Motion Control of a Hexapod Robot Over Uneven Terrain Using Heightmaps

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

March 2023

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 .....

Abstract

Motion Control of a Hexapod Robot Over Uneven Terrain Using Heightmaps

A.P. Lotriet

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

Thesis: MEng (EE)

March 2023

In recent times great strides have been made in the field of autonomous robotics, especially with regards to autonomous navigation of wheeled and arial 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.

Acknowledgments

Dedication

Table of contents

List of figures

List of tables

List of symbols

Constants

$L_0 =$	$300\,\mathrm{mm}$	

Variables

$Re_{\,\mathrm{D}}$	Reynolds number (diameter)	$[\,]$
x	Coordinate	$[\,\mathrm{m}\,]$
$\ddot{x}$	Acceleration	$[\,\mathrm{m/s^2}\,]$
θ	Rotation angle	$[\,\mathrm{rad}\,]$
τ	Moment	$[\,\mathrm{N{\cdot}m}\,]$

Vectors and Tensors

$\overrightarrow{v}$	Physical vector, see equation ...

Subscripts

a	Adiabatic
a	Coordinate

Notation

$[\![a..b]\!]$	Defines an integer sequence from a to b, including a and b, with a increment of 1.
$x \bigcirc \mathcal{A}$	Indicates that x is in reference frame $\mathcal{A}$.
$x \quad (\mathcal{A} \to \mathcal{B})$	Transforms $x \bigcirc \mathcal{A}$ to reference frame $\mathcal{B}$.[1] Thus, $[x \bigcirc \mathcal{A} \quad (\mathcal{A} \to \mathcal{B})] \bigcirc \mathcal{B}$.
$\mathcal{W}$	Indicates the world reference frame.[2]
$\mathcal{L}$	Indicates the local reference frame.[2]
$\mathcal{M}$	Indicates the map reference frame.[2]

[1] See appending **??** for transform definitions.
[2] See appending **??** for reference frame definitions.

$\mathcal{C}$	Indicates the camera reference frame.[2]

Abreviations

Chapter 1

Introduction

1.1 Background

There are many applications where vehicles are required to traverse rough terrain, such as in mines, rescue operations, agriculture, construction, etc. In many of these use cases rough 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 rough terrain.

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

1. Obtain a mathematical model of the robot, its actuators and its sensors.
2. Create a model of the robot in a simulation environment for development and testing.
3. Implement a vision based Simultaneous Localisation and Mapping (SLAM) system.
4. Develop a real time vision based dense mapping system for use in anchor point selection.

5. Develop a optimisation system to select optimal end effector anchor points based on the surrounding terrain.

6. Implement and test the system on the physical hexapod robot.

1.3 Methodology

When deciding how to determine optimal end effector 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 robots end effectors or measuring servo torque to determine when the end effectors were in contact with a surface. A previous paper by Erasmus *et al.* (2023) used a RGB-D camera by storing past snapshots to adjust the end effectors to the optimal height, it was decided that the primary sensing method for this thesis would also be a RGB-D camera but instead of storing snapshots, a height map 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 accurate simulation of the hexapod. The primary simulation packages that were considered are Gazebo, PyBullet and Multi-Joint dynamics with Contact (MuJoCo). Gazebo was a appealing choice due to the easy integration with 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 tripod gait state machine, IK system and control interface was implement, at this stage the hexapod was capable of walking on flat terrain.

Next the the system to generate the height map was implemented, this entailed sampling the RGB-D camera and comparing cells in the height map against the depth buffer. Once the height map was implemented it was possible to build the system responsible for end effector placement, this is covered in detail in **??**, after which collision checking for the generated end effector motion was implement, ensuring that the hexapod does not get stuck on pieces of terrain.

With this the system was realised in simulation, next the system was implemented and tested on the physical robot, discussed in detail in chapter 7

1.4 Scope and Limitations

As the hardware used was developed by Erasmus *et al.* (2023) this project will focus only on developing the necessary software to control het robot hardware.

The velocity control, tilt angle stabilisation and end effector motion planner was developed by the author, while the low level IK controller used was

developed by (Erasmus *et al.*, 2023). The scope of this project does not include autonomous waypoint navigation and thus requires a human operator to provide desired velocity commands. If no solution can be found for the given velocity command the system will not attempt to adjust the velocity command, the human operator will be required to adjust the command.

The local dense height map system was developed by the author, while the SLAM system used, ORB-SLAM3 was developed by Campos *et al.* (2021). It should be noted however that ORB-SLAM3 does generate a global sparse feature map of the environment, thus implementing waypoint navigation should be a trivial addition.

The sensors used in this project are limited to a single RGB-D camera, thus even with the generation of a local map, there could be cases where the system will not have height data around a desired anchor point. No torque or touch sensors are used to augment the system, thus if a leg were to collide with the terrain the robot will not adjust its trajectory. The system will however attempt to choose a step path based on the local heigh tmap such that no collision occurs. Additionally no Inertial Measuring Unit (IMU) is used, pose estimation is entirely handled by the 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 a overview of the hexapod hardware and the modelling thereof. This includes the robots mechanical form, sensors, on board computers and the simulation environment that is used.

Chapter 4 describes the environment mapping systems used, this includes the local dense height map and the sparse SLAM system.

Chapter 5 covers motion related topics, this includes the walking gait, IK and effector motion planning.

Chapter 6 describes the optimisation function and its various scores used to acquire the optimal end effector anchor points during each step taken.

Chapter 7 covers the hardware implementation process and software structure on the hardware.

Chapter 8 describes the various tests preformed and results obtained thereof.

Chapter 9 provides the conclusion of the research and any recommended future additions.

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

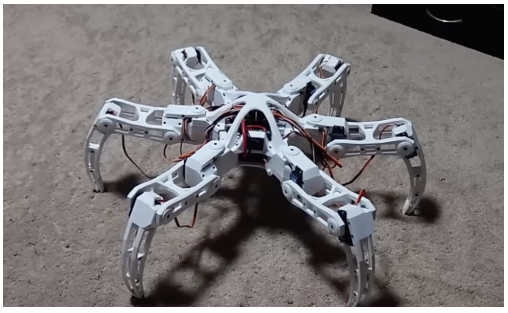


Figure 2.2: A circular hexapod

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

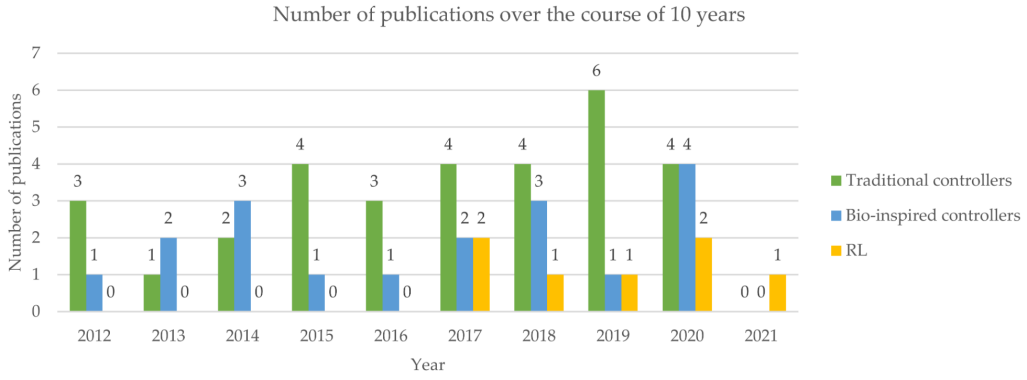


Figure 2.3: Trends of hexapod control schemes (Coelho *et al.*, 2021)

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 Learing

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) use 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 Robot Operating System (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 Related Works

TODO

2.6 Research Decisions

This research paper focuses on using vision based mapping to select the optimal anchor points for end effectors, thus it was decided that traditional kinematic control is used as it is the simplest and most predictable method of control.

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. Considering that the only relevant downside to MuJoCo 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, MuJoCo was chosen as the simulator.

Chapter 3

System Overview

This chapter aims top provide a high level overview of the system proposed in this paper, after which detailed descriptions are given in the following chapters. The goal of maneuvering rough terrain is achieved by a combination of 4 primary systems, namely, a mapping, foot placement optimisation, motion controller and a SLAM system. For a overview of the system, see Figure 3.1. The hexapod that the system is tested on can be seen in Figure 3.2.

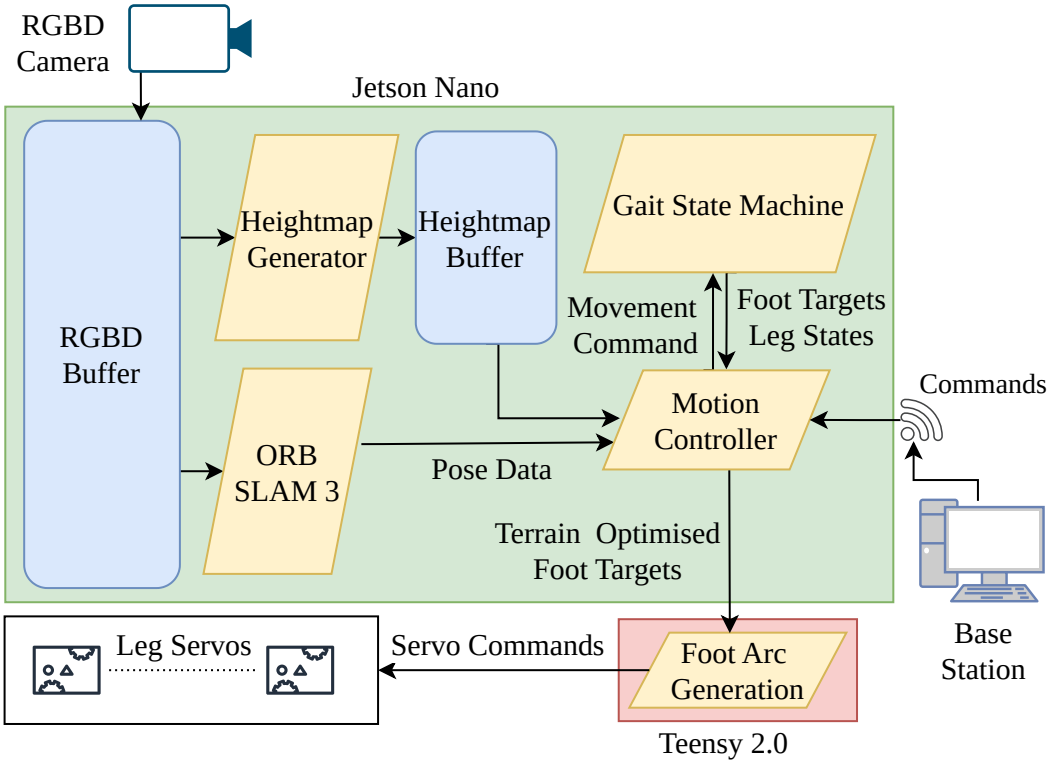


Figure 3.1: System Diagram

The mapping system utilises the RGB-D camera to construct a dense height-map 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 heightmp 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 of 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 target proposal system, a foot arc generator, and IK. The gait state machine selects the swinging and supporting legs during for each step, to achieve a tripod gait. The target proposal system proposes an initial target for all the feet based on the current step parameters. 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 foots destination, which if followed, results in a arc like motion to the destination. Finally to execute any movements, positions must be converted to angles, and velocity must be converted to angular rate, thus, and IK component is required. The motion system is further covered in chapter 5.

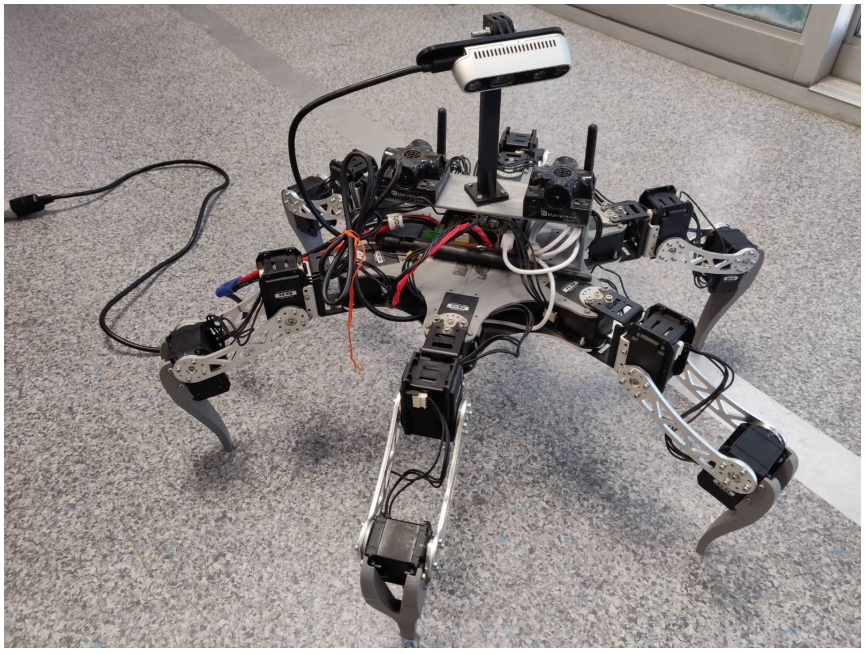


Figure 3.2: Physical Hexapod

Chapter 4

Mapping

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 primarily use of the sparse map is for localisation and extracting pose data, i.e. orientation, velocity and rate. While the dense map is used to analyse the terrain and find an appropriate point to place the three supporting feet. It is possible to also use the sparse map for autonomous navigation, however this use case in not covered in this paper. This chapter covers the design of the mapping system.

The localisation, sparse mapping and pose estimation is handle by ORB-SLAM3 as described in Campos *et al.* (2021). Since ORB-SLAM3 is not a system designed by the author, its design will not be covered in this chapter. Implementation and operation details will however be covered in chapters 7 and **??**.

4.1 Projection

In order to generate a heightmap from a RGB-D image, it is first required to project the RGB-D image into 3D space, this is necessary because a heightmap is essentially a 3D environment, that can be represented as a image due to the assumption of purely convex geometry.

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 3D space, assuming the camera is at the world origin and an zero rotation. 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.

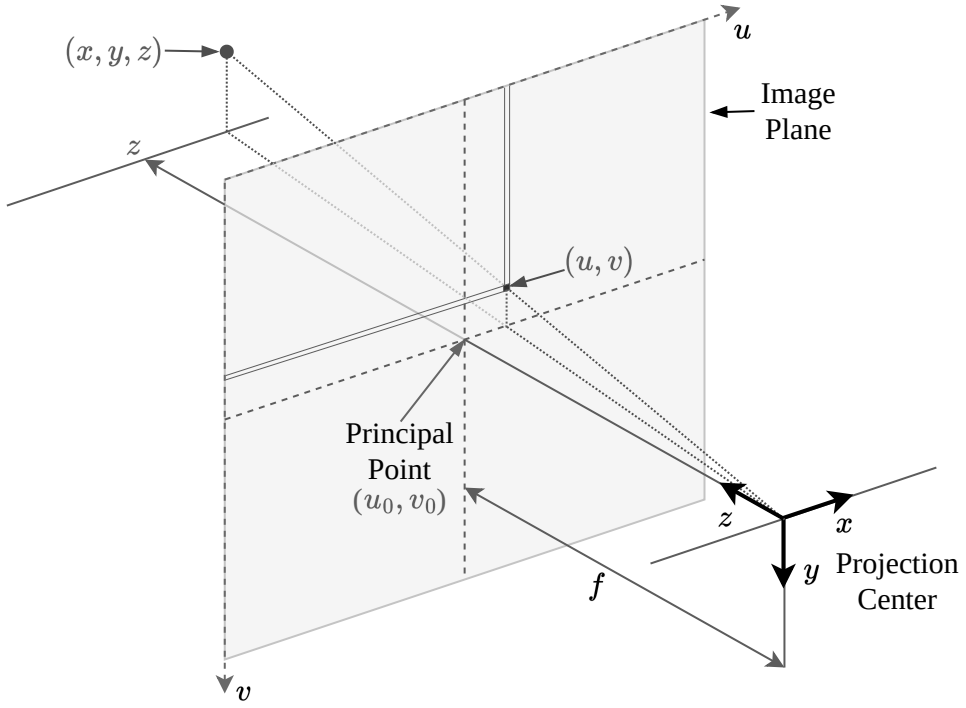


Figure 4.1: Camera Projection

Together the extrinsic and intrinsic matrices form the projection matrix, as shown in equation 4.1,

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

where $\boldsymbol{K}$ is the intrinsic matrix and $\begin{bmatrix} \boldsymbol{R} & \boldsymbol{T} \end{bmatrix}$ the extrinsic matrix, these are described in equation 4.3 and 4.4.

The projection matrix can be used to project a point on the image plane into world space as shown in equation 4.2.

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

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

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

where the focal length is represented by,

$$\alpha_x = f \cdot m_y$$

$$\alpha_y = f \cdot m_x$$

with m_x and m_y being the inverse of the width and height of a image plane pixel, f the focal length and u_0,v_0 being the principal point, optimally the center of the image plane. The skew coefficient, γ, is often, and in this case, 0.

The extrinsic matrix is as shown below,

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

where $\boldsymbol{R}$ characterises the camera's heading in world space and $\boldsymbol{T}$ the world origin expressed in the camera coordinate frame.

For ease of preprocessing points are first projected into the camera coordinate frame, in other words, the extrinsic matrix is omitted from equation 4.2. The resultant matrix equation is shown in equation 4.5.

$$\begin{bmatrix} u \\ v \\ 1 \\ 1/z \end{bmatrix} = \frac{1}{z} \begin{bmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{4.5}$$

From equation 4.5 x, y, z are found to be show in equation 4.6 to 4.8.

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

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

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

where $I_{u,v}$ is the depth image value at pixel coordinates u, v. For later use the variable $\boldsymbol{p_{cam}} \bigcirc \mathcal{C}$ is defined as in equation 4.9,

$$\boldsymbol{p_{cam}} = \begin{bmatrix} x & y & z \end{bmatrix} \tag{4.9}$$

4.2 Map Buffer

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.

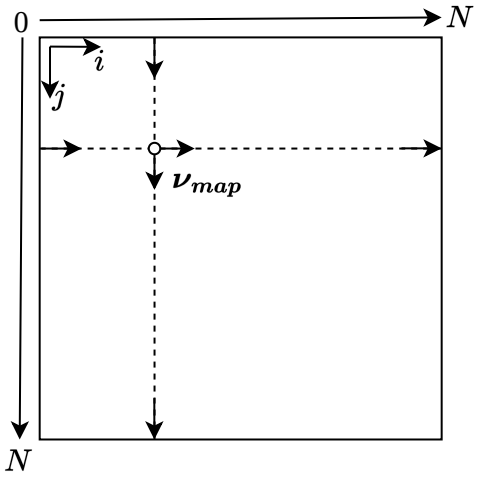


Figure 4.2: Memory Diagram

The heightmap buffer, $\boldsymbol{M}$, is of size $N \times N$ and is indexed by i, j. The mapping from the camera relative points, $\boldsymbol{p_{cam}} \bigcirc \mathcal{C}$, into map space is defined in equation 4.10,

$$\boldsymbol{p_{map}} = (\boldsymbol{q} \cdot \boldsymbol{p_{cam}} \cdot \boldsymbol{q}^{-1} S + \boldsymbol{\kappa_{map}}) \mod N \tag{4.10}$$

where $\boldsymbol{q}$ is the camera quaternion that rotates $\boldsymbol{p_{cam}}$ into world space, with its z axis pointing upwards, $\boldsymbol{\kappa_{map}}$ 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 4.11.

$$N = E \cdot S \tag{4.11}$$

where E is the size of the heightmap in cm.

The robots coordinates in the map space, $\boldsymbol{\kappa_{map}}$, can be found as shown in equation 4.12,

$$\boldsymbol{\kappa_{map}} = \boldsymbol{\kappa_{world}} \mod N \tag{4.12}$$

where $\boldsymbol{\kappa_{world}}$ is the coordinate of the robot in world space.

4.2.1 Between Map and Local Space

As the robot commands foot positions in local space, when optimising foot positions (see chapter 6) it is required to have a translation between map and local space. These translations are defined in equation 4.13 and 4.14 to 4.16.

Local to map space is a simple translation, very similar to equation 4.10,

$$\boldsymbol{\kappa_{map}} = (\boldsymbol{q_{bod}} \cdot \boldsymbol{\kappa_{loc}} \cdot \boldsymbol{q_{bod}^{-1}} + \boldsymbol{\kappa_{map}}) \mod N \tag{4.13}$$

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

$$\boldsymbol{\kappa_{temp}} = \boldsymbol{\kappa_{map}} \frac{1}{S} \tag{4.14}$$

where $\boldsymbol{\kappa_{temp}}$ is the map coordinates scaled back into world coordinates. Now $\boldsymbol{\kappa_{temp}}$ 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 4.15

$$\boldsymbol{\kappa_{temp}} \Leftarrow \begin{cases} \boldsymbol{\kappa_{temp}} & \boldsymbol{\kappa_{temp}} \leq \frac{1}{2}E \\ \boldsymbol{\kappa_{temp}} - E \operatorname{sgn} \boldsymbol{\kappa_{temp}} & \boldsymbol{\kappa_{temp}} > \frac{1}{2}E \end{cases} \tag{4.15}$$

where sgn is the signum function, which return either 1 or -1 depending on the sign of the operand. **Note** that equation 4.15 is applied to both coordinates in the vector $\boldsymbol{\kappa_{temp}}$ separately, in pursuit of readability this is not explicitly stated.

Now that $\boldsymbol{\kappa_{temp}}$ has been corrected for the circular nature of the map buffer, only the rotating into the local space remains.

$$\boldsymbol{\kappa_{loc}} = \boldsymbol{q_{bod}^{-1}} \cdot \boldsymbol{\kappa_{temp}} \cdot \boldsymbol{q_{bod}} \tag{4.16}$$

Chapter 5

Motion

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).

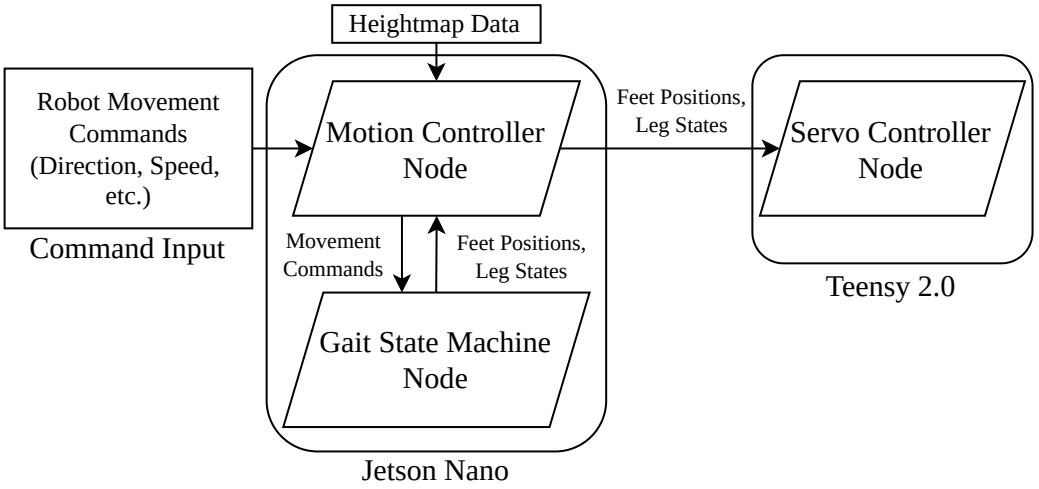


Figure 5.1: Motion System Overview

5.1 Gait 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 5.2. 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 5.1.1.

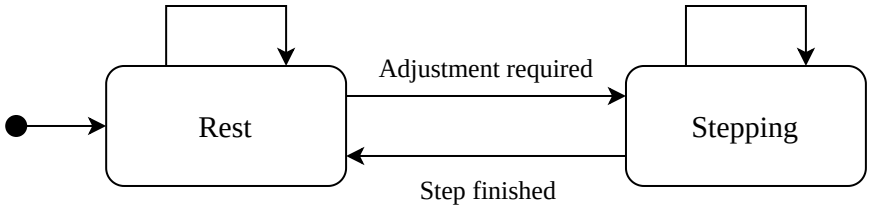


Figure 5.2: Gait State Machine

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 5.1.1. 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

5.1.1 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, $\boldsymbol{S_i}$, defined by equation 5.1, where a 1 indicates swinging and 0 supporting.

$$\boldsymbol{S}_{\boldsymbol{i}-\xi} = [i \text{ is even}] \tag{5.1}$$

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

$$i = [\![0..5]\!]$$

Figure 5.3 illustrates and example with sextant 1 being active. Thus legs 1, 3 and 5 are swinging, while legs 0, 2 and 4 are supporting.

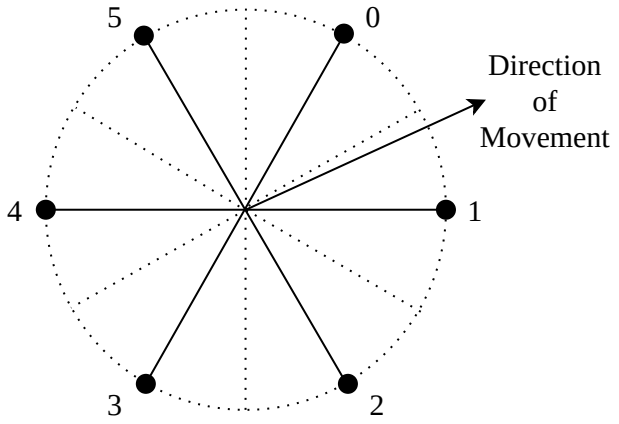


Figure 5.3: Leg sextants, with sextant 1 being active.

This of course would not be sufficient to define a walking gait, as at the end of each step $\boldsymbol{S_i}$ does not invert. Thus and additional step after equation 5.1 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 ξ. As shown in equation 5.2.

$$\boldsymbol{S}_{\boldsymbol{i}-\xi} = \begin{cases} \boldsymbol{i} \setminus \boldsymbol{S}_{\boldsymbol{i}-\xi} & l_\xi > L_\xi \\ \boldsymbol{S}_{\boldsymbol{i}-\xi} & l_\xi \leq L_\xi \end{cases} \tag{5.2}$$

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

$$i = [\![0..5]\!]$$

5.1.2 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 5.1.2.1 and 5.1.2.2 describe the process for finding the support and swinging leg targets; It should be noted that target matrices, denoted by $\boldsymbol{t}$, are common between these two sections, but are computed differently.

5.1.2.1 For Supporting Legs

For the supporting legs, first the required move vectors, $\boldsymbol{m} \bigcirc \mathcal{L}$, are calculated in equation 5.3,

$$\underset{6\times3}{\boldsymbol{m}} = (\boldsymbol{T} - l_{str}\boldsymbol{w_{dir}}) - \boldsymbol{t_{nom}} \tag{5.3}$$

where $\boldsymbol{T}$ contains the constant rest positions the feet, l_{str} is the desired stride length, $\boldsymbol{w}_{dir}$ contains the desired walk directions, and $\boldsymbol{t}_{nom}$ contains the nominal targets.

Then the optimised targets, $\boldsymbol{t_{opt}} \bigcirc \mathcal{L}$, are calculated as the addition of the previous targets and the move vector in equation 5.3,

$$\underset{6\times3}{\boldsymbol{t_{opt}}} = \boldsymbol{t_{prv}} + \boldsymbol{m} \tag{5.4}$$

where $\boldsymbol{t_{prv}}$ contains the previous, optimised, targets. Note that equation 5.4 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.

5.1.2.2 For Swinging Legs

For the supporting legs, the optimised targets in map space, $\boldsymbol{t_{map}} \bigcirc \mathcal{M}$ are calculated as in equation 5.5,

$$\underset{6\times3}{\boldsymbol{t_{map}}} = \boldsymbol{T} - \boldsymbol{\Delta\kappa} + (l_{str} + \operatorname{avg}\boldsymbol{m})\,\boldsymbol{w_{dir}} \quad (\mathcal{L} \to \mathcal{M}) \tag{5.5}$$

where $\boldsymbol{\Delta\kappa} \bigcirc \mathcal{L}$ is calculated as the difference between the robot's (body's) current position and its position at the beginning of the previous step,

$$\underset{1\times3}{\boldsymbol{\Delta\kappa}} = \boldsymbol{\kappa} - \boldsymbol{\kappa_{prv}} \quad (\mathcal{W} \to \mathcal{L}) \tag{5.6}$$

where $\boldsymbol{\kappa} \bigcirc \mathcal{W}$ is the robot's current position, and $\boldsymbol{\kappa_{prv}} \bigcirc \mathcal{W}$ is the robot's position at the beginning of the previous step.

Additionally, for use in equation 5.3, the swinging legs nominal targets, $\boldsymbol{t_{nom}} \bigcirc \mathcal{L}$, are calculated as in equation 5.7,

$$\underset{6\times3}{\boldsymbol{t_{nom}}} = \boldsymbol{T} - \boldsymbol{\Delta\kappa} + l_{str}\boldsymbol{w_{dir}} \tag{5.7}$$

Note that $\boldsymbol{t_{map}}$ is quite similar to $\boldsymbol{t_{nom}}$, the difference is that $\boldsymbol{t_{map}}$ is set such that it will align with $\boldsymbol{t_{nom}}$ at the end of the step, assuming the $(\mathcal{M} \rightarrow \mathcal{L})$ transform is applied. Thus, the omission of the average supporting move vector, avg $\boldsymbol{m}$, in calculating $\boldsymbol{t_{nom}}$.

5.2 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.

5.2.1 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. 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 5.4.

5.2.2 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 5.5 illustrates the field function and is described in section 5.2.2.1.

5.2.2.1 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

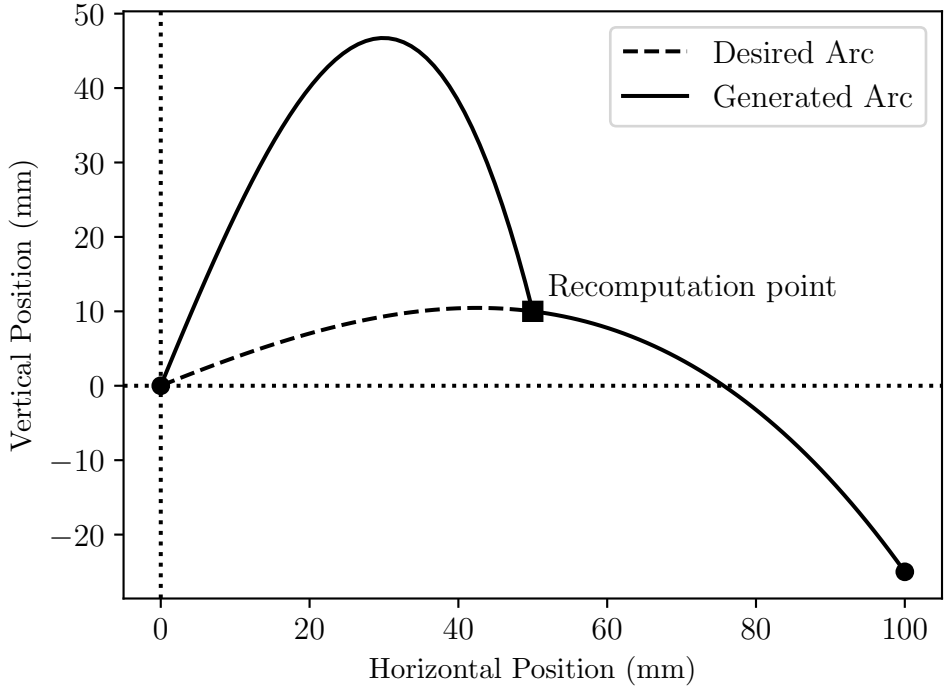


Figure 5.4: Existing arc recomputation problem

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 5.8 to 5.11, for the process of designing the flow function please see appendix **??**.

$$\begin{aligned} \rho(x,y) = \frac{\delta}{\delta x \delta y} \quad & f_a(x,y)x^2 + f_b(x,y)x + C \\ = \quad & 2f'_a(x,y)x + f'_b(x,y) \end{aligned} \tag{5.8}$$

where,

$$f'_a(x,y) = -\left|\frac{v_h}{x}\right| - |S(y)| \tag{5.9}$$

$$f'_b(x,y) = \frac{y}{x} - f_a(x,y) \tag{5.10}$$

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 5.11.

$$S(y) = \frac{0.515(y-q)}{1 + |y-q| - 0.505} \tag{5.11}$$

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 5.6 illustrates the sigmoid like function for different values of q.

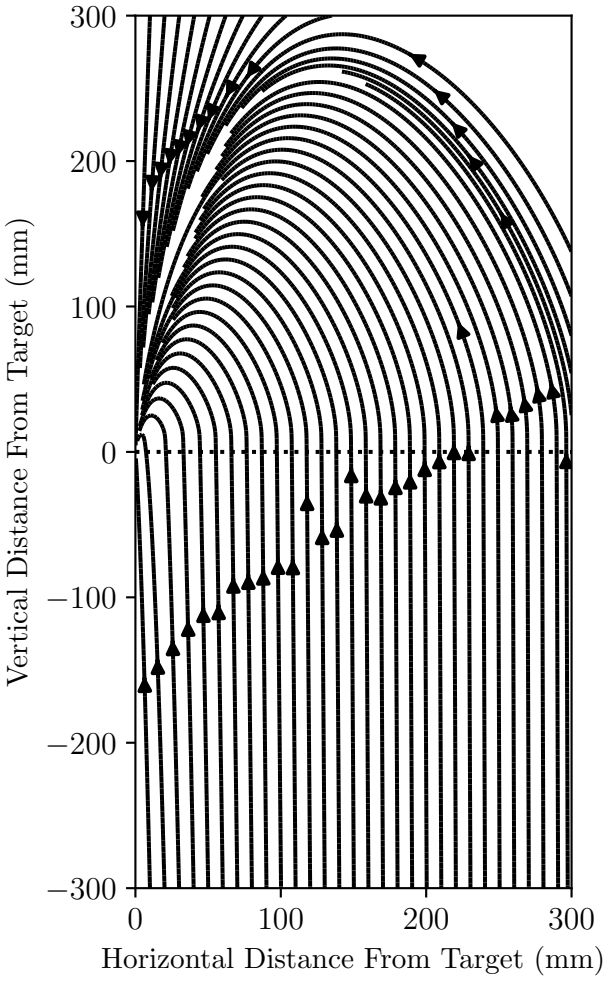


Figure 5.5: End effector movement path

Note that the 0.515 and 0.505 values in equation 5.11 are set to make its output range from roughly -1 to 0 over the active range.

5.3 Kinematics

When commanding a foot position, the servo controller requires a function to calculate servo angles. While the foot arc planner, see section 5.2, requires the current position of the feet to function. The IK and FK functions described in this section provide this functionality. Figure 5.7 illustrates the leg geometry and variables used in the IK and FK functions.

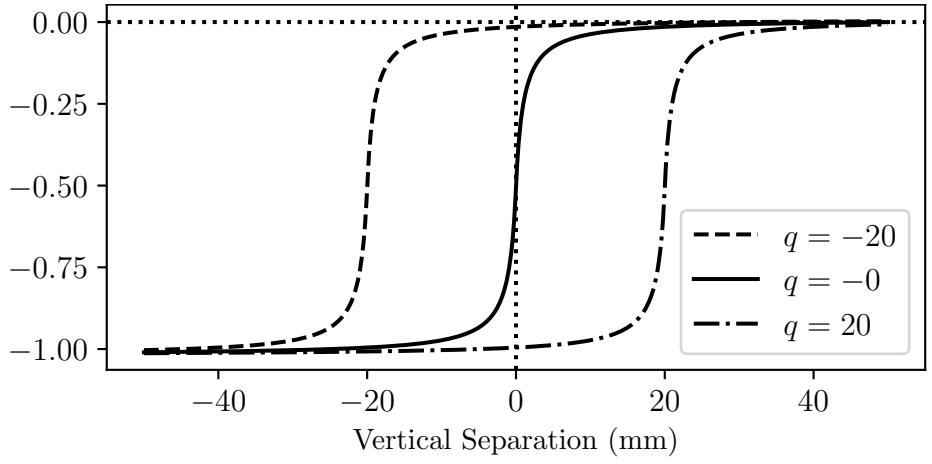


Figure 5.6: Sigmoid Like

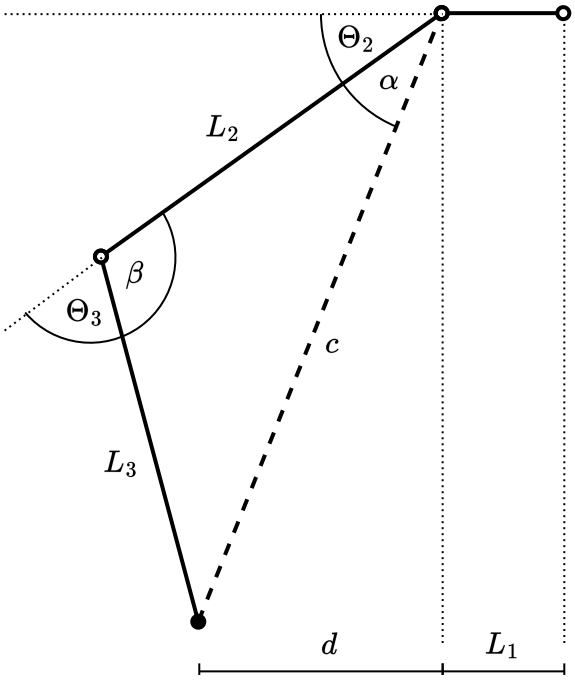


Figure 5.7: Leg Kinematics Diagram

5.3.1 Inverse Kinematics (IK)

The IK function calculates the leg servo angles, $\boldsymbol{\Theta} = [\Theta_1, \Theta_2, \Theta_3]^T_,$ required to move the foot to the given target position vector, $\boldsymbol{p_t} = [x_t, y_t, z_t]^T_.$ Equation 5.12

describes the IK function.

$$\boldsymbol{\Theta}(x_t, y_t, z_t) = \begin{bmatrix} \arctan\left(\dfrac{x_t}{y_t}\right) \\ \dfrac{\pi}{4} - \alpha - \arctan\left(\dfrac{y_t}{d - L_1}\right) \\ \dfrac{\pi}{2} - \beta \end{bmatrix} \tag{5.12}$$

Where α, β, c and d are calculate as shown in equations 5.13 to 5.16. For derivations of these variable please see **??**.

$$\alpha = \arcsin\left(\frac{L_3 \sin\beta}{c}\right) \tag{5.13}$$

$$\beta = \arccos\left(\frac{L_1^2 + L_2^2 - c^2}{2L_1 L_2}\right) \tag{5.14}$$

$$c = \sqrt{(d - L_1)^2 + z_t^2} \tag{5.15}$$

$$d = \sqrt{x_t^2 + y_t^2} \tag{5.16}$$

5.3.2 Forward Kinematics (FK)

The FK function calculates the position vector of a foot, $\boldsymbol{p_c} = [x_c, y_c, z_c]^T$, given the current angles of the leg servos, $\boldsymbol{\theta} = [\theta_1, \theta_2, \theta_3]^T$.

$$\boldsymbol{p_c}(\theta_1, \theta_2, \theta_3) = \begin{bmatrix} d\cos\theta_1 \\ d\sin\theta_1 \\ L_2\sin\theta_2 + L_3\sin(\theta_2 + \theta_3) \end{bmatrix} \tag{5.17}$$

Where d is calculated as shown in in equation 5.18.

$$d = L1 + L_2\sin\theta_2 + L_3\sin(\theta_2 + \theta_3) \tag{5.18}$$

5.3.3 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 5.19 define the derivative of the IK equations

(5.12), i.e. the angular rate, given the target movement speeds of a foot, $\dot{x}$, $\dot{y}$ and $\dot{z}$.

$$\boldsymbol{\omega}(\dot{x},\dot{y},\dot{z}) = \begin{bmatrix} \dfrac{-x\dot{y}+y\dot{x}}{x^2+y^2} \\ \dfrac{\left[(L_1-d)\dot{z}+z\dot{d}\right]\alpha + \left[(L_1-d)^2+z^2\right]\arctan\left(\dfrac{L_1-d}{z}\right)\dot{\alpha}}{(L_1-d)^2+z^2} \\ -\dot{\beta} \end{bmatrix} \tag{5.19}$$

With $\dot{\alpha}$, $\dot{\beta}$, $\dot{c}$ and $\dot{d}$ as shown in equations 5.20 to 5.23.

$$\dot{\alpha} = \frac{L_3\left[c\cos(\beta)\dot{\beta} - \sin(\beta)\dot{c}\right]}{c^2\sqrt{-\dfrac{L_3^2\sin^2(\beta)}{c^2}+1}} \tag{5.20}$$

$$\dot{\beta} = \frac{2c\dot{c}}{L_2L_3\sqrt{4-\dfrac{(L_2^2+L_3^2-c^2)^2}{L_2^2L_3^2}}} \tag{5.21}$$

$$\dot{c} = \frac{-(L_1-d)\dot{d}+z\dot{z}}{\sqrt{(L_1-d)^2+z^2}} \tag{5.22}$$

$$\dot{d} = \frac{x\dot{x}+y\dot{y}}{\sqrt{x^2+y^2}} \tag{5.23}$$

Chapter 6

Foot Placement Optimisation

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 done by applying various scores to the heightmap and selecting a point with the best score within an allowable deviation from the initial point received from the gait state machine.

6.1 Scoring

The scores considered are the gradient and proximity scores. The gradient score aims to reject points with high gradients 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.

6.1.1 Gradient Score

The gradient 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 gradient.

As the heightmap slope is not defined by a known function, the gradient 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 gradient score, S_g. Note that these equations represent a single scalar result of the Sobel operator using the Frobenius inner product, Horn and Johnson (2012). If evaluated over the whole heightmap this is equivalent to convolution.

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

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

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

where,

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

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

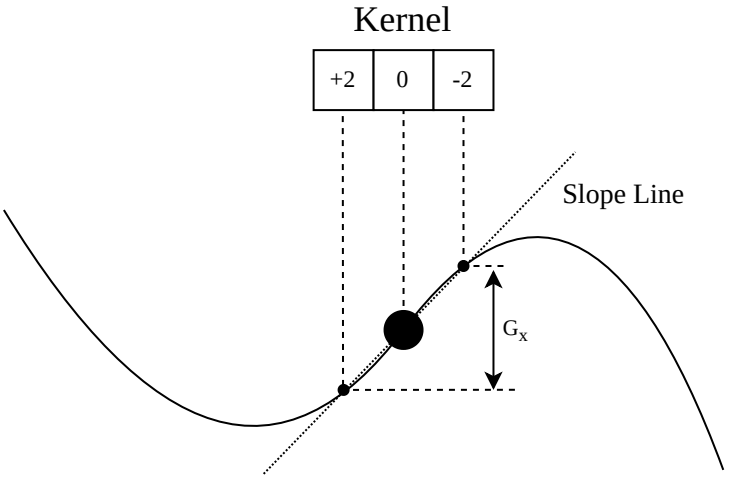


Figure 6.1: Slope Score

6.1.2 Terrain Proximity Score

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

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

where,

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

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

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

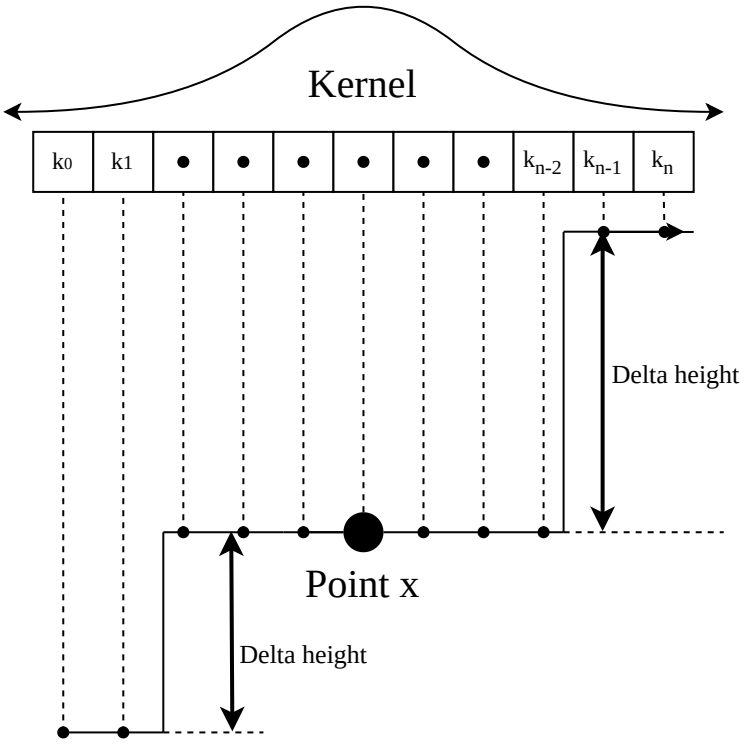


Figure 6.2: Proximity Score Diagrammatic Representation

6.1.3 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.

6.2 Method of adjustment

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.

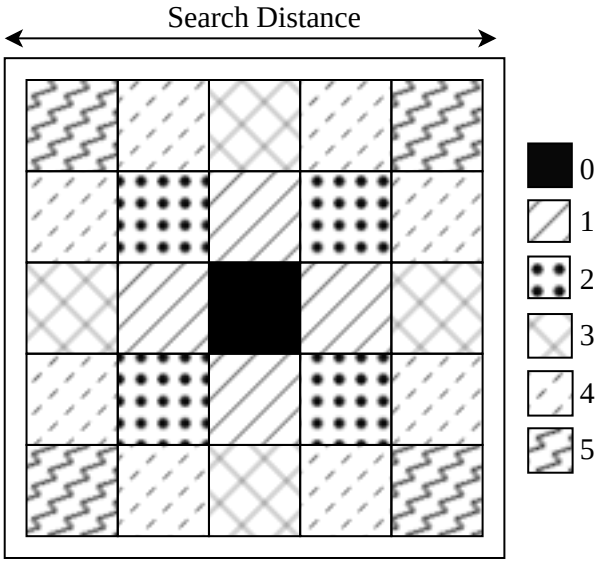


Figure 6.3: Radial search, shown for a 5 cell search diameter.

6.3 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 ore decrease its floor hight as it targets to step onto higher or lower terrain.

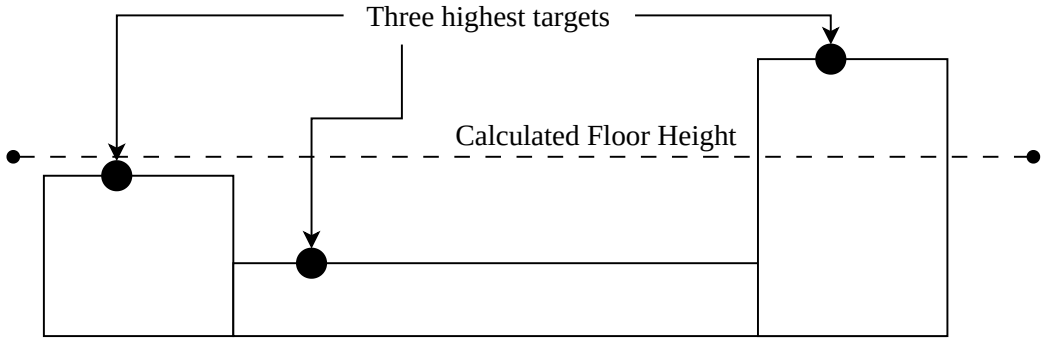


Figure 6.4: Floor height diagram.

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.

Chapter 7

Hardware Implementation

This chapter describes the process of implementing the system built in previous chapters on the physical robot.

Chapter 8

Results

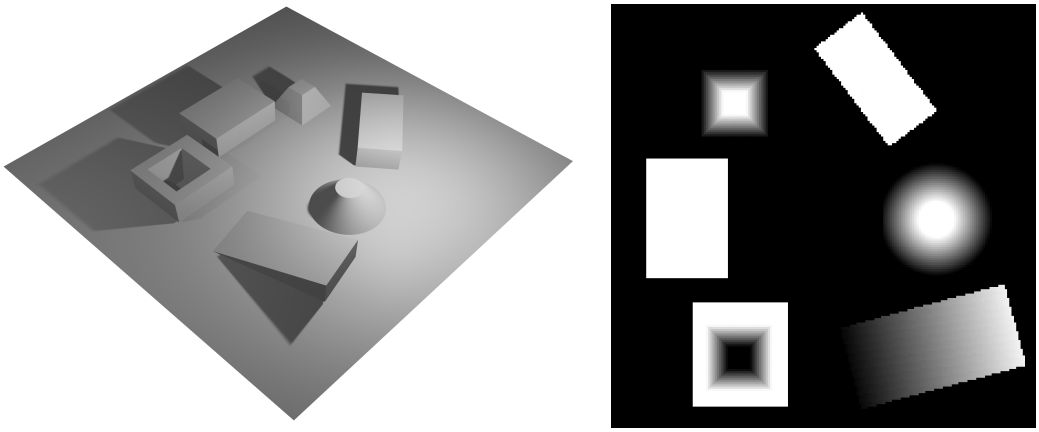
8.1 Requirements

8.2 Depth Camera

8.3 Heightmap

8.4 Walkability Scores

For testing the walkability scores described in section 6.1, 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 8.1.

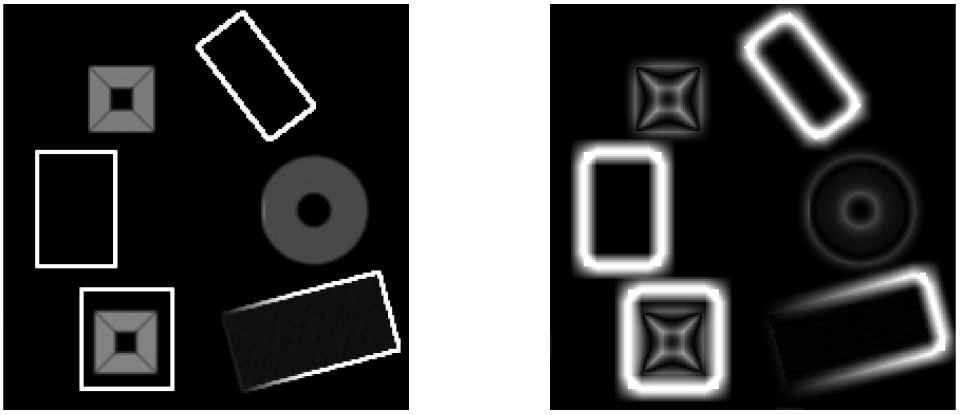


(a) 3D Terrain (b) Heightmap of 3D Terrain

Figure 8.1: 3D Model of Terrain and its heightmap.

The heightmap in figure 8.1b 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 8.2. Next the combined score is shown in figure 8.3, this is the score that is used to optimise foot end positions. The slope score can be seen in figure 8.2a,



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

Figure 8.2: 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 8.2b 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 8.3 the combined slope and edge proximity scores can be seen. This is simply a weighted addition of the two scores.

Finally, the radial search algorithm described in section 6.2 is used to optimise various nominal foot end positions. This is shown in figure 8.4. 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 8.4. 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

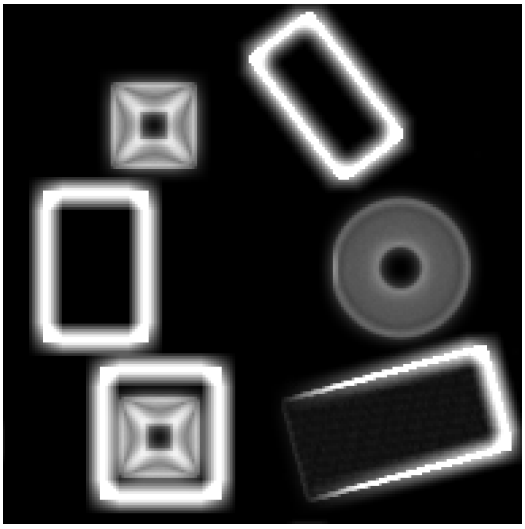


Figure 8.3: Full walkability score.

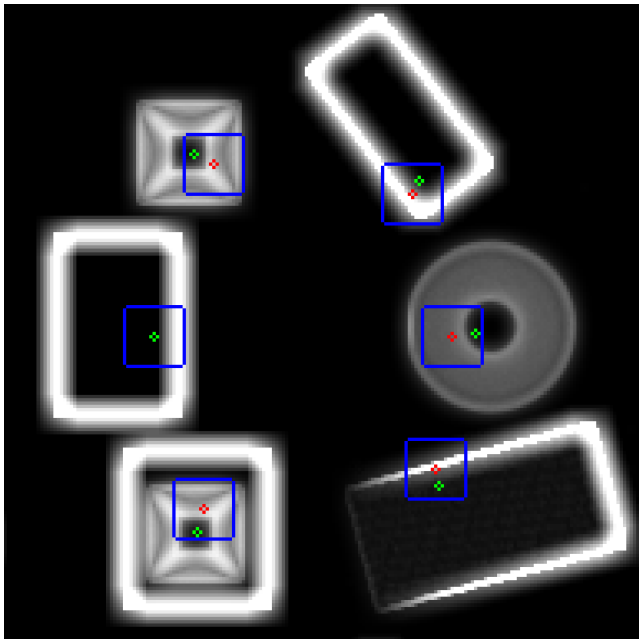


Figure 8.4: Walkability score optimisation test.

shows how the nominal position is place 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 to of the pillar and at the bottom of the hole.

The flat surfaces on the pillar an in the hole are large enough to not be

rejected by the edge proximity score, however, if the inclines sere 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.

8.5 Walking

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.

When presenting testing data, a snapshot of the 3D simulation environment will be shown, with the body and foot paths traced. Additionally key data points are also graphed, however only the height of a single foot will be graphed, this is to maintain clarity.

8.5.1 Flat Plane

The flat plane test aims to quantify nominal body height oscillations and to provide a baseline to compare subsequent test to. Figure 8.5 and 8.6 shows the test results of walking over a simple flat plane, as expected the the step sizes and arcs are uniform with no optimisation necessary.

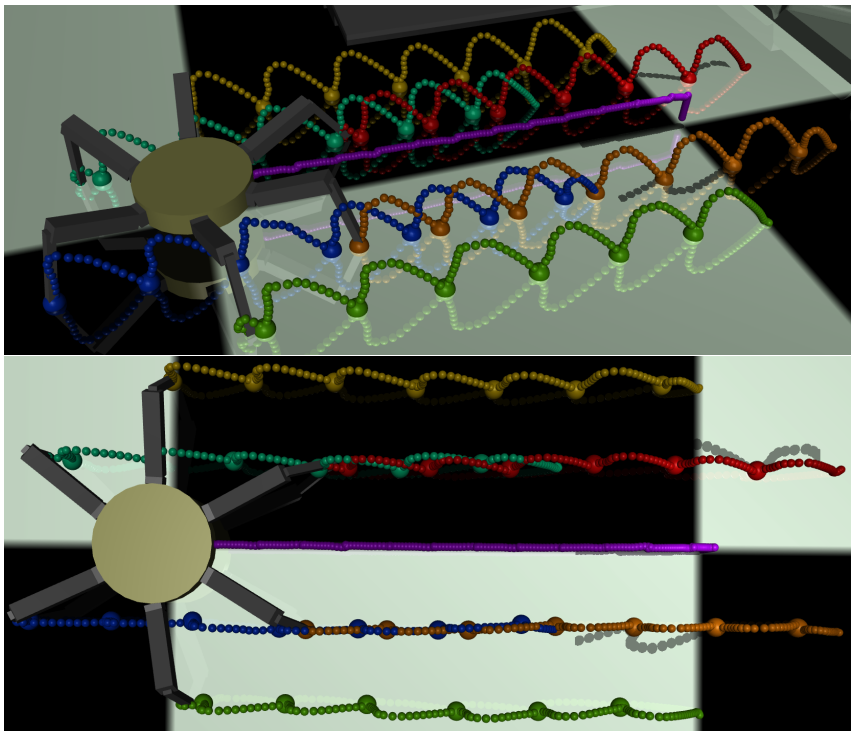


Figure 8.5: Flat plane walk test

From figure 8.6 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

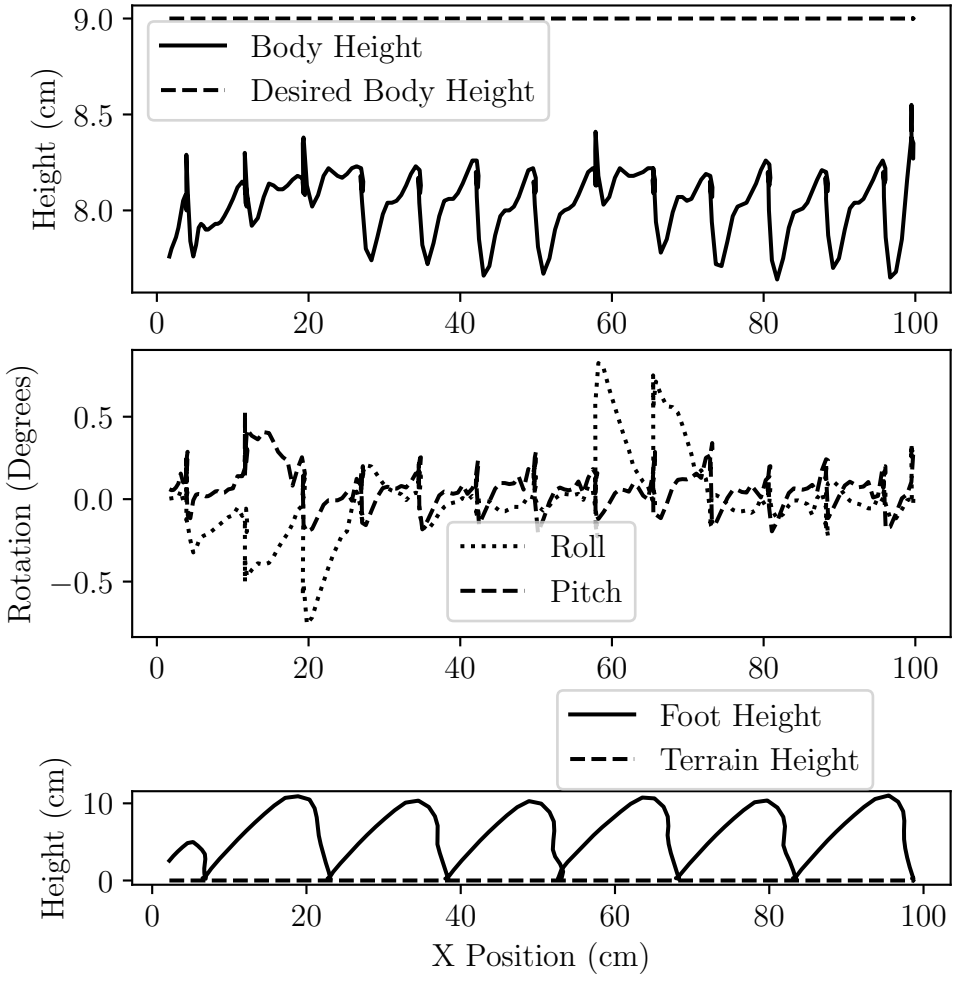


Figure 8.6: Body height (top). Body tilt (center). Position of one foot (bottom)

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.

8.5.2 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 8.7 and the data plots in figure 8.8.

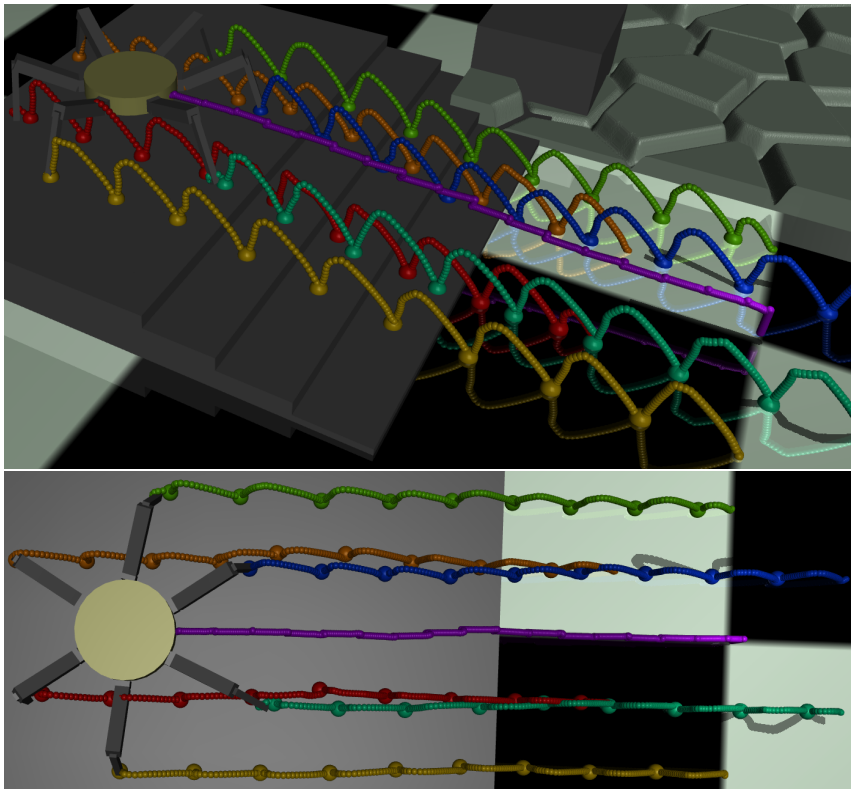


Figure 8.7: Stairs walk test

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 8.8 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 6.3.

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.

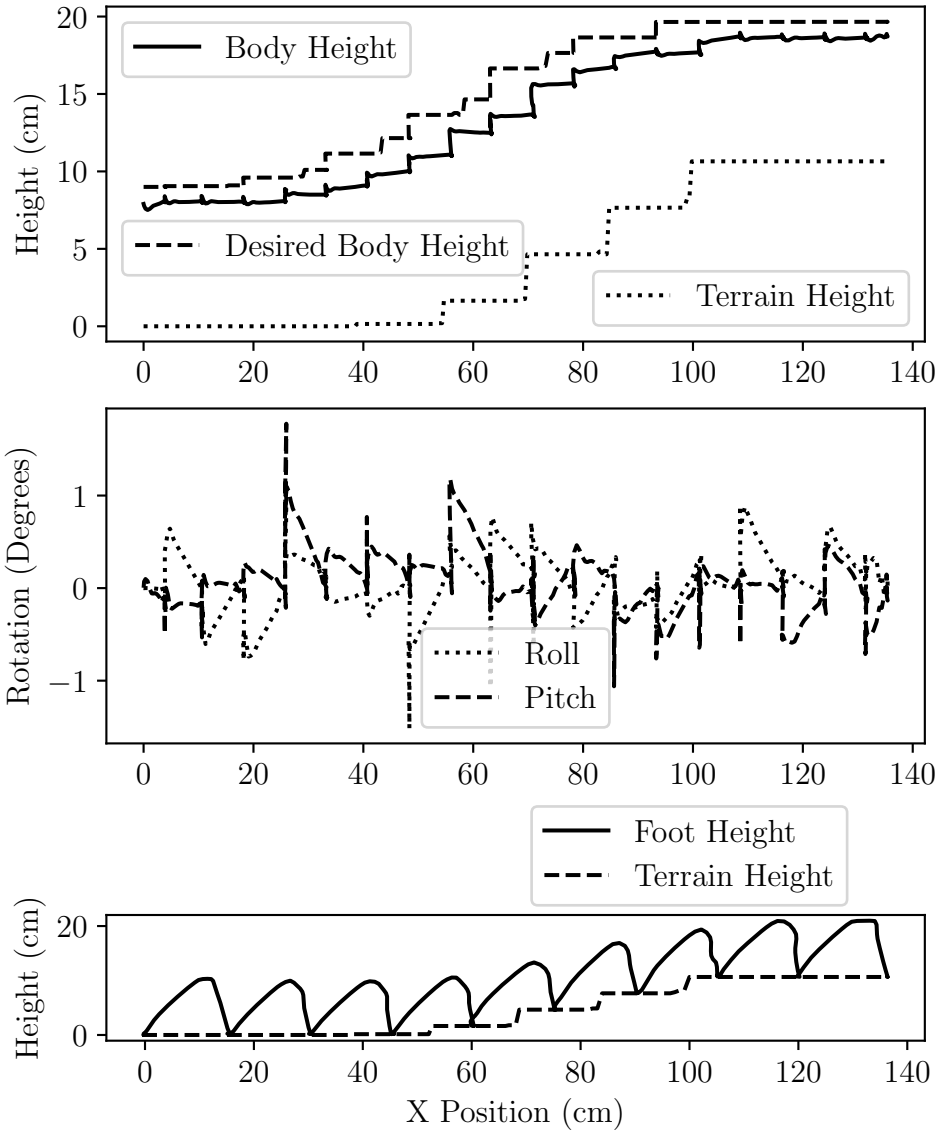


Figure 8.8: Body height (top). Body tilt (center). Position of one foot (bottom)

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.

8.5.3 Organic

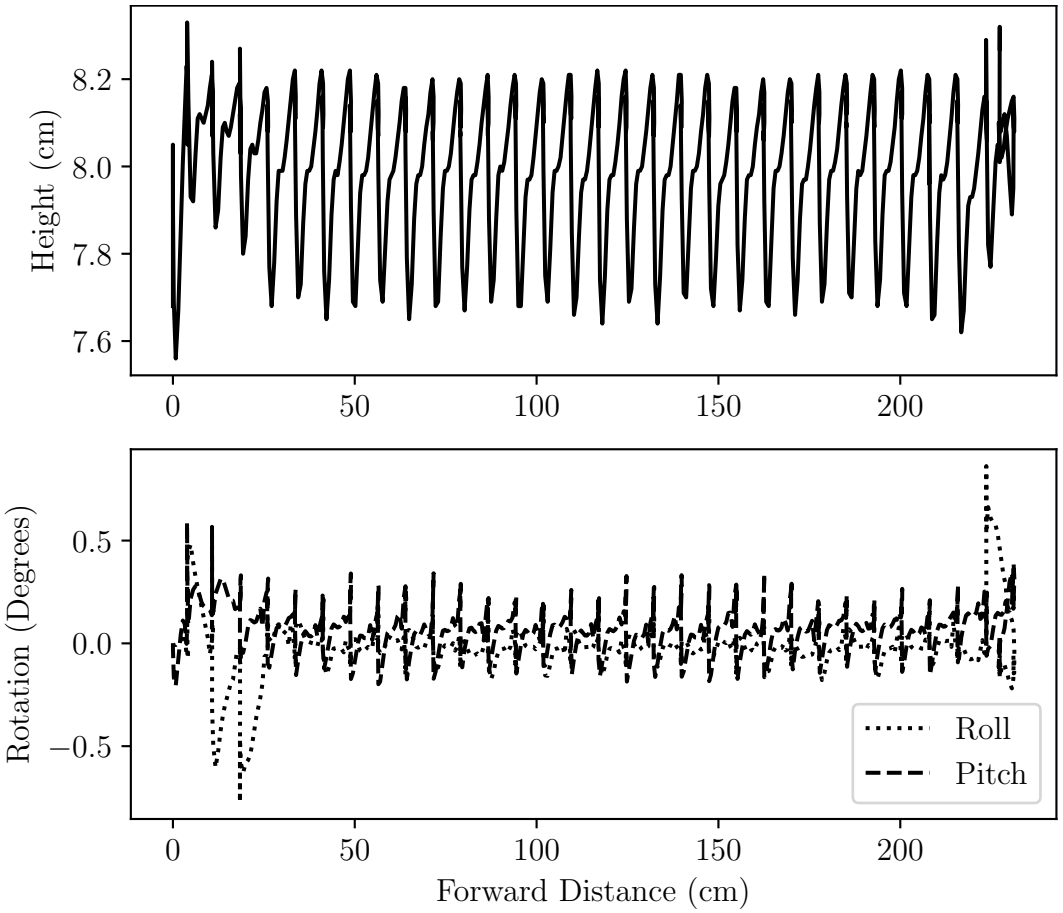


Figure 8.9: Body Data

Chapter 9

Conclusions

Appendix A

Mathematical proofs

A.1 Euler's equation

Euler's equation gives the relationship between the trigonometric functions and the complex exponential function.

$$e^{i\theta} = \cos\theta + i\sin\theta \tag{A.1}$$

Inserting $\theta = \pi$ in (A.1) results in Euler's identity

$$e^{i\pi} + 1 = 0 \tag{A.2}$$

A.2 Navier Stokes equation

The Navier–Stokes equations mathematically express momentum balance and conservation of mass for Newtonian fluids. Navier-Stokes equations using tensor notation:

$$\frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x_j}\left[\rho u_j\right] = 0 \tag{A.3a}$$

$$\frac{\partial}{\partial t}\left(\rho u_i\right) + \frac{\partial}{\partial x_j}\left[\rho u_i u_j + p\delta_{ij} - \tau_{ji}\right] = 0, \quad i = 1, 2, 3 \tag{A.3b}$$

$$\frac{\partial}{\partial t}\left(\rho e_0\right) + \frac{\partial}{\partial x_j}\left[\rho u_j e_0 + u_j p + q_j - u_i \tau_{ij}\right] = 0 \tag{A.3c}$$

Appendix B

Experimental results

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.

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.

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