Balancing and Locomotion of a Hexapod Robot Traversing Uneven Terrain

Jonatan Ekelund

LUND UNIVERSITY

Department of Automatic Control

MSc Thesis
TFRT-6047
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Printed in Sweden by Tryckeriet i E-huset
Lund 2018

Abstract

This master thesis project has developed a method for a six-legged robot to traverse uneven terrains. From a previous project, the robot already possessed basic locomotion which had to be extensively redesigned. Several new subsystems have been introduced, such as a continuous balancing system, a system estimating the slope of the ground the robot is walking on, and method for detecting when the robot's legs hits the ground.

This project designed a system that lets the robot balance its body as it moved using an inertial measurement unit. Several different controllers have been tested and compared. A lot of work has gone into gathering data and identifying how different controllers and sampling rates help the system.

To move across uneven terrain it is important to estimate the tilt of the ground that the robot is moving on. This estimate has been made by integrating the angles from the inertial measurement unit. However, a superior solution using distance measuring sensors has been developed in a simulated environment. The distance measuring sensors also made it possible to control the height of the robot more accurately than before.

A large part of the project has been spent on testing and comparing different methods of detecting when the legs of the robot hit the ground. No satisfactory solution was found using buttons or force sensitive resistors. Instead the legs' movement deviations were used to detect when a leg touched the ground. This was then successfully implemented in simulation.

Several new features, unrelated to the balancing but improving the movements of the hexapod, have been developed and implemented.

Also, a mathematical model for the maximum speed of hexapod walking on sloping ground has been designed.

Acknowledgements

I would like to thank Combine Control Systems and my supervisor, Jasper Gundersen, for making this thesis possible and teaching me so much about project planning, system design and problem solving. I also want to thank Andreas Tågerud, Sara Gustafzelius and Simon Yngve for all the support and help throughout the project.

I want to thank the Department of Automatic control at Lund University for providing me with the necessary software tools. I especially want to thank my supervisor at the department, Prof. Anders Robertsson, for all the late evening meetings, providing support and guiding me through this project.

Contents

1. Introduction **9**
- 1.1 Learning outcomes . . . 10
- 1.2 Resources . . . 11
- 1.3 Previous Work . . . 11

2. Theory **13**
- 2.1 Control Theory . . . 13
- 2.2 Programming . . . 14
- 2.3 Hexapod Locomotion on flat ground . . . 18
- 2.4 Basics locomotion of balancing while moving over angled and uneven terrain . . . 22
- 2.5 Hardware . . . 25

3. Method **31**
- 3.1 Programming . . . 31
- 3.2 Hexapod Locomotion on Uneven Terrain and Angled Surfaces 37
- 3.3 Simulation . . . 52

4. Result **54**
- 4.1 Balance . . . 54
- 4.2 Reality and Simulation Comparison . . . 64
- 4.3 Ground Angle Approximation . . . 74
- 4.4 Step impact detection . . . 77
- 4.5 Speed limitations . . . 78
- 4.6 Realtime analysis . . . 81

5. Discussion and conclusions **83**
- 5.1 Balance . . . 83
- 5.2 Ground Angle Approximation . . . 84
- 5.3 Step impact detection . . . 85
- 5.4 Speed limitations . . . 86
- 5.5 Real-time analysis . . . 86
- 5.6 Simulation . . . 88
- 5.7 Further Work . . . 89

Bibliography **90**

1

Introduction

Today, robots are an integral part of our society. Not only do we find robots in factories and warehouses, but robots also go to previously unreachable and dangerous places. Drones fly over volcanoes and radioactive hazard zones and the rover Curiosity is still working its way across the surface of Mars.

At Combine Control Systems (CCS) in Lund two master thesis projects have worked with the creation and development on a hexapod robot. The first master thesis project focused mostly on assembling the robot, but also built a control system for locomotion of the robot and rudimentary balancing. The second project installed a RGB-D camera and force sensors and created a system for the robot to move autonomously using path-planning. Both these projects focused only on making it possible for the Hexapod to move across flat ground. This project goal is to make it possible for the hexapod robot to walk across uneven terrain and obstacles while keeping its body horizontal.

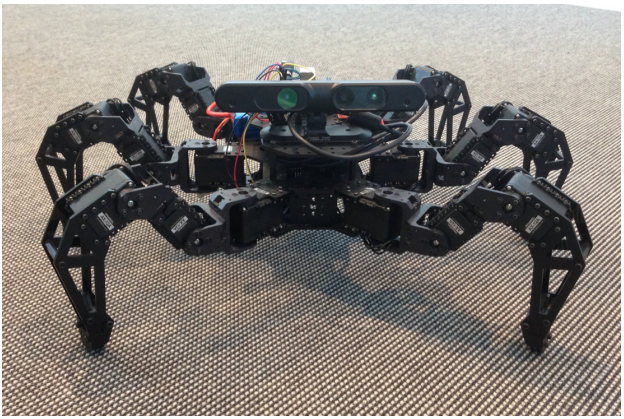

Figure 1.1 An image of the real Hexapod. [Malmros and Eriksson, 2016]

Goals and Problem Formulation

The goal of this project is to develop a control system that holds the hexapod's body horizontal while moving over uneven surfaces.

The hexapod is already able to balance in a very limited manner, being able to keep its body horizontal when standing still only if the surface is flat. Using several sensors the hexapod knows how its body is angled and can compensate for it by lifting and lowering its legs. The first objective of this project was to redesign the system so that the hexapod can walk and balance at the same time.

To balance while walking the Hexapod also needs to detect when its feet hit the ground. One function inherited from a previous project that could help with this is two force sensors installed on the two front legs. The sensors are used to stop the hexapod from walking off edges. If installed on all 6 legs the sensors could instead be used to detect when the legs touch the ground. This is an improvement recommended by a previous project.

To move the in parallel with the ground while keeping the of the hexapod horizontal one must estimate the angle of the surface under the hexapod. There are several different ways to do this, such as integrating the angle of the body or adding distance measuring sensors to the hexapod.

1.1 Learning outcomes

This master thesis project has given a deeper understanding of what it means to apply control theory to real world applications. A lot of time has been

spent trying to work around or improve hardware which in turn has given a lot of insights in what it means to plan projects and how to go about solving problems efficiently.

CCS uses model-based design extensively and a large part of this project has been to learn and work with model-based design.

1.2 Resources

CSS has provided all components and tools for the project. Workspace has been available at both LTH and CCS. The computer used has been my own and all MATLAB/Simulink and V-REP software used has been provided through LTH:s student licenses.

1.3 Previous Work

1.3.1 Thilderkvist and Svensson, 2015

Dan Thilderkvist and Sebastian Svensson were the first to work with the hexapod robot. They were the ones that constructed and programmed the hexapod's basic locomotion. They used the program V-rep extensively in order to simulate and test the robot before assembling the real robot and later to test their programming without risking damage to the hexapod equipment. To use V-REP, they first had to construct a detailed model using CAD-software.

The project resulted in an assembled hexapod robot, able to walk lifting one, two or three legs at a time dependent on the speed set by a controller. The hexapod had two computers on board, 18 motors, an inertial measurement unit (IMU) and a Bluetooth module to communicate with a hand-held controller. They also implemented a basic system enabling the hexapod to balance its body in a horizontal position while standing still.

1.3.2 Malmros and Eriksson 2016

Using the work of Thilderkvist and Svensson as a foundation Marcus Malmros and Amanda Eriksson continued the development of the hexapod, focusing on making the hexapod more autonomous.

To be able to scan the environment they installed an a RGB-D camera. Using the camera as well as the orientation of the hexapod they could use their data to make a grid representing the environment around the hexapod.

They then used an available path-planning algorithm, used by a previous master thesis project by S. Gustafzelius, to plan a route for the hexapod. The

route was then translated into specific instructions of speed and rotation for the hexapod.

Just like the previous project this project also used V-REP to simulate the hexapod. However, as a result when they moved their programming from the simulation run on their computer to the computer on the hexapod they realized that their model needed more processing power than the hexapod could provide.

1.3.3 Palm, 2017

Palm continued developing the work of Malmros and Eriksson, focusing on redesigning the system so that the path-planning and image analysis could be run without interfering with the locomotion of the hexapod.

Palm's project and this project ran in parallel. In order to ensure compatibility, the two projects have worked close to each other even though they focus on different aspects of the hexapod. Therefore some similarities in both theses could be expected.

Palm installed an additional computer on the hexapod, a Raspberry pi 3, and separated Malmros and Eriksson's system into three parts. The system now ran on the original BeagleBone Black, a Raspberry pi 3 and Palm's own laptop, this way the heavy work could be done on a more powerful computer.

Palm developed the path-planning designed by Malmros and Eriksson further, making it possible for a path to be set moving up or down. This way future projects could combine this project with that of Palm to create an autonomous hexapod able to traverse difficult terrain.

2

Theory

2.1 Control Theory

In this thesis there will be a lot of discussion regarding control systems. A control system adjusts the input of a process depending on the output.

There are many kinds of controllers but in this project the focus has been on P-, I- and D-controllers, also called proportional-, integral- and derivative-controllers. This section contains a brief explanation of how each of those work.

A P-controller multiplies the difference of the reference output and the actual output with a constant value and then adds it to the input. [Årzén, 2014] The calculation just described is shown in equation 2.1.

$$P = K(y_{ref} - y) \tag{2.1}$$

P is the value added to the input, K is a constant, y_{ref} is the value we want the output to be, called the reference and y is the actual output. So the difference between the value we want to have and the value we actually is calculated by $y_{ref} - y$ and then multiplied with a constant K. By choosing a high or a low K we can control how aggressively a system reacts to having the wrong output. If K is large, any change between y_{ref} and y will cause a large change in the input of the system and if K is small only a fraction of the difference between y_{ref} and y will be added to the input.

An I-controller integrates the difference between the reference value and the output and adds it to the input of the process. [Årzén, 2014] Equation 2.2 shows the calculations of an I-controller.

$$I = K/T_i \int_0^t (y_{ref} - y)ds \tag{2.2}$$

I is the value added to the input, K is a constant, T_i is a constant called the time constant of the integral, t is the current time, y_{ref} is the value we want the output to be, called the reference and y is the actual output.

By changing T_i it is possible to control how fast the integration will act. A small T_i will quickly add up errors and correct for them but it is also possible for the integration to become to add up too much and cause the system to correct too much. A large T_i makes the system slower to correct for errors but it won't cause problems by making the system overshoot it's reference value.

A D-controller estimates the time derivative of the output and subtracts it from the input.[Årzén, 2014] The calculations of a D-controller are shown in equation 2.1.

$$D = KT_d * d\frac{y_{ref} - y}{dt} \tag{2.3}$$

D is the value added to the input, K is a constant, T_d is a constant called the time constant of the derivative, y_{ref} is the value we want the output to be, called the reference and y is the actual output.

By changing T_d it is possible to control the rate at which the change of y is compensated for. In reality one can only calculate the derivative of $y_{ref} - y$ by comparing the latest value and the values before that one. By using several old values and calculating the mean change it is possible to make the system less susceptible to disturbances such as noise but the system also becomes slower to react.

The different controller types described are often combined. A PD-controller is a P-controller plus a D-controller, it is simply a sum of the two parts.

2.2 Programming

This section contains an introduction to the tools, programs and languages used in this project.

This project has worked mainly in Simulink and MATLAB but has also had to include functions implemented in C by using Simulinks own C integration support tools. The different methods are explained in subsection 2.2.2 and 2.2.3.

To deploy the compiled model on the hexapods own computer, a BeagleBone Black, the Simulink model had to be translated into C code; Which is done

using Simulink's own C code generation function explained later in subsection 2.2.4.

Using a simulation program called V-REP it is possible to run simulations of the hexapod robot, speeding up development and enabling more support for fixing errors. How Simulink and V-REP communicates is explained further in subsection 2.2.5.

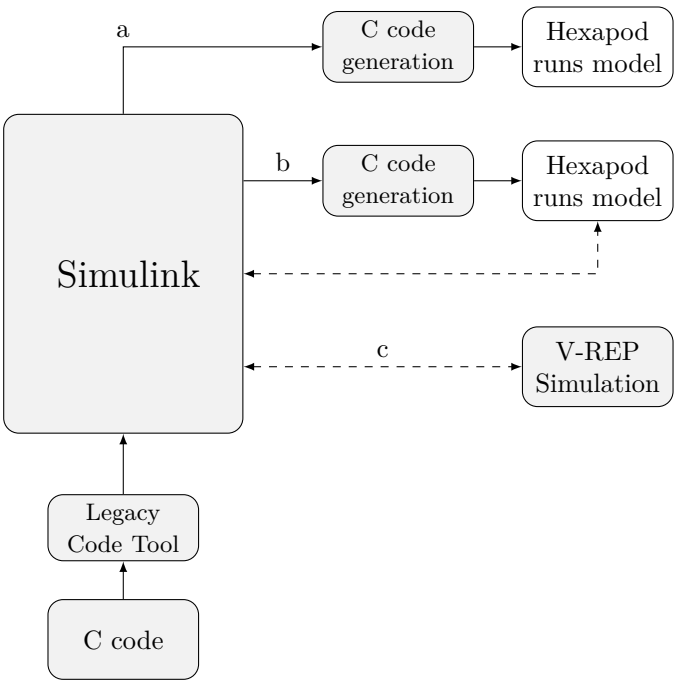

Figure 2.1 A representation of the different ways the Simulink code could be run, grey signifies everything that is located on the users computer, white signifies everything that happens on the hexapod itself and dotted lines shows communication. (a) shows the Simulink model being translated into C code by Simulink's C code generation tool and then deployed on to the hexapod. (b) shows the Simulink model being run in external mode, the model is still compiled and translated into C code that is then deployed on the hexapod, however after deploying there is communication going on between the external computer and the hexapod. (c) shows the Simulink model being compiled and run on the user's computer, the running Simulink model then communicates with the V-REP simulation in order to send new leg positions and gather sensor data.

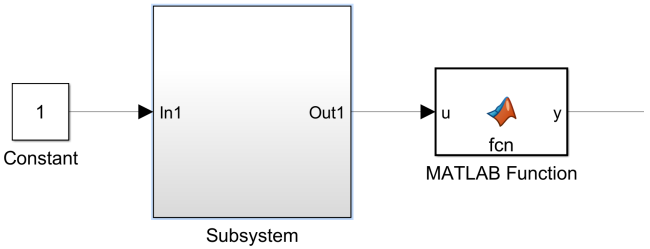

Figure 2.2 An example showing a constant value being sent to a subsystem (which could contain several different blocks and subsystems) and then the output from that subsystem being sent to a MATLAB function.

2.2.1 MATLAB and Simulink

MATLAB is a multi-paradigm numerical computing environment. MATLAB can be used for everything from plotting functions to implementing algorithms and running complex programs.

Simulink is a graphical extension of MATLAB used for modeling and simulation of systems. Simulink represents systems as a block diagram, making it very easy to understand the flow of dynamical systems by just looking at it. A block in Simulink could be anything from a MATLAB function to a subsystem that contains even more blocks, this way systems can be nested in each other and if something does not work as intended it is easy to isolate it and search it for errors. Between the blocks inputs and outputs run as wires. In following texts the type of Simulink block will be specified for clarity but it is important to understand that a block could be many different things.

2.2.2 Legacy Code Tool

The Legacy Code Tool is a function of Simulink that allows the integration of C and C++ code into Simulink models. The tool transforms the C code into a C MEX S-function that can be used in Simulink models.

The Legacy Code Tool is not the only method to do this in MATLAB and Simulink and it has some limitations. However, it is very easy to use and has worked well for earlier projects as well as this one.

In order to use the Legacy Code Tool one uses MATLAB code to specify the files used and which functions should be called at startup, clean up and so on. It is also possible to specify expected inputs and outputs.

2.2.3 S-function Builder

The S-function Builder is another method to incorporate C code in Simulink, transforming it into a C MEX S-function. It does the same thing as the Legacy Code Tool but instead of specifying everything with MATLAB-code there is a GUI interface. It is possible to write your C-code directly into the S-function Builder.

2.2.4 C Code Generation

To transform a Simulink model into C code there are two compilers included in Simulink, called the Simulink Coder and the Embedded Coder. The Simulink Coder generates ANSI C and C++ code from Simulink models and the Embedded Coder makes it possible to customize and optimize generated code.

For MATLAB there is instead the MATLAB Coder but this works only for MATLAB code. When generating C-code from MATLAB code included in a Simulink model one should be using the Simulink Coder instead.

The embedded coder has several add-ons which are necessary so make the generated code compatible and optimized for different types of systems and hardware. This project uses Embedded Coder Support Package for BeagleBone Black Hardware, Embedded Coder Support Package for ARM Cortex-A Processors, MATLAB Support Package for BeagleBone Black Hardware and Simulink Support Package for Beagleboard hardware(R2012a).

The workflow of the three different compilers are visualised in figure 2.3.

2.2.5 V-REP

V-REP is a simulation program specialized for robotics developed by Coppelia Robotics. It has extensive support for controlling models or objects using an embedded script, a ROS node, a remote API client or a custom built solution. This way controllers can be written in a wide variety of programming languages.

V-REP is intended mainly for fast prototyping and algorithm verification but could be used from everything from factory automation simulations to "safety double-checking".

The previous projects have used V-REP during development which is why it is used in this project.

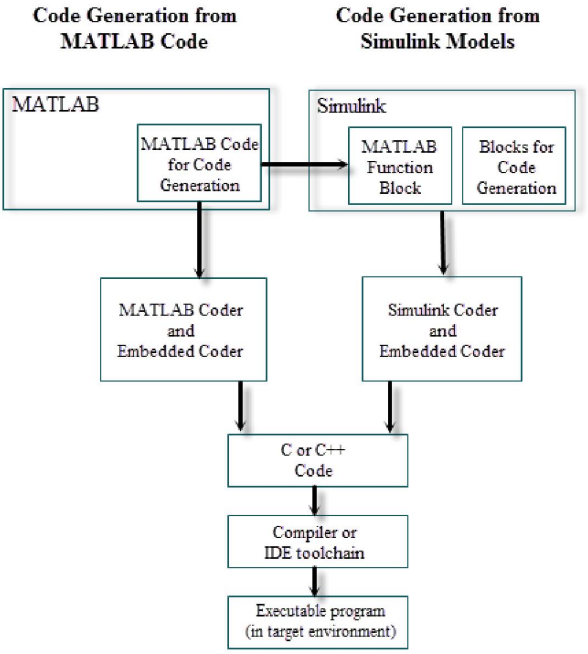

Figure 2.3 The workflow of the different compilers in Simulink and MATLAB [*Embedded Coder Manual for Matlab and Simulink*]

2.3 Hexapod Locomotion on flat ground

When a human walks, it is not the motion of putting one leg in front of the other that makes one move forward; It is the leg that is still on the ground that is pushing the body forward, the leg up front is only to make sure one does not just fall flat.

The same goes for the hexapod, it pushes its body forward by moving its legs backwards (relative to its body, they stay still on the ground). The hexapod has six legs and can lift up to three at a time and still keep its balance. Since the legs of the Hexapod are of finite length it is important for the Hexapod to takes steps as to avoid having legs reach their end point and being dragged behind.

When a leg takes a step it moves in a trajectory that resembles half of an ellipse, shown in figure 2.4. This was a choice made by Thilderkvist and Svensson [Thilderkvist and Svensson, 2015] as the the start and end of every

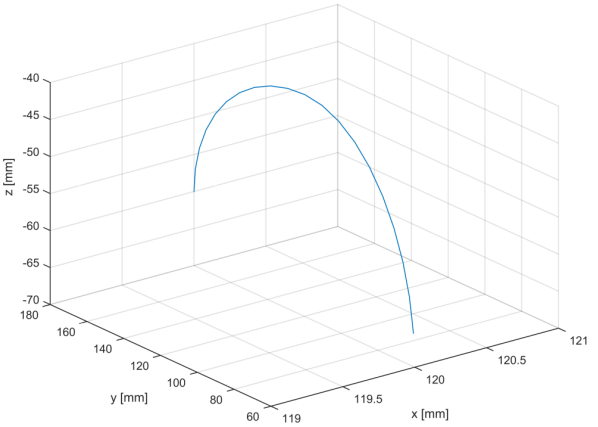

Figure 2.4 The step movement as implemented by Thilderkvist and Svensson. The leg is moving from (120, 60, -70) to (120, 180, -60). [Thilderkvist and Svensson, 2015]

step would be a vertical movement, making sure that the leg is elevated before it begins to move in any other direction and avoids potential obstacles.

The speed of the hexapod is determined by how fast it pushes itself forward. A faster speed makes it harder for the legs to keep up, putting demands on the steps the hexapod takes. There are only two parameters that one can adjust to insure the legs do not fall behind, the speed of the steps and the number of steps. The Hexapod can not move more than three legs at a time so there is an upper limit. The speed with which the legs move forward are in this project dependent on how fast one can move the legs without affecting the balance of the hexapod, this will be discussed further in the next chapter. It is also important to note that there is a limit to how long a step can be, the legs are of finite length.

The limit of how fast the hexapod can move is thus dependent on how fast it moves its legs forward.

2.3.1 Motion Control on flat ground as implemented by Thilderkvistand Svensson, 2015

Illustrated in figure 2.3.1 is the procedure for calculating the movement of the legs when moving the hexapod without any kind of control system for balance or force sensors, as implemented by Thilderkvist and Svensson [Thilderkvist and Svensson, 2015], the first project to work with the hexapod. The following is an explanation of the procedure based wholly on code and systems

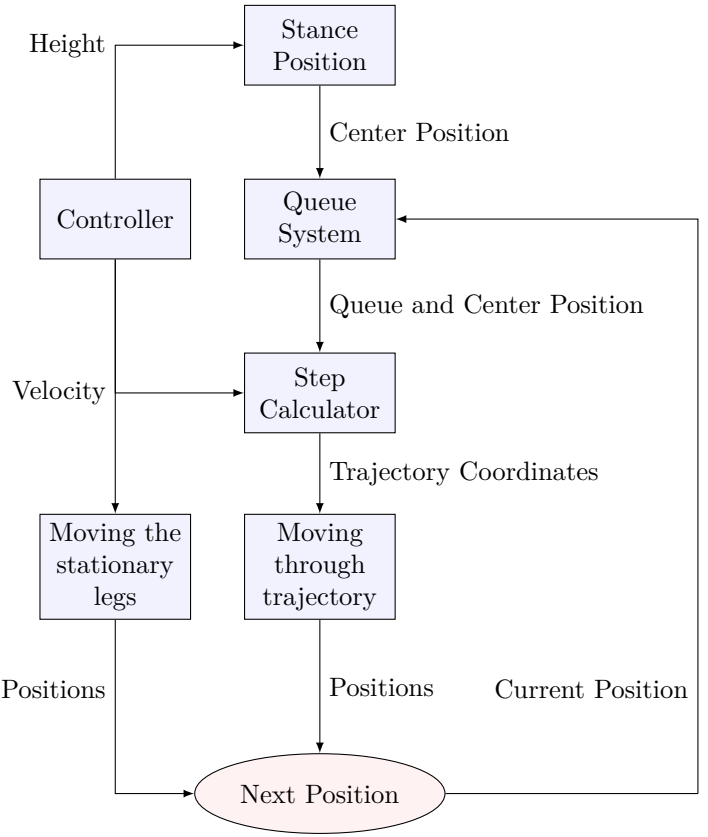

Figure 2.5 A representation of the procedure of calculating steps and movement of the hexapod.

inherited from previous projects.

To begin, the *center position* of the legs is calculated. This is done in the Simulink block *Stance Position*. The *center position* of the legs are the positions where the legs should be standing if the hexapod is standing still. The *center position* of the legs only change if the user changes the height of the hexapod. There is no special quality of the *center position*, it is just a chosen default position.

The *center position* are sent to another Simulink block called *Queue System* and used as a reference to calculate which leg is furthest back. *Queue System* then makes a *queue* containing the leg ID numbers ordered by how far away they are from where they would land if they took a step. This way no legs get

"left behind" as the hexapod moves. The *queue* as well as the *center position* get sent onward to the *Step Calculator*.

From the *controller* comes a velocity in two coordinates. The hexapod can move forwards, backwards and sideways. The rotation of the hexapod can also be changed by the controller. The *controller* can be the actual controller held in the hands of the human user or a planning algorithm as implemented by Malmros and Eriksson[Malmros and Eriksson, 2016].

The *Step Calculator* uses the *queue* from *Queue System* to select which legs should move next. Dependent on the velocity the hexapod can move one to three legs at a time. When a leg is chosen to take a step a flag is set to indicate that it is in the air. Next a *goal position* for the leg is calculated using the *center position* of the leg and the velocity set by the controller. The *center position* is used as a reference point for the step and the velocity determines how far forward the step should move.

When a goal position is set *Step Calculator* creates a path for the leg to follow from its present position to the *goal position* as a list of coordinates, moving in a half-elliptical manner. The list, called the *trajectory*, is then sent onward.

The program then steps through the coordinates of the list until it has reached the end of its *trajectory*. At the end of the trajectory the leg should have reached the ground and it is unflagged as being in the air.

While some legs are being lifted the legs still on the ground move in the opposite direction of the movement of the hexapod, thus pushing the body of the hexapod forward. The speed which the legs on the ground push the body forward is also dependent on the velocity set by the controller as to not move faster than the hexapod lifts its legs.

The new positions of the legs are then translated into angles for each leg's servos using inverse kinematics which is sent through the Arboti-X to the servos controlling the legs. The new leg positions is also sent up to the *Queue System* to calculate a new *queue* with which legs to move next.

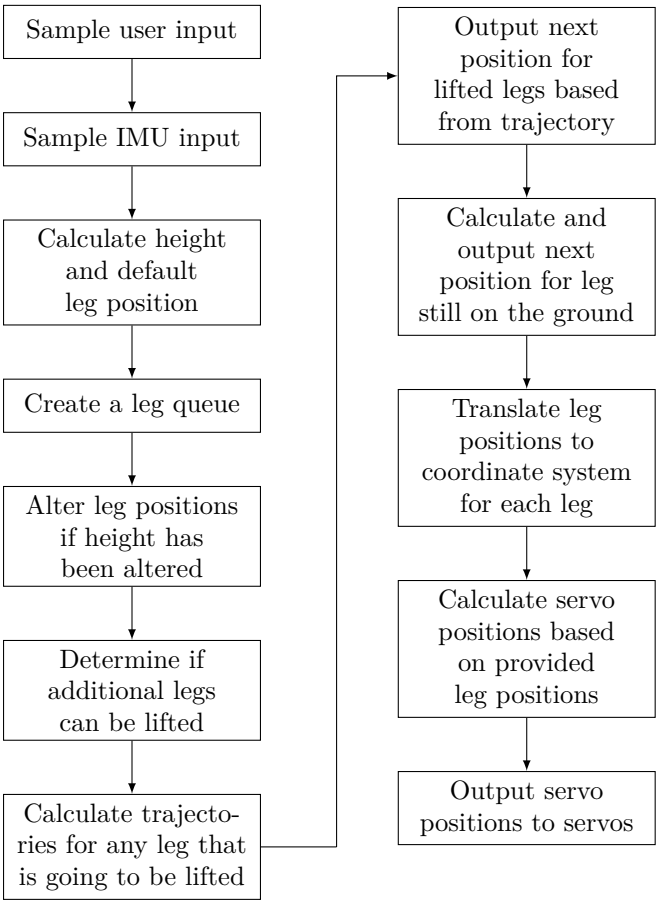

Figure 2.6 A representation of the procedure of calculating steps and movement of the hexapod. Based on a similar image from Thilderkvist and Svensson [Thilderkvist and Svensson, 2015].

2.4 Basics locomotion of balancing while moving over angled and uneven terrain

There are several different problems that need to be solved when making a robot balance while walking. This section functions as a brief introduction to the basics of these problems before discussing the solutions and the work that went into them in the next chapter.

The three main things in this section is balancing, surface estimation and climbing constraints.

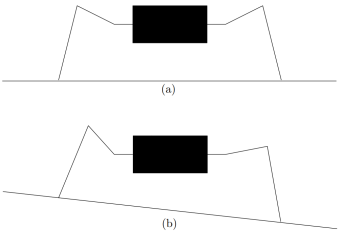

Figure 2.7 (a)Hexapod standing on level ground keeping the main body leveled. (b) Hexapod standing on a slope keeping the main body level by altering the heights of the leg positions. [Thilderkvist and Svensson, 2015]

2.4.1 Balancing

To keep the body of the hexapod horizontal there had to be some way to measure the angle of the body and adjust it, a so called angle control. Thilderkvist and Svensson [Thilderkvist and Svensson, 2015] began work on this kind of control system and implemented a basic controller.

To measure the position and angle of the Hexapod an inertial measurement unit (IMU) was installed on the hexapod. The IMU, a Sparkfun 9 Degrees of Freedom MPU-9150, contains an accelerometer, gyroscope and magnetometer [*Inertial Measurement Unit Data Sheet*]. Thilderkvist and Svensson, 2015 [Thilderkvist and Svensson, 2015], chose the MPU-9150 for its ability to perform sensor fusion with the accelerometer and gyrometer [Thilderkvist and Svensson, 2015]. Later in the project the magnetometer was also added to the sensor fusion.

Thilderkvist and Svensson [Thilderkvist and Svensson, 2015] represented pitch and roll using Euler angles (demonstrated in figure 2.8) and then calculated the height difference that needs to be moved in order for the legs to move onto the ground by using the measurements from the IMU and equation 2.4.

Using equation 2.4 it is possible to use the pitch and roll to calculate where a leg should be in order to compensate for the angle [Thilderkvist and Svensson, 2015].

$$P_z^{t+1} = P_z^t + P_x^t * tan(\alpha) - P_y^t * tan(\beta) \tag{2.4}$$

P_z is the legs end position on the z-axis, the axis parallel to the height of the hexapod, the P_x and P_y are the positions on the two axes x and y, the plane parallel to flat ground. The α and the β are pitch and roll as shown in

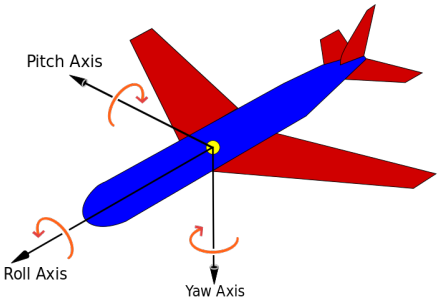

Figure 2.8 An ilustration of yaw, pitch and roll [*Picture ilustrating yaw, pitch and roll*]

figure 2.8. The new P_z^{t+1} is the adjusted height of a leg. To balance we only need to change the legs position height wise as the P_x and P_y are changed to move the legs forward.

This also works as an approximation when moving over an uneven surface and combined with some way of detecting when a leg meets the ground it is possible to always keep the feet on the ground and the Hexapod's body balanced.

2.4.2 Surface estimation

As implemented by Thilderkvist and Svensson the hexapod would walk up a hill just like it would walk on flat ground. When not keeping the body of the hexapod horizontal any direction is always parallel with the ground. However, as one tries to control the angle of the body it gets more complicated. As an example, when the robot moves up a hill the legs in contact with ground can not just push the body directly forward as it would push into the hill; The body needs to move upwards with the tilt of the hill. To do this, an estimate of the grounds angle is necessary.

2.4.3 Climbing constraints

The hexapod's maximum velocity is directly affected by how long time a step takes and how far back it can reach with the legs that are still in the ground. (The locomotion is discussed more thoroughly in section 2.3.1.) The problem is that as the legs move backwards or forward at an angle one must take into consideration the distance the leg will move vertically too. Therefore, it is important that the relationship between the maximum horizontal speed of the hexapod and the angle of the surfaces it is walking on is investigated.

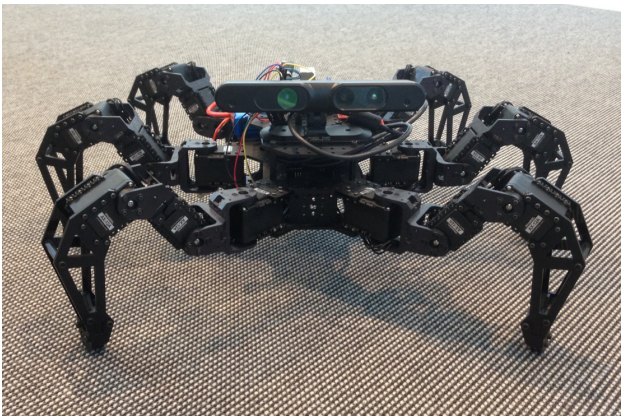

Figure 2.9 An image of the real Hexapod. [Malmros and Eriksson, 2016]

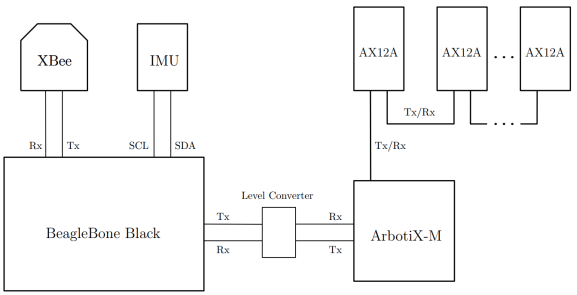

Figure 2.10 An overlay of the communication of the the different hexapod hardware, not including motion nor distance sensors. [Thilderkvist and Svensson, 2015]

2.5 Hardware

In this section we will discuss the different modules and hardware used in the project. Most of the hardware is implemented by a previous project but some of it was designed by this project and will be discussed further in the next chapter.

2.5.1 Force sensors

The previous project [Malmros and Eriksson, 2016](Malmros, M. and Eriksson A, 2016) used two force sensitive resistors on the tips of the front legs of the Hexapod to detect when or if the legs reached ground. This way the hexapod was able to sense if it was about to walk off an edge.

The resistors used were the FSR 400 produced by Interlink Electronics [*Force Sensitive Resistor Data Sheet*]. The resistance of the resistors lowers when compressed, such as the leg pushing against the ground. Without applying any force the resistance of the resistors is about 1MOhm [*Force Sensitive Resistor Data Sheet*], varying somewhat from resistor to resistor. By putting a low voltage over a force sensitive resistor it is possible to measure the amount of of force the resistance is being affected by as the voltage over the resistor changes, an electrical schematic is shown in figure 2.11. The behaviour of the resistance dependent on force is show in graph 2.12.

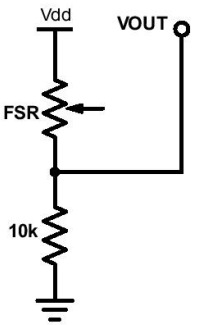

Figure 2.11 The electrical circuit used to measure when the force sensor is pressed [Malmros and Eriksson, 2016]

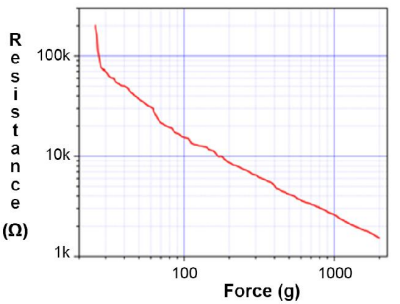

Figure 2.12 The force sensitivite resistors resistance as a function of force [*Force Sensitive Resistor Data Sheet*]

The change of resistance of the force sensitive resistors is dependent on a

air gap inside the resistor [*Force Sensitive Resistor Data Sheet*], bending the resistor or blocking the vent affects if the resistor functions.

Figure 2.13 An image of the sensor [AICamera]

The project also made use of mechanical buttons, specifically the B3F produced by Omron [*Mechanical Button Data Sheet*]. The electrical schematic for the button is seen in figure 2.14. Instead of a change of resistance, like the force sensitive resistors, the button is an open circuit when not pushed and a short circuit when pushed.

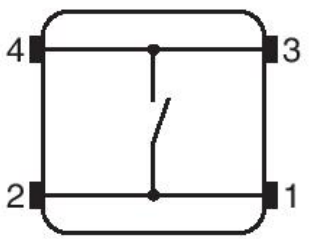

Figure 2.14 The electric scheme of the mechanical button [*Mechanical Button Data Sheet*]

2.5.2 Motors

The motors used on the hexapod are the Dynamixel AX-12A produced by Robotis. The motors have their own PID control system and a movement resolution 0.29°, [*Servo Data Sheet*].

2.5.3 BeagleBone Black (BBB)

The BeagleBone Black (BBB) is the main computer installed on the hexapod. The BBB was chosen by Thilderkvist and Svensson [Thilderkvist and Svensson, 2015] as the original ArbotiX-M did not have enough processing

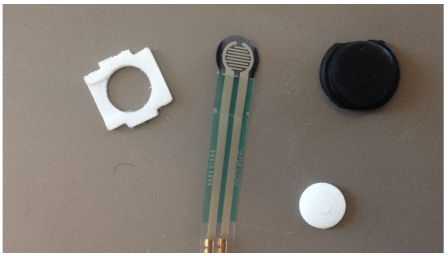

Figure 2.15 The 3D printed force sensor holder (white), the sensor and a rubber nub to increase the friction (black). [Malmros and Eriksson, 2016]

power to run their Simulink model. The ArbotiX-M was kept however since it had support for communication with the hexapod's motors.

The BBB is the main hub connecting all the hexapods sensors and systems. As seen in figure 2.10 connected to the BBB is a XBee, an Inertial Measurement Unit (IMU) and an Arbotix-M. The XBee provides communication with between a hand held controller and the BBB. The IMU measures the angle of the hexapod's body, explained further in subsection 2.5.4.

Not shown in picture 2.10, the force sensors that were used to detect when a leg would step onto the ground are also connected to the BBB using the BBB's analog input and output pins.

When this report describes the Simulink model being translated into C code and deployed on to the hexapod it is to the BBB that the code is sent and then run.

2.5.4 Inertial Measurement Unit (IMU)

The Inertial Measurement Unit (IMU) that is installed on the hexapod is a SparkFun Degrees of Freedom MPU-9150, which contains an accelerometer, a gyroscope and a magnetometer. Thilderkvist and Svensson installed the IMU and chose the MPU-9150 for its ability to perform sensor fusion with the accelerometer and gyrometer [Thilderkvist and Svensson, 2015]. Using already existing code the magnetometer was also added to the sensor fusion. The magnetometer helps with compensating for drift that might occur when only using the accelerometer and gyroscope, but it also slows down the start-up time of the IMU.

2.5.5 Proximity Sensor

In order to approximate the height of the hexapod as well as the angle of the ground beneath it two distance measuring sensors were provided by CCC. The sensors were the VL6180 sensor from Sparkfun [*VL6180 Guide*]. The

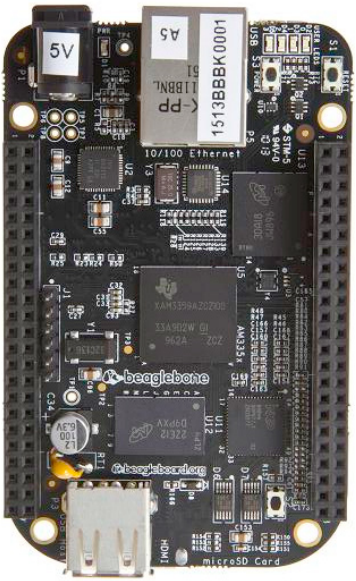

Figure 2.16 A picture of the BeagleBone Black. [*BBB Picture*]

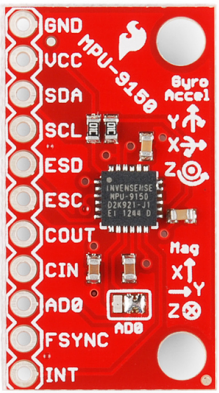

Figure 2.17 A picture of the IMU. [*IMU Picture*]

sensors use I2C communication and has a range of about eighteen centimeters. The sensors use a very precise clock to measure the time it takes light

to travel from the surface they are pointed at and back and are therefore very accurate and not very sensitive to noise. The sensors are not influenced by daylight and worked on most materials. The sensors were not used in the final product however.

3

Method

3.1 Programming

The main tools used in the development of this project have been Simulink and MATLAB, but several other tools and programs have also been used. Previous projects built the control system for the hexapod as a Simulink model, and what they could not do in MATLAB they did in C or C++. This project has been based on this model, however it had to be extensively redesigned. Through the course of this project several programs have had to be implemented using C and Bash, sometimes to replace defective functions implemented by previous projects. The main computer on the hexapod, a BeagleBone Black (BBB), provides a lot of functionality such as I2C communication and analog output and input pins. However, these functions could only be reached by using Bash commands or reading special files inside the BBB's file system which neither Simulink nor MATLAB supported. Bash commands can however be called using C code which then could be added to Simulink using Simulink's Legacy Code Tool. All of this is explained further in the following subsections.

3.1.1 Simulink and MATLAB Code structure

Before reading this section, it is recommended that one familiarizes oneself with the original Motion Control system Implemented by Thilderkvist and Svensson described in Subsection 2.3.1. This section can be considered as an overview of the changes that have been made to the Simulink model throughout this project. As this project resulted in two separate systems, one used to run the simulation and one used for the real hexapod, the differences between these two systems will be pointed out in this section.

The biggest changes to the Simulink Model has been done in one of the lower layers of the Simulink model. The Simulink subsystem called *MainController* takes velocity, rotation, buttons pressed, and in the case of the V-REP model, the distance sensors' height readings as inputs and then outputs the next

position for each leg as coordinates. The coordinates then get transformed into angles at which each motor in each leg should move and then sent onward to either a ArbotiX-M that is connected to the motors in the legs or sent into a V-REP model depending on if you are running the real hexapod or a simulation.

Inside *MainController* there are three Simulink subsystems called *Stance position*, *Queue system* and *Trajectory calculator*, seen in figure 3.3. Each subsystems contains a MATLAB function, called *calcCenterPos*, *LegQueue* and *calcTrajectory* respectively. The *Trajectory calculator* also contains a Simulink subsystem called *Force Sensors* and a MATLAB function called *Regulate IMU*.

Inside the block called *Force sensors* there was six Simulink functions from the add-on package Embedded Coder® Support Package for BeagleBone Black Hardware which were supposed to read the analog input signals received from the force sensors on the legs. As they did not work, they were replaced by C-code programs using the Simulink Legacy Code Tool.

The function *calcTrajectory* in the subsystem *Trajectory calculator* was extensively redesigned to implement the floor angle approximation, balancing control, height control, leg collision control as well as the control system used in the final version of the V-REP model to control the compensation distances for the legs. Several MATLAB functions were also redesigned such as the function *legLimit* that is used to decide how many legs the hexapod would use depending on the speed set by the controller and the function *makeLiftTrajectory* that creates a list with all the coordinates a leg will move through when taking a step.

A cut off controller was added to the MATLAB function called *Regulate IMU* in the model designed for the real hexapod. In the Simulink model designed for the simulation the function *Regulate IMU* was removed. A new function called *calcFloorAngle* was added to the Simulation model, *calcFloorAngle* used the distances received by the distance measuring sensors to estimate the angle of the ground under the hexapod.

calcCenterPos is a MATLAB function inside the Simulink subsystem *Stance Position*. *calcCenterPos* calculates the standard position the legs should be in when the hexapod is standing still, called *centerPosition* in the code; These positions are also used when calculating the final position, a leg should reach when taking a step. As seen in figure 3.3 the *centerPos* is calculated in *Stance Position* and sent onward to *Queue system* and then finally to *Trajectory calculator*. In previous projects, the center position had only been dependent on the height of the hexapod, which in turn was set by the controller. When implementing the floor angle approximation, which was done in the *Trajec-*

tory calculator block, the new *centerPosition* coordinates had to change with the the angle of the surface. The *Trajectory calculator* block got an added output sending the updated floor angle to the *Stance position* block, storing the value in memory between runs as the *Stance position* block always is executed before the *Trajectory Calculator*. The new stance position was calculated using equation 2.4 and function for changing the radius depending on the legs height was changed so that each leg had its own height and radius.

Thilderkvist and Svensson created MATLAB functions that communicate with V-REP the MATLAB to V-REP API. A function called *VrepCom* retrieves the tilt of the hexapod's body and sends it forward to the *Trajectory Calculator*. This was done once every ten executions and since the basis of the hexapods angles differed from the simulation "world" basis the angles were very hard to decipher. The angle basis of the hexapod was changed, the frequency at which the angle was sampled was set to once every execution, several distance sensors were added to the bottom of the hexapod and the API calls to read the distances measured by the sensors were implemented. The distance was then added to the outputs of *VrepCom* and sent forward into *Trajectory calculator* where it was further re-scaled and used to calculate its angle in the before mentioned function *calcFloorAngle*.

3.1.2 C code and Legacy Code Tool

Several of the BBB's functions are not supported by MATLAB nor Simulink. To integrate these functions into the model, programs were written in C and then added to the Simulink model using the Legacy Code Tool that is included in Simulink.

Legacy Code Tool is not the only way to include C code in Simulink. Thilderkvist and Svensson used S-function Builder to incorporate their C programs into the model. However, as Malmros and Eriksson used the Legacy Code Tool and as it seemed much easier to use and understand for future projects it was used also for this project.

Malmros and Eriksson used a Simulink add-on called Embedded Coder® Support Package for BeagleBone Black Hardware to gain access to the hardware functionality of the BBB. When trying to read the analog pins used in the step control using the included Simulink function from the BBB hardware add-on the function stopped working if the hexapod was restarted, even though everything else worked. A C program was written and tested first directly on the BBB and then copied and integrated into the Simulink model. There was some trouble with the Simulink compiler not finding the files necessary to integrate the C code so the code was put in a separate folder by itself.

This project includes the implementation of a C program that communicates with distance measuring sensors via an I2C bus. The program was not used in the final version of the Simulink model since the sensors did not support several sensors being connected on a single I2C bus. The C program was however used on the BBB to do tests with the sensor to determine if the sensors could be a viable option for future projects.

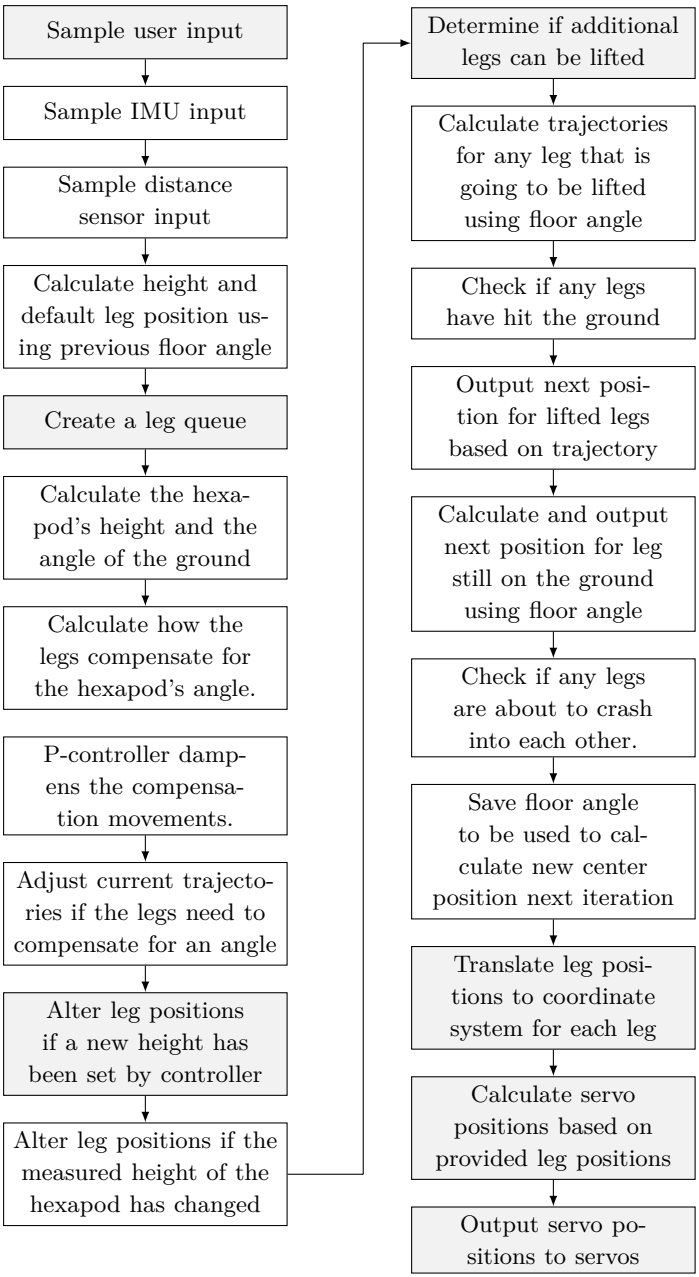

Figure 3.1 A representation of the new procedure of calculating steps and movement of the hexapod. New or redesigned operations are shown in white and the unchanged are shown in grey.

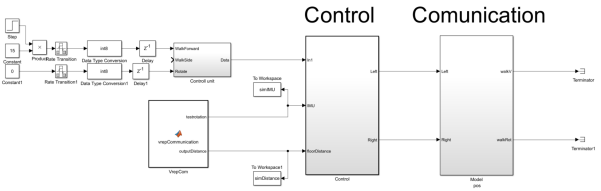

Figure 3.2 The top layer of the final Simulink model used for the simulation.

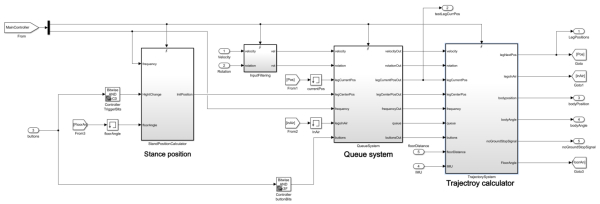

Figure 3.3 The fourth layer of the final Simulink model used for the simulation.

3.2 Hexapod Locomotion on Uneven Terrain and Angled Surfaces

This section describes the work that went into the locomotion and the different control systems of the hexapod. It begins describing the timeline of the project and then continues to describe each system in more detail in the following sections.

3.2.1 Project Timeline

The work began with replacing the worn out pressure sensitive resistors used in a previous project, running a few test to check that everything was still working and then continue to install resistors on the remaining legs. The code used to read the analog pins of the BBB connected to the resistors did not work properly and was replaced by a C script.

The first balancing control was implemented making the legs standing on the ground balance while the hexapod was moving.

There were constant problems with the resistors. Some of the resistors had been bent or damaged when running the hexapod and were replaced. A lot of work went into testing the sensors and looking for errors in the software or the circuitry connecting the resistors.

As explained in section 2.4.2, to move up a hill while keeping the hexapod's body horizontal an estimate of the angle of the hill must be made. By integrating the angles that the hexapod adjusted for it was possible to determine the total angle at which it had been moved and as the hexapod always began parallel to the surface it was standing on, it could work as an estimate of the tilt of the ground.

To further develop the balancing system control parameters were changed, and tests were made. The lifted legs' trajectory was adjusted in mid-air as the hexapod moved so that the legs would still find ground even if the angle of the ground changed.

The hexapod's balance control was coming together but the impact detection was still far from satisfactory. The resistors did not trigger as they should have. In order to compensate for their behaviour, it was decided to try to change the locomotion of the hexapod. First the hexapod was made to stomp its feet into the ground, this made the sensors react but the vibrations and errors the stomping caused made it an unsatisfactory solution. Next the legs were set to move more slowly at the beginning and end of their trajectory, the extra distance the legs would push down when not sensing any ground was also made slower.

While working at improving the leg movements several resistors had stopped working. The fault was caused by tearing and bending of the thin plastic the sensors were made of, and in an attempt to reinforce the resistors a layer of electrical tape was added around every resistor. A resistor testing procedure was also added to the start up of the hexapod.

The step impact detection was still unsatisfactory, and the resistors were replaced with mechanical buttons. To make it easier for the buttons to trigger once they hit the ground the locomotion of the hexapod was changed further.

By this point in time the project's development of the balance control of the hexapod had surpassed initial expectations. Without the impact detection the hexapod could still not move over uneven ground, but it moved very well across flat ground at an angle. It now became clear that the relationship between the maximum speed of the hexapod and the angle at which it was moving had to be estimated.

To avoid doing heavy calculations to calculate the maximum speed every iteration of the model, it was decided that the maximum speed per angle would be pre-calculated and put into a lookup table before running the model. This way the maximum speed could be checked every iteration without any heavy calculations. The pre-calculations were run in a program written in MATLAB.

The impact detection was causing problems for both the body angle control and the floor angle estimation. To fix this an attempt to use the legs height to control the height of the hexapod was implemented, the height control would help the impact detection as well as keep the body of the hexapod at a steady height. The new height control did not work as well as hoped and it was decided to install distance sensors on the hexapod instead.

A program was written making the BBB communicate with one of the distance sensor over a I2C port. The sensors lacked support for connecting several sensors onto the same I2C bus and therefore the sensors were never installed on the hexapod.

Several attempts were made to use the tilt of the hexapod to help determine when a leg hit the ground. Using the tilt made the system more susceptible to errors and since there was no good way to determine neither the angle of the ground nor the height of the hexapod accurately, the errors would affect the system to an even greater extent than expected.

The time allotted for the project's use of the hexapod was running out at this point in the project and since the task to install the distance measuring sensors had turned out to be much more complicated than first thought it was decided to move the model over to a simulation environment using V-REP.

The first task after moving the project over to a simulation was to improve and redesign the balancing system. It was much faster to run tests inside the simulation and a great number of control systems and parameters were built. It was decided that this project would continue development of the hexapo using a control system that would regulate the distance compensated for instead of the angle of the hexapod.

Four distance measuring sensors were added to the hexapod in the simulation. Each sensor's measured distance was collected from the Simulation to the Simulink model using V-REP's API. The height of the hexapod was then calculated as a mean value of the distances. Using trigonometry the angle of the ground relative to the hexapod was calculated.

The next step was to study how the legs would compensate when hitting the ground. A new impact detection system was designed, using the lifted legs movements to determine if it hit the ground.

3.2.2 Locomotion

The original system was designed for walking on flat ground and most of previous projects' work went into making the movements of the hexapod's legs smooth. Smooth leg movements do not necessarily mean not moving the body of the hexapod.

Several changes have been made to the way the hexapod moves. It is easy to focus on the work that went into controlling and compensating for the hexapod's movements, but a lot of work has gone into the movement of the hexapod; Making the system more stable and easy to control.

Before lifting a leg the whole trajectory of that leg is calculated. Every step has the shape of an arc, resembling half an ellipse. When a leg arrives at the end of its trajectory the leg will continue to move straight down until it detects the ground, or it reaches its maximum and stops the whole movement of the hexapod. Originally it would back away after not finding any ground but since we were using the resistors to sense where the ground would be and not as a way to avoid falling of "cliffs" like the Malmros and Eriksson did.

Two goals guided the work regarding the locomotion throughout the project; To minimize the vibrations and movements of the body and making it possible to detect when the legs hit the ground. Since the force resistors recommended by Malmros and Eriksson were far from functional many attempts were made to change the movements of the hexapod so that the sensors would trigger more easily. The first working alteration was to make the leg move into the ground quicker, this was a sure way to make the force sensitive resistors to react but as the legs hit the ground hard it caused too much vibrations and shaking.

Instead, a higher arc for the steps was designed, this way the feet would have very little vertical movement once they reached the ground. The step was made slower as well as slowing down the final part of the movement where the leg would move straight down after reaching its original trajectory. Several tests were made in order to understand how to minimize the vibrations and at the same time trigger the impact detection. It is important to note that with every new change came several hours of testing, especially since the resistors could stop working in the middle of a test and that many weeks of fruitless work went into trying to find a way to make the resistors work consistently.

As the force sensitive resistors were exchanged for mechanical buttons the main problem became the angle at which the leg hit the ground more than the impact itself. The trajectory was made longer, making the legs move completely vertical at the beginning and end of every step, making the leg slowly set down its foot and then if no ground was detected continue to slowly push down, much slower than before in order to decrease the effect it would have on the system.

When moving over to the simulation the hexapod no longer had any buttons or resistors to take into account. The movement of the legs were made simpler and focus was on minimizing vibrations but not affecting the maximum speed of the hexapod too much. The buffer that was added when working with the buttons was removed. Since the goal was to now detect a leg touching the ground by looking at the compensation of the legs, it was considered a possibility to make the legs quite a bit slower, especially at the end of its trajectory, so that the step impact would become slower and more distinct when compared to normal noise and disturbances. This was not necessary however, as the system managed quite fine with the normal steps.

3.2.3 Balance

As mentioned earlier Thilderkvist and Svensson, 2015 [Thilderkvist and Svensson, 2015] implemented a simple function for balancing. As discussed in Subsection 2.4.1, Equation 2.4 calculates the distance a leg needs to move in order to compensate for the angle of the body. A leg high up on a hill will move up and a leg further down will move down as shown in image 2.7. The hexapod has a IMU that uses several different sensors to determine at which angle the body of the robot is, an angle of zero means that the body is horizontal.

There are three angles that are measured, the pitch, the roll and the yaw, illustrated in Figure 2.8; Only pitch and roll are of importance when trying to keep the body horizontal.

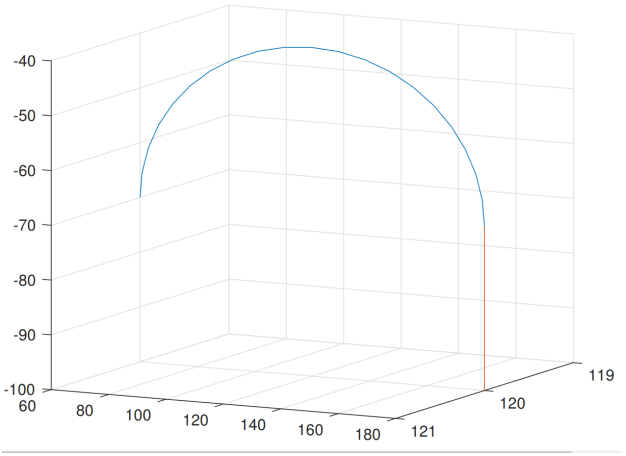

Figure 3.4 The movement of a step as implemented by Malmros and Eriksson. The original trajectory (blue) of the leg movement together with the extended trajectory (red). The start position is (120, 60, -70) and the goal of the ellipse is (120,180,-70). [Malmros and Eriksson, 2016]

So, the balancing system works by measuring the pitch and roll, calculating how each leg should compensate for it and then moving it accordingly. Thilderkvist and Svensson noticed that the compensation became unstable without any damping and found the that the system became stable when one multiplied the angles with a constant of 0.2, i.e. a P-controller.

Thilderkvist and Svensson's implementation was very rudimentary and restricted the hexapod to balance only when standing still on a flat surface. To let the hexapod balance while walking, the balance algorithm had to be integrated into the walking algorithm. While the legs still on the ground pushed forward they had to move up and down, using equation 2.4 and the current position of the leg to calculate how to move. The coordinates of the final destination of every step was also used with equation 2.4 and the change was added to the trajectory of the step; This way the legs moving in the air would not try to stop above or below the ground if the body had to be adjusted while the leg was in flight.

After integrating the balance control with the movement, the step impact detection of the legs became the top priority of the project. The body could keep itself horizontal but as explained in Subsection 3.2.5 the problem of being able to detect when a leg hit the ground was harder to solve than anticipated. Since the impact detection was essential for the hexapod move over uneven terrain it had to take precedence.

When the work of improving the balance control resumed, the first task was to make the system less susceptible noise and to minimize overshoots. Several tests were run using different damping constants but the change either made the system too unstable or too slow. Instead a so called cut-off controller was implemented; The measured angle was not allowed to change faster than 0.5 degrees between sample, making the angle sent onward into the last value ± 0.5 if the new value's change exceeded the limit. This way, small angles could still be corrected quickly, but for a large angle change there was a more linear and controlled correction. The large changes were mainly symptom of the dysfunctional step impact detection. The cut off filter helped make the system more robust and the project moved to improving other parts of the hexapod.

As the hexapod robot was not always available it sometimes became necessary to simulate the hexapod using V-REP. The simulation was much faster to use but since it seemed important to keep the project grounded in reality it was not used as extensively as it could; especially since a lot of errors and problems were caused by hardware. Later in the project the whole control system would be moved over to the simulation environment.

Without buttons or resistors in the simulation the focus shifted to improving the balance system. The plan was to test different ways of implementing the balance control and then use the best way to design a new impact detection system which would use the values from the balance control. Two different control systems were tested, one which regulated the , shown in Figure 3.2.3, and a new control system that would regulate the movements of the legs as they compensated for an angle, shown in Figure 3.2.3. Both systems were originally a P-controller but PI- and PD-controllers were also implemented and tested with different parameters.

It was decided to continue development using the new system that regulated the leg movements instead of the system regulating the angle. Since there were not any substantial differences in performance the new system was chosen as it was controlling a linear system which the angle did not. The type of controller used was a P-controller as it was the fastest and most stable controller and as the system dynamics contained an integrator no static errors would occur.

The simulation made it possible to change the sample rate of the IMU on the fly. Several tests were done to determine the effect it had on the control system as well as other systems.

Using the new more stable and quicker angle control,

several stress tests were made, to study how it would react to sudden changes

in angle and how it would compensate for the legs hitting the ground unexpectedly. The results were then used in the design of a new impact detection system.

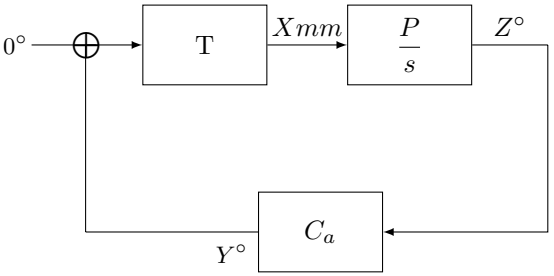

Figure 3.5 A simplified block diagram showing the old system using a controller to regulate the tilt of the hexapod. The diagram shows the angle controller C_a, trigonometric calculations block T, and the system dynamics $\frac{P}{s}$. The process is written as $\frac{P}{s}$ to emphasize the integrator part of the process. From the process we measure the angle of the hexapod's body, shown as Z°, this is send on to the controller C_a that makes alterations to the value dependent on the type of controller and outputs that value as Y°, a P controller for example would multiply the angle with the value K so that $Y = K * Z$. Y° is then compared to the angle we would like the hexapod to have, which is 0°. The comparison is then sent to T which calculates how the legs should move to compensate for the angle of Y°, this is done using the trigonometric functions explained in earlier sections. The distances the legs should move are shown as Xmm and are in mm, Xmm is then sent into the process which moves the legs.

3.2.4 Ground Angle Approximation

As explained in the theory chapter, Subsection 2.4.2, for the hexapod to move at an angle it is necessary to estimate the angle of the surface it moves across. There are several different ways to calculate and measure the angle of the ground.

The first surface estimation implemented was a sum of all the dampened angles sent in to the leg movement control. This worked as the system was run very quickly and could have been a sufficient estimate if the step impact detection would have worked better.

The impact detection had to be prioritized over the ground angle estimation since the bad impact detection affected the hexapod negatively in so many ways.

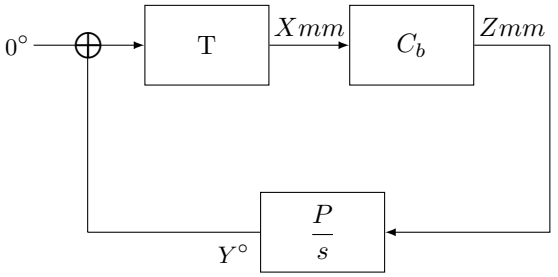

Figure 3.6 A simplified block diagram showing the new system using a controller to regulate the movements of the hexapod. The diagram shows distance controller C_b, trigonometric calculations block T, and the system dynamics $\frac{P}{s}$. The process is written as $\frac{P}{s}$ to emphasize the integrator part of the process. From the process we measure the angle of the hexapod, shown as Y°. The measured angle Y° is compared to the angle we want the hexapod to be at which is 0° and then sent to onward to T. T is a trigonometric transformation function that calculates the distances the legs should move to compensate for the angle of the hexapod. These distances, shown as Xmm, are then send on to the controller that makes alterations to the values and output that value as Zmm, as an example a P controller would multiply the distances with the value K so that $Z = K * X$. Zmm is sent to the process that in turns moves the legs.

However, once the work using physical sensors had reached its limits, the project went back to the surface estimation to find a solution. When the feet of the robot did not react when touching the ground, the leg would slowly push the hexapod off balance, yet since the balance system was working quite well by this point it would compensate for it resulting in the hexapod slowly moving upwards, stretching its legs. Problems like these also caused errors in the estimation of the angle of the ground showing just how sensitive the former angle calculations were. Every disturbance was added to the estimate and since we would get a input of zero when the body was angled correctly it would not be possible to add a low pass filter or something similar.

To correct for the upwards movement, a height control was implemented. Using the positions of the legs it was possible to calculate an approximation of the distance to the ground. The height control was affected by steps however since it could not use the legs in the air to and as soon as one leg left the ground the mean value changed. In the end it was only made into a safety control to avoid the legs from trying to reach beyond a maximum or minimum.

Two new methods of approximating the ground angle were devised. It could

be possible to calculate the angle of the ground using the positions of the legs. This design was scrapped however since there was no way to trust the step impact detection still as well as already having problems estimating the distance from the hexapod to the ground using the legs. Instead it was decided that several distance measuring sensors would be installed on the bottom of the hexapod. Using the distance measurements, it would be possible to both control the height of the hexapod as well as measuring the angle of the ground once the body of the hexapod got to a horizontal position.

A distance measuring sensor was borrowed from LTH and tested. The maximum distance was sufficient, and CCS ordered two so that the concept could be tested. However due to limited connections on the BBB and a lack of support to run several sensors using the same connection installing the sensors showed to be more complicated than anticipated. This in combination with the hexapod becoming needed elsewhere forced the project to move over to a simulated environment.

The distance sensors were easily implemented using V-REP and its V-REP to MATLAB API. Using Equation 2.4 it was possible to calculate the angle using the sensors position on the hexapod's body and the height to the ground. 4 sensors were installed on the bottom of the hexapod as shown in Figure 3.7. For each angle, roll and pitch, the sensors were divided in pairs and then the difference in distance between each pair was used to calculate the angle. The two pairs of calculated angles were then averaged and used as the new floor angle estimate.

As an example, when calculating pitch, the sensors would be paired as one from the front and one from the back, the height difference between each pair would then be used to calculate the angle, the mean value of the two angles from the two pairs was then used as the estimate of the angle of the ground. When calculating the roll of the body the sensors were instead paired up as two from the front and two from the back.

Once the sampling rate of the sensors were matched with the controller the height control and floor angle approximation worked very well which made it possible for the development of a new step impact detection system.

3.2.5 Step Impact Detection

The part of this project that has taken the most time has been the design of the step impact detection system. For the hexapod to be able to walk over an uneven surface it has to determine when its feet hit the ground. A previous project, Malmros and Eriksson [Malmros and Eriksson, 2016], had installed two force sensitive sensors on the front legs as well as a connection board, which circuit diagram can be seen in Figure 2.11. Since Malmros and Eriksson

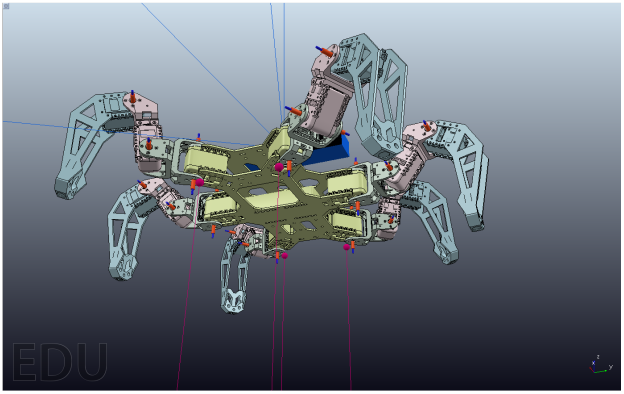

Figure 3.7 An image showing the placement of the proximity sensor on the hexapod in the V-REP simulation.

had recommended the resistors, "In the future, one goal could be to apply the sensors to every foot and integrate them with the balancing mode and thereby get the hexapod to re-balance as the legs are in different positions" [Malmros and Eriksson, 2016], as well as explained how well the resistors worked, "The sensors are, as can be seen in the Result section, precise and work as expected." [Malmros and Eriksson, 2016], it seemed like a natural start for this project.

The old resistors put on the hexapod by the previous project were worn out and replaced. Each of the resistors were coupled to two analog pins on the BeagleBone Black, one working as a source for the voltage over the sensor and the other as a sink. By measuring the voltage over the sink, it is possible to know if the force sensor is under pressure or not. To read the analog pin working as a sink the previous project had been using a Simulink function block from a BeagleBone Black support package [*Embedded Coder Support Package for BeagleBone Black Hardware*], called Analog Input, the function worked once the BeagleBone Black started but if the process was stopped or restarted the function stopped working. The bug seemed very inconsistent and it took a long time to pinpoint the cause, this could also be because as, I later found out, the resistors were far from precise.

The new function was written in C, by enabling the beaglebone's ADC driver it was possible to read the input values from specific files in the beaglebones directory. After working quite a bit with the legacy code generation to implement the sensor reading code into the model the sensors could finally be read properly, even after a restart.

The new resistors were quickly worn out, anything from a crease to a small

rip could make the resistor completely unusable and a lot of work went into looking for errors and replacing broken resistors. The force sensitive resistors are composed of a thin plastic with a conductive layer inside, making them very prone to melting if it took too much time to solder the connections. Since it usually was a crease caused by the hexapod moving unexpectedly the first attempt at protecting the resistors was to cover everything but the sensor pad in a thick layer of electric tape, making it almost impossible for the resistor to be bent out of shape. However, resistors were designed so that a small hole would let out air if the sensor pad was compressed. If the hole was blocked the sensor would stop reacting. A lot of time was spent unfurling non-functional resistors just to find them working again.

Figure 3.8 An image of the protected sensor.

Malmros, M. and Eriksson A had designed a 3D printed frame for the resistors as well as a button shape to be glued onto the resistors sensor pad, as shown in figure 2.15. There were enough frames, buttons and resistors for all the hexapods legs and after the initial tests all the legs had a resistor installed. The low design of the 3D printed buttons made it so that the pressure sensing pad under it was not always activated when the feet touched the ground at an angle. Rubber knobs were glued to the 3D printed buttons to make the buttons stick out more, so that they would take the weight of the hexapod once hitting the ground.

To not run tests in vain, this project has programmed a precheck routine that can check so that every sensor on the hexapod works when starting up the hexapod. One of the reasons for doing this was to make it possible for CCS to check the resistors before taking the hexapod to show it off. The precheck was hard to implement as even a fully functioning resistor would at times not react. The precheck was not used in the final version of the project as it only made the already slow code deployment and start up take even longer. After months of work trying to make the resistors function as described by Malmros and Eriksson with meagre results it was decided to replace the resistors with mechanical buttons. The rest of the system such

as the angle body control worked well at this point and the resistors were holding the whole project back.

The newly installed mechanical buttons did not wear out like the resistors though they still had a problem when touching the ground at an angle. The buttons would sometimes get caught at an angle making the yellow pin shown in Figure 3.9 push against its walls instead of moving straight down.

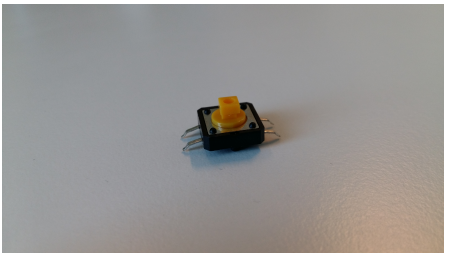

Figure 3.9 An image of the mechanical button.

A lot of new designs were thought of as possible alternative solutions to the impact detection. One alternative could be the use of height measurement, the angle from the IMU or of the balance compensation movement of the legs to detect the change once a leg hit the ground. Prototypes of these methods were tried out with inconclusive results as the floor angle estimate was too sensitive to the disturbances caused by the new tested impact detection.

Once the project moved over to using a simulated environment in V-REP and integrating the distance measuring sensors it became worthwhile to explore these methods further. This new floor angle approximation using distance measuring sensors was able to compensate for the disturbances caused by the impact detection.

Detecting the step impact using height difference did not look promising since the system quickly compensated to correct for any height differences and any error would have to become quite large before it significantly affected the height. Further tests were therefore made experimenting on how to detect the steps using either the angle or the compensation distances. Using the compensation distances made it possible to look at signals related only to one leg without any further calculations as well as having a direction directly related to the movement of the leg (we only want to react if the compensation wants to move the leg upwards). The controller had to be very sensitive and react to small changes to quickly detect when the leg hit the ground. The system is still not very robust, since errors accumulate and force the legs to move up from the ground thus unbalancing the hexapod. Several redesigns

were made to the impact detection such as trying to use integration to detect the constant change caused by a leg pushing into the ground but as the sample rate was too low for a slightly slower system to react before the hexapod would unbalance itself.

3.2.6 Speed Limitations

The hexapod can move in all directions, but the system built by previous projects represented the hexapod's velocity as a horizontal movement. It is possible to move vertically too but that is done as increments rather than continuously.

When walking at an angle, the legs in the ground move both horizontally and vertically, thus the hexapod can not keep the same horizontal speed when moving at different angles as the distance travelled vertically is added to the movement. If the hexapod moves too far away its legs will not be able to reach back far enough, (imagine you taking one giant leap with one of your legs while keeping the other in the ground). When contemplating this problem two alternatives were put forward. Either one could redesign the system and put in a block making the hexapod stop once it tried to reach a distance it could not, or one could calculate the relationship between speed and angle and thus always keep the speed under a maximum so that the problem could be avoided completely. It was decided to proceed with calculating the speed and angle relationship as it would make for a much more consistent movement. The fact that the task of analysing and creating a model of the relation between speed and angle was much more interesting and would give a greater understanding of the locomotion of the hexapod than redesigning the code of previous projects would also influenced the decision.

For each angle, the positions of the legs had to be calculated. The code written to calculate the center positions already did this but since it worked in an iterative manner it had to be implemented as a loop, changing the legs position until a sufficiently stable value was reached. To avoid calculating the maximum velocity for angles in vain a maximum and minimum limit for the legs' height was set. The limits were based on tests at high angles when the hexapod no longer could no longer balance its body.

Since the most plausible worst case scenario for the placement of the legs would be at start up this was decided as the scenario this model would be based on. At start up all the hexapod's legs stand at their center positions. The worst case scenario for each leg is that it is the last leg to be lifted of all the legs, the time it takes for the last leg to be lifted depends on how many legs are lifted at a time. If one leg is lifted at a time the last leg will have to wait 5 steps before it is its turn, two legs at a time results in a two step wait and with three legs at a time the last leg only has to wait for time it takes to

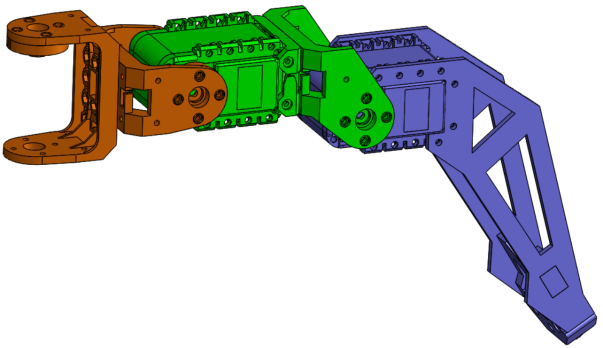

Figure 3.10 A CAD model of one leg of the hexapod. Each color represents one solid body, coxa (orange), femur (green) and tibia (blue). [Thilderkvist and Svensson, 2015]

do a single step. Since the time it takes to make a step is known, how far the last leg to be lifted will move can be calculated. It is simply a multiplication of the time it takes to make a step times a constant of 5, 2, or 1, and the velocity of the hexapod.

As the length of a leg and its position on the body is known, it is possible to calculate the maximum speed for each leg. Yet, since the hexapod's legs are not straight sticks the calculations were a bit too complex to be implemented in the Simulink model without risk of affecting the run speed negatively, so it was decided that the calculations would be made before starting the hexapod and put into a matrix that would act as a lookup table during the code execution.

Each leg consists of three parts as seen in Figure 3.10; the coxa, the femur and the tibia. The coxa only moves left or right while the femur and tibia can only move up or down. So, using the method described above it was possible to calculate the distance a leg would have moved while waiting to be lifted and add it to the center position at which the leg would start. The calculation of how far the leg would have to stretch to reach that position was more complicated. First the direction of the coxa had to be calculated, then the distance from the end of the coxa and the position of the its foot had to be compared to the length of the femur and the tibia combined. A demonstration is shown in Figure 3.11

When trying to verify the calculated maximum distances it was discovered that one could not simply check when the legs moved too far back but also see if a leg could reach as far forward as it needed to step at that velocity,

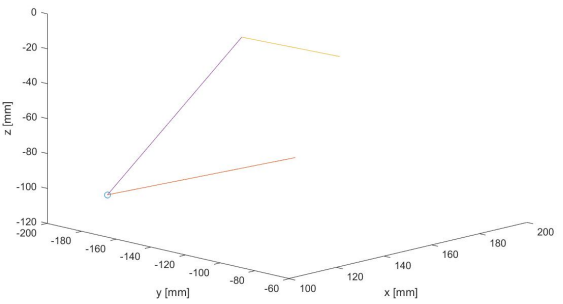

Figure 3.11 The movement of a leg waiting to be lifted shown as the orange line, the yellow line is the leg's coxa, the purple is the legs femur and tibia, the blue circle represents the position of the foot and the last position before the leg is lifted. [Thilderkvist and Svensson, 2015]

especially when walking downhill this would become a problem. Since every step moves the same distance forward from its center position as it would backwards if on the ground, that is to say legs per step multiplied with the time per step and the velocity, it was easy to add to the calculations.

3.2.7 Real-time Analysis

To find potential bottlenecks Simulink's own function execution time measuring was used. By configuring the parameters of the Simulink model, it was possible to record the execution time of all the Simulink functions. This was however only done using the Simulation as the project no longer had access to the hexapod at this point.

3.2.8 Leg Collision Avoidance

While stress testing the hexapod it was observed that when rotating and then trying to make the hexapod move forward quickly the legs would sometimes collide. Therefore, a leg collision avoidance system was designed, to avoid damage to the robot and as a preparation for future projects.

By making a linear function of each leg using the legs foot position and the position of its connection to the body and comparing how the endpoint of the leg with the smallest value on the y axis how close its end point x value was to the other legs x values for that y and then doing the same for the next to shortest leg comparing to the longest it was possible to set a minimum value and avoid any future collisions. This was done for each side of the hexapod, the assumption being that two legs from opposite sides would never collide.

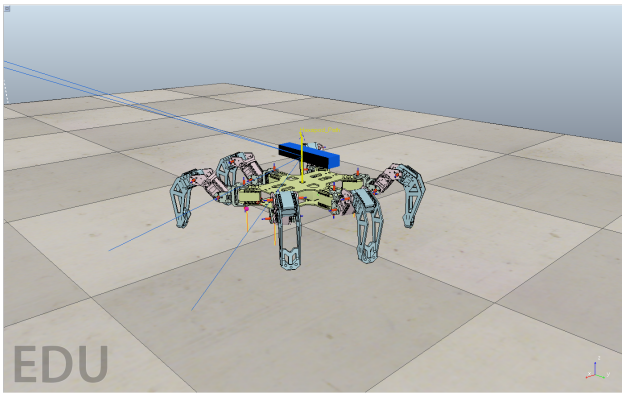

Figure 3.12 An image of the test environment and the hexapod in the V-REP simulation.

3.3 Simulation

Thilderkvist and Svensson built a 3D CAD model of the hexapod and put said model into V-REP so that they could work on developing the hexapod's movements easier. The next project to work with the hexapod, Malmros and Eriksson, then utilized the same 3D model but put it in a environment they could interact with. This model has then been re-purposed and used throughout this project. In order to read the angles of the hexapod's body a new orientation basis had to be set. In the later parts of the project four proximity sensors were added to the model. By tilting the floor on which the hexapod walked one could see how the system reacted when moving at specific angles. Pre-made environments could be added to the simulation, such as a hilly dessert environment, to do asymmetrical testing. Using building blocks available in V-REP it is also possible to build more challenging test courses.

Figure 3.13 An image showing the hexapod climbing a hill in the V-REP simulation.

4

Result

This section contains data relevant to this project and thesis. The data and the conclusions drawn from it will be discussed further in the next chapter. Most of the data is here to show the motivation and reason for certain design decisions described in the methods chapter. A lot of data was gathered before moving from the real hexapod to the simulation to be able to verify the simulation.

4.1 Balance

This section has several graphs showing the result of different balance controls systems used in a simulation while the hexapod is walking. All the control systems in this section benefit from five times the sample rate compared to the systems shown in section 4.2. Impact detection was not used in these tests and the hexapod walked across a flat surface at an angle. The grounds angle was approximated using distance sensors for all the tests in this section.

The following figures show the angle of the hexapod's body as it moves up a hill angled at $4°$, , taking steps one leg at a time and moving its body forward at the speed of $20mm/s$.

4.1.1 Angle controller

In these graphs the balance control system is regulating the angle before calculating the compensation distance for the legs to move.

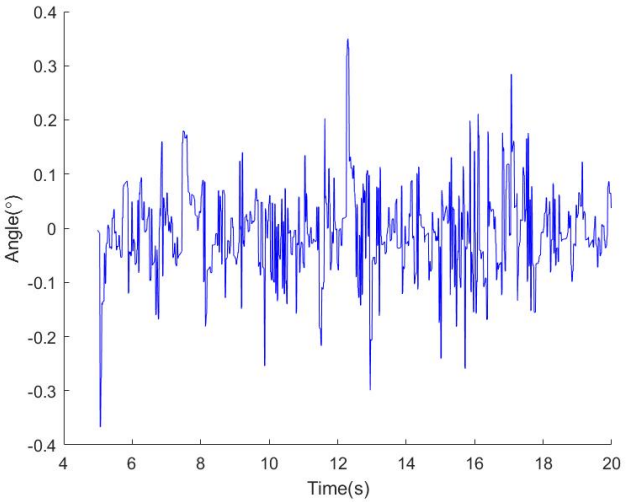

Figure 4.1 A graph showing the pitch of the simulated hexapod's body, the balance control system is regulating the angle with a constant of 0.2. Variance of the pitch is 0.0799°.

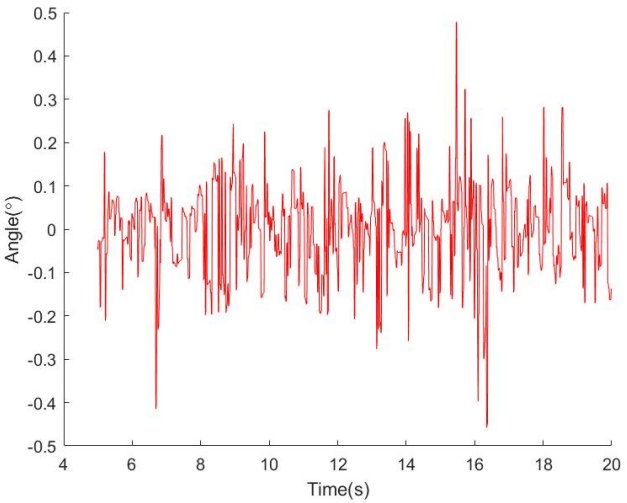

Figure 4.2 A graph showing the roll of the simulated hexapod's body, the balance control system is regulating the angle with a constant of 0.2. Variance of the roll is 0.1050°.

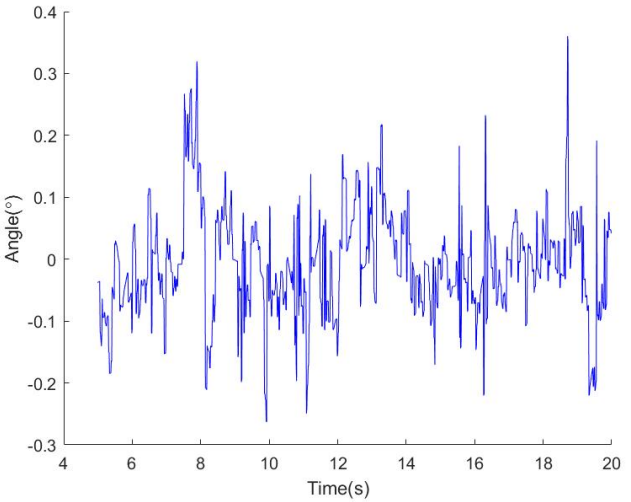

Figure 4.3 A graph showing the pitch of the simulated hexapod's body, the balance control system is regulating the angle with a constant of 0.1. Variance of the pitch is 0.0863°.

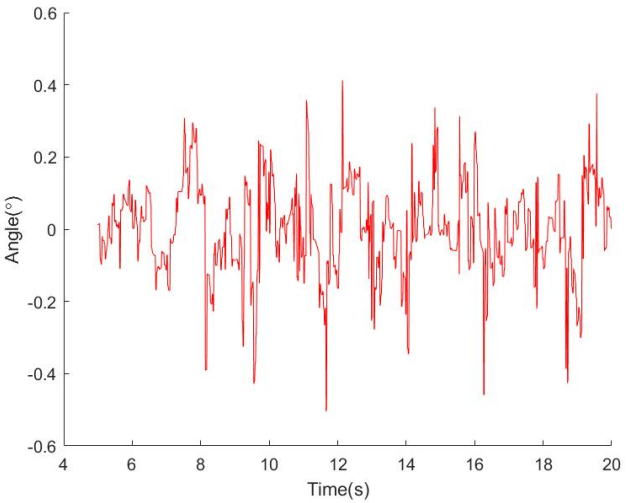

Figure 4.4 A graph showing the roll of the simulated hexapod's body, the balance control system is regulating the angle with a constant of 0.1. Variance of the roll is 0.1050°.

4.1.2 P-, PI- and PD-controller Comparison

In these graphs the balance control system is regulating the compensation distance for the legs to move and not the angle like the original system. This control system was used for the final version of the model.

4.1.2.1 P-controller The following figures show how different P-controllers balance the hexapod.

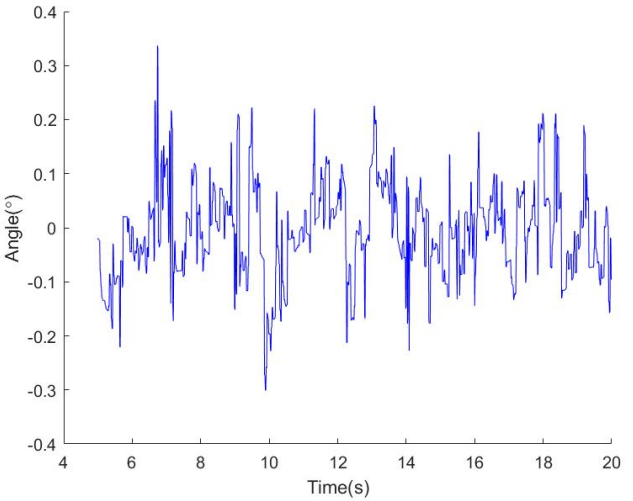

Figure 4.5 A graph showing the pitch of the simulated hexapod's body, the balance control system is regulating the compensation distance with a constant of 0.1. Variance of the pitch is 0.0883°.

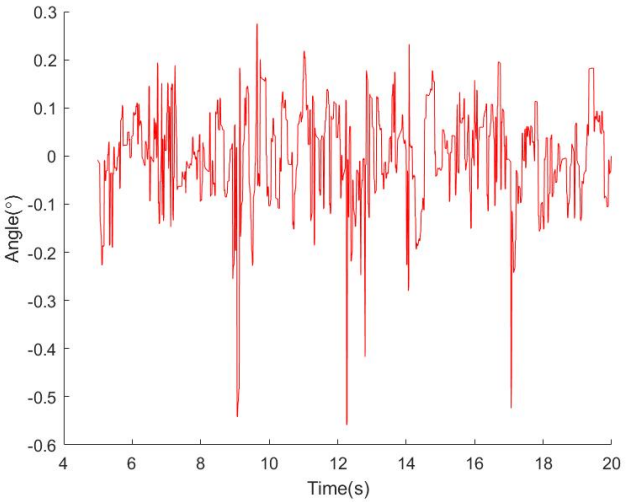

Figure 4.6 A graph showing the roll of the simulated hexapod's body, the balance control system is regulating the compensation distance with a constant of 0.1. Variance of the roll is 0.1026°.

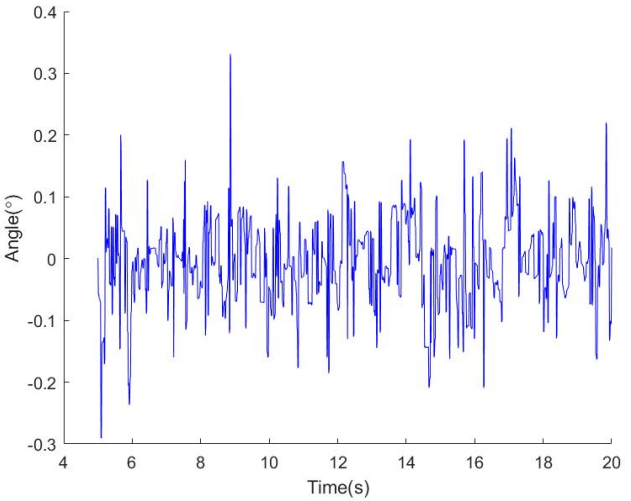

Figure 4.7 A graph showing the pitch of the simulated hexapod's body, the balance control system is regulating the compensation distance with a constant of 0.2. Variance of the pitch is 0.0723°.

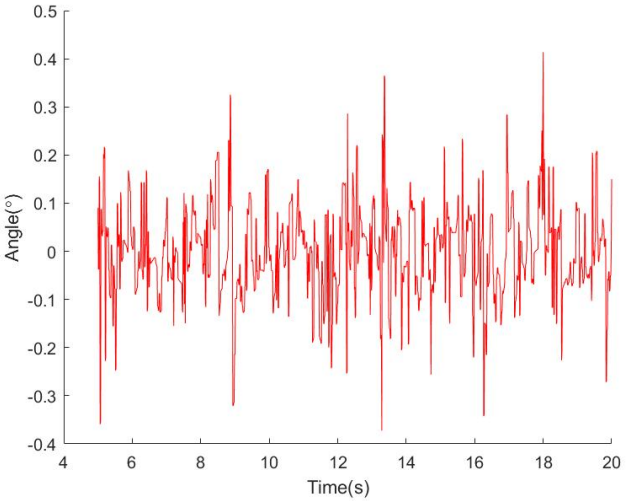

Figure 4.8 A graph showing the roll of the simulated hexapod's body, the balance control system is regulating the compensation distance with a constant of 0.2. Variance of the roll is 0.0971°.

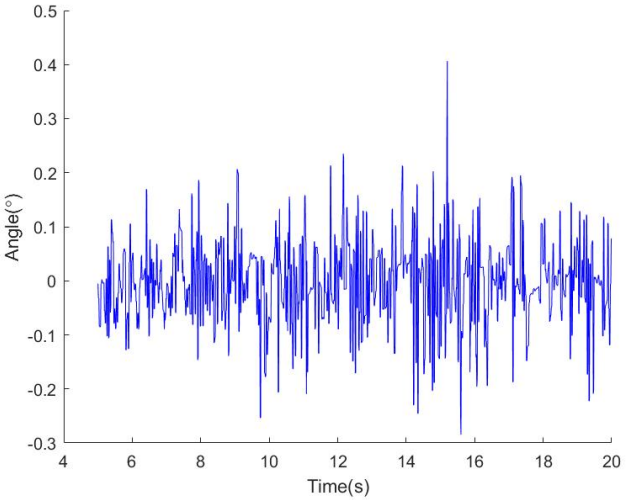

Figure 4.9 A graph showing the pitch of the simulated hexapod's body, the balance control system is regulating the compensation distance with a constant of 0.4. Variance of the pitch is 0.0799°.

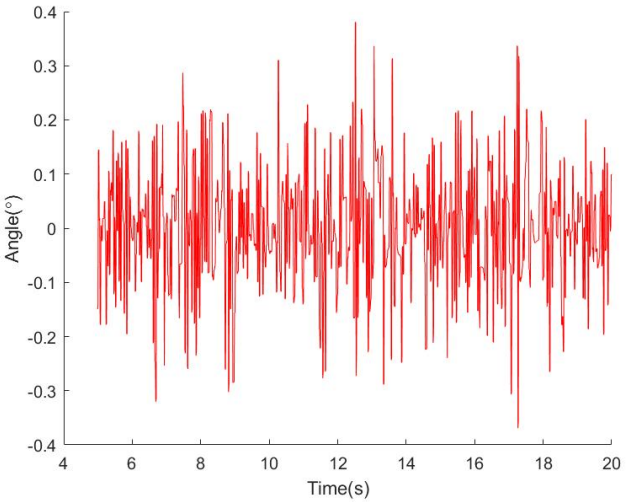

Figure 4.10 A graph showing the roll of the simulated hexapod's body, the balance control system is regulating the compensation distance with a constant of 0.4. Variance of the roll is 0.1124°.

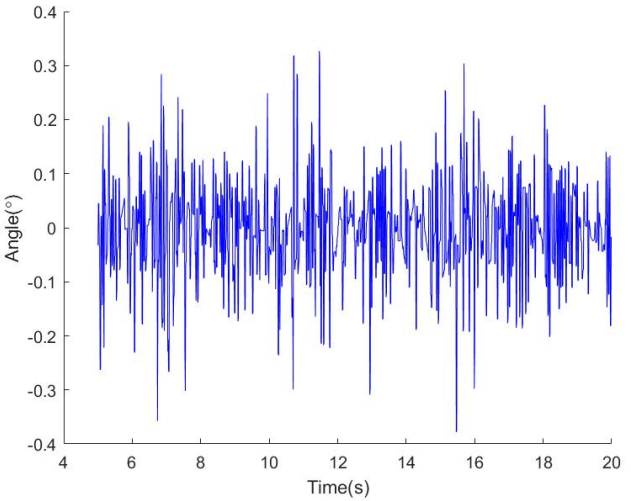

Figure 4.11 A graph showing the pitch of the simulated hexapod's body, the balance control system is regulating the compensation distance with a constant of 0.8. Variance of the pitch is 0.949°.

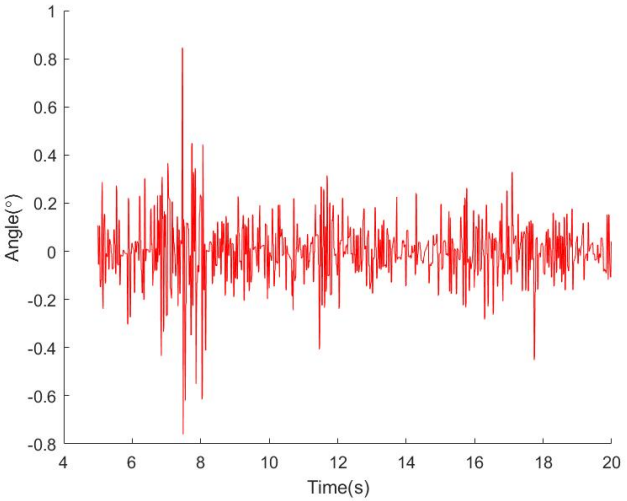

Figure 4.12 A graph showing the roll of the simulated hexapod's body, the balance control system is regulating the angle with a constant of 0.8. Variance of the roll is 0.1277°.

4.1.2.2 PI-controller The following figures show a PI-controller balancing the hexapod. The parameters are $K = 0.1$, $Ti = 8$ and $h = 0.025$. The variance is 0.0918° for pitch and 0.1283° for roll.

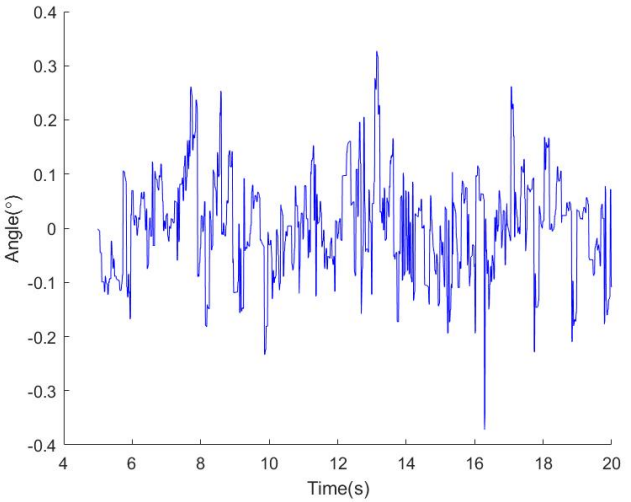

Figure 4.13 A graph showing the pitch of the simulated hexapod's body, the balance control system is regulating the compensation distance with a constant of 0.1.

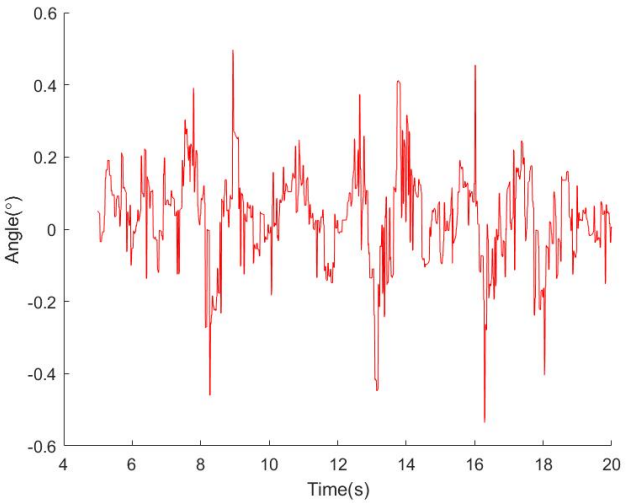

Figure 4.14 A graph showing the roll of the simulated hexapod's body, the balance control system is regulating the compensation distance with a constant of 0.1.

4.1.2.3 PD-controller The following figures show a low-pass filtered PD-controller balancing the hexapod. The parameters are $K = 0.1$, $Td = 0.1$ and $h = 0.025$ and $N = 20$. The variance is 0.1064° for pitch and 0.1302° for roll.

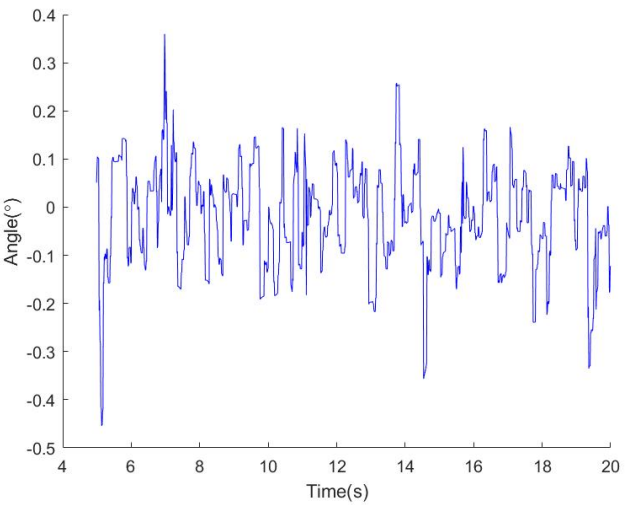

Figure 4.15 A graph showing the pitch of the simulated hexapod's body, the balance control system is regulating the compensation distance with a constant of 0.1.

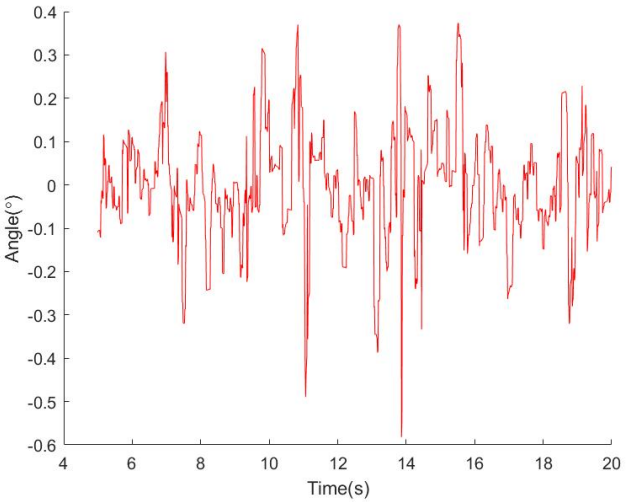

Figure 4.16 A graph showing the roll of the simulated hexapod's body, the balance control system is regulating the compensation distance with a constant of 0.1.

4.2 Reality and Simulation Comparison

The following figures show the angle of the hexapod's body as it moves in reality and in the simulated environment. These graphs were made to compare the simulation with reality and give an idea of how the work in the simulated system might function in reality.

For comparison reasons, the simulation is not running the last version of the Simulink model in the graphs of this subsection and is instead using an older model that matches the sample rate and control system of when the real hexapod data was gathered. They are both using a P-controller with the constant of 0.2 to regulate the angle of the hexapod's body before calculating the distance the legs should move to compensate.

4.2.1 Four Degree Pitch and One Leg at a Time

The following graphs show the pitch or roll of the hexapod's body as it walks without using impact detection. The hexapod is moving up a hill angled at 4°, taking steps one leg at a time and moving its body forward at the speed of $20mm/s$.

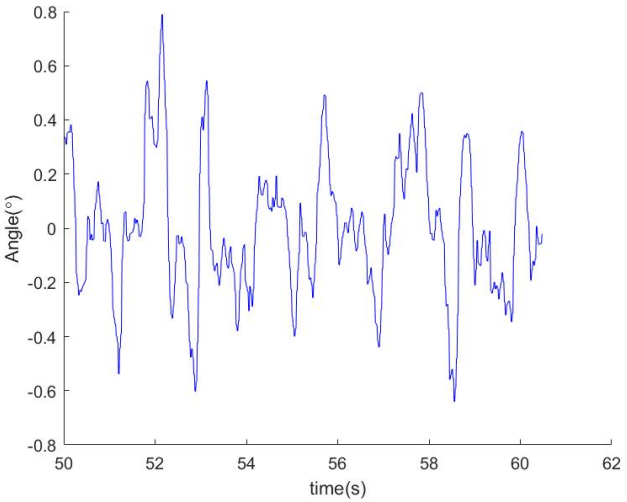

Figure 4.17 A graph showing the pitch of the real hexapod's body as the robot walks using a step impact detection. Variance of the pitch is 0.2510°.

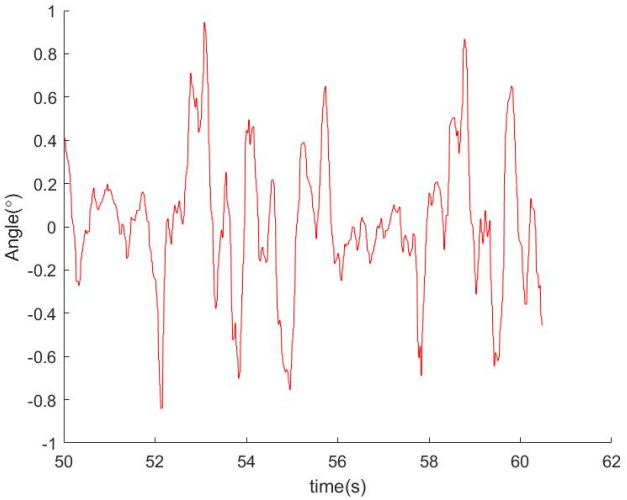

Figure 4.18 A graph showing the roll of the real hexapod's body as the robot walks using a step impact detection. Variance of the roll is 0.3171°.

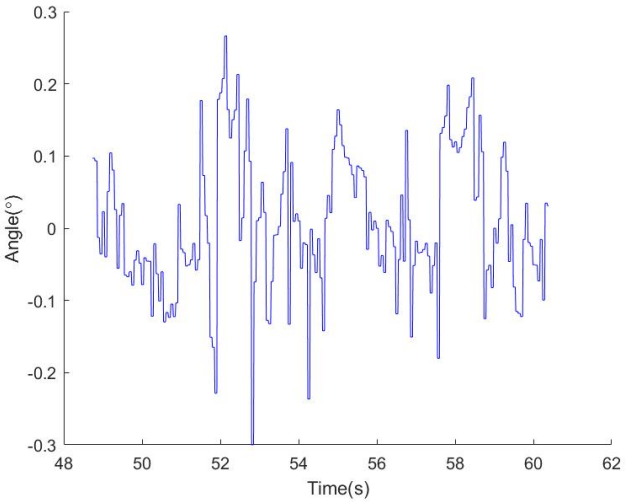

Figure 4.19 A graph showing the pitch of the simulated hexapod's body as the robot walks using a step impact detection. Variance of the pitch is 0.0970°.

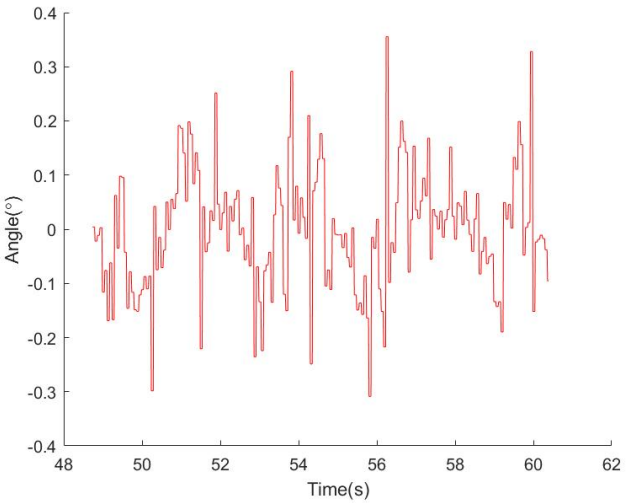

Figure 4.20 A graph showing the roll of the simulated hexapod's body as the robot walks using a step impact detection. Variance of the roll is 0.1109°.

4.2.2 Six Degree Roll and One Leg at a Time

The following graphs show the pitch or roll of the hexapod's body as it walks without using impact detection. The hexapod is moving across a surface angled 6° in the roll direction, taking steps one leg at a time and moving its body forward at the speed of $10mm/s$.

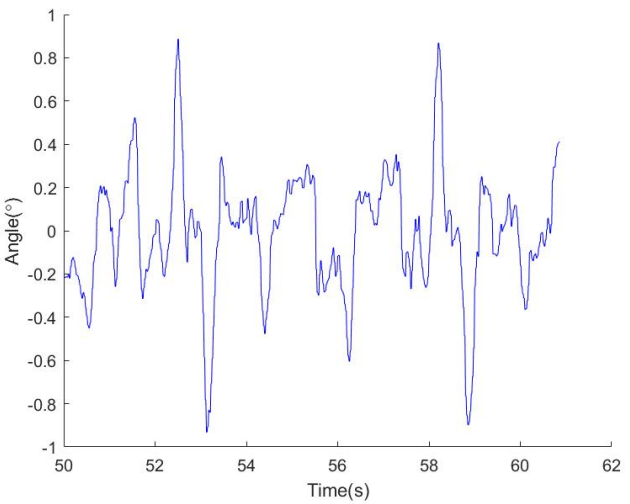

Figure 4.21 A graph showing the pitch of the real hexapod's body as the robot walks using a step impact detection. Variance of the pitch is 0.02824°.

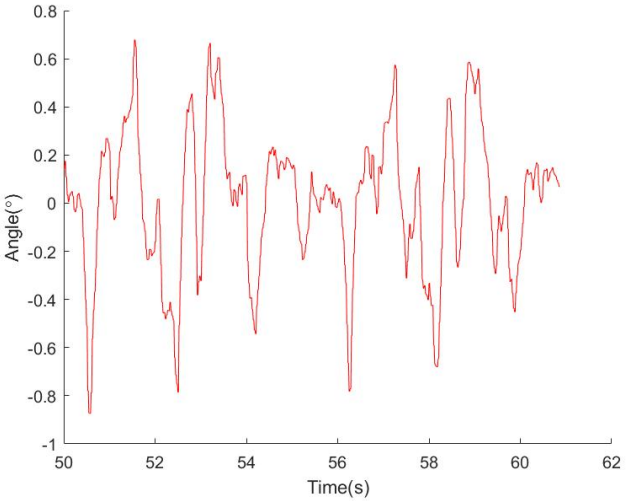

Figure 4.22 A graph showing the roll of the real hexapod's body as the robot walks using a step impact detection. Variance of the roll is 0.0.3004°.

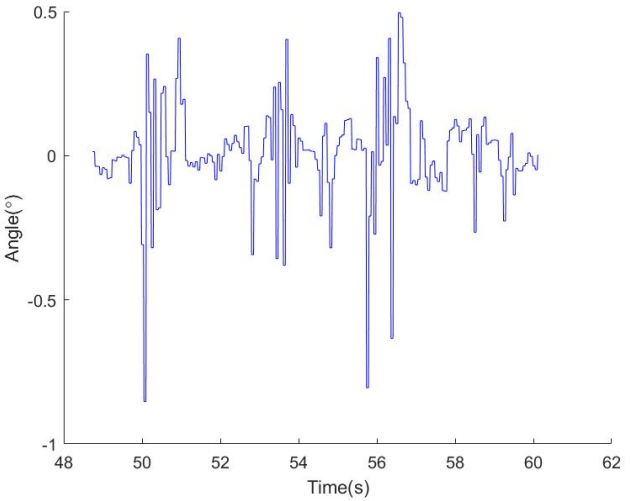

Figure 4.23 A graph showing the pitch of the simulated hexapod's body as the robot walks using a step impact detection. Variance of the pitch is 0.1722°.Variance of the pitch is 0.°.

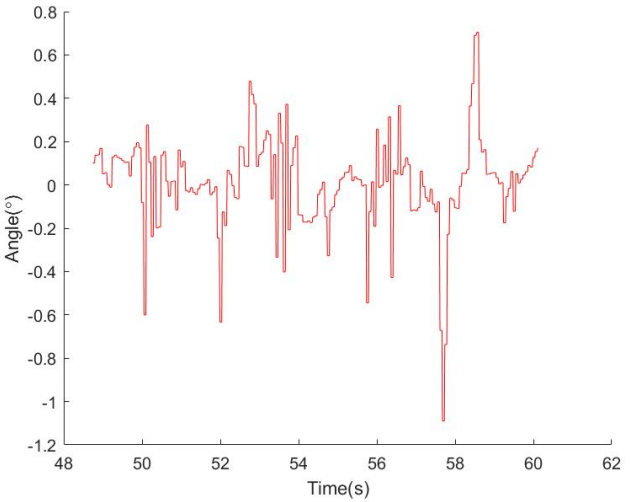

Figure 4.24 A graph showing the roll of the simulated hexapod's body as the robot walks using a step impact detection. Variance of the roll is 0.2130°.

4.2.3 Four Degree Pitch and Three Legs at a Time

The following graphs show the pitch or roll of the hexapod's body as it walks without using impact detection. The hexapod is moving up a hill angled at 4°, lifting three legs at a time time and moving it's body forward at the speed of $30mm/s$.

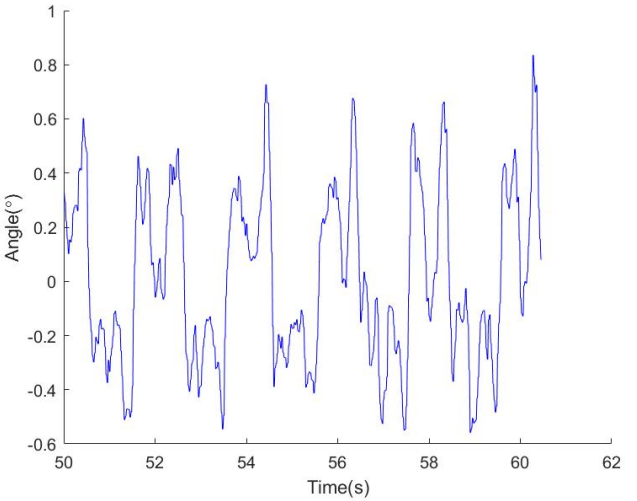

Figure 4.25 A graph showing the pitch of the real hexapod's body as the robot walks using a step impact detection. Variance of the pitch is 0.3169°.

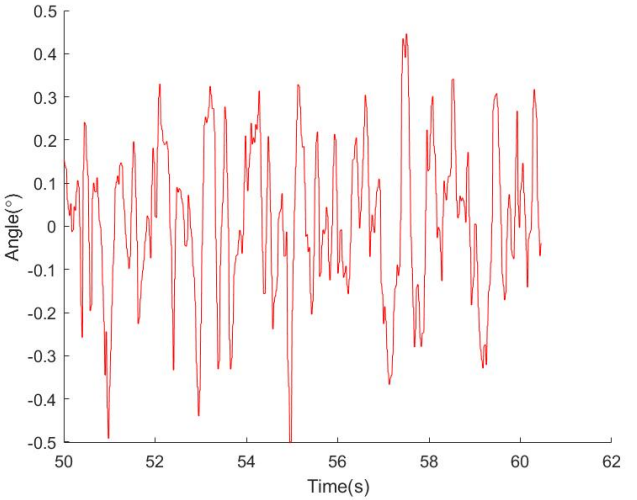

Figure 4.26 A graph showing the roll of the real hexapod's body as the robot walks using a step impact detection. Variance of the roll is 0.1785°.

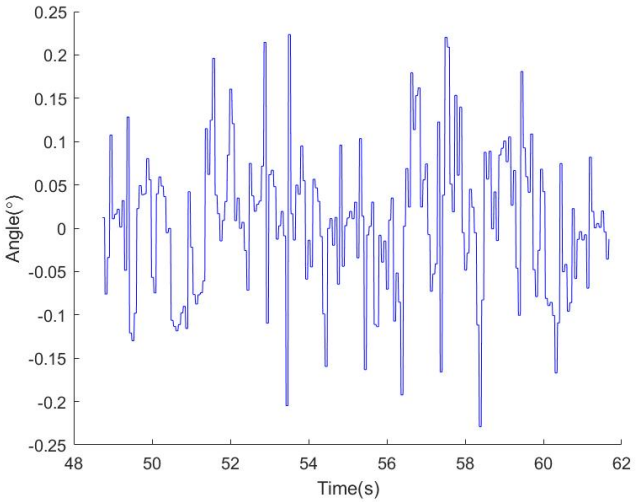

Figure 4.27 A graph showing the pitch of the simulated hexapod's body as the robot walks using a step impact detection. Variance of the pitch is 0.0820°.

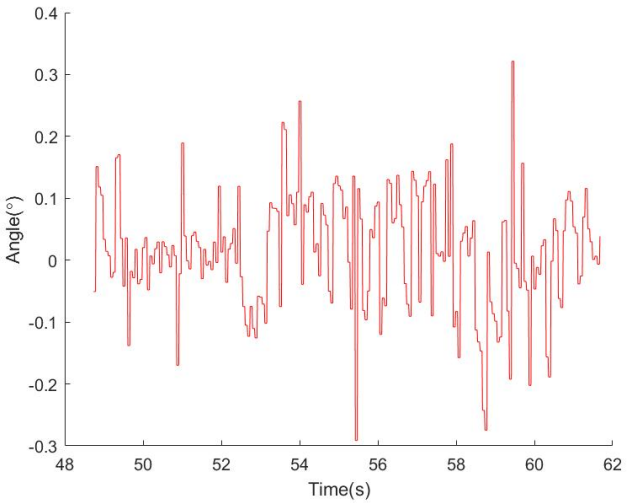

Figure 4.28 A graph showing the roll of the simulated hexapod's body as the robot walks using a step impact detection. Variance of the pitch is 0.0936°.

4.2.4 Six Degree Roll and Three Legs at a Time

The following graphs show the pitch or roll of the hexapod's body as it walks without using impact detection. The hexapod is moving across a surface angled 6° in the roll direction, lifting three legs at a time time and moving its body forward at the speed of $40mm/s$.

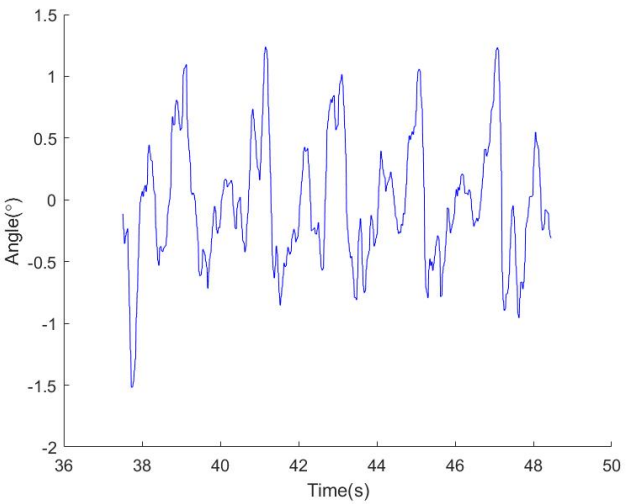

Figure 4.29 A graph showing the pitch of the real hexapod's body as the robot walks using a step impact detection. Variance of the pitch is 0.5090°.

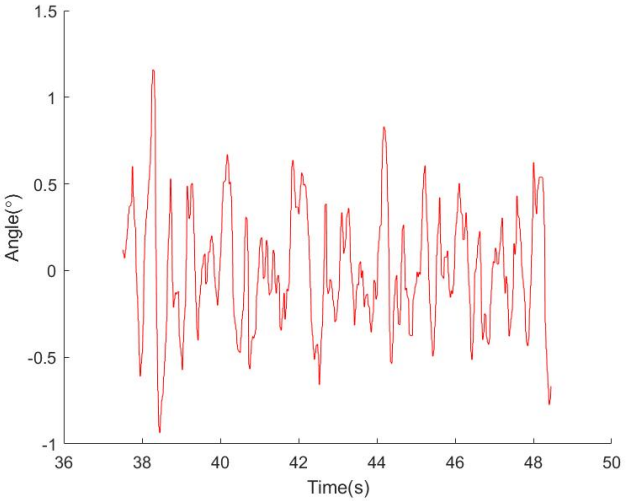

Figure 4.30 A graph showing the roll of the real hexapod's body as the robot walks using a step impact detection. Variance of the roll is 0.3473°.

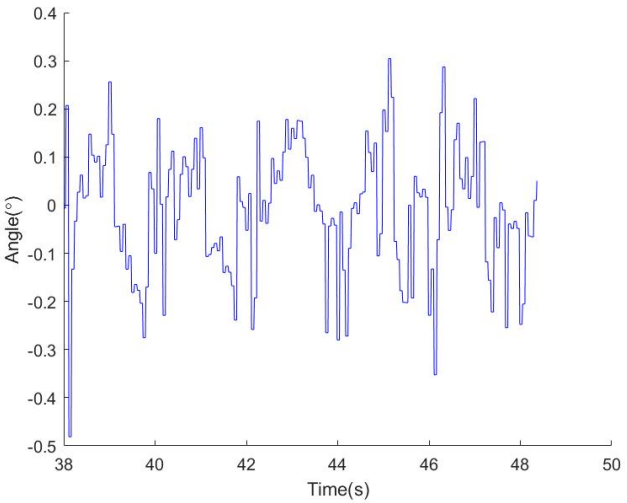

Figure 4.31 A graph showing the pitch of the simulated hexapod's body as the robot walks using a step impact detection. Variance of the pitch is 0.1331°.

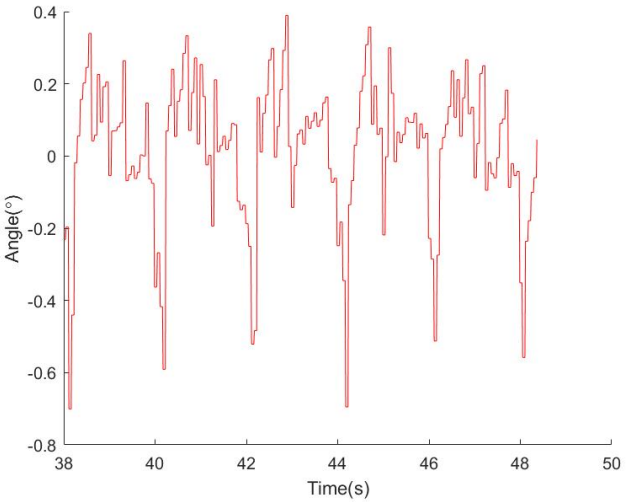

Figure 4.32 A graph showing the roll of the simulated hexapod's body as the robot walks using a step impact detection. Variance of the roll is 0.2016°.

4.3 Ground Angle Approximation

The following figures show the results from two different methods used to estimate the angle of the ground the hexapod walks on. The graphs show the pitch or roll of the estimate of the angle of the ground as the hexapod walks without using impact detection. The hexapod is moving up a hill angled at 4°, taking steps one leg at a time and moving its body forward at the speed of $20mm/s$.

Both systems run in a simulated environment and use an updated version of the Simulink model with a sampling speed 5 times that of the system running on the real hexapod.

The first system, results shown in figures 4.33 and 4.34, uses a P-controller with the constant 0.2 to regulate the angle sent by the hexapod. The controlled angle is then used to calculate the distance each leg should move to compensate for it. The system estimates the angle of the ground by adding up all the angles the body compensates for.

The second system, results shown in figures 4.35 and 4.36, uses a P-controller to regulate the distance calculated instead of the angle. The floor angle is then measured by distance measuring sensors.

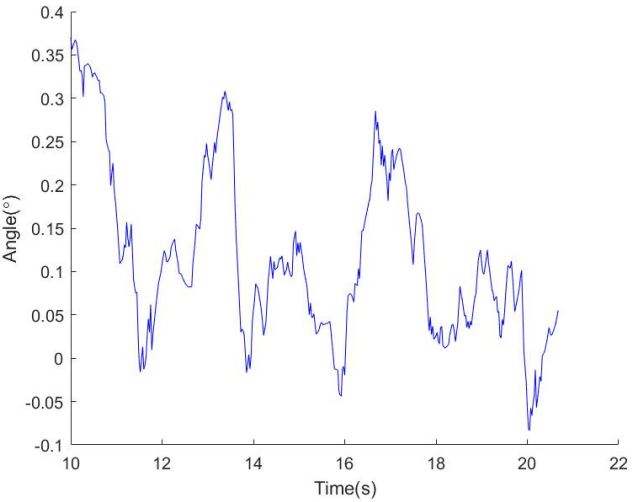

Figure 4.33 A graph showing the approximation of the grounds pitch angle as the robot walks. Variance of the pitch is 0.1003° and the average value is 0.1168°

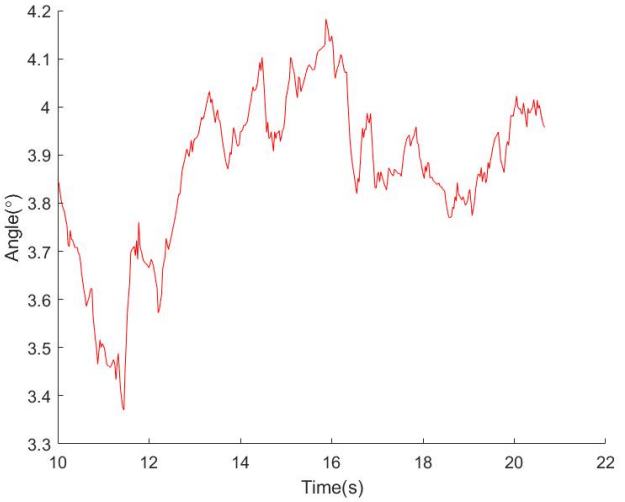

Figure 4.34 A graph showing the approximation of the grounds roll angle as the robot walks. Variance of the roll is 0.1677° and the average value is 3.8731°

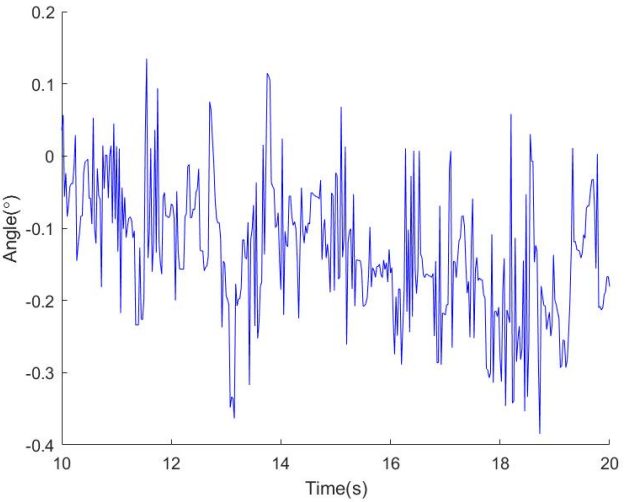

Figure 4.35 A graph showing the approximation of the grounds pitch angle as the robot walks. Variance of the pitch is 0.0929° and the average value is −0.1335°

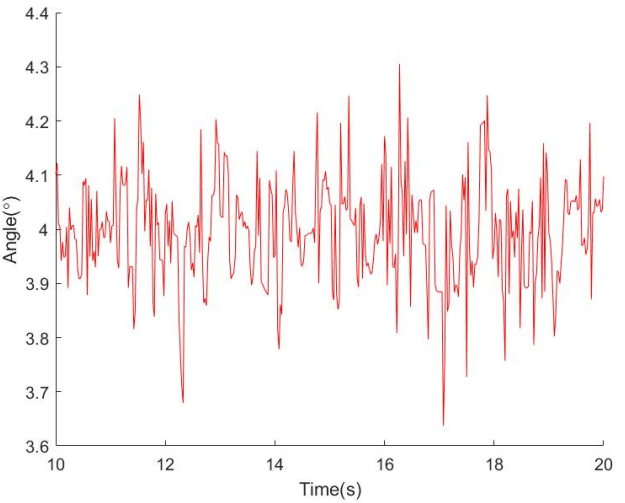

Figure 4.36 A graph showing the approximation of the grounds roll angle as the robot walks. Variance of the roll is 0.0975° and the average value is 4.0048°

4.4 Step impact detection

The following graphs show how the simulated hexapod's body is angled as it walks using impact detection. The hexapod is moving at 4° degrees angle, taking steps one leg at a time, at the speed of $20mm/s$ and using a P-controller with the constant of 0.1 to regulate the distance each leg moves to compensate for an angle. A leg taking a step react to a regulated distance making the leg move up with a value of 0.18. The variance of the pitch shown in figure 4.37 is 0.1510°and the roll in figure 4.38 has the variance of 0.3902°.

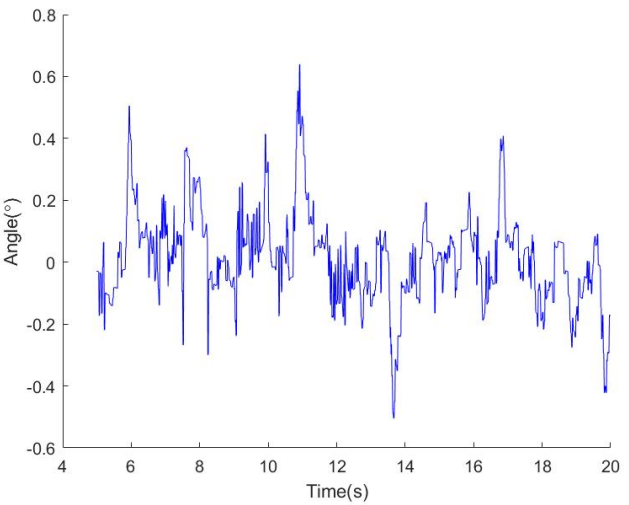

Figure 4.37 A graph showing the pitch of the simulated hexapod's body as the robot walks using a step impact detection. Variance of the roll is 0.1510°.

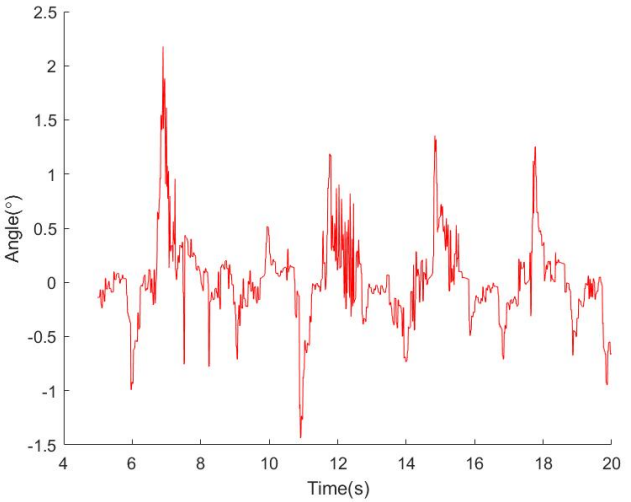

Figure 4.38 A graph showing the roll of the simulated hexapod's body as the robot walks using a step impact detection. Variance of the roll is 0.3902°.

4.5 Speed limitations

The following graphs show the relation between the hexapod's maximum speed and the angle of the surface it is walking on. Every circle is a data point. The empty space represents angles which would be too steep for the hexapod to stand on and still keep its body horizontal. Figure 4.39 and 4.40 shows the graph of the speed limit when using only one leg, figure 4.41 and 4.42 two legs and figure 4.43 and 4.44 three legs.

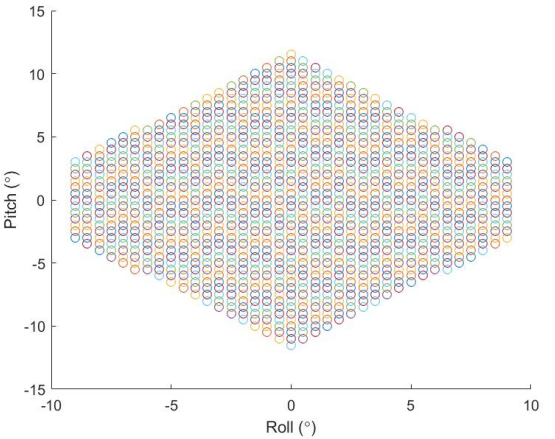

Figure 4.39 The "bird's eye view" of a graph showing the how the maximum horizontal velocity of the hexapod while taking steps with one leg at a time is dependent on the angle of the surface it is walking on.

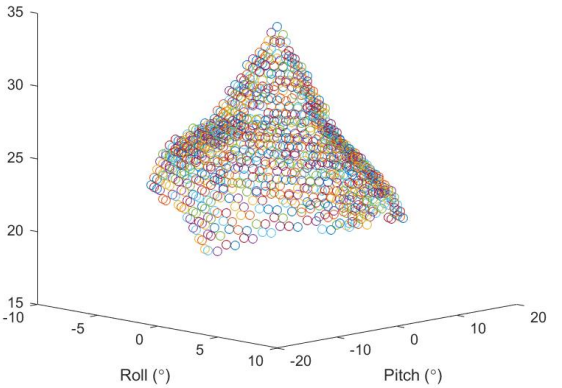

Figure 4.40 The side view of a graph showing the how the maximum horizontal velocity of the hexapod while taking steps with one leg at a time is dependent on the angle of the surface it is walking on.

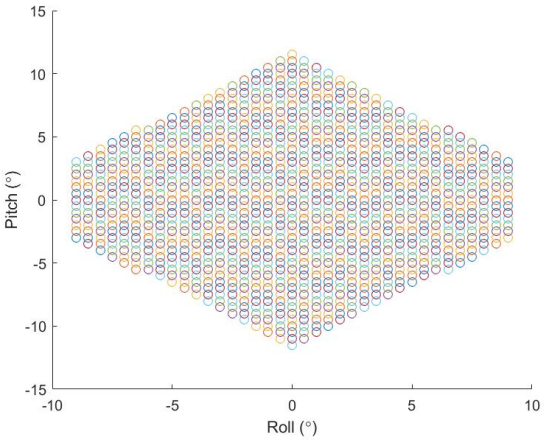

Figure 4.41 The "bird's eye view" of a graph showing the how the maximum horizontal velocity of the hexapod while taking steps with two legs at a time is dependent on the angle of the surface it is walking on.

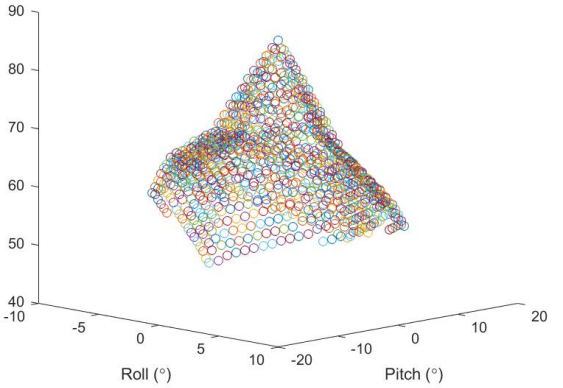

Figure 4.42 The side view of a graph showing the how the maximum horizontal velocity of the hexapod while taking steps with one leg at a time is dependent on the angle of the surface it is walking on.

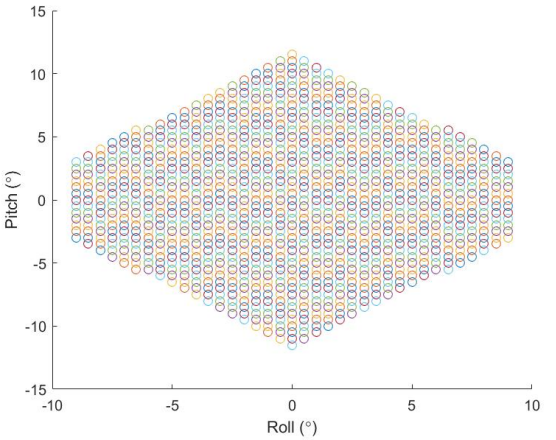

Figure 4.43 The "bird's eye view" of a graph showing the how the maximum horizontal velocity of the hexapod while taking steps with three legs at a time is dependent on the angle of the surface it is walking on.

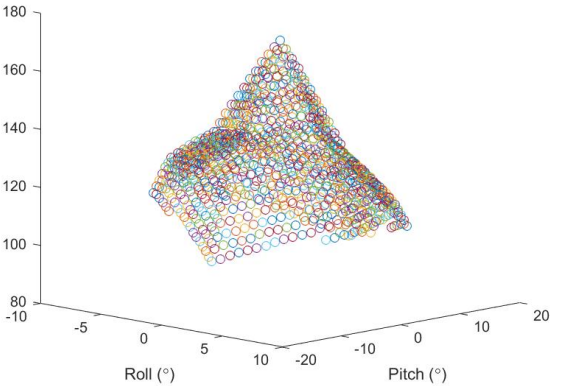

Figure 4.44 The side view of a graph showing the how the maximum horizontal velocity of the hexapod while taking steps with one leg at a time is dependent on the angle of the surface it is walking on.

4.6 Realtime analysis

Function:	Time (s)	Time(%)	Calls	Time/call
DevelopmentModel_distance/Control	4.03125000		2402	0.001678288926
Child functions:				
DevelopmentModel_distance/Control/LeftLegIK (TriggeredSubSystem.Outputs.Major)	0.43750000	10.9%	1601	0.000273266708
DevelopmentModel_distance/Control/RightLegIK (TriggeredSubSystem.Outputs.Major)	0.42187500	10.5%	1601	0.000263507183
DevelopmentModel_distance/Control/LeftLegIK1 (TriggeredSubSystem.Outputs.Major)	0.39062500	9.7%	1601	0.000243988132
DevelopmentModel_distance/Control/LeftLegIK2 (TriggeredSubSystem.Outputs.Major)	0.37500000	9.3%	1601	0.000234228607
DevelopmentModel_distance/Control/Body_Control/MainController /TrajectorySystem	0.34375000	8.5%	1601	0.000214709557
DevelopmentModel_distance/Control/RightLegIK1 (TriggeredSubSystem.Outputs.Major)	0.29687500	7.4%	1601	0.000185430981
DevelopmentModel_distance/Control/RightLegIK2 (TriggeredSubSystem.Outputs.Major)	0.21875000	5.4%	1601	0.000136633354
DevelopmentModel_distance/Control/Body_Control/MainController /QueueSystem (TriggeredSubSystem.Outputs.Major)	0.21875000	5.4%	1601	0.000136633354
DevelopmentModel_distance/Control/Body_Control/CoordBodytoLeg (TriggeredSubSystem.Outputs.Major)	0.18750000	4.7%	1601	0.000117114304
DevelopmentModel_distance/Control/Body_Control/MainController /InputFiltering (TriggeredSubSystem.Outputs.Major)	0.14062500	3.5%	1601	0.000087835728
DevelopmentModel_distance/Control/Body_Control/Rate Transition (RateTransition.Outputs.Major)	0.14062500	3.5%	4804	0.000029272481
DevelopmentModel_distance/Control/LeftLegIK1 (ToWorkspace.Outputs.Major)	0.07812500	1.9%	1601	0.000048797626
DevelopmentModel_distance/Control/Body_Control/MainController /StandPositionCalculator (TriggeredSubSystem.Outputs.Major)	0.07812500	1.9%	1601	0.000048797626
DevelopmentModel_distance/Control/Body_Control/MainController /floorAngle (Memory.Outputs.Major)	0.03125000	0.8%	1601	0.000019519051
DevelopmentModel_distance/Control/Body_Control/MainController /currentPos (Memory.Outputs.Major)	0.03125000	0.8%	1601	0.000019519051
DevelopmentModel_distance/Control/TriggerController/MainTriggerPulse (DiscretePulseGenerator.Outputs.Major)	0.03125000	0.8%	1601	0.000019519051
DevelopmentModel_distance/Control/LeftLegIK /Trigger (SignalConversion.Outputs.Major)	0.01562500	0.4%	1601	0.000009759525
DevelopmentModel_distance/Control/RightLegIK1 (ToWorkspace.Outputs.Major)	0.01562500	0.4%	1601	0.000009759525
DevelopmentModel_distance/Control/Body_Control/Data Type Conversion1 (DataTypeConversion.Outputs.Major)	0.01562500	0.4%	1601	0.000009759525
DevelopmentModel_distance/Control/Body_Control/Data Type Conversion2 (DataTypeConversion.Outputs.Major)	0.01562500	0.4%	1601	0.000009759525
DevelopmentModel_distance/Control/Vector Concatenate1 (Concatenate.Outputs.Major)	0.00000000	0.0%	1601	0.000000000000
DevelopmentModel_distance/Control/Vector Concatenate (Concatenate.Outputs.Major)	0.00000000	0.0%	1601	0.000000000000
DevelopmentModel_distance/Control/RightLegIK/Trigger (SignalConversion.Outputs.Major)	0.00000000	0.0%	1601	0.000000000000
DevelopmentModel_distance/Control/RightLegIK2/Trigger (SignalConversion.Outputs.Major)	0.00000000	0.0%	1601	0.000000000000
DevelopmentModel_distance/Control/LeftLegIK2/Trigger (SignalConversion.Outputs.Major)	0.00000000	0.0%	1601	0.000000000000
DevelopmentModel_distance/Control/RightLegIK1/Trigger (SignalConversion.Outputs.Major)	0.00000000	0.0%	1601	0.000000000000
DevelopmentModel_distance/Control/LeftLegIK1/Trigger (SignalConversion.Outputs.Major)	0.00000000	0.0%	1601	0.000000000000
DevelopmentModel_distance/Control/Body_Control/MainController /InAir (Memory.Outputs.Major)	0.00000000	0.0%	1601	0.000000000000
DevelopmentModel_distance/Control/TriggerController/Frequency (Constant.Outputs.Major)	0.00000000	0.0%	1601	0.000000000000
DevelopmentModel_distance/Control/Body_Control/Data Type Conversion3 (DataTypeConversion.Outputs.Major)	0.00000000	0.0%	1601	0.000000000000

Figure 4.45 A table showing how much time the different functions of the control system take to execute.

5

Discussion and conclusions

5.1 Balance

A lot of work has gone into finding an optimal controller for the hexapod body's angle control. The focus has been on testing several different variants of P-, PD- and PI-controllers. The hexapod's movements when adjusting for height corresponds to an integrator, accumulating adjustments. This is why it has worked so well using only a P-controller as control system and why adding an integration part to the controller has caused oscillations. As for the PD controller, a derivation part is often hard to implement and is often optional and turned off in many industrial applications[Årzén, 2014].

A future project could perhaps identify the process more thoroughly and design a better solution, but as this is only one of many parts of this project that kind of specialization has not been possible. Though I am skeptical of the value a derivation part would have, implemented on the real hexapod. Far more noise and disturbances could be expected in the real system and I do not believe the processing speed of the BBB or the sampling rate of the IMU would be high enough to effectively run a low pass filtered derivation controller.

A lot of thought went into the choice of regulator. Controlling the angle and then calculating the distance would not have worked as well as it did if it were not for how small the angles were. The hexapod reaches its physical limit of what it can stand on at an angle slightly beyond ten degrees. At smaller angles $tan(x) = x$ which why there was so little difference between the controlling the angle and controlling the compensation distance, at small angles they are both linear systems. In the end it was decided to proceed with the compensation distance control as it is linear no matter the angle and no future errors would occur because of it.

The goal has been to create a system that is fast but does not overshoot, which really is not a very uncommon goal for a control system. However, more

important than the design of the controller is the sampling and execution speed of the system. The angle variations of the hexapod are often minimal, often around a third of a degree, and happen quickly. As will be discussed further in Subsection 5.3 the errors accumulate which can in turn can make the hexapod unstable. Additionally, there is a lot of behaviour far beyond the scope of this project that creates variations and movement one would like to control for in a perfect implementation. The hexapod is not a rigid body and it has several joints and parts that are built in plastic.

Looking at the results, the speed and balance of the hexapod is beyond what I expected in the beginning of this project. It is clear that the system could be improved further but the work to do so would probably be a whole master thesis in itself. Some predictive control integrated with the movements of the legs could perhaps compensate for some of the spikes seen when lifting and putting down the legs.

Comparing the results in Section 4.1.2 it is clear to see that a P-controller with the constant of 0.2 is the most effective controller. The PD-controller was hard to implement without doing a deep analysis of the system dynamics and the PI-controller caused oscillations, seen in Figures 4.13 and 4.14.

5.2 Ground Angle Approximation

The first design used to approximate the angle of the ground below the hexapod was never intended as a final system. It was an experiment that showed potential. It was unexpected how much a leg hitting the ground too much was affecting the system.

The new system using distance measuring sensors is far superior. The comparison between the two methods are shown in Section 4.3. The graphs show that the average value is much closer to the real value and that the variance is lower when using the new method. The graphs also show the old approximation as much "smoother" than the new method. This could be remedied by applying a low-pass filter to the new method.

Given more time a circuit board could have been designed to allow for several distance measuring sensors to be connected to the BBB. However, the implementation in the V-REP simulation proved the concept.

The manual for the distance sensors was very misleading and vague in how it described several sensors working together, neither did any of the circuit schematics help. It at first seemed like one could solder certain connection on each resistor to set different addresses for each resistor but this was later found out to be false.

5.3 Step impact detection

The step control took a lot more time than anticipated. The original plan was to quickly set up the step control prepared and recommended by Malmros and Eriksson [Malmros and Eriksson, 2016] and then spend most of the time developing the balance system and perhaps implementing special obstacle and terrain handling features. When testing the resistors by themselves they were precise, and one could easily trigger them between two fingers. When set into the 3D printed frames and installed on the hexapod they were however completely useless. Weeks upon weeks of work went into trying to make the resistors detect the steps, changing the movements of the hexapod and trying to make the balancing system compensate for the errors. The resistors were functioning most of the time and there was nothing wrong with the 3D printed frames designed by Malmros and Eriksson, it was the fact that the resistors were so very unreliable. The resistors stopped working if they were bent toos much, if somehow the ventilation hole was blocked or if the fragile plastic got slightly damaged. All it took was for one of six resistors to not respond for a little while and a leg would try to push itself far into the ground. If they would not have worked at all it would have been easy to give up on them and find another solution, but the resistors worked at times. Sometimes the hexapod would walk perfectly for a while and then suddenly the resistors would stop responding. In hindsight, the resistor solution should had been abandoned much earlier.

Most of the work done in this project has been linked to problems and complications with hardware in one way or another. It has not helped that deploying a model onto the BBB can take several minutes, making it very time consuming to test new movements or control systems. The preparatory work for each test could be in the tens of minutes making for a very slow development process. A large part of this project has been to show what does not work, which is hard to show as a graph in the result chapter.

Replacing the resistors with mechanical buttons came with its own set of problems. If there had been more time to choose a more specialized button it probably would not have been as much of an issue, however it did lead into very interesting problems. Even if the buttons worked the errors of every step would accumulate, making the floor angle estimate drift. This led to the development of a new way to approximate the floor angle using distance sensors which in turn made it possible to remove the buttons and instead analyse the way the legs compensated for pushing against the ground in order to detect when it happened. This last development was done in a simulation and it has not been proven that it would work as well using the real hexapod. Looking at the data gathered from the real hexapod and comparing it the simulation would indicate that an impact detection as sensitive as the one implemented

in the simulation would be hard to achieve without increasing the execution rate of the step impact detection. The variance of the real system is higher than in the simulation and as seen in Figures 4.37 and 4.38 the step impact detection adds significant disturbances to the balancing. However, if one could speed up the rate at which the step impact detection is executed it could be possible to create a more complex system that could detect the difference between a leg hitting the ground and a common disturbance. The discussion about how to speed up the system is further discussed in Section 5.5.

One of the main problems with the impact detection was that there is no other way to determine if a leg has hit the ground and thus the system has to completely rely on the impact detection system. The robots only sense of touch is the sensors implemented for the impact detection. This is like a human walking blindfolded with completely numb legs.

5.4 Speed limitations

The calculated maximum speeds work very well when moving one leg at a time. However, the calculations do not try to predict collisions that could happen. This makes it impossible for the hexapod to achieve the theoretical maximum speed when lifting more than one leg at a time. The hexapod can move much quicker than anticipated and the maximum speeds given for using more than one leg at a time are incorrect if taken as the maximum that should be set for the hexapod. The speeds suggested by my calculations are far too high as legs taking a step risk colliding with the leg in front of it. Luckily, the anti-collision system that I designed avoids an actual collision, but also limits the maximum velocity. The goal of the calculations were never to be an exact maximum speed taking all the hexapods quirks and functions in consideration. The intention was to identify how the maximum velocity of the hexapod was affected by moving at an angle. The results are not false just because something else affects the maximum speed of the hexapod and I hope the observation that the angle has little effect on the movement of the hexapod when taking steps with several legs at a time will be of value to a future project.

5.5 Real-time analysis

The table shown in Figure 4.45 shows the most time consuming parts of the Simulink model's control system when using V-REP to simulate the hexapod. Since there is no way to improve the communication between V-REP and Simulink the real-time analysis focuses on the control system of the hexapod

in this project. However, it is important to mention that requesting and sending data between the simulation and the Simulink model takes more time than the whole control system. When running the model on the real hexapod the communication would not be expected to take as much time as in the simulation.

In Figure 4.45 we can see the function *TrajectorySystem* inside *MainController* taking approximately 8.5% of the total execution time of the whole *Control* system. Since the balance control, step trajectory calculations, the leg collision avoidance, step impact detection, floor angle estimation, height control, and leg movement calculations using the floor angle estimate are all part of the *TrajectorySystem* function it is not very surprising that it takes a substantial amount of time to execute. What is surprising is the fact that the inverse kinematics uses more than 50% percent of the total execution time of the control system. This is why the inverse kinematics designed by Thilderkvist and Svensson are by far the most significant bottleneck of the control system.

The speed of the system is sufficient for keeping the hexapod balanced but in order to improve the step impact detection it is necessary to increase the sample rate of the IMU and key components of the control system.

The system implemented on the real hexapod samples the IMU data at twice the rate than that of the control system. However, the sample rate is still too low to allow filtering or averaging. The balance controller is not the problem however as it works quite well at the current execution rate.

The subsystem most affected by the sample rate is the step impact detection. The impact detection needs to react to fractions of a degree to not create significant errors. The step impact detection that was designed for the simulation model detects when the compensation of the legs reaches a threshold. Even though several other more complicated designs have been tried the execution rate is simply too low to be able to effectively distinguish the difference between a leg pushing against the ground from a small vibration.

By separating the impact detection from the control system, it could be possible to design more advanced detection systems, such as integrating the angle signal so that the constant error that is caused by a leg pushing into the ground could be separated from ordinary disturbances.

As Thilderkvist and Svensson claim that their system already used 50% of the BBB processing power [Thilderkvist and Svensson, 2015] the first priority before trying to make a faster system would be to make the current one more effective. By pre-calculating the inverse kinematics in a similar manner to what this project did with the maximum speed dependent on angle one

could significantly lower the execution time. Additionally, a study of the limits to the leg movements would make it possible to avoid trying to move the legs to unreachable positions and make for a more robust hexapod.

It is unclear if it is necessary for the whole control system or just key functions to be executed at a higher rate. The precision of the movement of the legs is directly affected by the execution rate. The legs move at the speed set by the controller but if it is possible to send instructions to the motors more often it is possible to send smaller changes so that when we detect a leg hitting the ground the motors won't continue running the old instructions. In order to be able to stop the legs' movements before they cause significant errors the motors of the legs most be able to be stopped quickly.

To be able to say how quickly the step detection needs to be run and if it is possible to run the leg movements at a slower pace a much more detailed investigation of the system dynamics of the hexapod and the communication between the BBB and the ArbotiX-M would have to be made.

5.6 Simulation

V-REP has been an invaluable tool in this project. When the access to the hexapod robot became limited as the project was taking more time than anticipated V-REP made it possible for development to continue. Installing the distance measuring sensors took an afternoon in V-REP and as Simulink was run on the same computer as the simulation and no longer being deployed on the BBB it became much easier to work with the code.

Yet, it was quite late in the project that the simulation began to be used. It was not just that a lot of the early problems were related to the hardware it there was also a goal of developing the new balancing close to the hardware restrictions of the hexapod. My understanding of Malmros and Eriksson's project was that they began designing their system using V-REP to simulate the hexapod, when they later moved their system to the real hexapod it could not run as intended as the had implemented image analysis and algorithms that required processing power far beyond what the BBB supplied. I wanted to avoid the same fate for my project but in hindsight there would only have been to gain from using V-REP earlier in the project.

The comparison between reality and the simulation in Section 4.2 unfortunately shows a slight difference between the two. The variance when using the real hexapod is about 2-3 times larger than when using the simulation. This prompts caution when implementing the latest step impact detection system on the real hexapod. However as explained in Sections 5.3 and 5.5 there are solutions to the potential complications.

The physical hexapod is far more complicated than the simulation but that does not always make the simulation less susceptible to disturbances. There is a completely different weight and rigidity to the hexapod in the simulation, making it very easy for it topple over if not completely balanced. This has made the impact detection control implemented in the simulation more sensitive than I believe it would have to be if implemented on the real hexapod robot.

5.7 Further Work

The first step to improve the system would be to redesign the inverse kinematics used to transform the leg coordinates to the motor instruction. This could be done by pre-calculating all the possible positions for the legs which would drastically reduce the computational need. Making the control system execute at a higher rate is essential for any improvements to the balance and terrain handling capabilities of the hexapod.

Doing a more in-depth analysis of the system dynamics could make it possible to design a more specialized balance control but should not be prioritized over improving the step impact detection. The step impact detection is what makes it possible to move over uneven ground and not just tilted flat surfaces. By either moving out the step impact detection from the control system and making it execute at a higher rate or making the whole control system execute at a higher rate should make it possible to use methods such as integration to discern the constant disturbance of a leg pushing into the ground from ordinary changes.

The speed angle model could be further improved as it does not take in regard that the legs can collide. It could be possible to use the pre-calculated inverse kinematics discussed earlier to create a more holistic model.

Another future improvement would be to install the distance measuring sensors on the bottom of the hexapod as it would make the floor angle estimate more precise and make it possible use the height control used in the simulation possible on the real hexapod.

Bibliography

BBB Picture. URL: https://beagleboard.org/static/ti/product_detail_black_sm.jpg.

Embedded Coder Manual for Matlab and Simulink. URL: https://se.mathworks.com/help/pdf_doc/ecoder/ecoder_gs.pdf.

Embedded Coder Support Package for BeagleBone Black Hardware. URL: https://se.mathworks.com/matlabcentral/fileexchange/47891-embedded-coder-support-package-for-beaglebone-black-hardware.

Force Sensitive Resistor Data Sheet. URL: http://interlinkelectronics.com/datasheets/Datasheet_FSR.pdf.

IMU Picture. URL: https://cdn.sparkfun.com//assets/parts/7/3/7/6/11486-04.jpg.

Inertial Measurement Unit Data Sheet. URL: https://cdn.sparkfun.com/datasheets/Sensors/IMU/MPU-9150-Datasheet.pdf.

Malmros, M. and A. Eriksson (2016). *Artificial Intelligence and Terrain Handling of a Hexapod Robot.* Master's Thesis ISRN LUTFD2/TFRT–9999–SE. Department of Automatic Control, Lund University, Sweden.

Mechanical Button Data Sheet. URL: https://www.omron.com/ecb/products/pdf/en-b3f.pdf.

Picture ilustrating yaw, pitch and roll. URL: https://commons.wikimedia.org/wiki/File:Yaw_Axis_Corrected.svg.

Årzén, K.-E. (2014). *Reak-Time Control Systems.* Department of Automatic Control, Lund University, Sweden.

Servo Data Sheet. URL: http://support.robotis.com/en/techsupport_eng.htm#product/actuator/dynamixel/ax_series/dxl_ax_actuator.htm.

Thilderkvist, D. and S. Svensson (2015). *Motion Control of Hexapod Robot Using Model-Based Design*. Master's Thesis ISRN LUTFD2/TFRT- -5971- -SE. Department of Automatic Control, Lund University, Sweden.

VL6180 Guide. URL: `https://learn.sparkfun.com/tutorials/vl6180-hookup-guide`.

Lund University
Department of Automatic Control
Box 118
SE-221 00 Lund Sweden

Document name
MASTER'S THESIS

Date of issue
January 2018

Document Number
TFRT-6047

Author(s)
Jonatan Ekelund

Supervisor
Jasper Gundersen, Combine Control Systems
Anders Robertsson, Dept. of Automatic Control, Lund University, Sweden
Martina Maggio, Dept. of Automatic Control, Lund University, Sweden (examiner)

Title and subtitle
Balancing and Locomotion of a Hexapod Robot Traversing Uneven Terrain

Abstract

This master thesis project has developed a method for a six-legged robot to traverse uneven terrains. From a previous project, the robot already possessed basic locomotion which had to be extensively redesigned. Several new subsystems have been introduced, such as a continuous balancing system, a system estimating the slope of the ground the robot is walking on, and method for detecting when the robot's legs hits the ground.
This project designed a system that lets the robot balance its body as it moved using an inertial measurement unit. Several different controllers have been tested and compared. A lot of work has gone into gathering data and identifying how different controllers and sampling rates help the system.
To move across uneven terrain it is important to estimate the tilt of the ground that the robot is moving on. This estimate has been made by integrating the angles from the inertial measurement unit. However, a superior solution using distance measuring sensors has been developed in a simulated environment. The distance measuring sensors also made it possible to control the height of the robot more accurately than before.
A large part of the project has been spent on testing and comparing different methods of detecting when the legs of the robot hit the ground. No satisfactory solution was found using buttons or force sensitive resistors. Instead the legs' movement deviations were used to detect when a leg touched the ground. This was then successfully implemented in simulation.
Several new features, unrelated to the balancing but improving the movements of the hexapod, have been developed and implemented.
Also, a mathematical model for the maximum speed of hexapod walking on sloping ground has been designed.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
1-91

Recipient's notes

Security classification