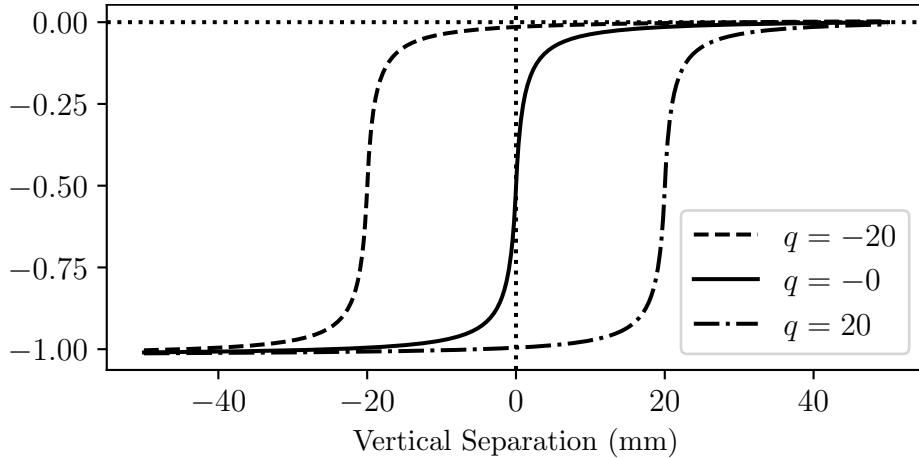


Figure 5.14: Sigmoid Like

Note that the 0.515 and 0.505 values in equation 5.24 are set to make its output range from roughly -1 to 0 over the active range.

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 use in locating the position with the best score. Finally the process of choosing a reference floor height based on the terrain is described.

6.1 Overview

The best possible anchor 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 cells 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, i.e. to reject points inside holes or close to ledges.

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.

Equation 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, see Appendix ?? for a breakdown. 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.

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

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

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

where,

$$i = \llbracket x - 1, x + 1 \rrbracket$$

$$j = \llbracket y - 1, y + 1 \rrbracket$$

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

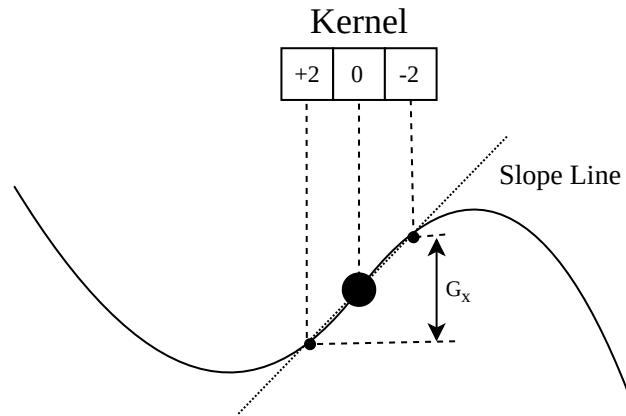


Figure 6.1: Slope Score

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 \mathbf{K} , of size n by n . The distance around inclines that is rejected depends on the chosen size of the kernel \mathbf{K} , the standard deviation of \mathbf{K} and the height differences. This score is described in Equation 6.4.

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

where,

$$i = [\lfloor \frac{1}{2}n \rfloor, x + \lfloor \frac{1}{2}n \rfloor]$$

$$j = [\lfloor \frac{1}{2}n \rfloor, y + \lfloor \frac{1}{2}n \rfloor]$$

with x and y being the indices of the cell whose score is currently being evaluated, \mathbf{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.

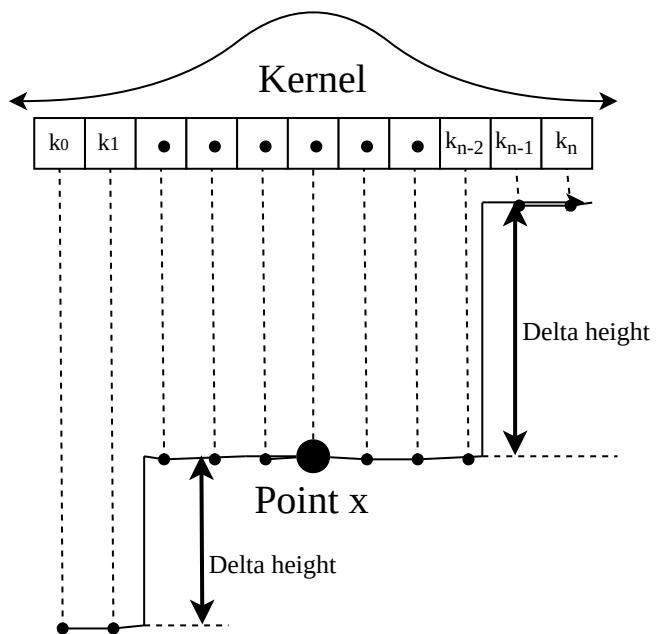


Figure 6.2: Proximity Score Diagrammatic Representation

6.2.3 Constraints

It is important to constrain the possible anchor points to confer 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 ?? 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.

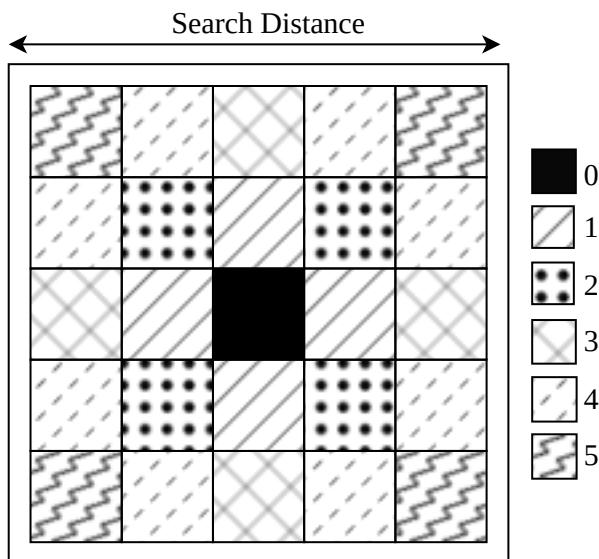


Figure 6.3: Radial search, shown for a 5 cell search diameter.

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, 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 paper uses the average of three highest target positions out of all the robots legs. This allows the robot to preemptively increase or decrease its floor height as it targets to step onto higher or lower terrain.

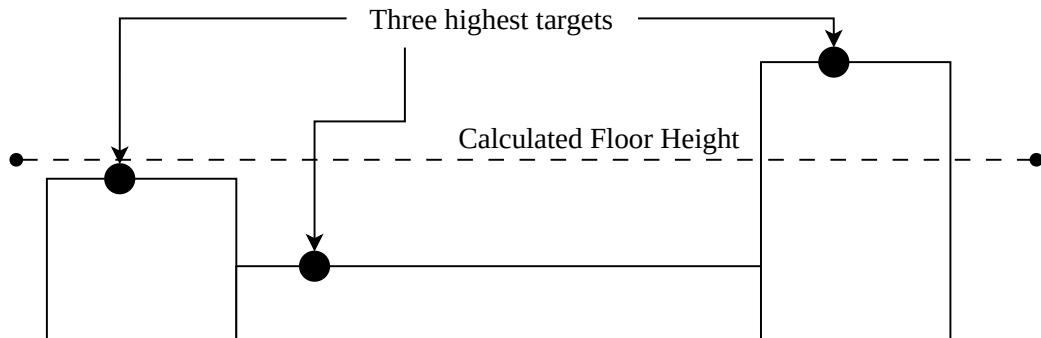


Figure 6.4: Floor height diagram.

While this method works well to preemptively adjust height and to optimise the body height against foot height, it does present one problem. If there is a tall piece of terrain in the path of the robots 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 robots body, excluding the legs and feet, is checked against the heightmap and if anywhere in the bounding box is higher than the previously set floor height, the floor height is adjusted to this new value.

6.5 Scoring Test

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 chosen, this terrain and its heightmap is shown in figure 6.5.

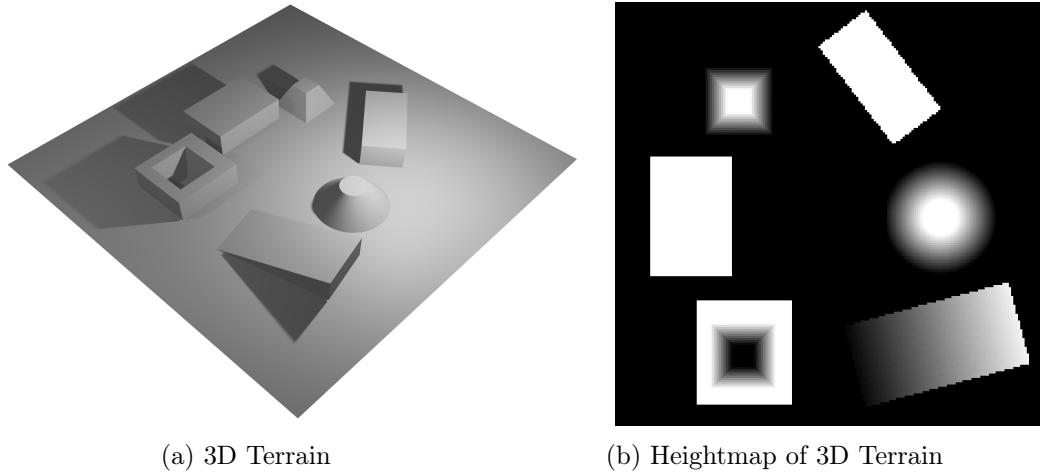


Figure 6.5: 3D Model of Terrain and its heightmap.

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 score is applied to the heightmap separately from each other, 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 slope score can be seen in figure 6.6a,

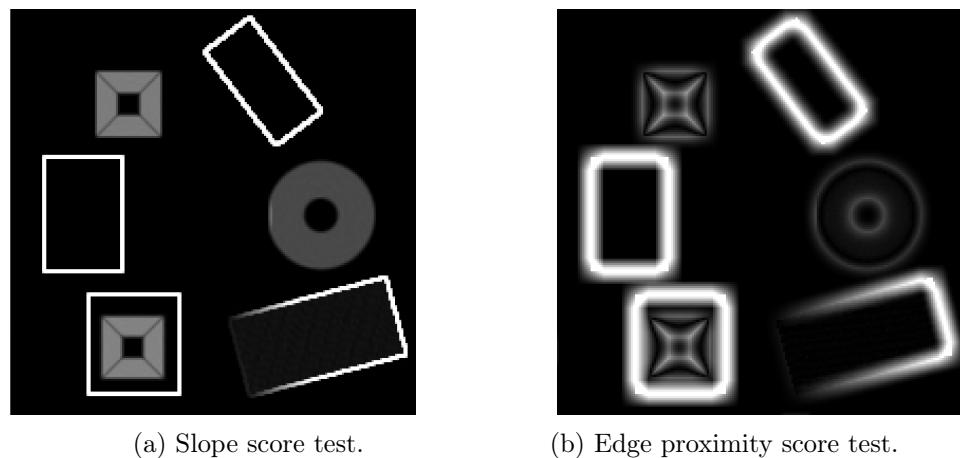


Figure 6.6: Slope score and edge proximity score tests.

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 a weighted addition of the two scores.

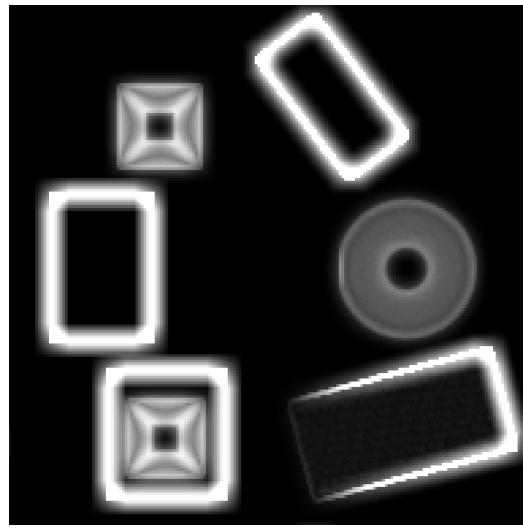


Figure 6.7: Full walkability score.

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.

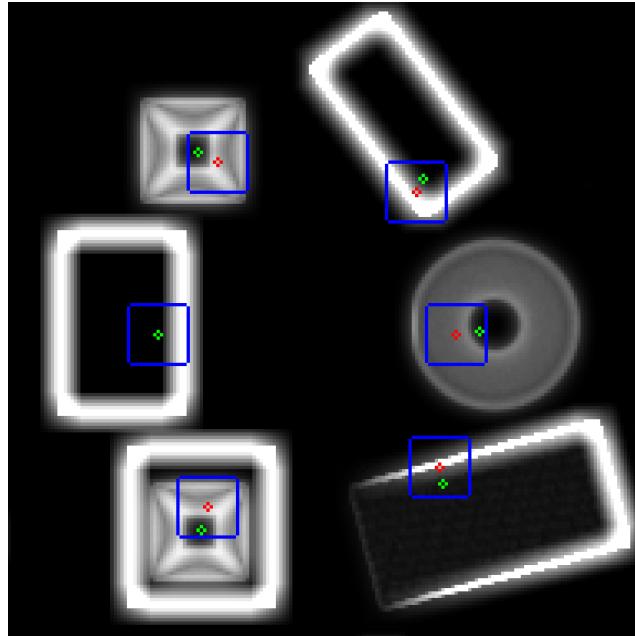


Figure 6.8: Walkability score optimisation test.

If the nominal foot position is in an acceptable position, the position will not be further optimised, this can be seen with the left most nominal position in figure 6.8. If however the nominal position is too close to an edge, such as with the top-right example, the nominal position is optimised to the closest acceptable position. The optimised position does not need to have a perfect score, as can be seen in the bottom-right example, this example is similar to the aforementioned top-right, but it adjusts onto the ram, which has a non-perfect, however acceptable, walkability score. Next the right most example shows how the nominal position is placed on a piece of terrain that is too steep, this it is optimised onto the flat surface on top of the round pedestal.

Finally, the top-left and bottom-left examples show the case of a nominal position placed on a sloped pillar and a sloped hole respectively. In both cases the nominal position is optimised to the flat surface at the top of the pillar and at the bottom of the hole.

The flat surfaces on the pillar and in the hole are large enough to not be rejected by the edge proximity score, however, if the inclines were steeper or the flat surface smaller it is possible that these nominal positions would be unsolvable, as there is no appropriate foot end placement inside the search radius. In this case the robot would need to make a course adjustment.

6.5.1 Hardware Scoring Test

Using the same colour scheme as above, the hardware heightmap found in section 4.6 is shown in figure 6.9a. The scoring system is then applied to this heightmap as shown in figure 6.9b.

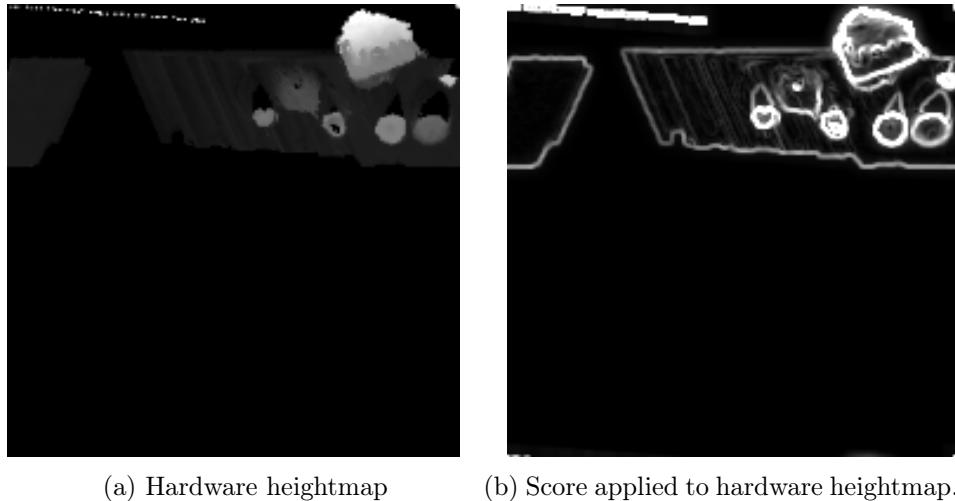


Figure 6.9: Heightmap generated from hardware and the corresponding score map.

As can be seen from figure 6.9 the scoring system applied to the hardware generated heightmap generates similarly to when it is applied to a ideal heightmap. The primary difference that should be noted is the score artifacts created around the mapped areas edge, seen as a outline around the mapped area. Additionally to the outline around the mapped area, within the mapped area light score lines can be seen. 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 position, mostly height, estimation, as explained previously in section 4.6.

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 correction to be made. This can be seen in figure 6.10 which shows a couple of nominal test points applied to the hardware generated score map.

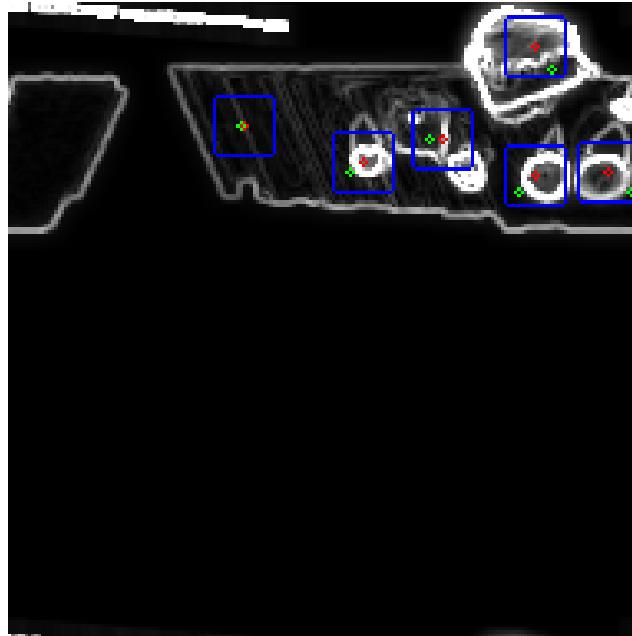


Figure 6.10: Test points on hardware score map.

As expected, the test points applied to the hardware score map behaves similarly to the ideal case shown in figure 6.8, with all the nominal points being adjusted to appropriate new positions. 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 warrant 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

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.

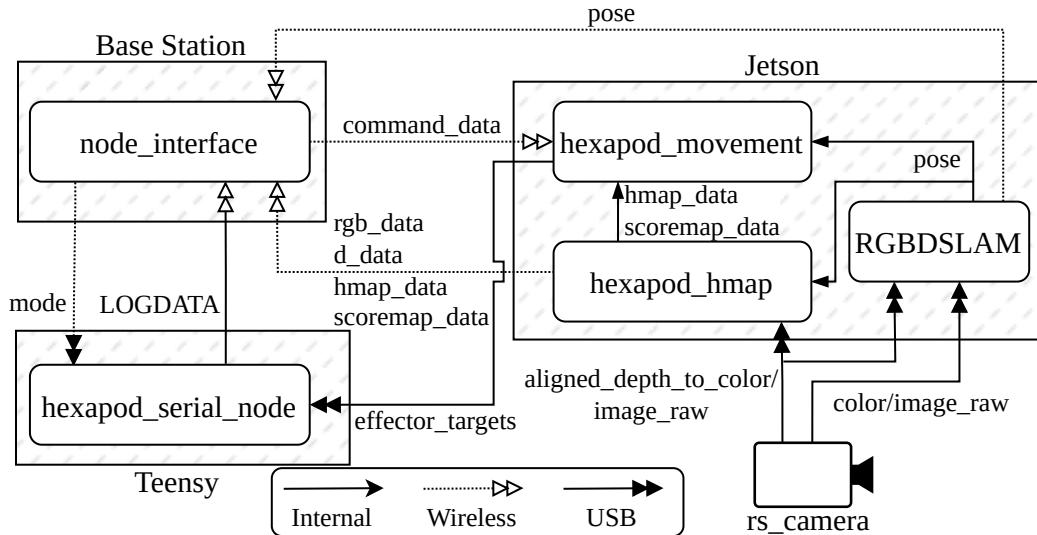


Figure 7.1: ROS nodes and communication.

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 Test

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.

8.1 Flat Plane

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 the test results of walking over a simple flat plane, as expected the step sizes and arcs are uniform with no optimisation necessary.

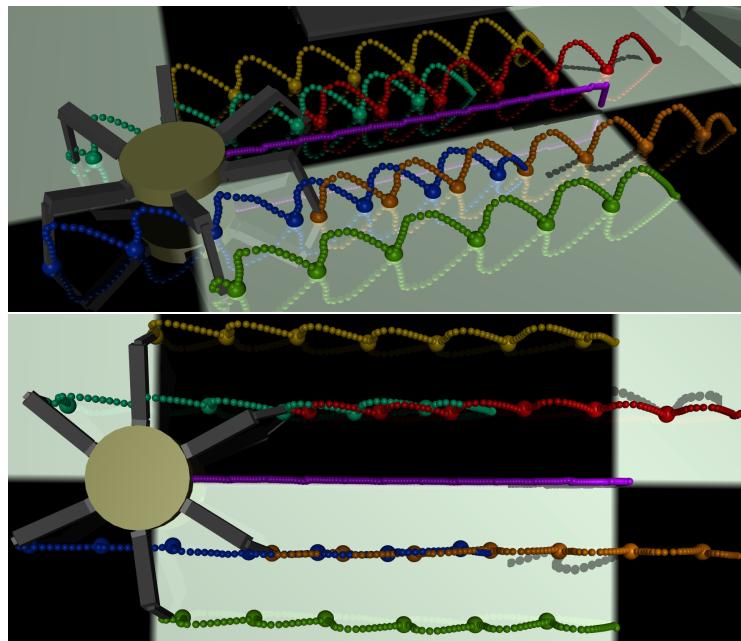


Figure 8.1: Flat plane walk test

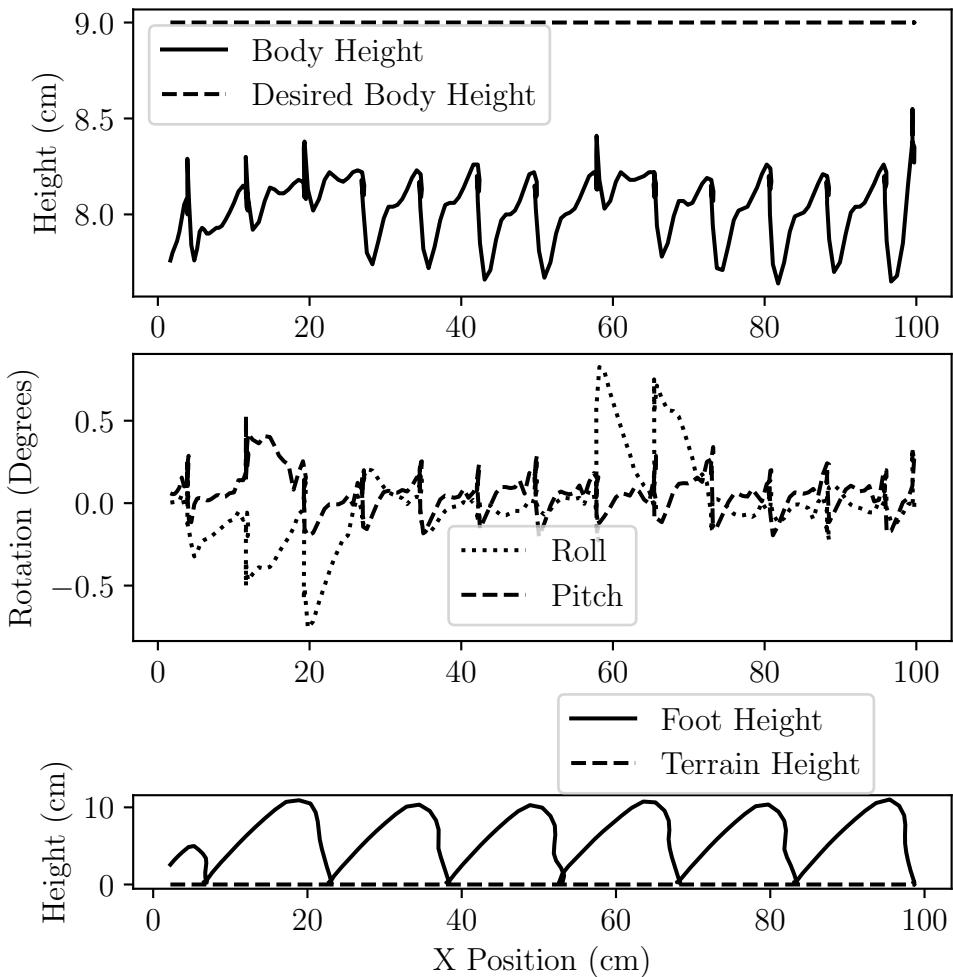


Figure 8.2: Body height (top). Body tilt (center). Position of one foot (bottom)

From figure 8.2 it can be seen that there is a constant error of about 1cm in the body height of the robot, this can be easily solved by adding a constant offset or adding integral control to the leg servos. It is also clear that as the robot walks there are some oscillations in the body height, this is to be expected and the oscillations are lower than the 10% desired body height specification.

Next it can be seen that the tilt of the body, in both roll and pitch, tends to stay below 0.5° , which is well within specifications. It is also possible to further improve the tilt error by incorporating accelerometer based feedback control to the feet heights. Finally, the height of one of the legs are shown, from this it is clear that the robot takes uniform, stable steps.

8.2 Staircase

The staircase test demonstrates the ability of the robot to adjust the height of its feet in order to maintain a level body, additionally the ability of the robot to automatically adjust its height relative to the floor is also demonstrated. A 3D image of this test can be seen in figure 8.3 and the data plots in figure 8.4.

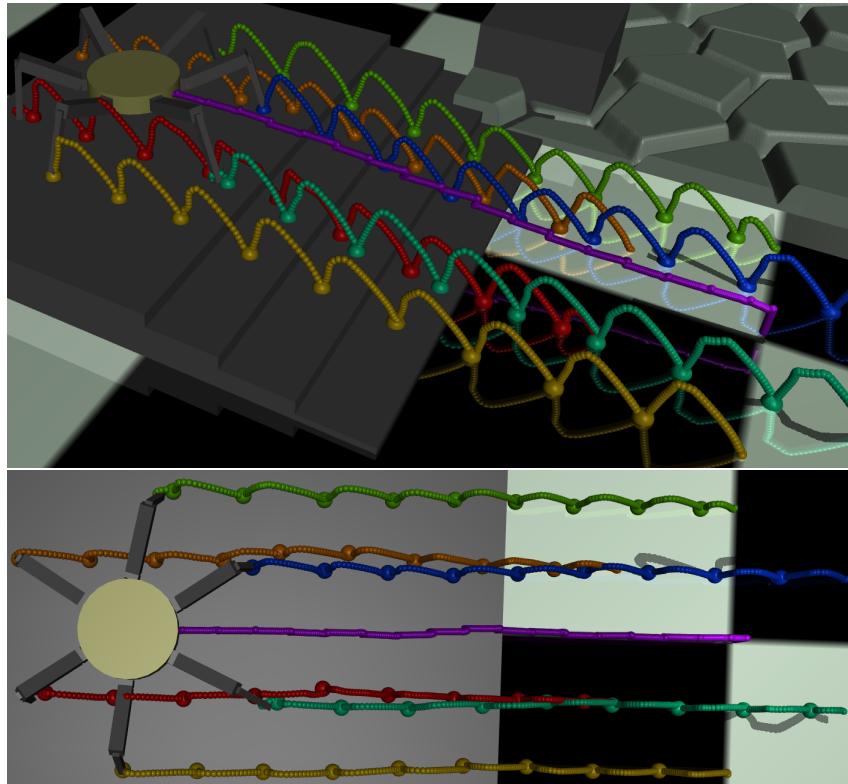


Figure 8.3: Stairs walk test

Similarly to the flat plane test, as expected, the constant body height error is still present. However, from this test it can be seen in figure 8.4 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.

Body tilt, while more prominent than in the flat plane test, is still quite low as it largely stays below 1.0° , with intermittent jumps approaching 2.0° . This is still within the specifications, and as previously stated could easily be improved by incorporating accelerometer based feedback control on the foot height.

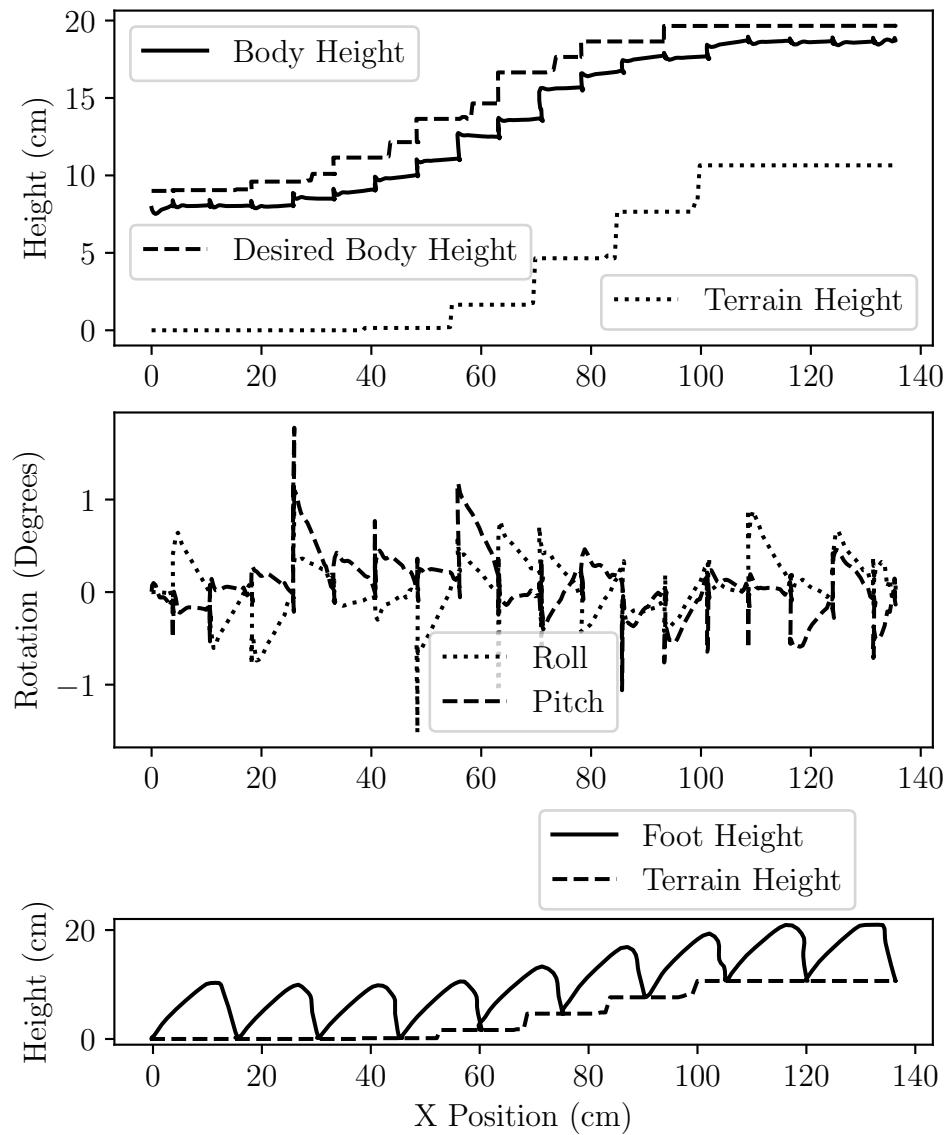


Figure 8.4: Body height (top). Body tilt (center). Position of one foot (bottom)

Finally, the foot height plot clearly shows how one of the feet of the robot adjusts its height and step arc based on the terrain. Otherwise the foot arc looks very similar to that of the flat plane test, which is optimal.

8.3 Organic

The organic test aims to show how the robot can place feet on an appropriate spot on the terrain, while maintaining level and a certain height. Figure 8.5 shows the 3D view of the test.

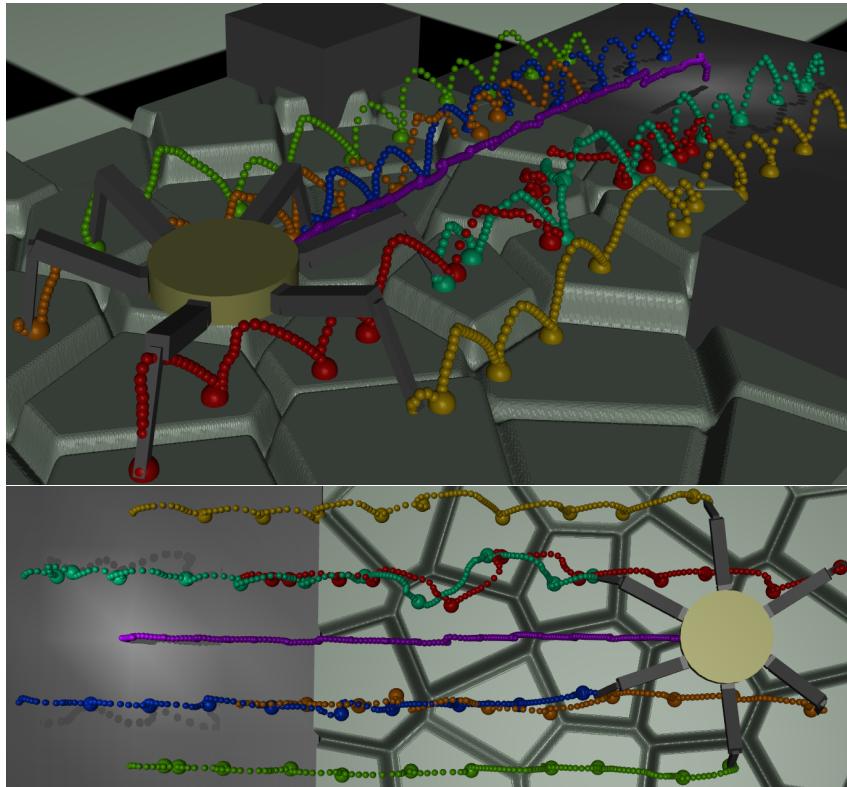


Figure 8.5: Organic walk test.

From figure 8.6 it can be seen that the body height error seen in the flat plane test is still present and more severe. But the body does stay within a reasonable margin from the desired height.

For most part the body tilt is similar to that of the stairs test, however 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 end 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.

Finally, when looking at the foot position plot in figure 8.6, please note that the plot is a projection into the X-Z plane, and thus any adjustment made in

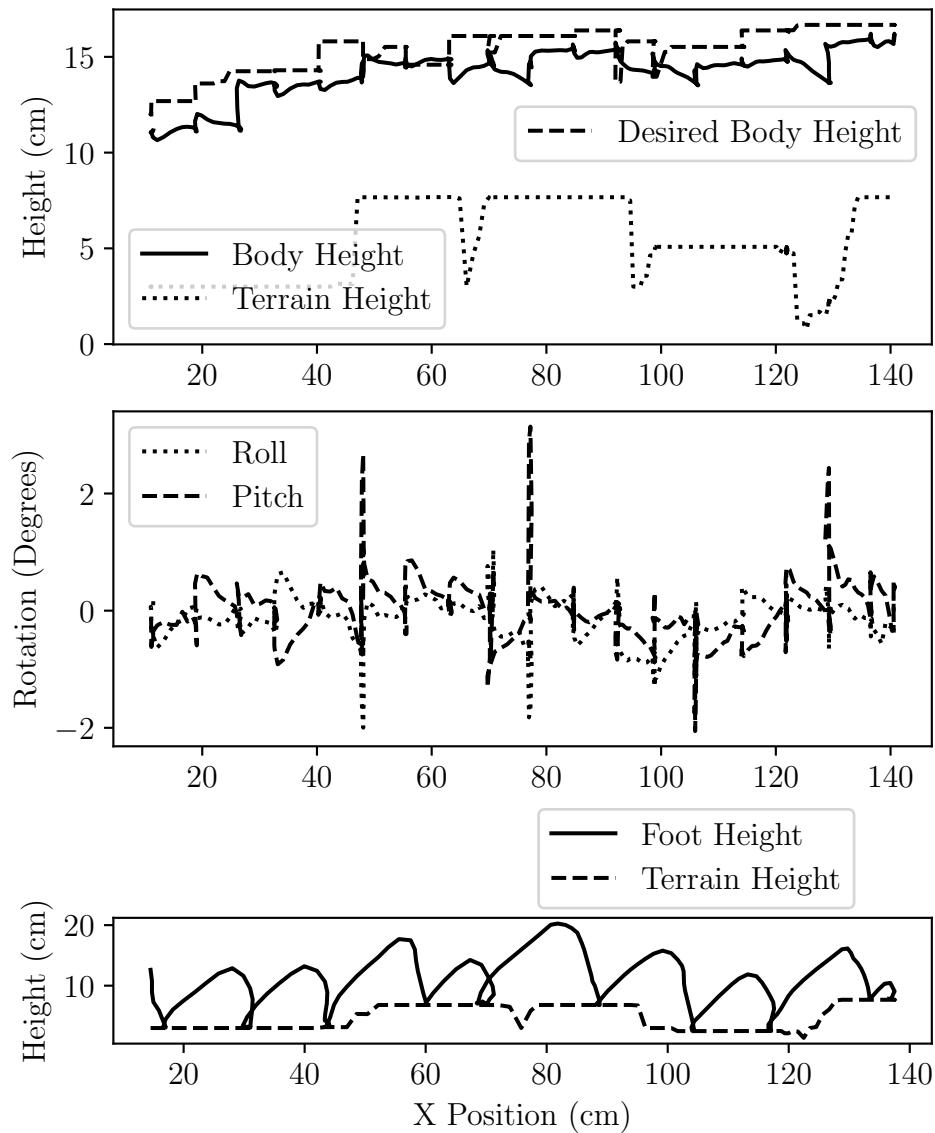


Figure 8.6: Body height (top). Body tilt (center). Position of one foot (bottom)

the Y-axis is not shown, the foot shown has been chosen to include minimum Y-axis variation. Thus, please also see the 3D view in figure 8.5. From these two figures it is clear that the foot end positions are, in addition to height, adjusted in the horizontal plane, the step arcs are appropriately adjusted and remain relatively smooth, similar to the flat plane test.

Chapter 9

Conclusions

Throughout this thesis a system to map terrain and score terrain for walkability was designed and implemented. Then a baseline tripod gait walking system was implemented, this system generated foot positions that would allow the hexapod robot to blindly walk over flat terrain. The walkability was then used to adjust the nominal foot positions from the baseline walking system to the terrain, in both the horizontal and vertical planes.

Various tests were performed to validate the effectiveness of these systems in simulation and in hardware. The tests revealed that the mapping and scoring systems performed well, with possible improvements in the frequency of pose updates from the SLAM system to reduce noise and artifacts in the heightmap.

The walking testes, performed in simulation, showed that the system is capable of successfully navigating uneven terrain with large vertical displacements, such as stairs, and horizontal undulations, such as an assortment of rocks.

As the system focuses exclusively on the mapping and feet motion planning aspect of the robot, future work could include macro scale path planning and waypoint navigation. Furthermore, terrain could be classified by not only its geometric shape, but also by its material characteristics, such as differentiating between sand, rock, dirt, etc.

Appendix A

Reference Frame Transforms

Reference frame transforms are defined by the shorthand $T_{AB}(\mathbf{x}^A)$, meaning vector \mathbf{x}^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 described in this appendix.

A.1 Camera to Map

Transforming a vector from camera, to map space requires rotating the vector by the camera quaternion, adding the camera's global position to the vector, scaling the vector into the map by multiplying with the map scaling factor, and finally modulating the vector to the confines of the map. As shown in equation A.1.

$$\begin{aligned}\mathbf{x}^{\mathcal{M}} &= T_{\mathcal{CM}}(\mathbf{x}^{\mathcal{C}}) \\ &= (\mathbf{q}_{\text{cam}} \cdot \mathbf{x}^{\mathcal{C}} \cdot \mathbf{q}_{\text{cam}}^{-1} S + \mathbf{p}_{\text{rob}}^{\mathcal{M}}) \mod N\end{aligned}\tag{A.1}$$

where $\mathbf{x}^{\mathcal{C}}$ is the vector in camera space, \mathbf{q}_{cam} is the camera quaternion that rotates $\mathbf{x}^{\mathcal{C}}$ into world space, with its z axis pointing upwards, $\mathbf{p}_{\text{rob}}^{\mathcal{M}}$ is the coordinate vector of the robot in map space, and S is the scaling factor used to relate heightmap cells to cm. The relationship between N, S and E are characterised by equation A.2.

$$N = E \cdot S\tag{A.2}$$

where $E \times E$ is the size of the $N \times N$ heightmap buffer in cm.

A.2 World to Map

The world to map transform is similar to camera to map transform, except that the rotation is not required, as the map and world frames are already aligned. This is shown in equation A.3.

$$\begin{aligned}\mathbf{x}^{\mathcal{M}} &= T_{\mathcal{W}\mathcal{M}}(\mathbf{x}^{\mathcal{W}}) \\ &= \mathbf{x}^{\mathcal{W}} S \mod N\end{aligned}\quad (\text{A.3})$$

where S 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.3, with the exception of using the body quaternion instead of the camera quaternion. This can be seen in equation A.4.

$$\mathbf{x}^{\mathcal{M}} = (\mathbf{q}_{\text{bod}} \cdot \mathbf{x}^{\mathcal{B}} \cdot \mathbf{q}_{\text{bod}}^{-1} S + \mathbf{p}_{\text{rob}}^{\mathcal{M}}) \mod N \quad (\text{A.4})$$

where $\mathbf{x}^{\mathcal{B}}$ is the vector in local space, \mathbf{q}_{bod} is the robot body's quaternion that rotates $\mathbf{x}^{\mathcal{B}}$ into map space, with its z axis pointing upwards, $\mathbf{p}_{\text{rob}}^{\mathcal{M}}$ is the coordinate vector of the robot in map space, and S 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, \mathbf{x}_{temp} , is defined as a helper,

$$\mathbf{x}_{\text{temp}} = \mathbf{p}_{\text{rob}}^{\mathcal{M}} \frac{1}{S} \quad (\text{A.5})$$

where \mathbf{x}_{temp} is the map coordinates with the inverse map scaling applied. Now \mathbf{x}_{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.6

$$\mathbf{x}_{\text{temp}} \Leftarrow \begin{cases} \mathbf{x}_{\text{temp}} & \mathbf{x}_{\text{temp}} \leq \frac{1}{2}E \\ \mathbf{x}_{\text{temp}} - E(\text{sgn } \mathbf{x}_{\text{temp}}) & \mathbf{x}_{\text{temp}} > \frac{1}{2}E \end{cases} \quad (\text{A.6})$$

where sgn is the signum function, which return either 1 or -1 depending on the sign of the operand. Note that equation A.6 is applied to both coordinates in the vector \mathbf{x}_{temp} separately, in pursuit of readability this is not explicitly stated.

Now that \mathbf{x}_{temp} has been corrected for the circular nature of the map buffer, only the rotating into the local space remains. As shown in equation A.7.

$$\begin{aligned}\mathbf{x}^{\mathcal{B}} &= T_{\mathcal{MB}}(\mathbf{x}^{\mathcal{M}}) \\ &= \mathbf{q}_{\text{bod}}^{-1} \cdot \mathbf{x}_{\text{temp}} \cdot \mathbf{q}_{\text{bod}}\end{aligned}\quad (\text{A.7})$$

where \mathbf{q}_{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.8 shows this transformation.

$$\begin{aligned}\mathbf{x}^{\mathcal{L}_i} &= T_{\mathcal{BL}_i}(\mathbf{x}^{\mathcal{B}}) \\ &= \mathbf{Q}_i^{\mathcal{B}} \cdot \mathbf{x}^{\mathcal{B}} \cdot \mathbf{Q}_i^{\mathcal{B}^{-1}} - \mathbf{R}_i\end{aligned}\quad (\text{A.8})$$

where $\mathbf{Q}_i^{\mathcal{B}}$ 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 entirely 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 operating 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 publishers 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.
- 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.
- 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>
- 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.
- 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*.
- 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.