Motion Control of Hexapod Robot Using Model-Based Design

Dan Thilderkvist

Sebastian Svensson

LUND UNIVERSITY

Department of Automatic Control

MSc Thesis
ISRN LUTFD2/TFRT--5971--SE
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 2015

Abstract

Six-legged robots, also referred to as hexapods, can have very complex locomotion patterns and provide the means of moving on terrain where wheeled robots might fail. This thesis demonstrates the approach of using Model-Based Design to create control of such a hexapod. The project comprises the whole range from choosing of hardware, creating CAD models, development in MATLAB/Simulink and code generation. By having a computer model of the robot, development of locomotion patterns can be done in a virtual environment before tested on the hardware.

Leg movement is implemented as algorithms to determine leg movement order, swing trajectories, body height alteration and balancing. Feedback from the environment is implemented as a internal measurement unit that measures body angles using sensor fusion.

The thesis has resulted in successful creation of a hexapod platform for locomotion development through Model-Based Design. Both a virtual hexapod in SimMechanics and a hardware hexapod is created and code generation to the hardware from the development environment is fully supported. Results include successful implementation of hexapod movement and the walking algorithm has the ability to walk on a flat surface, rotate and alter the body height. Implementation also contains a successful balancing mode for the hexapod whereas it is able to keep the main body level while the floor angle is altered.

Acknowledgements

Thanks go out to the Department of Automatic Control at the Faculty of Engineering, LTH, part of Lund University, for providing tools and a workshop. Special thanks goes to our supervisor at the department, Anders Robertsson, who guided us through the project and taken the time for meetings on a weekly basis. We would also like to acknowledge Combine Control Systems AB for providing an office to work in and the hardware to work with. We also like to thank our supervisor at Combine, Simon Yngve, who guided us in the usage of Simulink and MATLAB, but also had meetings with us and helped us structure the project. Lastly we would like to thank Fredrik Habring and Juan Sagarduy at MathWorks for answering questions regarding software and provided solutions to related issues.

Contents

1

Introduction

1.1 Background

A common workflow today for developing control systems is using Model-Based Design. An interesting way of testing the limitations of Model-Based Design is to use it in different projects. Combine Control Systems AB in Lund has shown interest in using this method to develop control of a hexapod.

By using it to develop locomotion and movement patterns for a hexapod it will be possible to evaluate the benefits of such an approach in robotics. If results are positive, Combine can then use the developed system to showcase the business idea of Model-Based Design.

Combine Control Systems AB has business partners in software development such as MathWorks and National Instruments. The nature of their business model presents them with several opportunities to attend technical fairs where they showcase their business idea.

A hexapod (from the Greek hex for "six" and pous for "foot") refers to a six legged robot. Hexapods have recently become increasingly popular and are now available to fairly cheap prices. Combine has previously shown videos of existing thesis work [Ohlsson and Ståhl, 2013] on a hexacopter. Due to the dangers of having high speed rotating blades close to people the hexacopter could never be brought to fairs. This is one of the reasons Combine has shown interest in a safer system like a hexapod.

Due to the novelty of the project, no hardware existed at Combine prior to this work. Therefore part of the thesis involves choosing a hexapod platform. A hexapod platform usually consists of the six legged robot, a micro-controller, a wireless handheld commander and an open source control program. Model-Based Design and automatic code-generation is to be used in order to replace the supplied control algorithm and thus be representative of modern control implementation. Combine Control Systems AB has a lot of cooperation with MathWorks and as such plenty of knowledge and support on their software. Simulink and Embedded Coder has good support for code-generation to micro-controllers such as Arduino, Raspberry Pi and

BeagleBone Black. This provides good opportunities to develop control algorithms in Simulink and use SimMechanics for a visual representation.

One of the strengths of walking hexapods is their potential to handle uneven terrain. With good and adaptive control they have an advantage over wheeled vehicles in hard-to-manoeuvre areas. An algorithm for handling terrain is to be developed and tested based on accelerometers and gyroscope placed on the hexapod body.

1.2 The hexapod

The hexapod used in this thesis is a PhantomX AX Hexapod Mark II, see Figure 1.1. It is developed by Interbotix Labs and is a second revision of their hexapod PhantomX. Using three servos per leg up to a total of 18 servos all together the PhantomX provides 18 degrees of freedom. The processing unit is an Arduino based board called ArbotiX-M using a 16MHz ATmega644p procesor. The ArbotiX-M communicates with the 18 Dynamixel AX-12A Servos over a half duplex UART serial communication channel. This communication is specified to 1 Mbit per second but can be scaled down. A handheld remote called ArbotiX Commander communicates wireless with the hexapod using XBee modules. These XBee modules can also be used to communicate with the robot through a computer with a virtual remote.

The retailer, Trossen Robotics recommends an open source software engine for the hexapod called NUKE. It is a fairly small (17.41 kB) program written i C/C++ that can be downloaded to the processor through the Arduino IDE. Full functionality is provided to the hexapod with the NUKE Engine. Forward/backwards mobility, sidestep, turning and changing locomotion pattern (also called gait) can be controlled from the handheld remote. The different gaits that can be chosen are movement using one, two or three legs simultaneously.

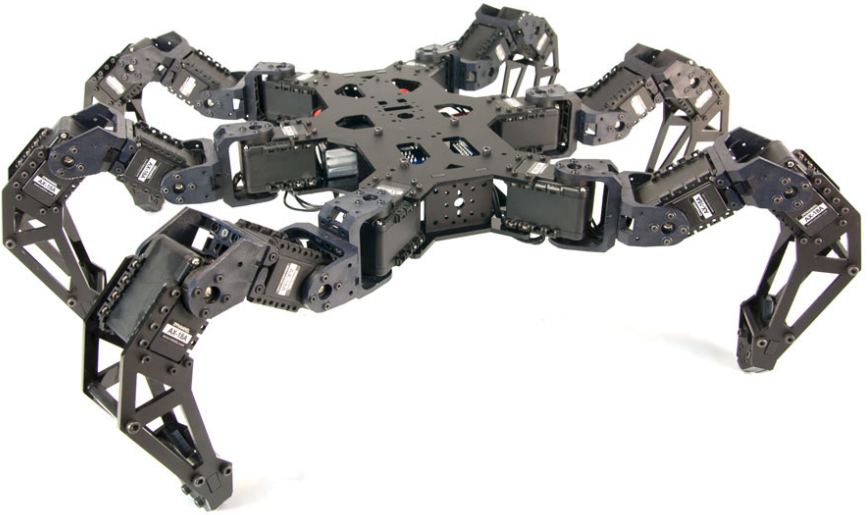

Figure 1.1 PhantomX AX Hexapod Mark II as retailed by Trossen Robotics [*PhantomX AX Hexapod Mark II Kit*].

1.3 Goal

The main goal of the thesis is to program a walking hexapod robot using Model-Based Design. This is supposed to highlight the strengths of working with a computer model as a first testing stage. In today's industry it is a lot more cost and time efficient to test control methods on a computer model of a plant before testing on a real plant [Ahmadian et al., 2005]. By having both a computer running the model and the real hexapod at a fair, Combine will be able to show their business model in an advantageous way. Therefore, the visual aspect of the computer model is of importance as well.

Since the project was started from scratch the first milestone was to find a suit-

able hexapod platform. Criteria for a good hexapod platform are several. It is preferred for the processing unit to have good support through Simulink and Embedded Coder. The hexapod is preferred to be visually similar to an arthropod and for it to be able to navigate terrain a high ground clearance is important. In order to avoid limiting the control, servos with higher speed and update frequency were desired. As generated code is fairly unreadable, a processing unit that offers debugging during execution is highly preferred. To avoid unnecessary system-latency, as few hardware components as possible were preferred. This will also facilitate further thesis work on the platform.

In order to visualize the model in SimMechanics a CAD model of the hexapod were constructed using SolidWorks. Control for the hexapod movement created in Simulink could then be used on both the computer model and the hardware platform. Since the same controller runs on both model and hardware, verification of the model could be done by applying the same input to both and comparing the results.

As a scientific challenge, a final aim was to have the end product hexapod navigate uneven terrain. With help of on-board accelerometer, gyroscope and magnetometer the movement and orientation of the main body is to be measured. From this measurement the possibility to identify objects and walls may exist. Utilizing this information the walking pattern could adapt to the environment.

1.4 Model-Based Design method

Traditionally system development consists of several steps. Usually a team of system engineers define and create system specifications. These are handed as documents to the software engineers that interprets them and implement software code in the preferred language. The next step is then to test the implementation on hardware. Usually a lot of errors are first realized at the testing stage, but have to be corrected all the way back to phase one. Not until testing has been successful can the system go into production. [Ahmadian et al., 2005]

Model-Based Design is a modern way of improving upon this approach. In order to cut down on both developing costs and time, modern companies can use software like Simulink to design and build both plant models and control in a virtual environment. By using Model-Based Design developers can create systems by using mathematical models of parts and their interaction with the environment [Ahmadian et al., 2005]. This can eliminate a lot of errors from interpretation between system designers and software designers. It also opens up a lot more options for the developer as access to a virtual test plant let them test ideas without committing to hardware. Systems developed using Model-Based Design also make it easier to reuse working systems. As an example, Dongfeng Electric Vehicle (DFEV), part of Chinese Dongfeng Motor Company managed to create a battery management system for buses using Model-Based Design. This was done in less than 18 months with

only a six person team with different engineering backgrounds because they could develop it all in Simulink using test data. The same system has later been reused for developing battery system for their sedan model [GreenCarCongress, 2009]. The ability to reuse systems and created code is one of the advantages for Model-Based Design compared to the more conventional methods.

Another advantage of modern software for Model-Based Design is the ability to generate code automatically from the model. This way it is possible to eliminate manual code errors and enable Model-Based Design through all steps towards the target system [Combine, 2013]. Code generation also closes the door for human interpretation of design parameters and opens up several options for code optimization. This way of reducing the amount of steps needed to production results in better project managment and makes products reach the market faster [Ahmadian et al., 2005]. Automakers at the SAE Convergence 2010 conference in Detroit claims that Model-Based Design has saved them 40 percent in development and 60 percent in validation. The ability to reuse C code seamlessly has become a great asset for new products in the automotive industry [Murray, 2010].

2

Theory

2.1 Modelling and Verification

The Simulink environment

Simulink is a software where models are constructed by connecting different blocks in a GUI. The blocks can be grouped into different subsystems to create different abstraction levels of a model. For the ability to reuse subsystems, they can be placed in libraries. With use of libraries it is possible to update all linked subsystems in one place.

When the block model has been constructed it can run in several different modes. Commonly used simulation modes are normal, Software-in-the-loop (SIL), Processor-in-the-Loop (PIL) and external mode. Normal mode is used when running simulation on a computer. When code generation is used SIL and PIL can be used to evaluate code generated from a Simulink model. In SIL mode code is generated and run on the same computer used for simulation in normal mode. PIL mode differs from SIL mode because code is generated for the real hardware e.g. an embedded system. The generated code is downloaded and executed on the target hardware. SIL and PIL simulation modes can be used for the whole model or only some subsystems in a model. A combination of SIL and PIL blocks can be used to compare performance between code running at target hardware. In external mode code for the model is generated and deployed to the target hardware. During simulation signal data for the hardware can be sent to the computer to monitor different signals. When using code generation, code for a model can also be deployed to hardware as a standalone application. The model can be started, restarted and terminated from MATLAB command line.

SimMechanics

SimMechanics is a software developed by MathWorks to model mechanical systems. A model of a system is built up using solid bodies, which are connected to coordinate frames. These coordinate frames are then connected to each other using rigid transforms or joints. Joints allow different degrees of freedom (DOF) between

two frames while rigid transforms allow zero DOF. The two frames that connected a joint are known as base and follower frame. Joints can be actuated using torque or motion input. The follower frame then moves relative to the base frame. When motion actuation of a joint is used the block is fed with position, velocity and acceleration as a function of time. It is possible to sense motion and torque for each joint.

To illustrate how SimMechanics works a model of a servo is shown in Figure 2.1. A visualization of this system can be seen in Figure 2.2. This servo model consists of two solid bodies which are connected by a revolute joint. This joint allows the servo horn to be rotated relative to the servo body.

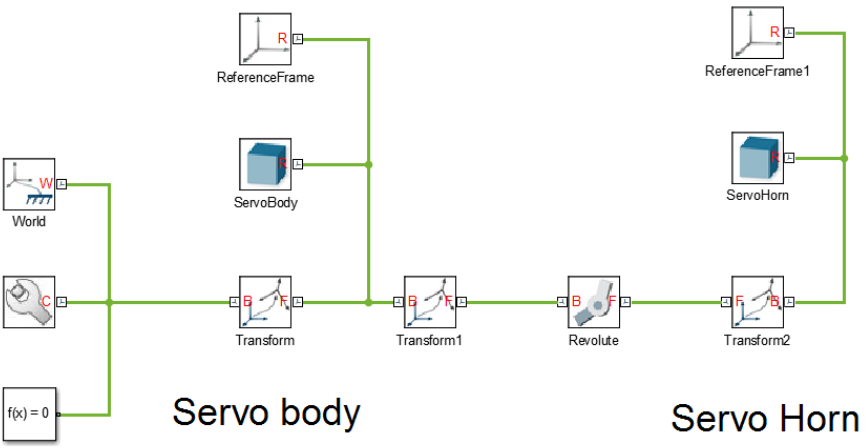

Figure 2.1 SimMechanics model of a servo. Two solid bodies, body and horn, are connected using a revolute joint.

To each solid body physical properties such as mass, center of mass and moments of inertia can be assigned. These properties are then used when simulating the system.

To improve modelling capabilities in SimMechanics, MathWorks has developed a tool called SimMechanics Link. This tool allows the user to export CAD models and then import them into SimMechanics. From the CAD model, visual appearance, physical properties and coordinate frames are imported.

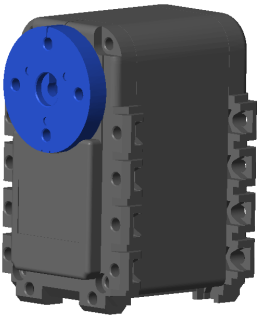

Figure 2.2 Visualization of servo model in Figure 2.1. Servo body (gray) and the rotating disc, the servo horn (blue).

Modelling of physical systems

Three commonly used methods for modelling physical systems are white-, grey- and black-box modelling [Leifsson et al., 2008]. The difference between these methods is the amount of knowledge that is known about the physical systems. In white-box modelling the inner structure of the system is known and modelled by using first principle like e.g., Newtons second law of motion (2.1). The opposite method to white-box modelling is known as black-box modelling. Using this approach the system is modelled as an input/output system see Figure 2.3. Input and output data are measured from the system and then the functional structure and parameters for those are estimated. Functional structure could be determined by use of different transfer functions see e.g. (2.2). The order and placement of the poles and zeros of the chosen transfer function are then determined to match measured data.

However models often fall somewhere in-between white- and black-box models and are then referred to as grey-box models. When using grey-box models, the functional structure of the system is known. For example, consider a simple low-pass filter in Figure 2.4. The transfer function for the system is known (2.3) but the capacitance C and resistance R need to be determined.

$$\mathbf{F} = m \cdot \mathbf{a} \tag{2.1}$$

$$S_1(s) = \frac{s+a_1}{s+b_1}, S_2(s) = \frac{1}{s+b_1}, S_3(s) = \frac{1}{(s+b_1)(s+b_2)} \tag{2.2}$$

$$S(s) = \frac{1}{1+RCs} \tag{2.3}$$

Figure 2.3 A representation of black-box model.

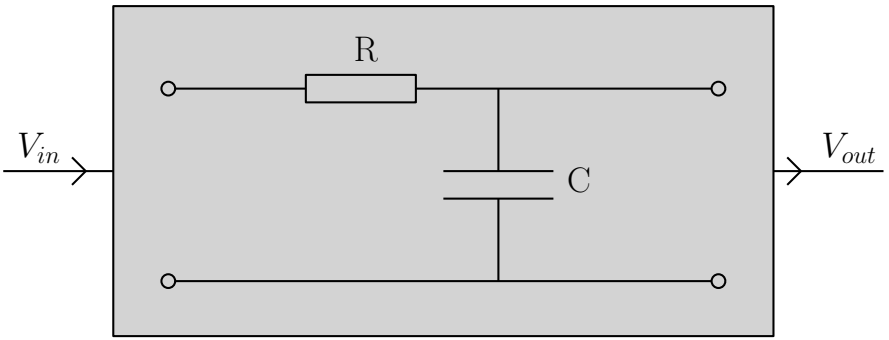

Figure 2.4 A representation of a low-pass filter which can be modelled as a grey-box model.

2.2 Walking theory

In order for the hexapod to walk, several algorithms need to work together to form a complete controller. The end product at every time interval is the position set-point for each servo. Walking patterns need to be chosen, swing trajectories calculated and leg position constraints updated.

Depending on velocity, different gaits are selected by a controller. To execute these gaits each leg will have a stand phase and a swing phase [Campos et al., 2010]. Whereas the stand phase is when the leg has ground contact at all time. During the swing phase a trajectory between two stand positions must be properly calculated by the controller. Due to size of hardware such as leg length, servo positions and body width, certain constraints will restrict the possible leg positions. The positions of each leg will also affect remaining legs possible position space.

Due to resemblance between a hexapod robot and legged insects a lot of inspiration can be taken from insect locomotion and biometrics.

Gaits

To move a hexapod in any direction the legs has to push it in that way, resulting in legs getting further away from the hexapod body. In order for this to continue the legs have to be lifted and moved back into the vicinity of the body. This can be done

in several different ways and possibilities increases with the amount of degrees of freedom [Ridderström, 2003]. The most common way of creating gaits is by manual programming [BELTER and SKRZYPCZYŃSKI, 2010]. More sophisticated methods exist and some of them include mimicking stick insects [Fielding, 2002], evolving patterns using genetic algorithms [BELTER and SKRZYPCZYŃSKI, 2010] or using artificial neural networks [Dürr et al., 2004]

Some of the most common gaits used are metachronal, ripple and tripod gait. They are used for slow, medium and fast movement respectively [Campos et al., 2010]. The basic difference between these gaits are the usage of one, two or three legs simultaneously in swing phase, Figure 2.5.

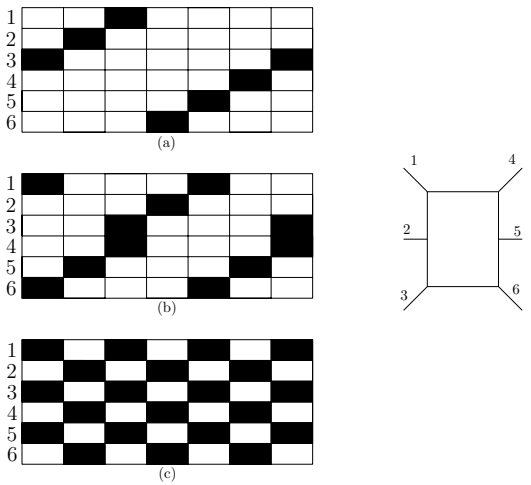

Figure 2.5 Gait diagram for three common gaits that can be implemented. Legs are numbered from front to back with 1-3 for left legs. Black symbolises swing phase. (a) Metachronal, (b) Ripple, (c) Tripod.

The metachronal gate provides the most stability due to more legs on ground [Campos et al., 2010]. It can be described as a back to front propagating gait, moving one leg at a time. Each leg has a separate swing cycle adding up to 6 different cycles until the hexapod has returned to the start state. Ripple gait is very similar to metachronal but allows for a faster movement speed. Back to front propagation is done simultaneously on each side with one pause cycle. Diagonal legs swing simultaneously and the middle legs swing during a pause for the other side. This adds up to four cycles with the sides two cycles offset of each other. Tripod gait allows for the fastest movement and only contains 2 cycles. Ipsilateral anterior and posterior legs, and the contralateral middle leg combined works as two separate tripods. One tripod swing while the other keep the balance of the tripod.

To allow the hexapod to turn, the used gait have to be modified during turning.

In order for insects to turn there exist some methods that can be used for hexapods as well. The basic method consists of modifying the step length on one side [Fielding, 2002]. By decreasing/increasing step length, one side will move slower/faster and thus allow for turning. Another common method is to lower swing frequency on one side [Fielding, 2002]. By lowering the frequency so that one step is lost on the inside, insects can achieve a turn of around 20 degrees [Fielding, 2002]. For tight turns and turning on the spot a combination of the two is common. Also legs stepping backwards allow for very tight turns or easy turning on the spot. Another way for a hexapod to turn similar to step length alteration is to rotate the legs around the body center. Rotating legs on ground around the main body center will push the main body in a rotational manner. In order for rotation to work it is important that legs rotate at equal angular velocity and around the same rotational center-position (main body center, or z-axis in the coordinate system). When legs get too far out of position they can simply be brought back into position by their swing phase. Means for rotating in a Cartesian coordinate system are rotation matrices R (2.4) [Spong et al., 2006, p. 42].

$$R_z(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.4}$$

Rotation around the z-axis can be achieved by θ degrees by multiplying an arbitrary point by (2.4).

Amongst insects, different behaviours have been observed for starting, stopping and standing still. Most insects tend to stop with their legs on ground, but observations where legs are left in air or reversed exist [Fielding, 2002]. Some insects also tend to rearrange legs at a stop in a symmetrical and stable way. Though this entails the need for a couple of steps before normal walk can be coordinated [Fielding, 2002]. Rearranging legs to a symmetrical and stable position for the hexapod requires the knowledge of such a position.

Inverse kinematics (IK)

To achieve leg movement, the angles for each servo needs to be determined. This can be done using inverse kinematics.

Inverse kinematics is commonly used in control of different robotics applications [Spong et al., 2006, p. 54]. Inverse kinematics calculates, for a given point in space, the angles for each joint. A simple inverse kinematic problem is shown in Figure 2.6. A servo located at the origin has a link l_1 attached to it. The problem is then to calculate the angle α required to position the tip of l_1 at point p_1. The solution to this problem is (2.5) were (x_1, y_1) are the coordinates of p_1.

$$\alpha = \arctan\left(\frac{y_1}{x_1}\right) \tag{2.5}$$

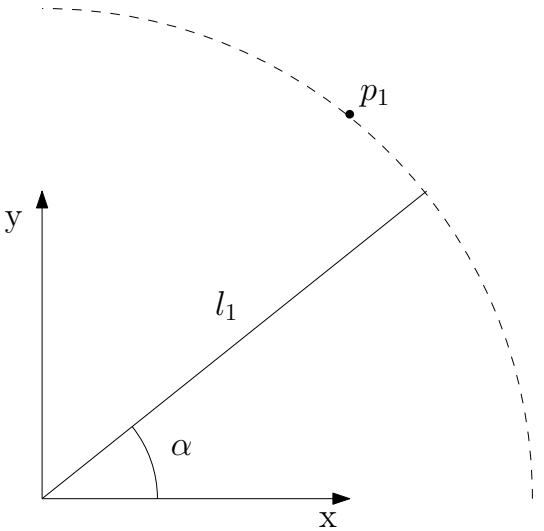

Figure 2.6 A simple inverse kinematics problem.

Trajectory generation

Legs of the hexapod have two states, stand and swing state. The difference of these states is if there is ground contact or not. As the name implies, the stand state has ground contact and pushes the main body whereas the swing state has no ground contact and legs swing into position. These states need trajectories of how they are supposed to move. One option for this is to send trajectories to each servo from a controller as done by [Campos et al., 2010]. With an inverse kinematics algorithm though, the servo trajectories will be calculated by the algorithm, using one trajectory for each leg. The only trajectories needed then are the one trajectory for each leg telling the IK where to put the leg.

Stand trajectories are easily generated by using the desired velocity. In order to move the hexapod, the legs need to move in the direct opposite of the desired velocity direction. Using that fact, one can calculate next position P_i^{t+1} for each leg in stand phase by using its current position.

$$P_i^{t+1} = P_i^t - \frac{v}{f} \tag{2.6}$$

Whereas t is the discrete time index, i is the leg number, P is position for leg i at time t in the main body coordinate system, v velocity and f the current sample frequency of the controller. The frequency is there to normalize the movement in order to keep the hexapod moving velocity from being affected by frequency changes.

During the swing phase the controller needs to lift the leg, move it in the direction of hexapod movement and then put it down again. As suggested by [García-

López et al., 2012], it is possible to do a trajectory in the form of an ellipse, using only the upper half. The advantage of a circular movement when creating a lift trajectory is that the leg will be fairly elevated before further movement in the walk plane. In this way it will be easier to avoid potential object collisions during the swing phase. Using the equation for an ellipse (2.7).

$$\left(\frac{x}{a}\right)^2 + \left(\frac{y}{b}\right)^2 = 1 \tag{2.7}$$

The constant a determines the distance from the ellipse center to the cross section between the ellipse and the x axis and b determines the corresponding distance on the y axis. One can get the equations for the separate coordinates that can be used to generate an ellipse (2.8).

$$\begin{cases} x &= a \cdot \cos(\theta) \\ y &= b \cdot \sin(\theta) \end{cases} \tag{2.8}$$

Using θ in the interval [0°, 180°] will generate the upper half of an ellipse. By translating such an ellipse trajectory to the current leg position the constants a and b can be used to shape the swing. The horizontal length of a trajectory will be twice the size of a and the maximum height of a swing will be equal to b.

Movement smoothing

The nature of the servos chosen is these they will move between two reference angles with a constant angular speed. Maximum angular speed is a configurable parameter for the servos [Robotis, 2006]. Servo identification shows that maximum angular speed corresponds to the used constant angular speed if not effected by very heavy loads. Though even for heavy loads the angular speed is still constant if the load is constant. The hexapod legs are designed in such a way that when they move between two leg positions not all three servos move the same angular distance. If all servos move at same speed this will introduce a jerk movement due to servos reaching their reference value at different times. In Figure 2.7 the servos of one leg are shown with theoretical set points and theoretical expected trajectories due to predefined angular speed. The problem can be seen here as servos reaching their set point before the next sample and thus need to stay stationary.

The lower the update frequency for the position set points, the worse the jerk movement is expected to be. This is due to the higher possibility of servos reaching their destination prior to the next sample. Default setting for the servos are to move with physically maximum angular speed (no speed limit) [Robotis, 2006].

The servos offer the ability to change maximum angular speed on-line in the same way as they receive position reference [Robotis, 2006]. This provides a potential solution to reducing the jerky motion. If the set point update frequency is known, the time until next update will be known. Therefore the maximum angular

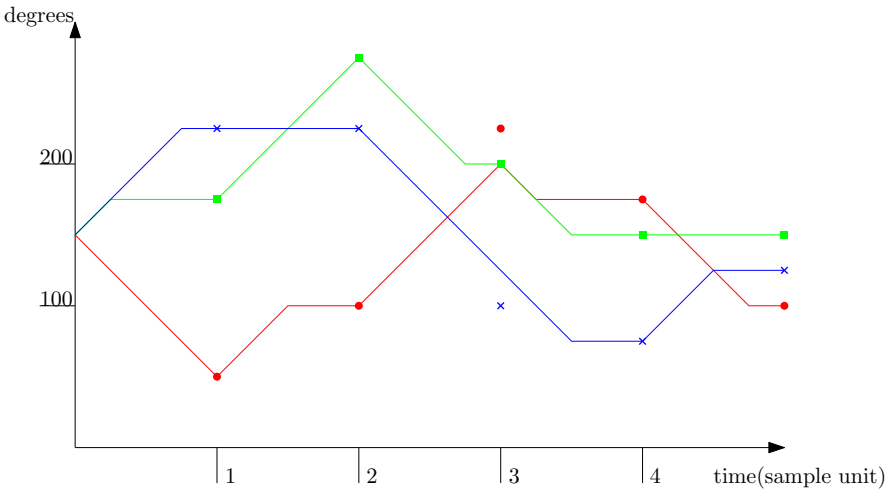

Figure 2.7 Theoretical trajectories for the three servos of a leg with constant speed. The angular speed of the servos is set to 100 degrees per sample (fairly exaggerated) to show that servos reaching their set-point prior to next sample will have to wait.

speed can be set so the servo reaches the set point at the time of next update. A theoretical expectation of the result can be seen in Figure 2.8.

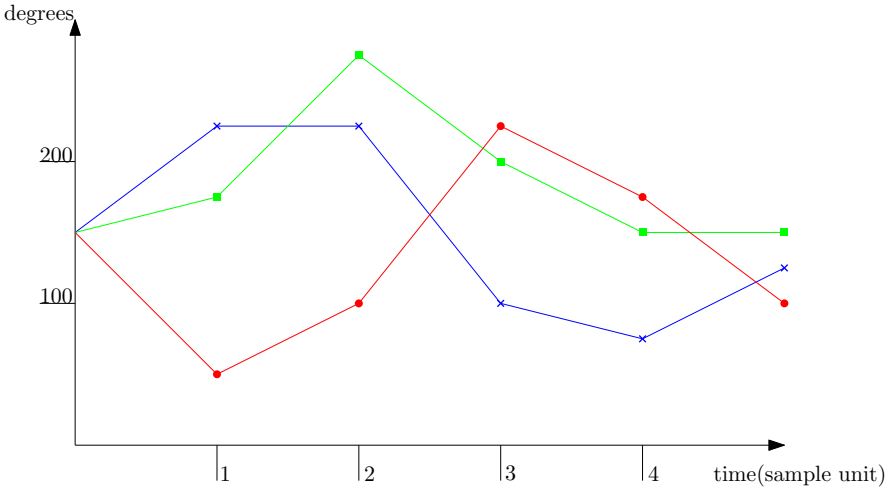

Figure 2.8 Theoretical trajectories for the three servos of a leg with dynamical speed. The angular speed is here calculated at each sample so the servo will reach its set point at the time of next sample.

Another advantage of setting the speed of servos to adapt according to the set-point is that the full leg trajectory will be a straight line between set points. If a leg set-point is given to the IK and servo speed is not set, the IK will calculate servo set-points that might be reached desynchronized resulting in leg trajectory between samples being a curve between two set points. Servos are instead set to reach their set-point simultaneously by using individually set maximum speed. The resulting leg trajectory will then be a straight line between the two set-points.

2.3 Handling terrain

Terrain handling

To handle terrain, obstacles need to be detected. Several different approaches can be considered to handle this problem. Different kinds of distance measurement units can be used e.g. IR- or ultrasonic-sensors. Successful work using this approach has been done on a 12 dimensional hexapod RHex [Lin et al., 2006]. Another approach is to use image analysis [Wei et al., 2012]. Image analysis methods can give a detailed estimation of the environment at the expense of requiring large amount of computation. Due to limited resources in our system, image analysis methods were not investigated further.

Once an obstacle is detected it needs to be taken into account when generating new trajectories for each leg. Due to this, an estimation of the height of the obstacle needs to be fed to the trajectory generation.

Body stabilization

In this thesis, body stabilization is defined as the ability to control roll and pitch angles of the hexapods main body. Euler angles [Spong et al., 2006, pp. 53-57] is one method of measuring angles of a body in 3 dimensional space. In Figure 2.9 one way of defining Euler angles is presented. This definition for roll, pitch and yaw angle is used in this work. The IMU placed on the main body has accelerometer, gyroscope and magnetometer. These sensors are used for sensor fusion to calculate the Euler angles to the controller.

A desired state for the main body can be to stay level independently of the terrain layout. In order to achieve this the controller can compensate the leg positions when deviations occurs, see Figure 2.10. The Figure shows two different ground angles and corresponding hexapod leg positions to keep it level.

In order for the hexapod to stay level, it needs to alter the heights of the legs on both sides. By using trigonometric functions this can be calculated (2.9).

$$\Delta h = a \cdot \tan(\alpha) \tag{2.9}$$

In the equation Δh is the difference in height for a leg due to ground angle, a is the distance from body center to the leg position along the y axis and α is the ground

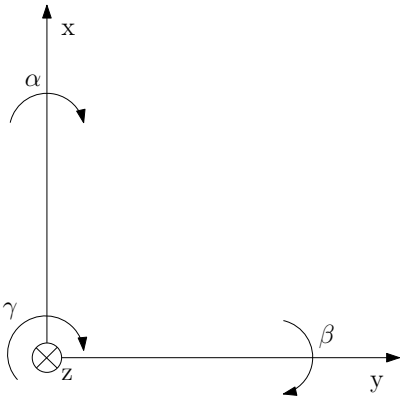

Figure 2.9 Euler angles roll α, pitch β and yaw γ. The arrows defines positive rotation.

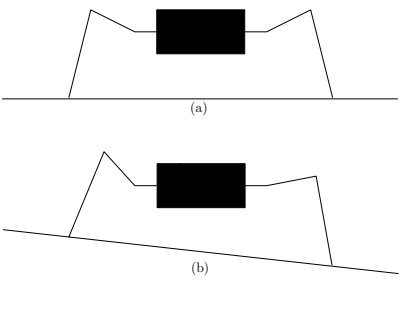

Figure 2.10 (a) Hexapod standing on level ground keeping the main body leveled. (b) Hexapod standing on a slope keeping the main body level by altering the heights of the leg positions.

angle. The same equation holds along the x axis with the use of the β angle. Doing this for both angles α and β for all legs, the hexapod main body will stand level if the ground angles changes (under the assumption that the hexapod is standing still).

2.4 Code generation

One benefit of using Model-Based Design is the ability to generate code automatically from Simulink models [Lambersky, 2012]. Using automatic code generation, hand coding errors and development time can be reduced [Ahmadian et al., 2005]. Code generated from a Simulink model in this thesis used two MathWorks products, namely Embedded Coder and Simulink Coder.

The workflow for Simulink Coder is shown in Figure 2.11. From the Simulink model a .rtw file is generated. This file contain a text description of the Simulink model. Simulink uses this file and a .tlc file to generate C or C++ code and makefiles. By changing the .tlc file it is possible to change the structure of generated C or C++ code for the model.

The other product (Embedded Coder) allows optimization of generated code by using different objectives e.g., RAM usage, execution efficiency or ROM usage. Embedded Coder is integrated with Simulink Coder as shown in Figure 2.12. Embedded Coder also makes it possible to set up a custom toolchain when generating code to target hardware.

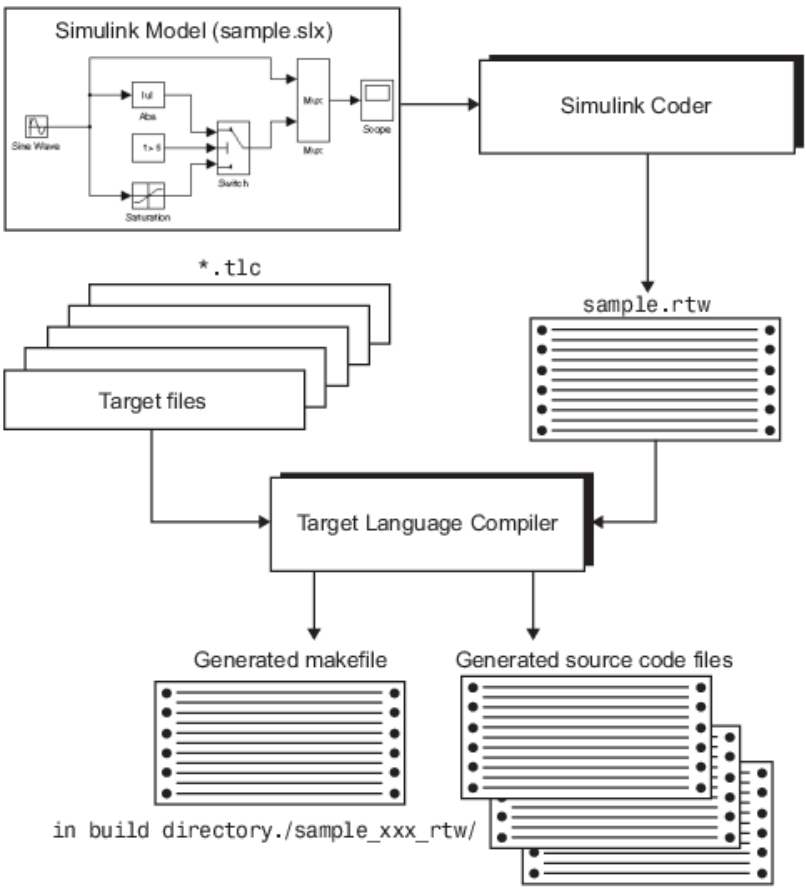

Figure 2.11 Workflow of Simulink Coder. Image source: [MathWorks, 2015b].

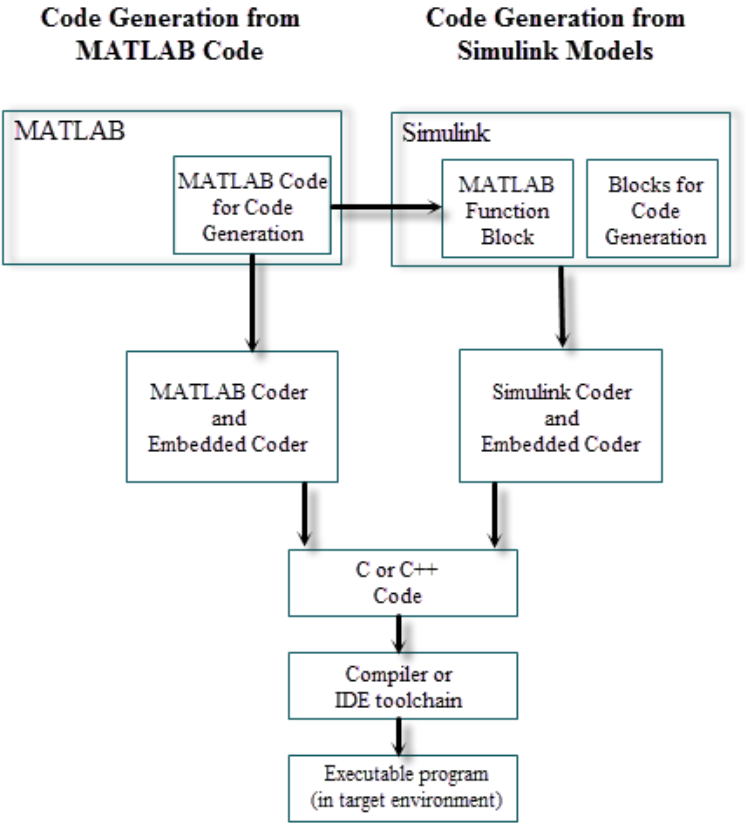

Figure 2.12 Workflow of Embedded Coder. Image source: [MathWorks, 2015c].

3

Method

3.1 Model-Based Design methodology

Model-Based Design (MBD) is used as a development process throughout this project. An essential part in MBD is a model of the system, which is used throughout the design process. The MBD design process can be divided into several different workflow approaches. Two of them are the V-model [Baumgart et al., 2010, p. 5] and the Ten Step Method [Jensen et al., 2011]. The V-model is chosen as the design process in this project, an overview is illustrated in Figure 3.1. As a first step in using the V-model, a requirement analysis is done. During this phase, goal-specifications for the system are determined. This is done in Section 1.3, based on this specification hardware for the project is chosen, Section 3.2. Based on the selected hardware a high-level design of the whole system is defined in Simulink, which is the second step in the V-model. This model is defined in a top-down approach. A structure consisting of several subsystems is used to achieve various different abstraction layers.

The process then continues by defining a detailed descriptor of each component defined in the previous step. First a CAD model of the hexapod is constructed in SolidWorks CAD environment, Section 3.4. This model is then exported to SimMechanics where development can be continued using Simulink. To control the different parts of the hexapod, a servo model is constructed from system identification methods, Section 3.4. The control system for the hexapod can be implemented when the model has been constructed. During this stage different control strategies are evaluated against the SimMechanics model. When controller performance has been tested in software with good results, it is tested on the embedded platform. This is mainly done in what is known as external mode in Simulink. Using Embedded Coder, Simulink generates C code to run on the hardware platform. The code is downloaded to the hardware, where it is possible to monitor signals during execution. For the code generation processes to work, interface code between different hardware components need to be defined and tool-chains for compiling need to be setup. This work is described in Section 3.9 and 3.10.

During the different development steps code is tested from a bottom up approach. First each component is tested. When the result of this is satisfactory the

component is integrated with other components in the system. Through the process the model is refined to the detail level required by the controller.

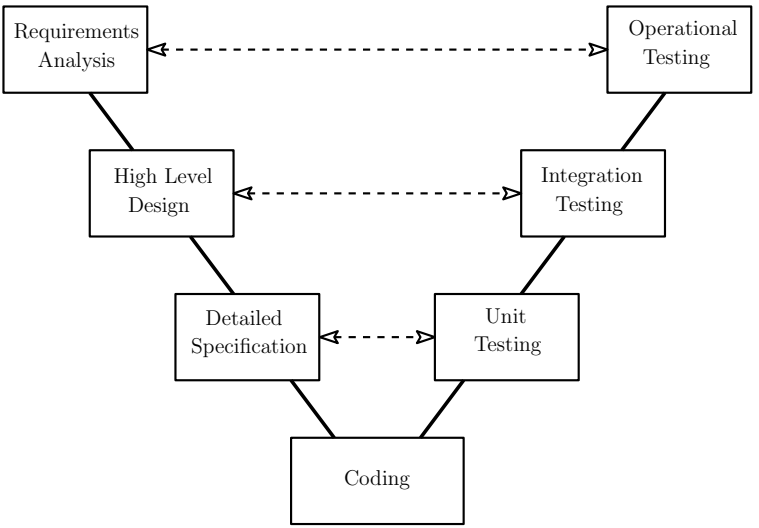

Figure 3.1 An overview of the V-model. The process starts from the upper left corner and follows the V shape. To each step tests are assigned on the right side. Image based on illustration found at [Embedded360, 2010].

3.2 Choosing of hardware

The first part of this thesis was to determine what hardware to work with. There were two different options here. A full hexapod kit could be bought with body frame, servos, battery and processor. The other option was to buy all parts separately. The exact hardware that was chosen is mentioned in Section 4.1 but has already been introduced in the beginning of the report. This section exists to illustrate the process of choosing hardware and which criteria that were considered. In the end the hexapod was chosen mostly based on the preferred servo. The hexapod was bought as a complete kit and the original processing unit ArbotiX-M was later upgraded to the faster BeagleBone Black mentioned in this comparison section. Additional motivations for chosen hardware can be found under the discussion section.

The different parts of a hexapod to be chosen are the chassis, servos, the processing unit and an accelerometer/gyroscope measurement unit. These parts were individually analysed and are presented in the coming subsections. Several criteria were considered when evaluating parts but only the more relevant is presented here. Most important parts are the servos and the processing unit since these need to work

Chassis name	*Phoenix*	*Phantom AX Hexapod Mark II*	*DFRobot robot kit*	*T-HEX 4DOF*	*AXHexapodHW*	*MAH3-R*
Buy-able without servo	yes	yes	no	yes	yes	no
CAD available	hobby CAD	hobby CAD	no	no	no	no
Ground clearance (cm)	13.97	medium-high	medium	11.76	low-medium	17.78
Robustness	medium	medium-high	medium	medium	high	medium
Needed servos	18	18	18	24	18	18
Visual aspect (1-5)	5	4	4	2	1	2

Table 3.1 Comparison between different hexapod chassis. Where no data existed regarding ground clearance and robustness, an estimation was done based on pictures and videos. The visual criteria is highly personal but based on the looks compared to a spider.

together and will limit the possible control implementations. Important criteria of these are communication, processor speed and the availability of feedback.

Hexapod chassis

Different companies sell different hexapod chassis and most of them are located in USA. The main criteria for a terrain walking hexapod are decided to be ground clearance, availability of CAD model, amount of servos and also visual aspect. Table 3.1 shows a comparison of chassis under consideration. Some chassis are not sold without the full kit of servos, battery and processor. For some criteria like robustness, no data existed, in this case an estimation was done based on videos from www.youtube.com on the scale from low to high.

Servo name	Hitec 645MG	Dynamixel AX-12A	Uptech CDS 5516
Max current (mA)	450	1400	1300
Operating voltage (V)	4.8-6.0	9.0-12.0	6.6-16.0
CAD available	yes	yes	partly
Max torque (kgcm)	9.6	16.5	16
Max speed (s/60°)	0.2	0.169	0.18
Range	180°	300°	300°
Angular resolution	0.18°	0.29°	0.32°
Communication	PWM	UART	UART
Feedback	no	yes	yes

Table 3.2 Comparison between different servos. Communication is how the main hexapod processor communicates with the servos. The ability of feedback is provided by servos available to measure on-line data by themselves.

Servo

Servos were compared under the criteria: torque, rotation speed, rotation range, communication type, resolution and feedback availability. The comparison can be viewed in Table 3.2. The main difference between the two servos to the right compared to the one to the left is that they are smart servos. Smart servos mean that they have an integrated microcontroller (MCU) with a regulator that controls the servo. This opens up several additional options, for example options on how they respond to a step in reference signal. Current speed, position and torque can also be measured on-line from the servos. These measurements are what can be seen as feedback. The smart servos have a higher torque and thus a higher operating voltage and max current. That in combination with the processor leads to higher power consumption.

Processor

Lately a lot of micro controller boards have become available to customers. Known brands such as Arduino, Raspberry PI and BeagleBone Black have been compared in Table 3.3. Processor speed, Simulink support and the amount of certain input/output ports are important factors that need to be taken into consideration when choosing. Input/Outputs are important because there is a need for several communication channels like servo and remote.

Processor	BeagleBone Black	Raspberry PI A+	Raspberry PI B+	Arduino Due	Arduino Mega
CPU frequency (MHz)	1000	700	700	84	16
RAM (MB)	512	256	512	96KB	8KB
Operating voltage (V)	3.3	3.3	3.3	3.3	5
I/O	•	•	•	•	•
UART	4	1	1	4	3
SPI	1	1	1	1	1
I2C	1	1	1	1	1
Existing Simulink support	•	•	•	•	•
UART	no	no	no	yes	yes
PWM	yes	no	no	yes	yes
XBee	no	no	no	no	no
Video Capture	yes	yes	yes	no	no

Table 3.3 Comparison between different micro controllers. The amount of certain inputs and outputs have been taken into account. Also the support that Simulink can provide is important.

Internal Measurement Unit (IMU)

To balance and detect obstacles in the environment an IMU is to be used. An IMU unit usually consists of accelerometer, gyroscope and sometimes also a magnetometer. To easily integrate an IMU into the system two requirements need to be taken into consideration. First of all, it should be possible to mount the IMU onto the hexapod. The other requirement is which communication interface that is used. Because of these two requirements, two different breakout boards were considered, SparkFun Degrees of Freedom MPU-9150 [SparkFun Electronics®, 2015a] and SparkFun 9 Degrees of Freedom IMU Breakout - LSM9DS0 [SparkFun Electronics®, 2015b]. These two boards have IC which contains accelerometer, gyroscope and magnetometer. The MPU-9150 board was chosen due to its digital motion processor (DMP) [InvenSense Inc, 2013, p. 10] which is able to perform sensor fusion with the accelerometer and gyrometer. Existing software found at Github [Pansenti, LLC, 2012] makes it possible to integrate the magnetometer and combine it with the DMP output.

3.3 Assembly

The Phantom AX hexapod is received as a building kit, see Figure 3.2. Assembling can be divided into two parts, one of the mechanical and one of the electrical. The kit also offers some customization.

Figure 3.2 The Phantom AX Hexapod Mark II is supplied as a kit.

Mechanical assembly

Assembly instructions for the kit is provided online [*PhantomX Hexapod Assembly Guide*]. Important during assembly is to use an adhesive for the bolts due to the vibrating nature of the hexapod. In addition to the supplied battery of 2200 mAh an additional battery of 4200 mAh is mounted on top the hexapod. This provides the ability to continue testing when one battery goes empty. The optional top deck mentioned in the instructions are later mounted on top of the hexapod in such a way that the extra battery fits beneath it. This provides for mounting additional sensors and electronics on the top deck. The BeagleBone Black and IMU are mounted here when upgraded from the ArbotiX-M processor and also the XBee is moved up to the top deck. The hole pattern on the top deck did not fit that of the BeagleBone Black and IMU. Additional holes were drilled in the top deck to fasten these parts properly. Other parts mounted on the top deck were fastened using Velcro tape.

Electrial assembly

The electrical assembly of the system can be divided into two circuit systems; power and communication. In Figure 3.3 all power connections from the battery are shown.

The hexapod can be powered from one of two available batteries (4200 or 2200 mAh). According to [Coley, 2014, p. 82] no voltage is to be applied to any I/O pin before start-up of the BeagleBone Black. Switch S1 in Figure 3.3 is used to make sure that the BeagleBone Black powers up before the ArbotiX-M.

The BeagleBone Black uses an on-board Power Management IC [Coley, 2014, p. 41]. This IC converts the incoming 5.3 V to required voltage by the board and also supplies XBee, IMU and the level converter with voltage and current.

The communication connections are showed in Figure 3.4. UART and I2C are used between different components in the system. Because BeagleBone Black uses 3.3 V logic and ArbotiX-M uses 5 V logic a level converter is placed between these circuits.

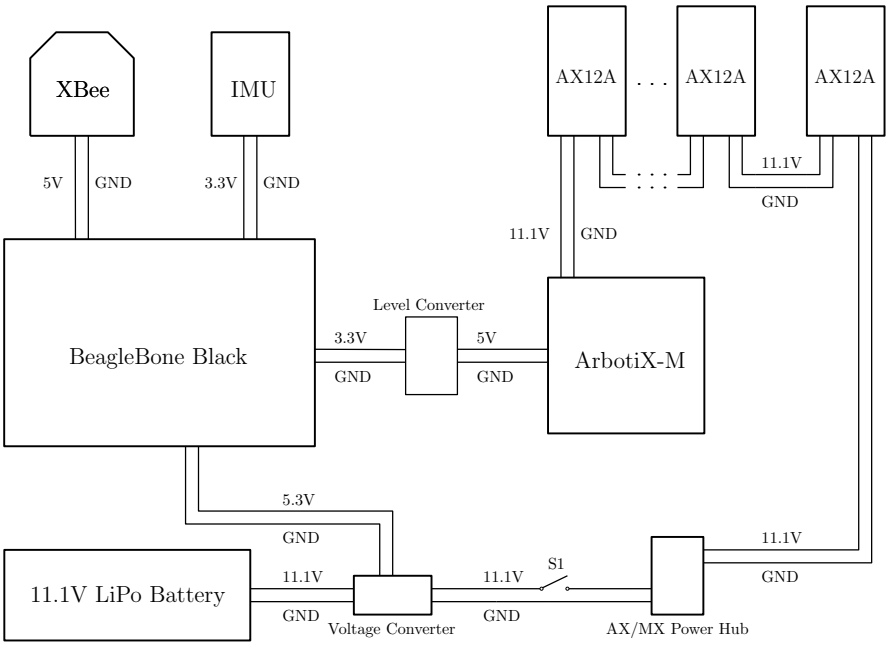

Figure 3.3 An overview of how the power is supplied to the system.

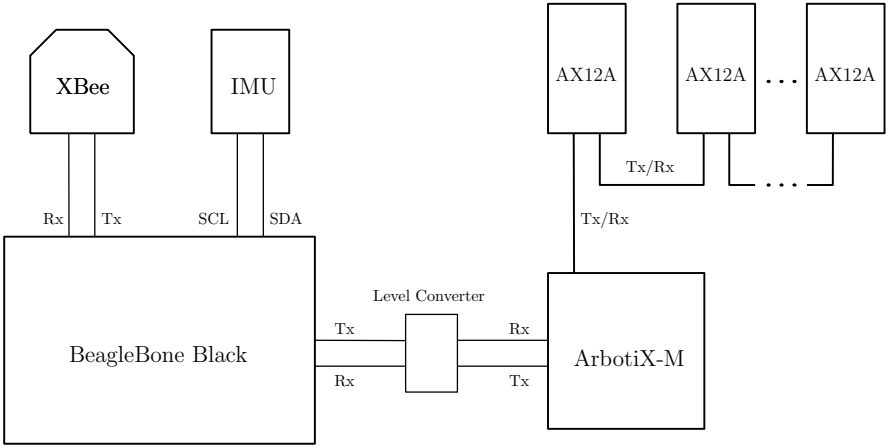

Figure 3.4 An overview of how communication in the system is connected. Each line represents one electrical wire.

3.4 Modeling hexapod and servo

CAD model

The hexapod is first modelled in SolidWorks and then exported to SimMechanics using SimMechanics Link. Some models of different parts of the hexapod were found online at [ROBOTIS INC, 2015] and [Hendricks, 2014].

The individual parts of the hexapod are assembled to different solid bodies. To improve simulation time it is important to keep the number of solid bodies low. A CAD model of one leg can be seen in Figure 3.5. The leg modelled using three solid bodies coxa (orange), femur (green) and tibia (blue). The main body of the hexapod is modelled as one solid body and is showed in Figure 3.6.

The different parts of the hexapod can be connected via joints in either SimMechanics or in SolidWorks. To get more control of how coordinate frames are assigned, the coordinate frames are created in SolidWorks. When they have been exported to SimMechanics the frames are connected by joint blocks. To each servo a coordinate frame is assigned according to Figure 3.7. This frame is then used to connect the servo body to the servo horn in SimMechanics using a revolute joint. A model of one hexapod leg in SimMechanics can be seen in Figure 3.9.

The six legs are then connected to the main body with the use of revolute joints. This model can be seen in Figure 3.10. The exported hexapod can be viewed inside MATLAB and a picture of this can be seen in Figure 3.8.

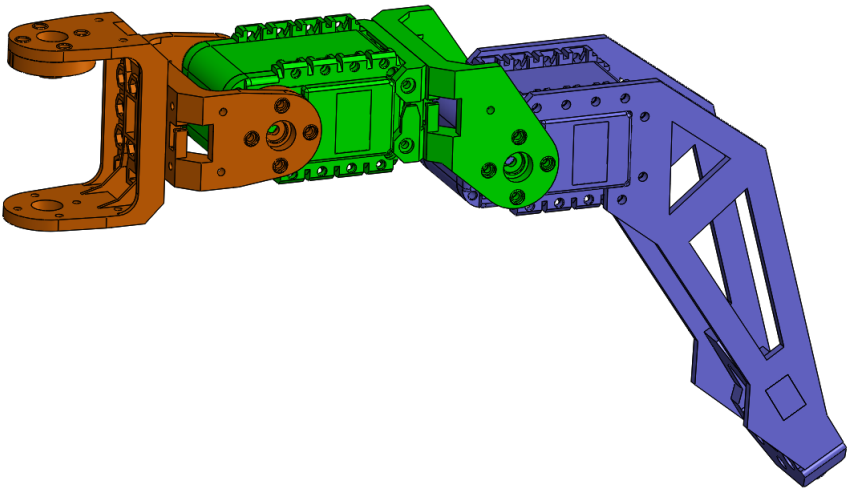

Figure 3.5 A CAD model of one leg of the hexapod. Each color represents one solid body, coxa (orange), femur (green) and tibia (blue).

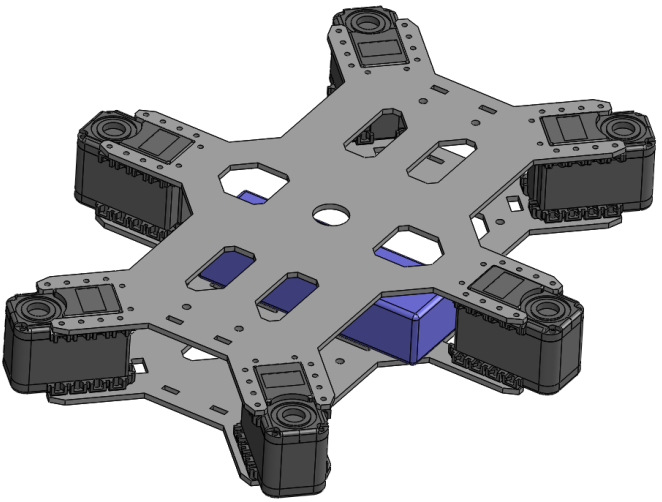

Figure 3.6 A CAD model of the main body of the hexapod. The blue box represents the battery.

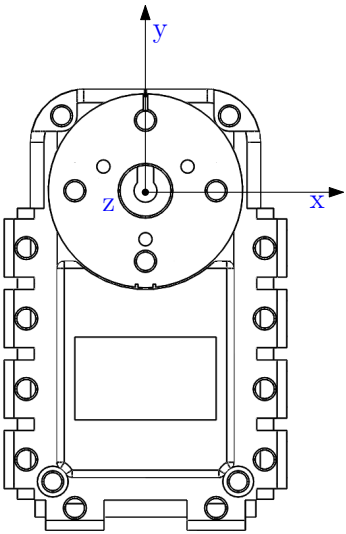

Figure 3.7 Coordinate frame to connect servo to servo horn on each servo.

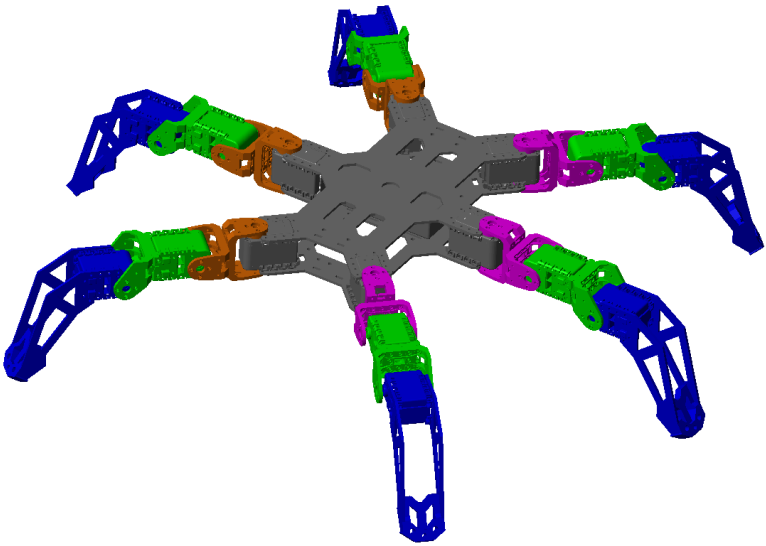

Figure 3.8 A visualization of the hexapod in MATLAB.

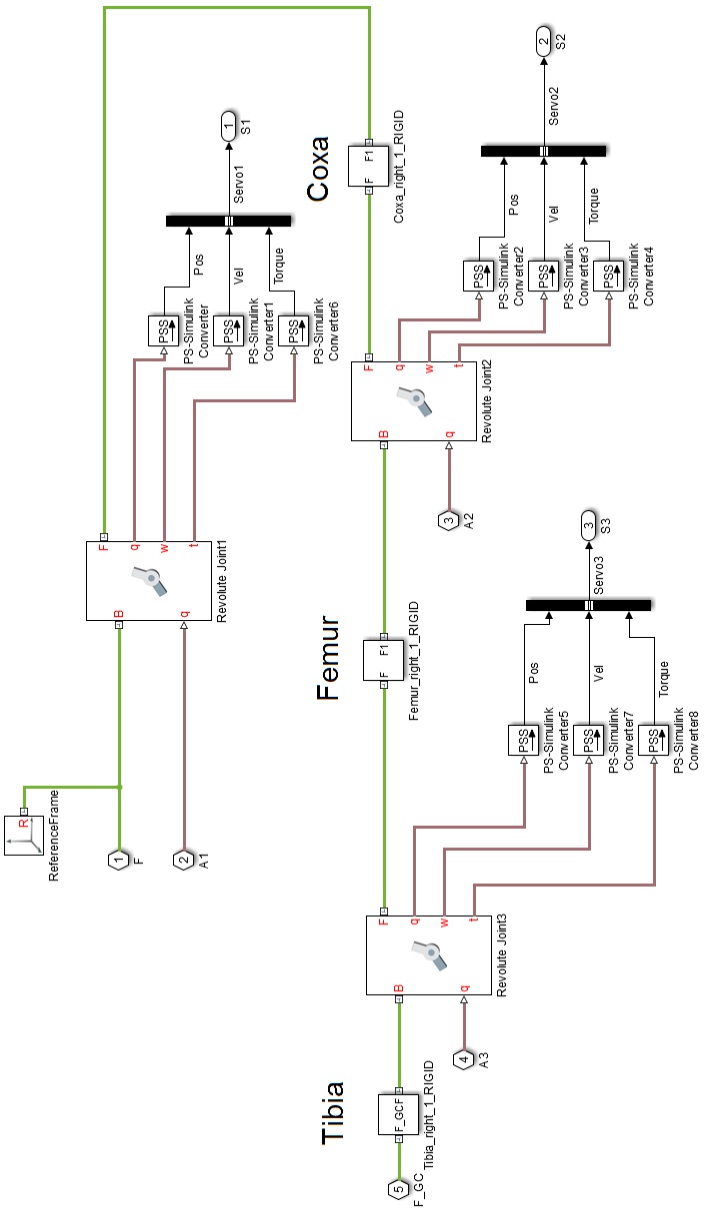

Figure 3.9 Model of one leg of the hexapod in SimMechanics. To each joint actuation and sensing are connected.

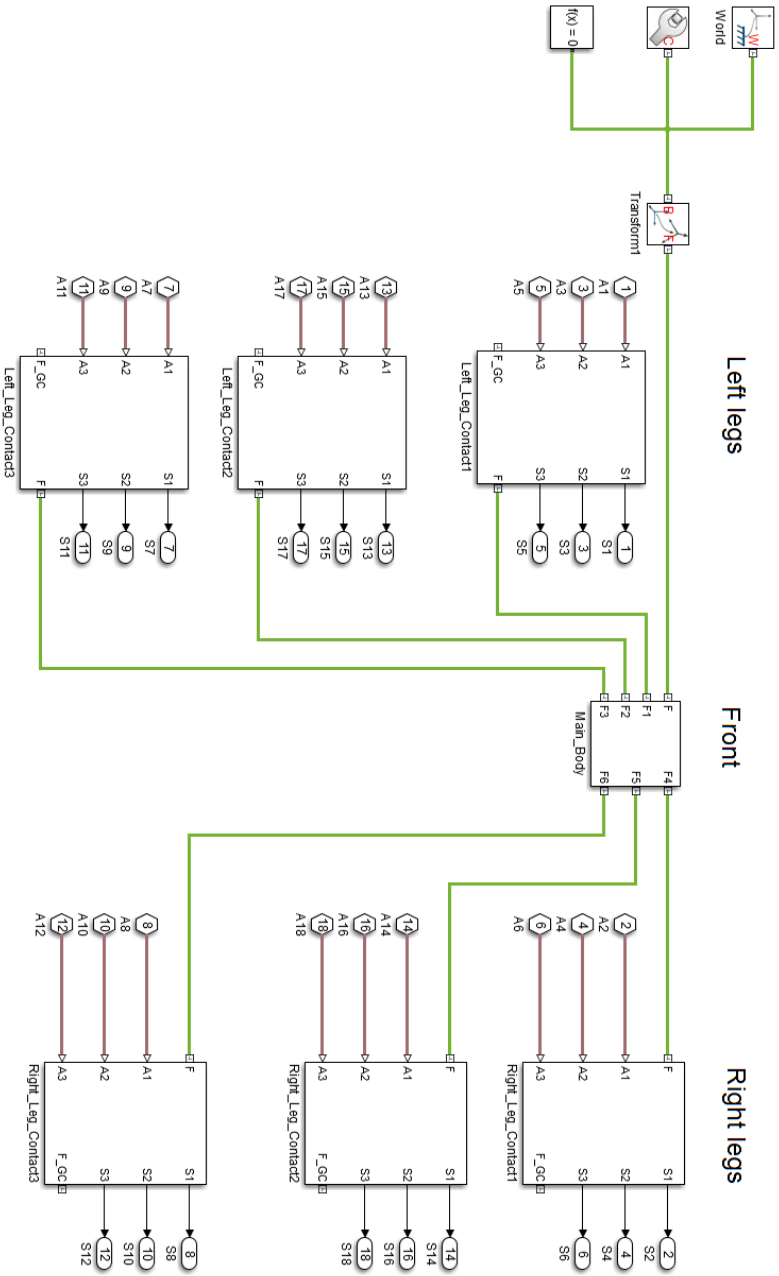

Figure 3.10 Model of the hexapod in SimMechanics; one main body is connected to the six legs.

Modeling of contact forces in SimMechanics

To make the SimMechanics model able to interact with the environment, contact force is modelled. Blocks or methods for performing this do not exist in SimMechanics. As a first approach an add-in library for contact forces is used [Miller, 2014]. This library can model contact forces between 2D objects. In Figure 3.11 a model of how contact forces are modelled are showed. Contact between two bodies are modelled as a spring-damper system.

The most important contact force is considered to be contact between the floor and the hexapod. Because of the limitations of this library, only contact between the six feet and the floor is modelled. This is done by placing a coordinate frame at each feet. By using two circle to line blocks a sphere(feet) to plane(floor) contact force is modelled. A friction model is also used to make the hexapod move across the floor.

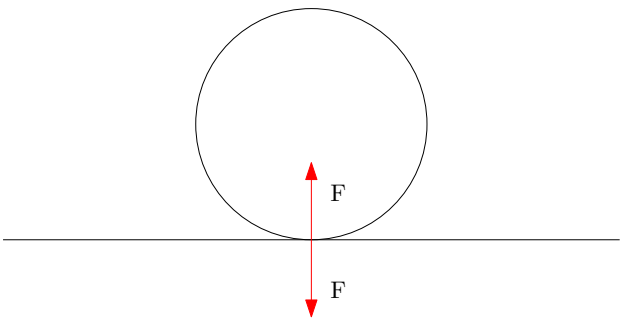

Figure 3.11 Illustrtion of contact force model.

Servo identification

Dynamixel AX12A servos use an internal MCU to perform speed and position control of the servo horn. The structure of the regulation systems is unknown. Based on the documentation [Robotis, 2006] of the servos it is assumed that the angle of the servo is measured in steps of 0.29°. The input to the servo model is the same as the real servos, which is position set-points in integer steps from 0 to 1023. Figure 3.12 shows the position-sampling model of the servo horn. The output of the servo sampling circuit is then fed to a controller which regulates the speed of the servo based on the position error.

The speed control of the MCU can be turned off, the servos then run at maximum speed when the position error is large enough. For estimating the parameters of the PID regulator a test rig was constructed according to CAD Figure 3.13. Two different servo horns were also created in order to apply different loads to the servo Figure 3.14.

A series of step responses were conducted and data are collected using the ArbotiX-M card. The tests are performed with different loads and with either speed

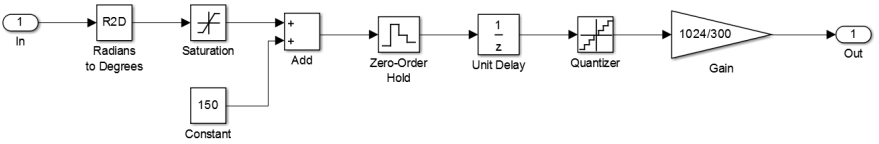

Figure 3.12 Model of sampling circuit of servo.

control switched on or off. After this process was complete, the different step responses were studied, and structure of the regulator is decided. The regulator was then tuned using Simulink Design Optimization. This program performed parameter estimation by using optimization methods. When parameter estimation was completed, validation was needed. This was done by measuring a step response from the servo mounted on the hexapod. A simulation response was then compared against this measurement.

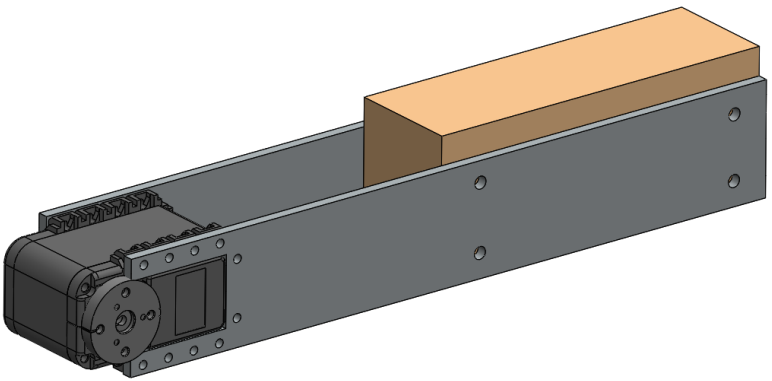

Figure 3.13 A CAD model of the test rig used for servo estimation.

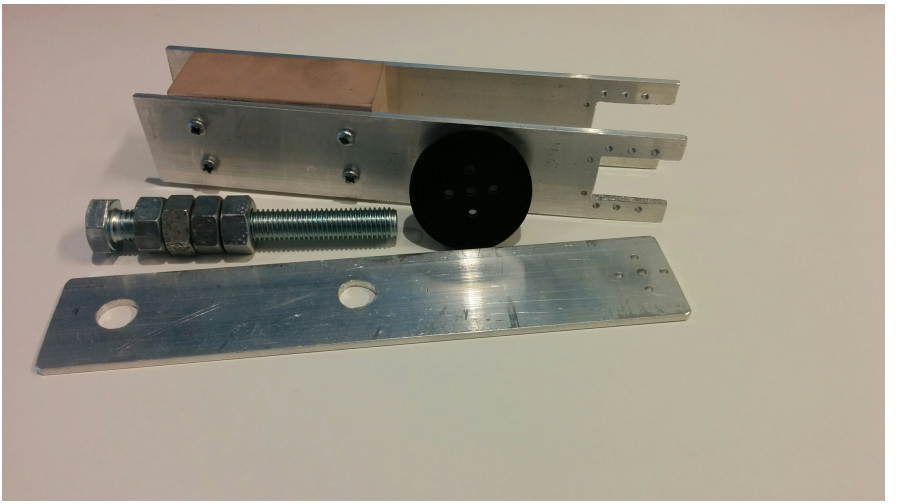

Figure 3.14 The constructed test rig used for servo estimation. Servo wheel (black), servo rod (front) and weight for servo rod (nuts and bolts).

3.5 Control implementation

The controller implementation is separated into two parts, main controller and inverse kinematics. The main controller handles user input and determines leg tip positions at every sample. The input comes from the hand-held remote when using the hexapod or a coded testing sequence in case of model testing. In both cases the output is the positions of the tip of each leg in the hexapods own coordinate system. When leg positions are mentioned in the report it is the leg tip position that is referred to. The other part is the inverse kinematics that utilize these tip positions and calculate servo angles to achieve correct leg stance. Because each leg will not be put in the same way each leg needs its own IK controller, though the IK controllers for each side of the hexapod are identical. To allow easier calculations for the IK and use it for all legs on the same side each leg has its own coordinate system. A translation between the main body coordinate system and the legs individual coordinates is therefore done after the main controller. An overview of the full controller can be seen in Figure 3.15.

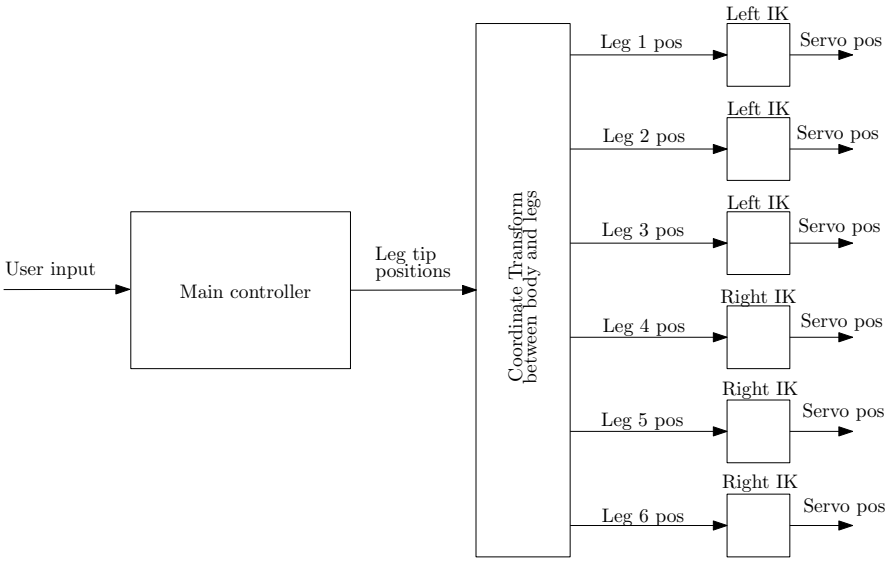

Figure 3.15 The controller system with user input from left to a main controller that calculates leg positions. Translation of the coordinate system is then needed before the inverse kinematics calculate individual servo positions.

Main Controller

The main controller is the brain of the hexapod. Its task is to calculate new positions for each leg. These positions need to be based on the user input from the hand-held remote. Joysticks and buttons are what the user have to play with, see Figure 3.16. The remote already has a supplied program loaded that sends a package of information every 33 ms. Information sent in this package are the positions of the joysticks and the state of the buttons. Current video games where vehicles are driven usually utilize the left stick for acceleration and the right stick for rotation. Natural for the hexapod is then to utilize the left stick for velocity in arbitrary direction and right stick for rotation. The trigger buttons are used for hexapod height alteration and the normal buttons can be used to change between different controller modes.

The goal of the hexapod controller is to navigate unknown terrain in a controlled way. Hard coded gait patterns fall out of favour due to the requirement of adaptation to unknown terrain. A more dynamic gait pattern will ease walking in unknown terrain and in arbitrary direction. Another goal is for the hexapod to choose gait pattern based on user inputs, for example how many legs to lift simultaneous. The controller is designed in such a way that there is a trajectory function that calculates the next positions for all legs and determines their need to be lifted. To aid in this, a function that composes a queue of legs to be lifted next exists.

The main purpose of this function is to determine which leg is the "most be-

Figure 3.16 The hand-held remote called ArbotiX Commander. It provides 6 buttons, 2 trigger buttons and 2 joysticks.

hind", in sense of walking direction, and place it first in the queue for lifting. There is also a function that provides the controller with information about what is a "default stance", i.e., a stable stance for the motionless hexapod. It is also around this stance that the leg movement should be performed in order to achieve good stability during walking. It is fully possible to walk the hexapod using a static default stance, but in order to change height continuously the default stance needs to adapt continuously, necessitating a separate function. The full design can be viewed in Figure 3.17.

Default stance position is calculated by the first function. This is done in such a way that the function has a predefined list containing these positions for different discrete heights. This list is saved in cylindrical coordinates based on each legs attachment point to the main body, and contain position data for each 10 mm in height. The initial height is known and further height changes will be achieved by registering when the trigger buttons are pressed. The height alteration speed is set to be 20 mm per second and is not user customizable. Some calculations are needed in order to achieve continuity in default stance positions for heights not being a multiple of 10 mm and thus not in the list. To calculate these positions linear interpolation is done between the two closest positions in the list. Before being usable these positions need to be translated to the main body's Cartesian coordinates. Current height and corresponding stance positions are then forwarded to other functions.

Creating an ordered queue of which leg to lift next can be implemented in several ways. A basic approach would be to push a leg into the queue bottom after

it has been lifted and pop the leg to move next from the top of the queue. This will however resemble hard-coded gait patterns a lot because legs will move sequentially. Another approach is to determine legs position in the queue based on how far away they are from their default position. Using this the direction of movement needs to be taken into account else legs being ahead of their default position will be considered in the same way as legs being behind. A third approach is to consider which legs will violate some boundary condition first. Since legs are physically restricted to a certain area this could be used as a boundary. Since velocity and rotation is known for a certain time instance, this could be used to calculate the amount of time until the leg will violate such a boundary. Combinations of the approaches could also be used in order to create a weight function that takes all parts into consideration. More in-depth regarding the implementation of the methods are described in Section 3.7.

In the last part of the controller, positions for the legs are updated. Provided with a queue of the legs and user inputs the function determines how many legs to lift. The legs in stand phase are moved using (2.6). In case of rotation the unlifted legs positions are multiplied by the rotation matrix (2.4) to achieve rotation, see the full move equation (3.1) for trajectories in the main body's coordinate system. This will allow both directional and rotational movement of the hexapod body simultaneously.

$$P_i^{t+1} = \left(P_i^t - \frac{v}{f} \right) \cdot R_z \left(\frac{\theta}{f} \right) \tag{3.1}$$

The controller determines, based on the speed and rotation speed, how many legs it is allowed to lift. When a leg is decided to be lifted a trajectory function calculates its full trajectory instantly and saves it for future samples to utilize. The trajectory is calculated as the upper half of an ellipse (2.8). An example of a trajectory from a position (120, 60, -70) to (120, 180, -60) can be seen in Figure 3.18. Note that the y direction is the same as the forward direction.

The function allows for the option to set the trajectory time i.e., the lift duration. In order to be able to make trajectories between different heights separate values on the variable b is used for the first 90 degrees and the last 90 degrees. A basic concept of the trajectroy function is presented in Listing 3.1. This show the steps done in the function but is in no way a complete code for it.

```
function [trajectory] = makeLiftTrajectory(pos, goal, ...
    frequency, stepTime)

%   Initialize constants and buffer size for the trajectory
    init();

%   Calculate width (a*2) and height (b) of ellipse
    a = abs((goalPos - pos))/2;
    b1 = 30;    %height is always the same for first quarter
    b2 = 30 + heightdiff(goal, pos); %second quarter of ellipse

%   Generate the ellipse
    for i = 0:pi/2
        x = a*cos(i);
        z = pos(z) + b1*sin(i);
    end
    for i = (pi/2):pi
        x = a*cos(i);
        z = goal(z) + b2*sin(i);
    end

%   Sample the ellipse to achieve correct stepTime
    sampling = linspace[0:length(x):stepTime*frequency/1000];
    [x, z] = [x(sampling), z(sampling)];

%   Lastly project the ellipse x-axis on the movementvector
    [x, y] = projection(x, goalPos - pos);

%   Return a full trajectory for the main controller to carry ...
    out in coming samples
    trajectory = [x; y; z];
end
```

Listing 3.1 Basic concept code for the trajectory generation function.

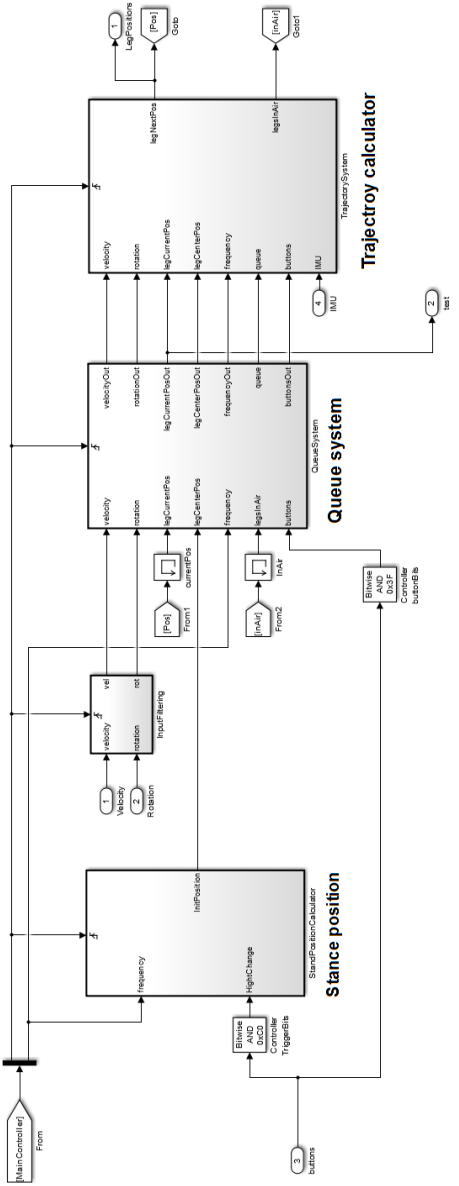

Figure 3.17 Design of the main controller, where one function keeps track of stance position based on current height, one function that calculates a queue of what order the legs should be lifted and lastly were the trajectories for each leg is calculated based on information from previous functions and user input.

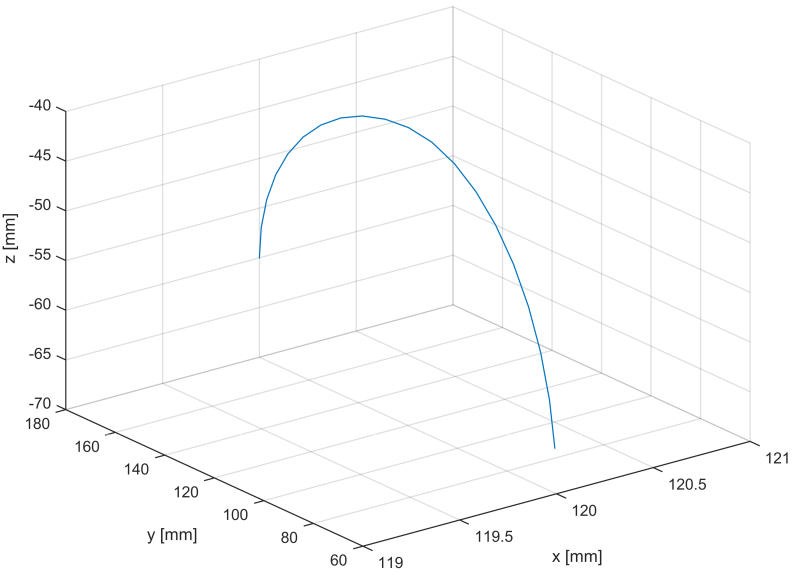

Figure 3.18 The resulting trajectory generated from (120, 60, -70) to (120, 180, -60).

Leg IK

The goal of the inverse kinematics is to position the foot of the leg at point $p_1(x_1, y_1, z_1)$. To achieve this three servo angles need to be calculated. Using the coordinate system and notation presented in Figure 3.19, the angle for the coxa servo is calculated by (3.3) were arctan2 is defined according to (3.4). When $\gamma = 0$ the servo is at position 150°. For positive γ servo angle is increased, this corresponds to clockwise rotation in Figure 3.19.

$$\gamma = -\operatorname{atan2}(x_1, y_1) \tag{3.2}$$

$$\gamma_{Coxa} = 150° + \gamma \tag{3.3}$$

$$\operatorname{atan2}(y,x) = \begin{cases} \arctan\frac{y}{x} & x > 0 \\ \arctan\frac{y}{x} + 180° & y \geq 0, x < 0 \\ \arctan\frac{y}{x} - 180° & y < 0, x < 0 \\ 90° & y > 0, x = 0 \\ -90° & y < 0, x = 0 \\ \text{undefined} & y = 0, x = 0 \end{cases} \tag{3.4}$$

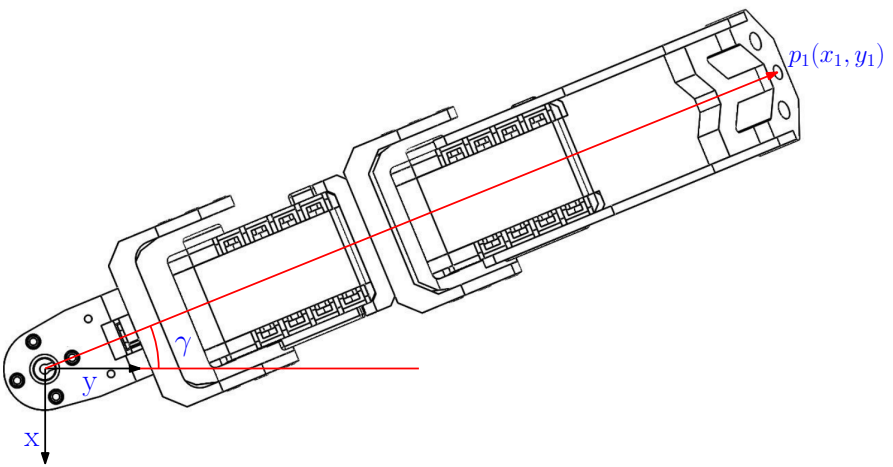

Figure 3.19 Notation used for calculation the coxa angle (3.3).

To calculate the angles for femur and tibia servo, notation according to Figure 3.20 is used. The ξ axis points along the vector from origo to p_1 in Figure 3.19.

First L_1 and L_2 are calculated according to (3.5) and (3.6). Using the law of cosines the femur and tibia angle can be calculated according to (3.9) and (3.10). In Figure 3.20 femur and tibia are at an angle of 150°. Increasing γ_{Femur} and γ_{Tibia} makes point p_1 move in positive z direction. To calculate offset_{Femur} and offset_{Tibia} the servos is set to 150°. Then coordinates of p_1 is found to be $p_1 = (0, 209.4, 111.4)$ mm using the CAD model. The constants in Figure 3.20 used when doing the inverse kinematics calculations are listed in Table 3.4.

$$L_1 = \sqrt{x_1^2 + y_1^2} - L_{Coxa} \tag{3.5}$$

$$L_2 = \sqrt{L_1^2 + z_1^2} \tag{3.6}$$

$$\alpha = \text{atan2}(z_1, L_1) \tag{3.7}$$

$$\alpha_2 = \arccos\left(\frac{L_{Femur}^2 + L_2^2 - L_{Tibia}^2}{2 \cdot L_{Femur} \cdot L_2}\right) \tag{3.8}$$

$$\gamma_{Femur} = 150 - (\alpha_2 - \alpha + \text{offset}_{Femur}) \tag{3.9}$$

$$\gamma_{Tibia} = 150 - \arccos\left(\frac{L_{Tibia}^2 + L_{Femur}^2 - L_2^2}{2 \cdot L_{Tibia} \cdot L_{Femur}}\right) + \text{offset}_{Tibia} \tag{3.10}$$

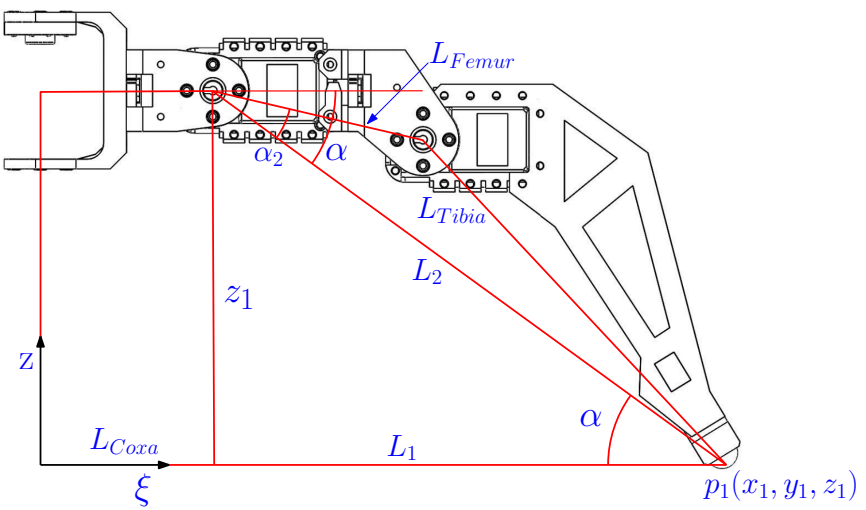

Figure 3.20 Notation used for calculation the femur (3.9) and tibia angles (3.10).

Name	Value
L_{Coxa}	53 mm
L_{Femur}	66 mm
L_{Tibia}	133 mm
offset_{Femur}	13.7°
offset_{Tibia}	147.6°

Table 3.4 Parameters used in leg IK calculation.

3.6 Constraints

There are constraints that effect the hexapod that the controller should take into consideration. Constraints that apply directly to the individual servos due to mounting brackets will be referred to as hard constraints. These constraints affect the angular range of the servos and are constant throughout all movement.

The other part of constraints needed to be taken into consideration are referred to as leg constraints. They will limit the space in which the inverse kinematics are able to put the legs. These limits are a combination of physically impossible leg positions and mathematically inaccessible positions. A last contributor will also be the effect of other legs' positions.

Hard servo constraints

The servos Dynamixel AX-12A used by PhantomX AX Hexapod Mark II has a rotation range of 300°, Table 3.2. Though the mechanical assembly of the PhantomX provides a physical impossibility for the servos to reach their full range. If for some reason during testing the controller sends a position to a servo that is not reachable it will hit the mounting bracket in an attempt to reach that position. Since the servos have an internal controller they also have built-in safety routines. In the event of hitting something unmovable with full torque the servo will enter an error state rendering it unmovable by itself to protect the electrical engine. Not until the servo is restarted will it return to normal functionality. Even though the safety routine exists there to help with these occasions it would be preferable if they did not occur.

An effective way to prevent servos from trying to reach impossible positions is to introduce constraints to their reference signal. In order for these constraints not to effect anything else in the controller, they are implemented as late as possible in the controller sequence. The last part of the controller is the leg IK which calculates the servo positions as a number between 0 and 1023 due to their 10 bit resolution. This is where such constraints are implemented.

The constraints need to be determined experimentally or taken from previous controller for the Phantom hexapod. The servo easily lets the user read servo positions. By putting servos manually at constraint positions while continuously reading their positions allows for getting exact constraints for the servos, Table 3.5. Since

the signals to servos are never calculated in degrees there is no need to translate the constraints to degrees. Legs on the left and right sides are mirrored but otherwise identical. Therefore there is only a need to determine different constraints for left and right legs.

Servo	Max	Min
Left Coxa	850	180
Left Femur	850	170
Left Tibia	880	280
Right Coxa	850	180
Right Femur	850	170
Right Tibia	700	130

Table 3.5 Constraints for servos on the left and right side respectively. All legs on one side are identical why there is no need to have different constraints for them. The constraint values are not in degrees but in values between 0 and 1023 due to servos working with 10 bits.

Leg constraints

The size of the parts coxa, femur and tibia are not flexible for the hexapod. Because of this each leg will not reach longer than these parts combined. Practically, this will create a sphere with a certain radius around the coxa rotation center that is accessible space for the leg. Outside of this area the leg IK will not be able to calculate servo angles, cause they do not exist. In the same sense there exists positions close to the coxa rotation center that is inaccessible because of the lengths of the leg parts. Cylindrical coordinates represent a good way of representing these constraints because leg trajectories often occur in the same hight plane. For every height this means there will be two circles around the coxa rotation point representing the outer and inner bound for the legs to be able to be positioned in. But because the coxa servos are attached to the hexapod main body, of course the full circle is not accessible. The circles will be limited to an angular interval in which leg movement is reasonable. This interval is saved as a constant and cannot be changed by the user. Likewise the inner and outer radius are saved as a list. Because there are two servos changing the legs radial position there will be different radial intervals for different leg heights. These radial intervals are measured by pushing the leg IK to its limit for discrete heights of 10 mm. To achieve continuity these intervals will be linearly interpolated by the controller.

There exists, in a sense, leg positions that are accessible and have a negative radius. This means that the length of the femur and tibia allows to put the leg tip behind the coxa rotation point. But the trigonometrical structure of the implementation of the inverse kinematics algorithm makes them inaccessible. The implementation

renders an error if given such positions. This is a known flaw that could be fixed but is chosen not to be. Since there is nothing to gain from putting legs in such positions under the hexapod these errors are rather avoided by setting the inner limit to always be positive. For a graphical representation of the constraints we refer to Figure 3.21.

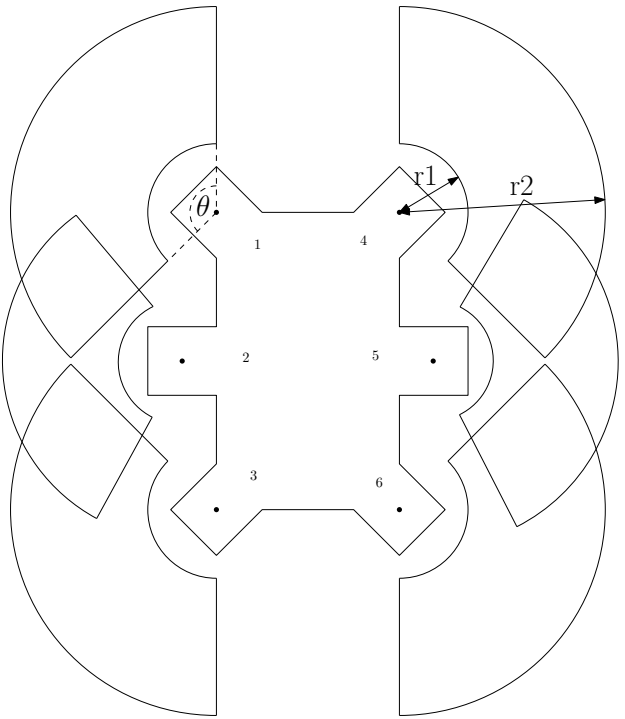

Figure 3.21 Each leg has its own restricted space where it is allowed to move in. It is restricted by an inner (r1) and outer (r2) boundary, but also by an angular restriction θ. As can be seen, the areas overlap.

As is seen in Figure 3.21, the areas of possible leg positions overlap. This will introduce another problem because legs might bump into each other. A way of reducing these risks the legs have an additional constraint that corresponds to the y coordinate of adjacent legs, see Figure 3.22. Since legs have some volume the constraint is exaggerated a bit from the actual y position to avoid complications. Due to the two different coordinate systems some problems might occur. However, leg positions are calculated in the main body Cartesian coordinate by the controller, so constraints will only be saved in cylindrical coordinates, but translated when used.

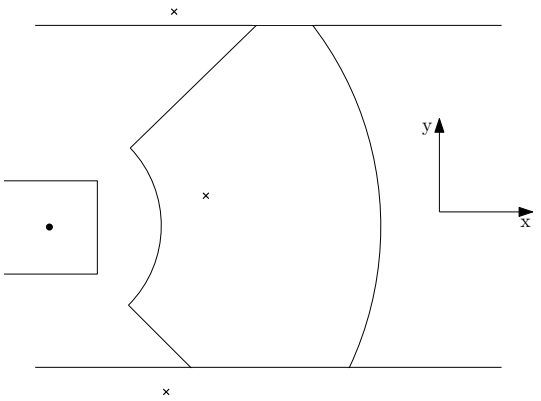

Figure 3.22 The restrictions for leg number 5. Leg positions are marked by x. The boundary will be further compromised by the position of adjacent legs.

3.7 Implementation of walking algorithms

Due to the existence of several different implementations of walking/balancing in the controller, the different options are developed as different walking modes. These modes can be switched between using the buttons on the hand-held remote. The reason to develop different modes is to be able to keep modes that work properly as a visual option while trying to develop them further in another mode. The main difference between these modes is the difference between trying to develop smart walking patterns and developing good body stabilization. Stabilization and terrain handling will be covered in the next section.

The workflow of the controller goes from input through leg position calculation to servo position calculation by the IK, see Figure 3.23. The parts here that actually differ between different modes lie mostly in how the queue is created and how the controller handles lifting of legs. These implementations differ in how they utilize leg constraints, leg positions and user input.

To keep stability and have a reset option, a function is created for user inputs of zero velocity and rotation. When the user tells the hexapod not to move (joysticks' value of zero), the hexapod is supposed to lift legs and put them back to default position. This is done one at a time and is done in the order of which leg is furthest from its default position. By having this function the user is able to just release the joysticks in order for getting back stability if legs are put in an unsatisfactory way.

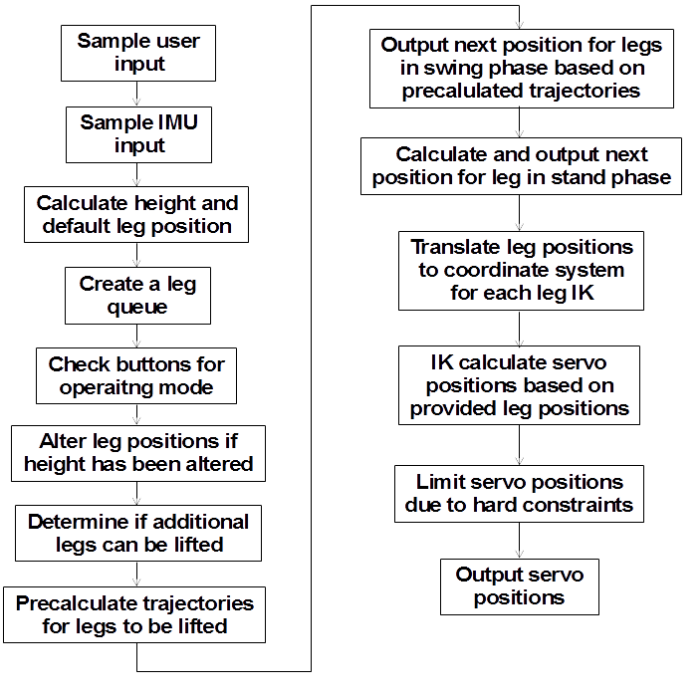

Figure 3.23 This is the basic workflow of the hexapod controller. Different modes chosen by the remote buttons will change how the queue is created and change the algorithm for lifting legs.

Creating a queue

Mentioned in Section 3.5 several methods for creating the queue are tested. Focus is on a method where legs are put into queue based on their distance from their default position. Initially the method in which legs are put in a queue as first in first out (FIFO) is implemented. This method was used in the early stages of development and later discarded. It was mainly used to test that code generation worked and that the hexapod responded in a controlled manner. When development changed from the ArbotiX-M to the BeagleBone Black the method was discontinued.

Creating a ordered queue of which leg to move next can be based on the legs' current distance from the default stance position and the current velocity. The absolute distance from the legs' current foot position to a position s seconds ahead of default position is calculated (3.11).

$$distance = |(D_i + s \cdot v) \cdot R_z(s \cdot \theta) - P_i^t| \tag{3.11}$$

D is the default position for a leg, s distance in time to the comparison position

and *distance* is the value of how far behind the leg is. Legs are then put in queue with the largest distance value first. If a leg is currently lifted it will not be taken into account for in the queue and is automatically put last. This method of creating the queue makes it more reliable for changes in moving direction whilst at the same time not requiring a lot of computations. This method is referred to as mode 1 and is the default mode for the hexapod.

Introducing the leg constraints mentioned in Section 3.6 the queue creating function can be developed to utilize these. Since both current position and boundary are known the distance to the boundary can be measured. Ordering the queue based on this distance provides a different approach than previous function. To measure the distance in a correct manner, virtual steps are taken from the current position as in (3.1). The amount of steps until the boundary is crossed, is the amount of samples until the leg will be outside. To distinguish this method it will be referred to as mode 2. Instead of measuring in samples the distance is measured in time by changing the division by frequency in (3.1) to division by parts of a second. This will allow for lesser steps to be taken until crossing and thus lesser calculations but at the cost of time resolution.

Changing between different gaits

The supplied control program that came with the hexapod kit utilized the remote buttons to change between the basic gaits shown in Figure 2.5. But in order to create more dynamic locomotion, the hexapod needs to change between gait patterns based on velocity and rotation. To get more dynamic movement the hard coded locomotion mentioned in theory is simplified to the pure usage of one, two or three legs. Changing between these simplified gaits is done by a function that determines the amount of legs the hexapod is allowed to have lifted simultaneously. If the quota is filled the hexapod will not carry out any lifting that sample, but if not the controller will calculate trajectories for additional legs and start lifting them.

Because of the simplified and dynamic locomotion there is a need for some leg lifting constraints. This is in order for the hexapod not to lift legs in such a manner that it loses balance. A limit of maximum 3 legs lifted simultaneously is a necessity. To further limit leg lifting, constraints that prevent two adjacent legs on the same side to be in air simultaneously is needed. To stay balanced further limitations are not needed. But if the two front or back legs are lifted simultaneously there is risk of losing balance. Since it is easier to prohibit this rather than creating a balancing algorithm for it, this is the preferred approach.

The function that determines how many legs are allowed in swing phase can be created in different ways. A simple approach with low computational cost is to have a lookup table. Based on the velocity and rotation there will be a quota of allowed legs in the lookup table. The limits for when to increase/decrease the quota are experimentally produced to work for all directions of velocity.

Another approach is to utilize the time distances calculated in the queue function

of mode 2. Creating an algorithm that determines how many legs to move based on the amount of time until they cross the boundary will generate longer periods between lifting legs. This is because legs will not be lifted until it is necessary. Because distance to boundary is known this allows for the controller to have a safety feature to avoid faulty positions. If a leg is about to exit its boundary and is not liftable due to other legs being lifted, the hexapod can halt its movement until lifting is possible.

Trajectory end positions

The function that generates trajectories for the swing phase of the legs needs a trajectory end position. Start position of the trajectory is always the leg's current position, but end position can be chosen. The main goal for the swing phase is to put the leg in an advantageous position based on the movement the leg will do in its later stand phase. Depending on the amounts of legs in swing phase simultaneously there will be different amounts of time in stand phase for a leg.

Since default stance positions for the legs are based on stability it is preferred for the legs to move around these positions when in stand phase. By using the default position, current velocity and rotation it is possible to calculate and end position ahead of the default position. This can be done by using (3.12).

$$endposition = (D_i + s_1 \cdot v) \cdot R_z(s_2 \cdot \theta) \tag{3.12}$$

This will generate an end position for a leg s seconds ahead of the default position if $s_1 = s_2$. If trajectories last for 1 second, legs will swing every 5th second when only moving one leg simultaneously. In the same sense legs will swing every 2nd second for two legs swinging simultaneously and every second when moving three legs simultaneously. By using s to calculate a position half the stand time ahead, legs will move around the default position during constant velocity.

Another approach is to move legs based on the leg constraints. When a leg needs to be lifted a path inside the boundary can be created based on current user input. This calculated path, is a possible path for the future stand phase of the leg going through the default position to achieve stabilization. An example of such a calculation can be seen in Figure 3.24.

Here the controller can use the start of this path as the end position for the trajectory. In that way stand phase will be maximized in length and the hexapod might save power due to not having to lift legs as often. Another way is to use a part of the path and choose an appropriate end position on the path to still have stand phase trajectory go through default position.

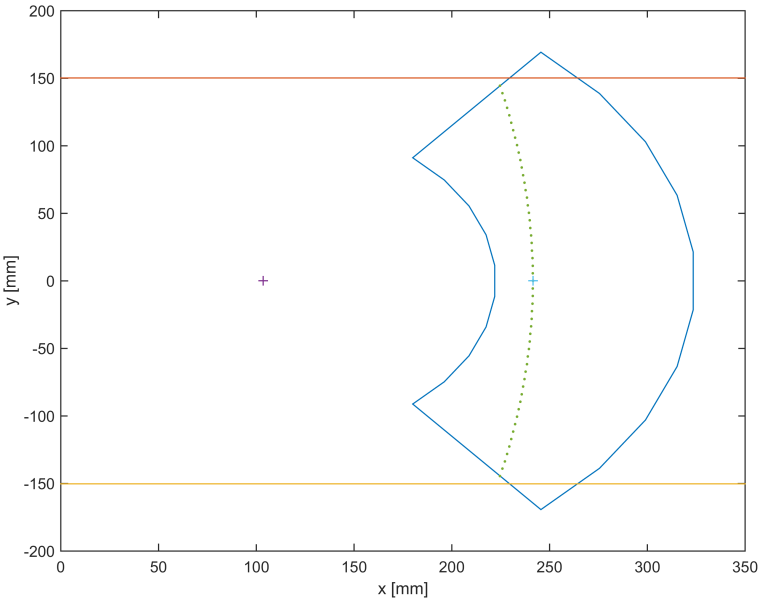

Figure 3.24 A path inside the boundary for leg five in stand phase. The green dots represent the path every tenth second. The blue plus is this leg default position for the current height. The upper lines are constraints based on the position of adjacent legs.

3.8 Stabilization

Terrain handling

Due to limitations in computing power and implementation time, image analysis method for terrain is not considered. Instead an IMU is used to estimate the environment around the hexapod.

The first approach to detect obstacles are to use accelerometer data. When a change in the accelerometer signal is detected and the command signal for move direction remains constant, an obstacle could be considered detected. This method is tested by walking into a wall at different speeds.

Because of the difficulties to detect obstacles with the method mentioned above another approach is used. When a leg of the hexapod steps onto a obstacle the main body of the hexapod will start tilt. This tilt is detected using the IMU and sensor fusion.

Balancing

The hexapod has a mode in which it will perform actions to keep the body leveled. If the main body is not leveled the IMU will send the offset angle to the controller. In order to compensate, (2.9) can be used to calculate differences in leg height on both sides. When standing still balancing is achieved by utilizing this, see (3.13)

$$P_i^{t+1}(z) = P_i^t(z) + P_i^t(z) \cdot \tan\alpha - P_i^t(z) \cdot \tan\beta \tag{3.13}$$

Where P is the leg position and the angles are defined in Figure 2.9. To later be able to walk and balance simultaneously the balancing also updates a floor angle parameter to keep track of the current ground angles. The balancing is only implemented as a feature when standing still. In order for movement not to compromise balancing it is implemented as a button mode on the remote, mode 6.

3.9 Communication

The structure of communication in the system is showed in Figure 3.4. The BeagleBone Black is the main computer in the system. Because of this as many parts as possible is connected to this board. The Dynamixel servos uses half-duplex, special connector and a custom communication protocol for communication with other boards. Because of the existing software framework on the ArbotiX-M it is used as a communication relay in the system.

IMU gyro and accelerometer

The MPU9150 is connected to the BeagleBone Black via the I2C bus. The I2C bus consists of two wires for communication and uses a master/slave communication model [NXP Semiconductors N.V, 2014, p. 3]. The BeagleBone Black acts as master and the MPU9150 act as Slave device. At start-up the sensitivity of the accelerometer and the gyroscope is setup and the IMU is taken out of sleep mode. After this the accelerometer and the gyroscope is sampled at 100 Hz. In the Simulink support package for BeagleBone Black no I2C driver is included so an S-Function is written in C to support communication with the IMU.

IMU angle estimation

MPU9150 contains a processor DMP [InvenSense Inc, 2013, p. 10]. Sensor fusion of accelerometer and gyroscope is performed by the DMP [InvenSense Inc, 2012, pp. 5-6]. The result of the sensor fusion is a quaternion which is read by BeagleBone Black. The quaternion is then merged with measurements from the magnetometer to compensate for drift.

All of this is done by code developed by [Pansenti, LLC, 2012]. The downloaded code from Github contains two demonstration programs to run from the

terminal. One program performs calibration of the accelerometer and magnetometer. Calibration is performed by recording maximum and minimum output of the accelerometer and the magnetometer for each axis. The data is collected by moving the IMU through six different orientations. The six orientations corresponds to positive and negative x,y,z axis points towards the floor in a Cartesian coordinate frame. Then the maximum and minimum readings for each axis of the accelerometer and magnetometer are used as calibration data. Calibration of the gyroscope is performed online after 8 seconds of zero motion [InvenSense Inc, 2012, p. 8]. The other program mentioned above configures the IMU to use the DMP and print Euler angles in a terminal. This program is tested in the Clode 9 IDE [Cloud9 IDE, Inc., 2015] on the BeagleBone Black. Clode9 is a IDE which runs on the BeagleBone Black and is accessed through the web browser. This demonstration code is rewritten to fit inside an S-Function Builder block in Simulink.

ArbotiX-M

To establish communication between the BeagleBone Black and the ArbotiX-M, UART with a baud rate of 115200 is used. Communication protocol between the two processors is created and the main functionality of the ArbotiX-M card is to read and write commands to the servos. Often all of the servos are updated at the same time. Because of this a message to set values of all 18 servos is constructed according to Table 3.6. When the ArbotiX-M receives this message it interpret it and sends the message to the servos.

Byte number	Value	Description
1	0xFF	Start byte
2	3	Instruction
3	0 to 255	Number of bytes in message
4	0 to 31	Starting register to write to
5	1 to 13	Number of bytes to write to each servo
6 - 7	0	Data for servo with ID 1
...	...	...
39 ... 40	0	Data for servo with ID 18

Table 3.6 Example of message for writing data to each servo.

To perform model verification of the servo model implemented in Simulink a message for reading data is implemented according to Table 3.7. The ArbotiX-M reads data from an arbitrary number of servos and then sends it back to the BeagleBone.

Byte number	Value	Description
1	0xFF	Start byte
2	4	Instruction
3	0 to 255	Number of bytes in message
4	0 to 31	Starting register to read from
5	1 to 13	Number of bytes to read from each servo
6	1 to 18	Number of servos to read from (n)
$7 \ldots 6+n$	1 to 18	Id of servos to read data from

Table 3.7 Example of message for reading data from servo.

ArbotiX-M to servo

Dynamixel AX12A is connected in a daisy-chain configuration using a linear topology. Only one wire is used for sending and receiving data. To achieve this on the ArbotiX-M card the Tx and Rx line from one UART port are combined. The software on the ArbotiX-M then switches the receiver and transmitter module of the UART to achieve half duplex communication. The baud rate between the AX12A and the ArbotiX-M card is 1 Mbps [Robotis, 2006].

XBee

The remote control that is delivered with the hexapod uses a XBee module to communicate with the BeagleBone Black. From the BegleBone Black the XBee module is seen as a UART port. On the remote an Arduino card is used to sample the user input(joystick and buttons) at 30 Hz and then sends them to the XBee module. This is done by the code included in the hexapod kit. Each message from the remote is sent as 8 bytes message according to Table 3.8. On the BeagleBone Black one UART port is used to communicate with the XBee module.

The communication with the XBee module is implemented in Simulink using the S-Function Builder block. First the following approach is used: At each sample instance the system reads eight bytes from the serial port and then interprets the message. The S-Function is sampled at a period of 0.3 seconds. This resulted in code that blocked the main control loop. Solving this is a S-Funciton that first checks the number of bytes available. If the eight bytes is not available the S-Function is finished and the bytes are read the next time the S-Function runs. The S-Function is sampled at a period of 0.0125 seconds. The joystick values are changed from the interval 0..255 to $-128..127$ for each axis.

Byte number	Value	Description
1	0xFF	Start byte
2	0 to 255	Right joystick vertical
3	0 to 255	Right joystick horizontal
4	0 to 255	Left joystick vertical
5	0 to 255	Left joystick horizontal
6	0 to 255	Buttons
7	0	Extras
8	0 to 255	Checksum

Table 3.8 Bytes in a message from the Commander.

3.10 Code generation

ArbotiX-M

To generate code from Simulink to the ArbotiX-M card the following approach is used: code from a subsystem in the Simulink model is generated as a C function. Using a .tlc file, a main program is also generated. By customizing this .tlc file call to a function which sends new position to the servos is called. The structure for the main program is showed in Listing 3.2.

```
1  int main(){
2
3      // Initialization code
4
5      while(true){
6
7          //Call to generade simulink code
8          generatedFunction();
9
10         //Send new references to servo
11         //Handwritten code...
12         sendPos();
13
14         //delay to next iteration
15     }
16 }
```

Listing 3.2 Structure of ArbotiX-M main program

BeagleBone Black

A support package for Embedded Coder is used when generating to the BeagleBone Black [MathWorks, 2015a]. This software made it easier to set up code generation

for BeagleBone Black than for the ArbotiX-M card. Support for running in external mode made it easier to verify the generated code. In external mode it is possible to monitor different signals in real time. To make use of the different communication interface on the BeagleBone Black custom C code is integrated using S-Function blocks.

Development of S-Function blocks for BeagleBone Black

S-Function Builder block is used to include C code and construct a Simulink block. The S-Function Builder block consists of a Simulink block were the C code is written in different tabs. In Figure 3.25 and 3.26 an S-Function Builder block which implements communication with MPU9150 is showed. The S-Function dialog window consists of seven different tabs see Figure 3.26. The "Initialization" tab is used to create discrete and continuous state of the S-Function. The "Data Properties" tab is used to set up input/output ports and parameters of the block. The "Libraries" tab is used to include references to external files used by the S-Function (header,source code,object and library files). In the "Outputs" tab, code that is executed each time Simulink evaluates the block is entered. This is defined by the sample time of the block. The "Discrete Update" tab is used to update discrete states of the S-Function. When implementing different S-Functions a discrete state is used to perform initialization code (opening and setting up ports). The Outputs tab is used to read and write data from the communication interface. When implementing different S-Functions in this work much help and advise is found in [Dustin Kahawita, 2014] and [Giampiero, 2014].

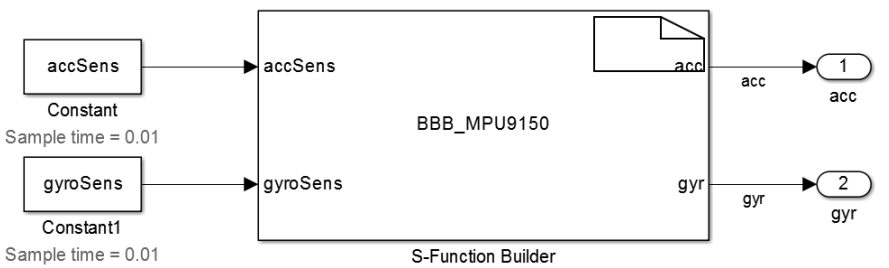

Figure 3.25 S-Function builder block in Simulink.

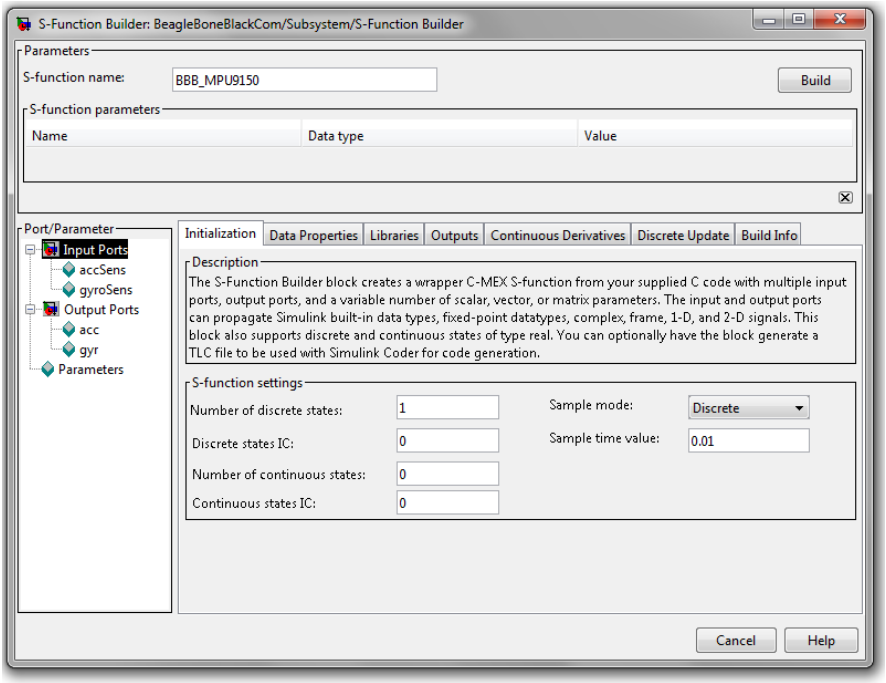

Figure 3.26 Dialog window for S-Function Builder block.

Verification of generated code

To verify the generated code PIL simulation is used. Code is generated for the control block and put on the BeagleBone Black. The control block is then executed on both PC and BeagleBone Black driven by the same input. A representation can be seen in Figure 3.27. Position references for the servos of one leg are measured from both control blocks. The differences between the measurements are then calculated. This method will detect if numerical differences exist between simulation running on PC and generated code running on BeagleBone Black.

Another important aspect is to consider is execution time of the generated code. To measure this, code is generated and executed on the BeagleBone Black. For each sample rate in the system Simulink generates one task in the generated code. The time required to run each task is measured by counting the number of ticks used by the CPU.

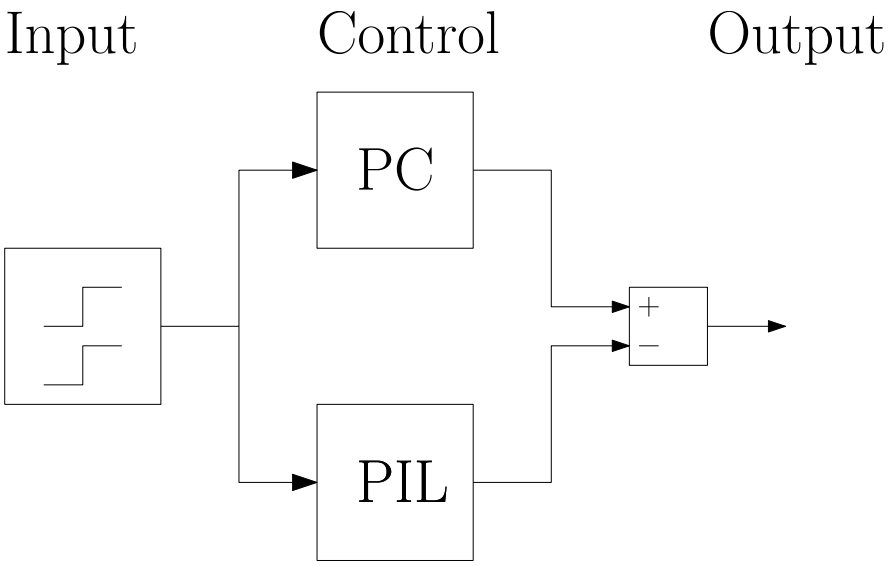

Figure 3.27 A illustration of how the generated code is verified. The block PC and PIL represents the control block.

4

Results

4.1 Chosen hardware

PhantomX AX Hexapod Mark II was the chosen hexapod platform. It was bought as a kit including the Dynamixel AX12 servos and an Arduino based MCU named ArbotiX-M. Also included was a battery of 2200 mAh, a remote and two circuit boards for wireless communication called XBee. The XBee uses the frequency 2.4 GHz and has an effect of 1 mW. The kit was shipped from USA and unfortunately delivered with a faulty MCU. This resulted in days of debugging before the American supplier agreed to send a replacement.

In order to upgrade the MCU a BeagleBone Black Rev C was later bought and code generation target changed to this board. The old MCU ArbotiX-M was still used as a communications relay to the servos due to pre-existing communication protocols. At the same time as the BeagleBone Black was bought a level converter was bought to cope with the different operating voltages of the two boards.

The need of an Internal Measurement Unit resulted in the buying a breakout board with MPU-9150 [SparkFun Electronics®, 2015a]. The modified PhantomX AX Hexapod Mark II with BeagleBone and IMU can be viewed in Figure 4.1.

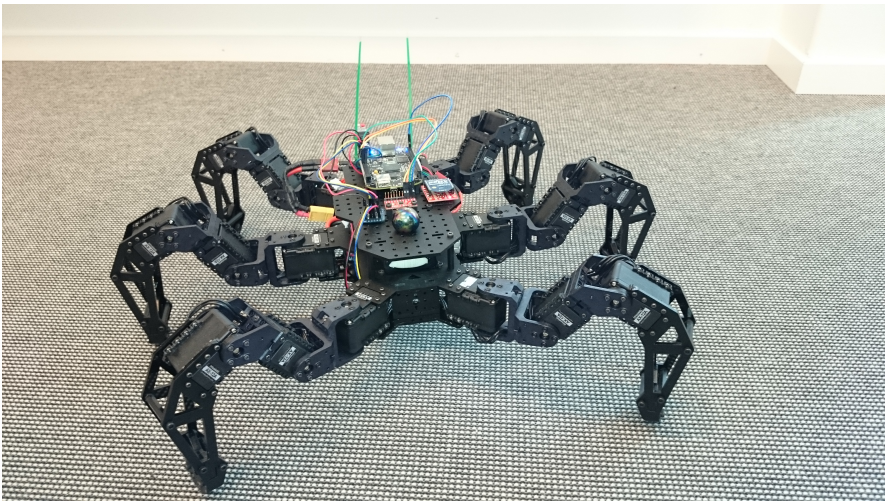

Figure 4.1 The modified hexapod containing BeagleBone and IMU on the top deck.

4.2 SimMechanics model

The hexapod was successfully modelled in SolidWorks and then later exported into SimMechanics. In SimMechanics the complete hexapod was assembled with joints using connection frames. A result of the created model and basic control of legs can be viewed as a video link found in section B.

Servo measurement

Result of step responses using the circular servo horn and two different loads are showed in Figure 4.2. Result of a step responses with circular disk and constant load of 303 g is plotted in Figure 4.3. As can be seen in the figures a stationary error exists, this is probably due to the compliance used in the servos [Robotis, 2006]. The controller gives a output torque which is dependent on the position error. When a greater load is applied to the servo, the output voltage does not produce enough torque to turn the servo.

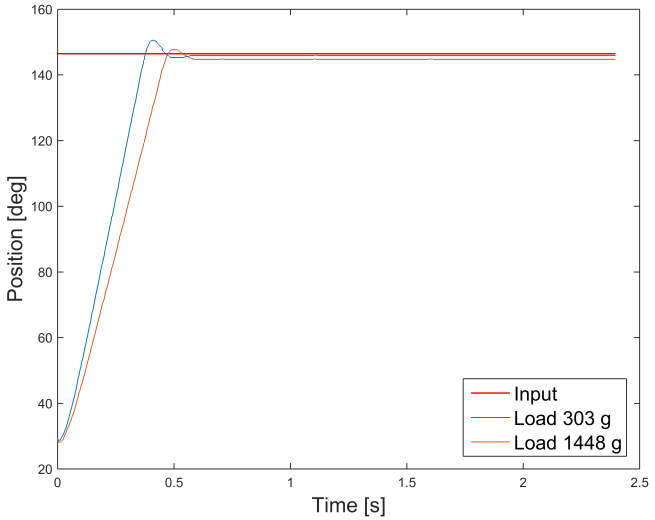

Figure 4.2 Step response with two different loads. Sample rate is 250 Hz.

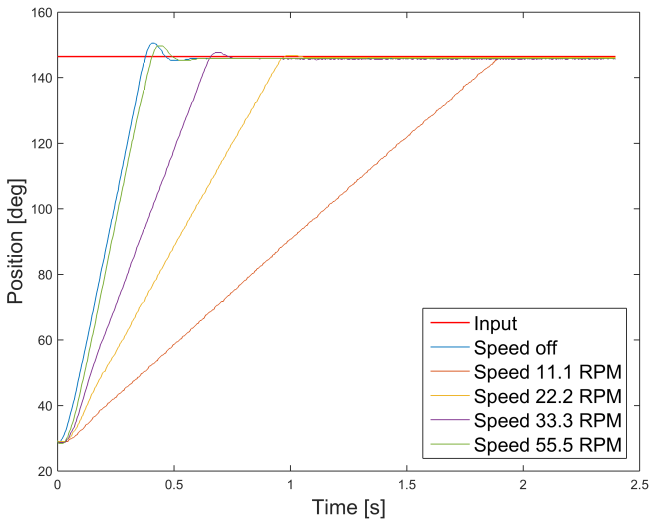

Figure 4.3 Difference between speed regulation modes. Sample rate is 250 Hz.

Servo identification

A single servo is modelled as a discrete PID regulator from position error to speed. This PID regulator also has a saturation on the output. The discrete transfer function for the regulator is given in (4.1).

$$C(z) = P + I \cdot T_s \frac{1}{1-z} + D \frac{N}{1 + N \cdot T_s \frac{1}{z-1}} \tag{4.1}$$

In Figure 4.4 the result of the parameter estimation is presented. It is a comparison between the model response and data used for estimation. To validate the parameter estimation a model response is compared against a step response from a servo mounted on the hexapod, Figure 4.5.

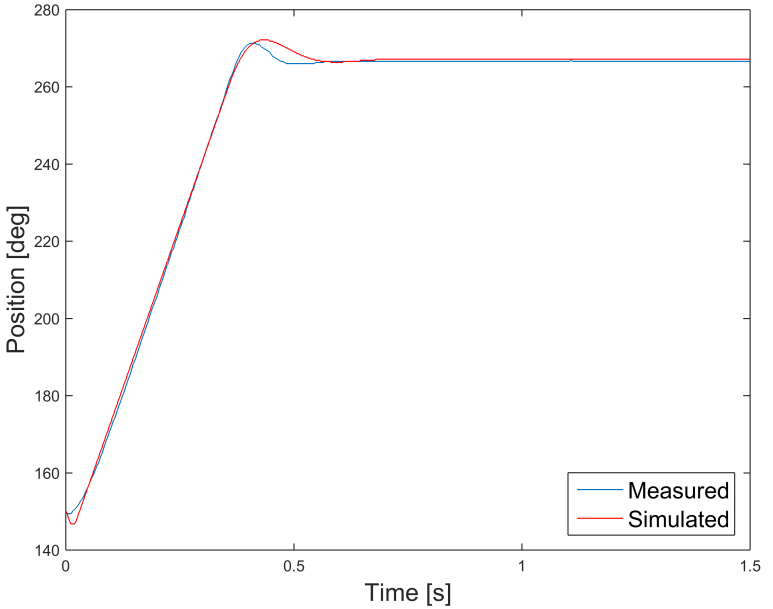

Figure 4.4 Result of servo estimation. Sample rate of measured data is 250 Hz. Load of servo is 303 g and no speed control is used.

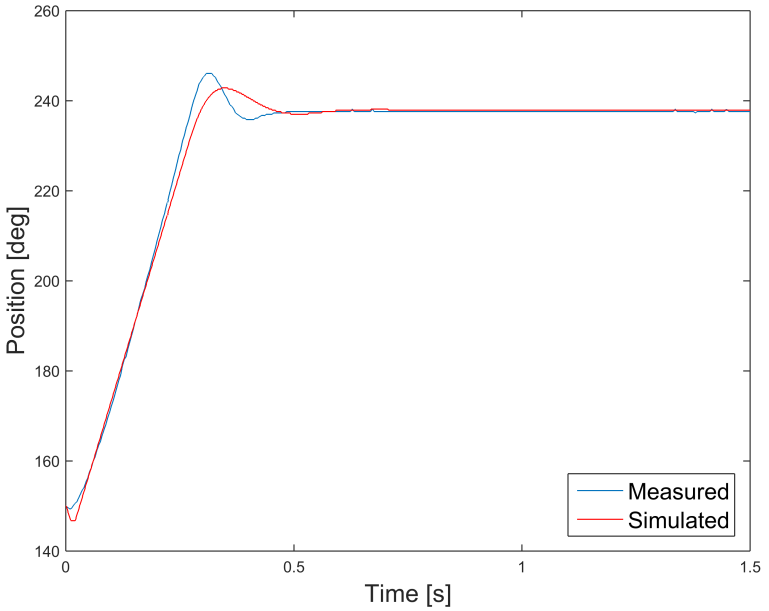

Figure 4.5 Result of validation servo estimation. Sample rate of measured data is 250 Hz. No speed regulation is used.

Contact force modelling

When contact forces were taken into consideration the computation time for the simulation was increased substantially. Tolerances used during the simulation needed to be decreased to achieve accurate simulation results. Often during simulation the hexapod ended up in strange positions and it was hard to get accurate results. Due to the need of prioritizing other parts of the project no more time was used to further investigate contact forces. For a visual representation of the contact force modelling refer to section B.

4.3 Control performance

Performance is divided into two sections, the initial performance using the ArbotiX-M board and the later usage of the BeagleBone Black. The ArbotiX-M card was discontinued as processing unit about 3 months into the project. A last section presents the results from trying to make smooth movement for the legs.

ArbotiX-M

Initially control was implemented on the original processing board, ArbotiX-M. The first implemented controller was a hard-coded locomotion pattern that allowed for walking straight. This confirmed the functionality of the servos and the ability to code generate. Further development included the IK and a controller for movement in arbitrary direction. The result of this was a hexapod able to walk but very jerky. Result was still satisfactory though, since the hexapod was able to put legs at appropriate positions, the IK was confirmed working. The controller built here was working at 10 Hz and this frequency was increased in order to cope with the jerky movement. Updating servo positions at 40 Hz made the hexapod move more softly, but it had a delayed response to user input. This was confirmed to depend on the ArbotiX-M not finishing calculations during designated sample time. A sample time of 20 Hz was able to be achieved with the ArbotiX-M still having time to finish calculations during sample time. For the resulting movement see section B.

The original controller NUKE was investigated and confirmed to run on a frequency of 30 Hz. But when investigated further, during usage of the NUKE controller it only calculate positions every second sample.

BeagleBone Black

The upgrade to BeagleBone Black resulted in a lot of communications protocol had to be implemented. ArbotiX-M only usage now was to relay reference positions to the servos and read data from the servos. The ArbotiX-M managed to keep up with position relaying when the BeagleBone works at 40 Hz. Reading data from the servos was implemented as a functionality but never used for control due to lack of time.

The queue was first implemented based on the distance calculated by (3.11). In constant velocity this would render a queue of which the leg moved last was placed first in the queue. In case of a velocity change, legs would start moving in another direction and the queue composition would change in order to comply with the new direction. Different values of the variable s were used but a value of 2 gave best results in sense of distance behind in the movement direction. Movement was still a bit unsatisfactory, so (3.11) was changed to (4.2) where time-distance s was individual for velocity and rotation. Using s_1 as 3 and s_2 as 1 generated better movement than previous value.

$$distance = |(D_i + s_1 \cdot v) \cdot R_z(s_2 \cdot \theta) - P_i^t| \tag{4.2}$$

This queue was combined with the trajectory function and a leg lifting function were different speed allows for different amount of lifted legs. In Table 4.1 a lookup table of allowed legs in air depending on speed and rotation can be seen.

The lookup table was the result of tests on both the model and the hardware. By walking at different speeds the needed amount of legs could be visually seen on the hardware and the model, when legs collided. In the model if a leg was put outside

Movement	Legs allowed to lift
$\|v\| = 0mm/s, \|rot\| < 5°/s$	1
$\|v\| = 0mm/s, \|rot\| < 18°/s$	2
$\|v\| = 0mm/s, \|rot\| > 18°/s$	3
$\|v\| < 5mm/s, \|rot\| = 0°/s$	1
$\|v\| < 40mm/s, \|rot\| = 0°/s$	2
$\|v\| > 40mm/s, \|rot\| = 0°/s$	3
$\|v\| > 0mm/s, \|rot\| > 0°/s$	3

Table 4.1 Lookup table for controller that determines the number of legs allowed in air simultaneous. Absolute values for velocity and rotation are used to cope with movement in negative direction.

of its physical boundary the IK would generate an error and thus additional legs needed to be lifted. These two methods are how the table was created.

Trajectory end position for this implementation was calculated as in (3.12). Values of s_1 were equal to s_2 and were chosen depending on amount of legs lifted. Variable s was chosen to correspond to half of the time the leg would spend in stand phase.

Resulting locomotion of the hexapod with these configurations was decent and referred to as mode 1. The walking was performed without errors for movement with no quick changes in user input. For a view of the movement see section B. Though if quick input changes were done the locomotion collapses in such a way that legs might position themselves badly. No constraints were taken into account for in the controller and due to that, legs would at some times try to position themselves outside the physically possible area. The result of that are leg servos positioning themselves in maximum angel, i.e., legs pointing upwards. The result of walking sideways was a fairly jerky movement where legs seem to forget to move when they position themselves underneath the hexapod body. Rotation and walking simultaneously at high velocity and high angular speed collapsed the movement into legs bumping into each other or pointing upwards as described previously.

Calculated trajectories can be compared against the resulting trajectory. Since there is no way of measuring the actual trajectory on the hexapod platform it was measured in the virtual model, the result is showed in Figure 4.6.

A queue based on time until legs violate constraints proves to be difficult to use and is referred to as mode 2. The queue was composed and legs were supposed to be lifted when they neared boundary edges. This resulted in lesser legs lifted, but the controller wanted to lift several or all legs simultaneously in such a way that it loses balance. It also made the hexapod forget to move legs in a strange way making it practically immobile. Trajectory end positions were here based on the start of the path calculated in Figure 3.24. There was a function for this mode that

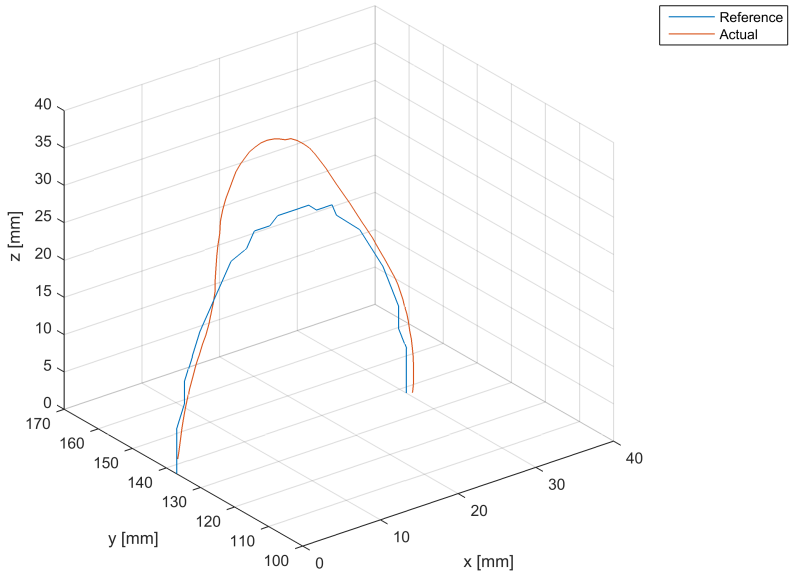

Figure 4.6 Virtual representation of the main controller trajectory (blue) and the resulting actual trajectory (red) as carried out by the IK in the virtual model.

halts movement before a leg was about to exit its boundary. The result was that the hexapod halts a lot and sometimes freezes in halt position, though legs never threw errors in such a way that they point straight upwards.

The function that resets legs when joysticks are released has shown to be very useful. It allowed for handling most error situations by just releasing the joysticks. The only error states not fixable by the function was when servos entered error state. In this situation the servo power needed to be reset manually.

Movement Smoothing

There exist no result for using the theory described in Section 2.2. Because of the results from servo identification this method were never developed. Results from servo identification showed that when servos have a set maximum angular speed, a period of delay was introduced after receiving a set point. If Figure 4.3 is zoomed in around the origin it shows a delay that depends on maximum speed, see Figure 4.7. The delay ranged from 10 ms for no maximum angular speed up to 50 ms for a maximum angular speed set to 11.1 RPM.

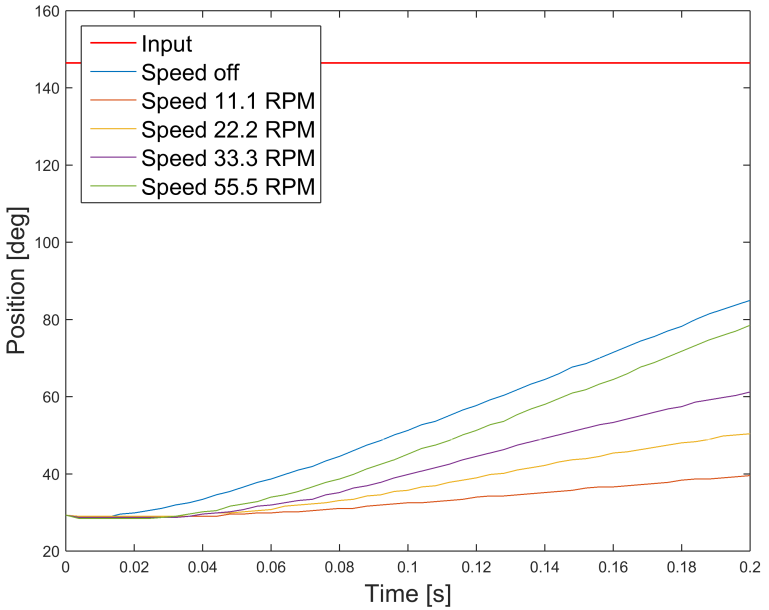

Figure 4.7 Same as Figure 4.3 but zoomed in to only contain information regarding the first 200 ms. This figure illustrates the resulting delay introduced when a maximum speed is set. In the graph is can be seen that a delay exist before the servo responds to a new set point.

4.4 Stabilization

Terrain

Terrain identification was almost not investigated. This was because the accelerometer and gyroscope data from the IMU was very wobbly during walking. Time and difficulty never allowed for creating functions to identify leg object contact from the IMU data during walking. Tests of colliding with walls were done to see results on IMU data. Using this method made it hard to detect obstacles because the response from the accelerometer differed depending on which movement speed was used when the collision took place.

IMU-DMP

During startup of the IMU-DMP the gyroscope needed to be calibrated. Because of this it took some time before the angles were stabilized, this process can be seen in Figure 4.9. To observe and estimate drift of the calculated Euler angles, the IMU was mounted on the hexapod and Euler angles were logged for 9 minutes. For this

measurement initial gyroscope calibration was removed. To measure the drift of the three angles a first order polynomial was fitted to the measurement $y = k \cdot x + m$. The coefficients for the first degree term is presented in Table 4.2. The fitted polynomial and measurements for the Euler angles are showed in Figure 4.8. During balancing, only roll and pitch angles were used in the controller. Due to not using the yaw angle, the larger drift had no impact on control.

Angle	Rate of change [deg/s]
Roll	$-3.91 \cdot 10^{-6}$
Pitch	$-2.17 \cdot 10^{-6}$
Yaw	$-3.51 \cdot 10^{-3}$

Table 4.2 Drift of Euler angles.

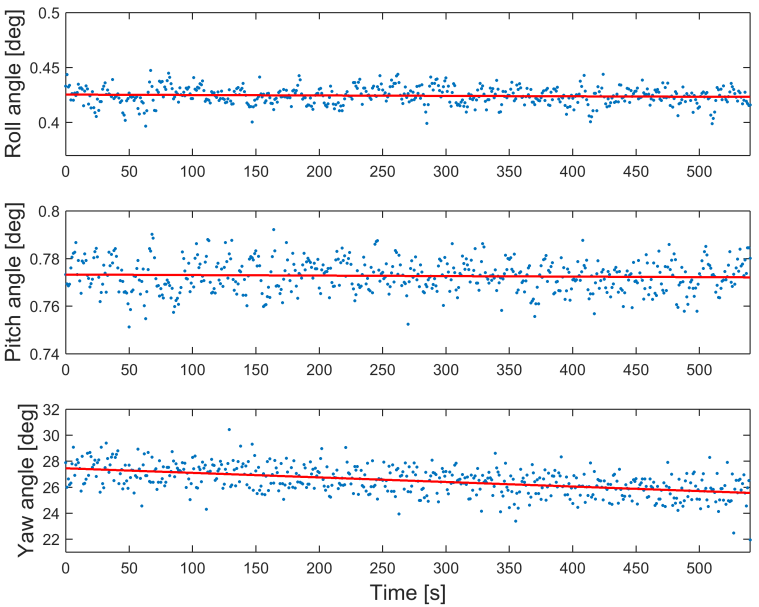

Figure 4.8 Drift of roll, pitch and yaw angle. Measurement is done after initial calibration of the gyroscope. Sample rate is 1 Hz.

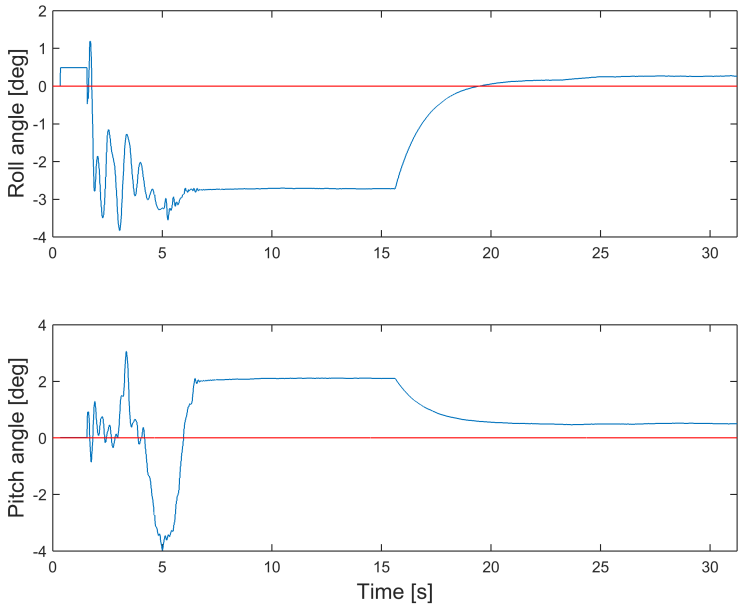

Figure 4.9 Roll and pitch angles during start-up. The gyroscope is calibrated after about 15 seconds of no movement. Sample rate is 80 Hz.

Balancing

The usage of (3.13) resulted in a balancing algorithm that made the hexapod oscillate back and fourth ferociously when the floor was not level. Dividing the α and β angles by a constant lowered the ferociousness of the oscillating. By dividing the IMU data by five a stable balancing algorithm was implemented. The performance of the balancing algorithm can be seen in Figure 4.10 The hexapod was able to keep the main body level for floor angles that were not too large. For a visual representation of the resulting balancing, see section B. When the floor angle was too large the hexapod had trouble keeping the body level and the hexapod also started sliding on the floor.

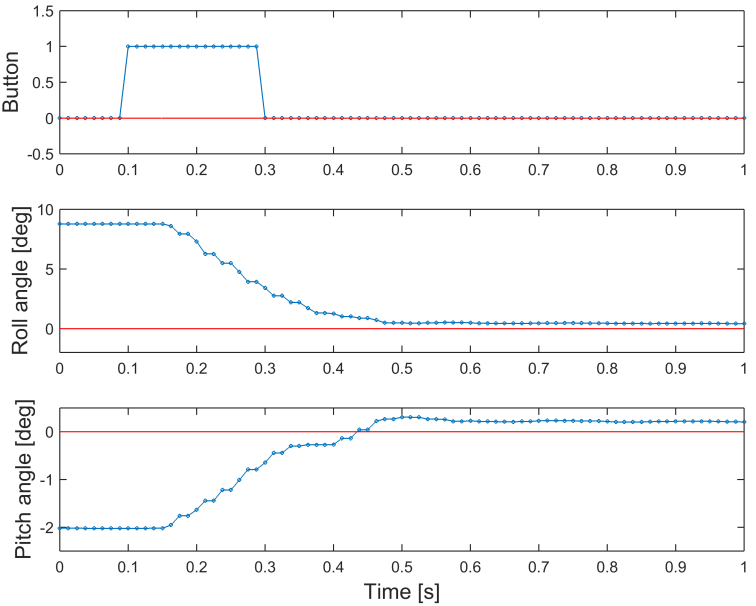

Figure 4.10 Performance of the balancing algorithm. The hexapod starts at a tilted table, at 0.1 s the stabilization algorithm is activated. At 0.6 s the hexapod has stabilized the main body. Sample rate is 80 Hz.

4.5 Performance of generated code

When performance of the generated code was evaluated two aspects were considered; controller output and execution time. Output from the controller running on a laptop and on the BeagleBone Black was compared using PIL simulation. The result of the measurements can be seen in Figure 4.11.

Due to the real-time nature of the system timing requirements are important. Simulink divides the code to run in a number of tasks. Execution time for this tasks were measured in PIL simulation mode with the code running on the BeagleBone Black.

As a first approach the controller were executed at four different sample rates with resulted in four different tasks, Figures 4.12 and 4.13. Because of the bad performance, the system were change to only run at two different sample rates. When this was done the code for reading the remote was also rewritten to improve performance. The improved code resulted in better timing performance, Figures 4.16 and 4.17.

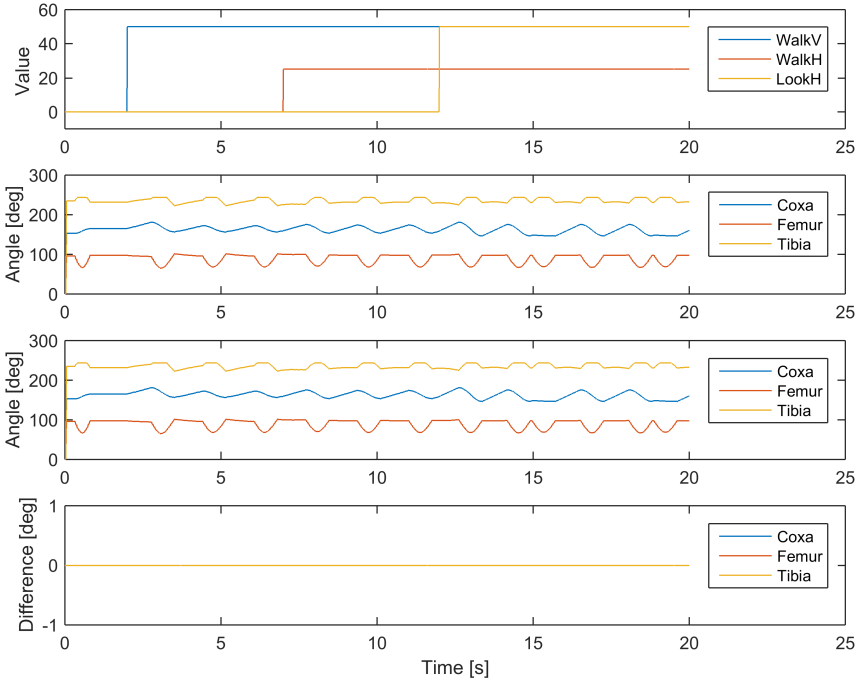

Figure 4.11 Comparison between control output of the Simulink system and the same control systems' generated code running on BeagleBone Black. The same input is simulated for both the Simulink system and the generated code. From top, plots shown are simulated control input, Simulink system output, generated code output and difference between the two outputs.

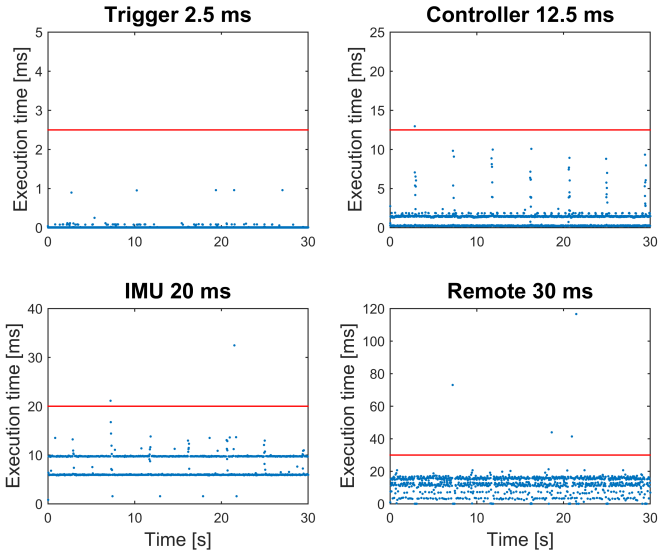

Figure 4.12 System running at four different sample rates. Title shows what part of the system that is running and time limit for each task.

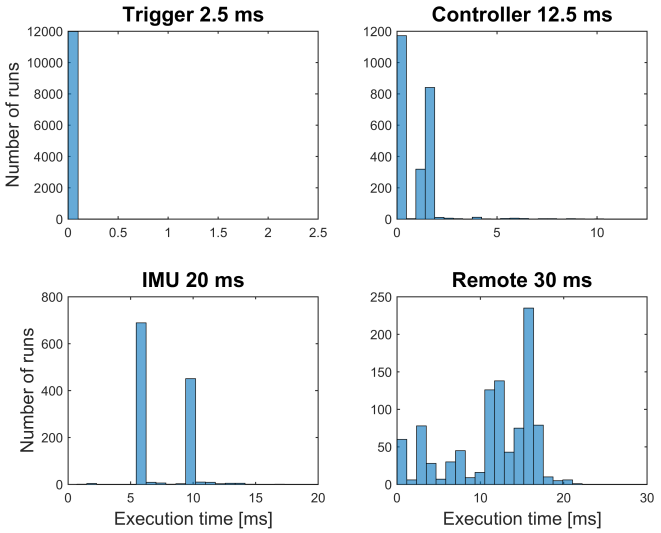

Figure 4.13 Distribution of execution time from Figure 4.12.

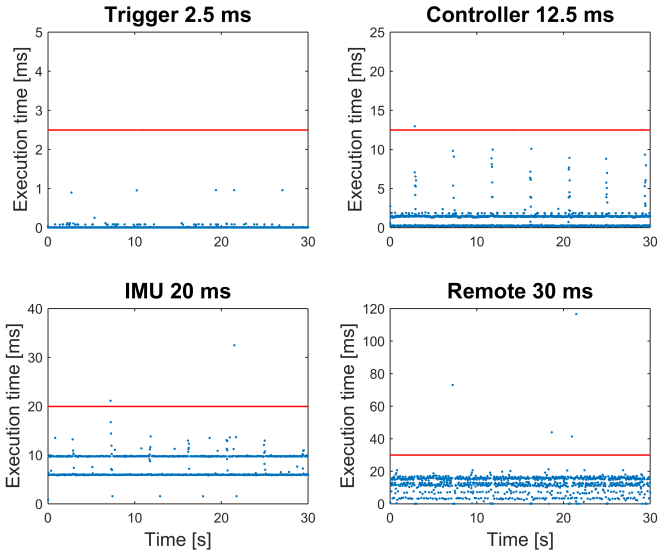

Figure 4.14 System running at four different sample rates. Title shows what part of the system that is running and time limit for each task.

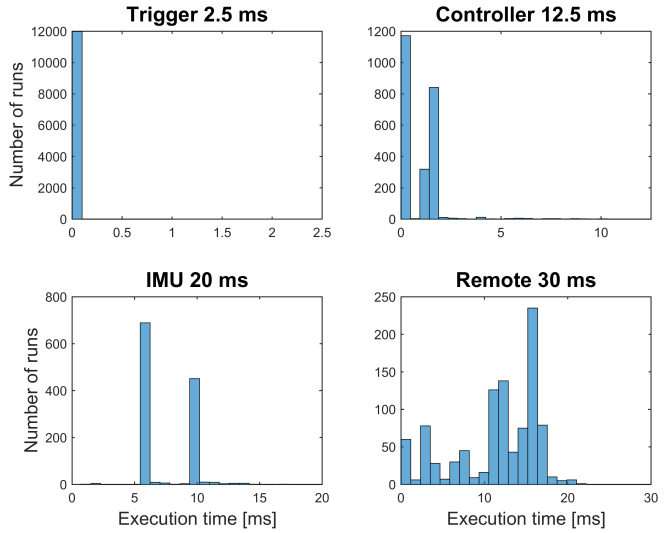

Figure 4.15 Distribution of execution time from Figure 4.12.

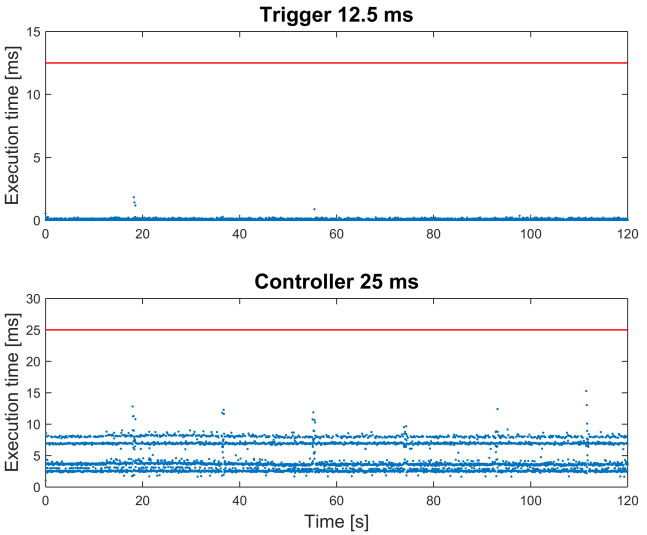

Figure 4.16 System running at two different sample rates. Title shows what part of the system that is running and time limit for each task.

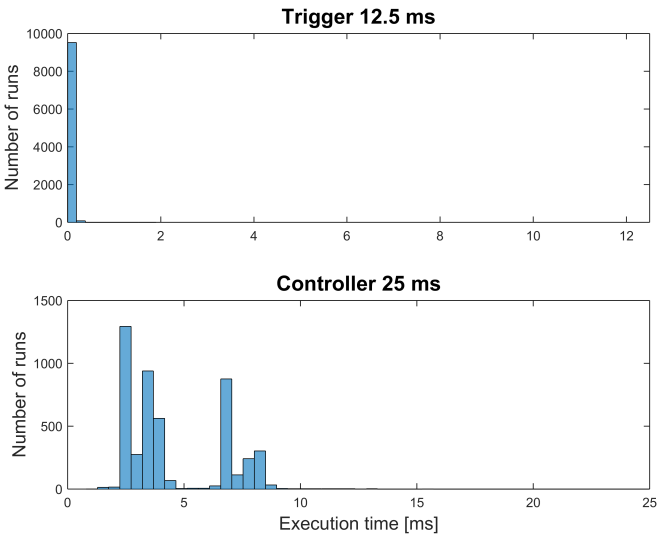

Figure 4.17 Distribution of execution time from Figure 4.16.

5

Discussion and Conclusions

5.1 Expectations

The projects stretched over several phases of software development and hardware integration and the main goal to achieve a terrain navigating hexapod was never fully achieved. But a stable platform for control development was created and further thesis work on the platform will hopefully be able to achieve terrain walking. Model-Based Design were very successfully used in the project and the utilization of a visual computer model aided the locomotion development in a very valuable way.

Hardware

The hexapod platform created was based on the PhantomX kit mentioned earlier. This kit was chosen mostly due to the fact that the kit utilizes the smart servos Dynamixel AX12A. These servo allows for on-line measurement of servo states due to their inboard MPU. By being able to measure data such as servo position, servo speed, and servo torque the idea was to be able to get feedback from the environment. Due to lack of time this data was never used in the controller. The only time data was read from the servos was when doing model verification. Unfortunately, the additional data being sent over the communication channel had severe effect on the hexapod performance. Because of this the servo feedback readings were never expected to work during the current sampling frequency without improving upon the communication somehow.

Included in the kit was the ArbotiX-M that was supposed to be the controller of the system. Already at the early stage of choosing hardware there were suspicions that the computational power would not be enough. It was desired not to add too many different controller boards to the project. Because of the unusual half-duplex communication other boards than the ArbotiX-M were expected to force the project into communication development. That not being in the scope of the project the

control implementation were initiated on the ArbotiX-M. As it was realized that the 16 MHz were not going to be enough another board had to be bought. To focus more on control and sensor usage the ArbotiX-M was still used as a communication relay link in the system even though the goal was to minimize delay. This proved to compromise the servo data readings, but was never addressed as a major problem due to the controller not utilized servo feedback.

Model-Based Design

Model-Based Design was the most successful part of the project. Using the computer model it was possible to initiate control implementation in Simulink even before the hexapod kit was assembled. If the Model-Based development actually saved the project time was hard to establish since half of the project has been development of the model and setting up the development environment. But further work will probably be done on the platform and since the development environment works properly additional projects will definitely save time if utilizing it. The visual part of the model was extremely useful when developing locomotion and balancing. Controller algorithms could easily be tested and verified without downloading them to the hardware which would have taken much longer time and worn the hexapod. Extensive savings in effort were done by being able to distinguish the extremely unreliable patterns from the promising ones by just watching the model.

Unfortunately the model was never able to move in a realistic manner on virtual terrain. The implementation of contact between objects such as hexapod and ground were not supported well enough by SimMechanics. Contact forces were expected to be integrated into the model, but the time required for this was decided not worth investing. It was enough to have the model show implemented locomotion without the feedback of ground beneath it. The advantages of the model were worth all the time invested in creating the model.

Locomotion

The development of locomotion was expected to reach a state where a dynamical walking pattern would be able to walk on uneven surfaces and still keep the hexapod main body stable. Unfortunately, this goal was never achieved, mostly due to the lack of time or the unexpected amount of time needed to assemble a working system. Development of terrain handling was never initiated at all due to more focus on developing a working locomotion pattern. The final walking pattern developed as mode 1 showed promising when used for low and medium speeds and no quick input changes. It was able to seamlessly change the amount of legs used based on the user input and walk in a controlled manner in any direction. Development of a height alteration feature was implemented and accessed by trigger buttons on the remote. The height alteration was implemented as a future means for the hexapod to utilize when navigating terrain. If large objects would obstruct the path and walking around were not an option, altering the height would provide another option.

Since the terrain handling was never implemented the height alteration remained a user feature only. In the same way balancing the hexapod was implemented as a means for controlling the level of the body when walking uphill/downhill. Due to lack of time the balance algorithm was only implemented as a very basic controller and only works when the hexapod is not moving. It was accessed as a feature by a mode button press that disconnected the joysticks in order for movement not to compromise the balancing.

For quick joystick changes the hexapod behaved in a partly chaotic manner and legs did unexpected kicks and sometimes the full hexapod folded itself together. As this was discovered to depend on the physical boundary of leg placement a constraint algorithm was implemented to remedy this behaviour. The constraint algorithm by itself did not show much promise. Unfortunately, time did not allow for the implementation of a combination of the two walking algorithms.

A result from the hard constraints implemented in the IK were that even if legs folded or bumped into each other they were able to recover when better reference positions were sent to them. In combination with the repositioning of leg to default stance when standing still this allowed for fast recovery of the hexapod if the joysticks were released.

5.2 Discussion

Model-Based Design

Having not only a virtual hexapod as a mathematical plant, but also using the visual representation allowed fir examining built locomotion patterns in an intuitive way. The algorithms for locomotion were implemented using Simulink blocksets and MATLAB Functions. From the Simulink environment, C code could be generated both to the initial Arbotix-M processing card and to the later upgrade replacement BeagleBone Black. By using the more advanced BeagleBone Black a Simulink external mode allowed for monitoring of signals while the simulation was running.

Using Model-Based Design it was possible to develop software without having access to the actual hardware. This allowed several developers to test and evaluate designs of different control systems. By using CAD models inside SimMechanics several different mechanical system could be evaluated before they were even built. This would save money that would have been required to construct the real models. However workload for people who builds models would be reduced. This lead to less people required to build prototypes. As product development becomes more digital less work will be spent prototyping real world models.

Compared to programming embedded systems by hand coding, Model-Based Design offered the ability to easier test and evaluate different designs by using subsystems and subsystems libraries. If the hardware was changed the code could be changed by changing code generation target.

SimMechanics model

The existing CAD parts of the hexapod made it easier to model and assemble the complete hexapod in Solidworks. The integration with SimMechanics through SimMechanics Link made it possible to export the hexapod. When combining CAD models it was of importance to place connection frames correctly in SolidWorks. This was important to consider when using existing SolidWorks assemblies for export to SimMechanics. The benefit of using the CAD model was the ease to model more complex geometries in SolidWorks. Weights and moment of inertias were also calculated by SolidWorks and then exported. In this project weight of different parts of the hexapod were measured. In the end, this resulted in a more detailed model than needed. One of the reasons for weighing all the parts was to get correct torque from each joint. But the torque measurement from the servos were not used because of two reasons, low update frequency and inaccurate measurement. Inaccurate measurements were a result of the use of current measurement for torque estimation instead of a real torque sensor. When the SimMechanics model was completed integration with the rest of Simulink environment were possible.

Modelling contact forces in SimMechanics was at the moment a complex task. The methods used in this project required significantly more computation time. When using Model-Based Design it was good to have fast simulation times. This reduced development time when many simulations were required. This was the case when tuning controllers using different optimization techniques for e.g., parameter tuning. When verifying controllers it was also important to have correct and fast simulations. Design of models was always a trade-off between detail and simulation speed.

Servo modelling

As shown in Figure 4.2 the top speed of the servo was reduced when a greater load was applied to the servo. In both cases the speed of the servo was linear, this suggest that the control output of the servo saturated.

When using speed regulation on the servos a delay was introduced in the system, Figure 4.7. For this reason speed regulation of the servos were not used. Only position references were sent to the servos. This resulted in a PID controller with speed saturation as a servo model.

The maximum speed of the servo was dependent of the load of the servo, Figure 4.2. As the weight for the different leg servos were in the same magnitude different speed saturation were not modelled. As can be seen in estimation and validation results, Figures 4.4 and 4.5, the saturation speed of the model was slightly lower than the measured result in Figure 4.5. This was due to the load on the servo used in the estimation was greater than the load in the validation data. A better approach of modelling this would be to collect estimation and validation data for each servo (coxa, femur, and tibia). By using these data three different saturation speeds could be determined, one for each leg servo. Another complication was that the maximum

speed of the the coxa servo is dependent on the position of the femur and tibia servo. When the positions of the tibia and femur changed, the moment of inertia for the coxa servo changed. Because of the change of inertia the maximum speed of the servo was changed. A servo model which outputs maximum torque for a given speed limit was one approach of solving this problem. By then using a 2-DOF controller with position and speed error as input and torque as output would give a more accurate servo model.

There were also some difference in the size of the overshoot between the simulated and measured servo response. However, the model was considered to be detailed enough for the control implementation since locomotion patterns were easily evaluated in the model.

Control performance

Compared to the original open source controller the one developed with Model-Based Design could be considered equally as good or even better. Both of them allowed for movement in any direction, rotation and the changing of locomotion pattern. In addition, the one created as part of this thesis allowed for height alteration, seamless gait transitions and balancing when standing still. Unfortunately the created one still had errors that showed as legs doing unpredicted kicks and sometime legs were not moved into position. Legs also had a tendency to bump into each other during quick movement or rapid direction changes. Though the original controller showed the same leg bumping behaviour for rapid user input changes. In order to cope with some of the problems of walking and rotating simultaneously a rotation speed limit was introduced. The limit sat in when walking and rotating simultaneously and limited the maximum allowed rotation speed depending on the current walking speed.

Balancing was achieved in a satisfactory manner when implemented as (3.13). The oscillatory behaviour first achieved was avoided by using a gain of 0.2 on the IMU data when calculating new leg positions. No measurements of the oscillatory behaviour exist since the hexapod had to be turned off to prevent hardware damage. But results of the working balancing as showed in Figure 4.10 gave a good step response that settled in about 0.5 seconds using the gain of 0.2. If stabilization were to be used during walking the ground angle would probably not change like a step but rather as a slope. In that case the current performance should be sufficient. If more time would have been available a full system identification of the balancing could have been done. Then the balancing controller algorithm could have been optimized using the Simulink toolbox to design the controller. Since the balancing was achieved by altering the leg heights, legs tended to slide a bit on the surface. In an uneven terrain this would maybe compromise the balancing since legs might slide down from object they are standing on and thus compromise the balancing. By achieving balancing by rotation the legs around the x- and y-axis instead their positions on the surface will remain the same. The advantage of altering leg heights

instead of rotating them around the body was that they would remain inside the leg constraints and thus not result in position errors. This should be considered if the balancing is to be done on an uneven surface.

The implemented controller was not been optimized for a low calculation cost. A sample frequency faster the 40 Hz has not yet been tried and not felt necessary. At the current state the controller has sufficient time for the calculations of servo positions each sample. But if the controller would grow more advanced and for example include a camera some optimization might be necessary.

The inverse kinematics for the legs of the hexapod needed to run six times each sample. If the update frequency of the controller were to increase or a less powerful CPU were to be used, calculation problem with sample time may occur. If this is the case performance could be increased by using lookup tables for trigonometric functions. Fix-point implementation could also be use to increase performance. This could not only be implemented for the IK but for the whole controller, but as the IK is the algorithm that runs the largest amount of times this is where to start.

Generated code

The comparison between the generated code and the model running on a laptop, Figure 4.11, showed that the output matched. The output of the control block was rounded off into degrees in steps of 0.29°. This reduced difference in machine precision of the two hardware platforms. The equivalence between the two models also showed that the control block met the real-time requirements. However this did not take the interfacing with the XBee, IMU and ArbotiX card into account. The evaluation of the whole system in PIL, Figure 4.16, showed that the real-time requirements were met.

Running the Simulink model at several different sample rates showed that the real-time requirements were not met, Figure 4.12. This was due to the fact that code for reading the remote blocked the rest of the system when it waited for all eight bytes to arrive. When this problem was identified, the code for reading the remote was rewritten. When this was done the sample rate of the controller was also changed to reduce the number of sample rates in the system. The result of this showed that the real-time requirements were met, Figure 4.16. When building Simulink model for execution on embedded system it was of importance to keep the number of sample rates in the system as low as possible. However, some situations might require a computational intense filtered to run at a slower rate. So, in conclusion, many different aspects need to be taken into consideration.

As presented in Figure 4.17, the execution time fell into two different groups. This was a result of the triggering system used, since some subsystems were only executed every second time.

Code generation

Automatic code generation for two different hardware platforms has been used. Both required different modification to the code generation process to generate code that worked with the custom process. For both implementations some of the code needs to be handwritten and integrated with the generated code.

The benefit of using automatic code generation is the possibility to verify generated code against simulation result from Simulink.

Integration of C-code into Simulink using S-Functions took some time to get used to. But when one S-Function had been developed and the creation of S-Functions were understood, developing more S-Functions was easy. It is worth mentioning that the development time of the S-Functions in this project required more time than expected. This code was also specific for the hardware used as coder target. If for example the hardware were to be replaced by a Raspberry PI, some similarities exists but some differences were also present. For example the Raspberry PI had fewer UART ports than the BeagleBone Black. Also the baudrates that could be used is dependent on the clock used for UART.

Developing UART communication in Linux was complicated because many different settings existed through the Unix API termios [Kerrisk, 2015]. In this project this was encountered when reading data from the remote. Eight bytes of data that was sent from the remote needed to be read at one time. To solve this first a code that blocked the rest of the controller was used. This resulted in some spikes in execution time performance, see Figure 4.12. Later another approach was used; first the number of bytes available were checked. If eight bytes were not available no data was read. By running this code at a rate of 80 Hz the performance of the generated code was improved, see Figure 4.16.

A good way of testing the developed S-Function was to place it in a separate Simulink model. By then using external mode in Simulink it was possible to evaluate performance. PIL simulation mode could also be used to evaluate execution time. When these tests are done the S-Function can be integrated into the main model.

5.3 Conclusions

Overall this thesis highlights the major stages in the development process of a control system using Model-Based Design. Because of the wide scope of this work, knowledge from many different engineering and scientific disciplines was required. This resulted in the use of CAD-tools, electrical engineering, control, mathematics, physics and programming. A lesson learned from this project was that it is important to have a broad understanding of the system which is going to be modelled. When e.g., modelling the servos, knowledge if speed regulation were to be used or not was important.

Using CAD modelling in combination with some setup time and customization of the CAD model to integrate with SimMechanics. It is also of importance to use the correct level of detail in the CAD-model. Too many solid bodies in SimMechanics resulted in longer simulation times. A more detailed model required more development time which could been spent in developing other parts of the system.

The modelling of the Dynamixel servos is a complex task with many different aspects to consider. The developed model in this project is considered to be detailed enough for the control development.

Developing locomotion for a hexapod has been a tough task that has required a lot of testing. It can not be expressed enough how useful the virtual model has been. When code generation was changed from ArbotiX-M to BeagleBone Black the locomotion development could be continued even though code generation was not set-up for the new platform. By having the model available as a repository, control development can be done anywhere and without access to the actual hardware and still get feedback on performance.

Handling terrain with a hexapod was a hard task and can by itself be considered a thesis subject. The vast complexity of creating dynamical locomotion required a lot more time than was available during this work. But even though most effort have been put into creating a platform for Model-Based Design a working walking pattern was created. This proves that Model-Based Design and code generation works for such a delicate system as a 18 degrees-of-freedom hexapod.

5.4 Future improvements

A lot of things can be improved upon the hexapod and the controller implemented. This section will cover some of the ideas that were never carried out due to lack of time.

- With more time the controller would be developed further with more functionality. Implementation of balancing would be further developed to combine walking and balancing simultaneously. By changing (3.1) to account for the floor's current angle this could probably be achieved. Implementing a function that could identify if a leg is placed on ground or not would help developing terrain handling. By utilizing either the servo feedback or the IMU data, identification of ground contact would allow for the swing phase to end when the swung leg gets in contact with the terrain. In that manner walking would not have to be done on a flat surface.

- The current walking algorithm still contains a lot of errors. By creating a more advanced algorithm that incorporates the effects of leg constraints, balance, power consumption and visual aspect the resulting locomotion would be more reliable. An option could be a weighting function that takes into account the cost in power and balance when deciding to lift a leg into swing phase.

- The option to use a camera for navigation was fully dismissed in this work due to the lack of time and our limited knowledge of image analysis. Though the BeagleBone Black supports the usage of a camera and Simulink already contain support for incorporating a camera feed into the controller without having to create additional block-sets.
- The current state of the hexapod looks a bit like a temporary solution for wiring on the upper deck. Wires could be drawn in a more organised manner and the creation of a body shell would improve a lot upon the visual aspect of the hexapod. This would also protect the open electronics from being damaged.
- An easy solution to the identification of ground contact for the legs could be to exchange the rubber feet to rubber buttons. By having buttons send a state signal the controller would have constant knowledge of ground contact for all legs. A disadvantage might be that these buttons would have to be very robust in order to not break. Also the buttons might have difficulties for standing on angled surfaces.

Bibliography

Ahmadian, M., Z. Nazari, N. Nakhaee, and Z. Kostic (2005). "Model based design and sdr". In: *DSPenabledRadio, 2005. The 2nd IEE/EURASIP Conference on (Ref. No. 2005/11086)*, p. 8. URL: http://ieeexplore.ieee.org.ludwig.lub.lu.se/stamp/stamp.jsp?tp=&arnumber=1575352.

Baumgart, A., P. Reinkemeier, A. Rettberg, I. Stierand, E. Thaden, and R. Weber (2010). "A model–based design methodology with contracts to enhance the development process of safety–critical systems". English. In: Min, S. et al. (Eds.). *Software Technologies for Embedded and Ubiquitous Systems*. Vol. 6399. Lecture Notes in Computer Science. Accessed: 2015-05-22. Springer Berlin Heidelberg, pp. 59–70. ISBN: 978-3-642-16255-8. DOI: 10.1007/978-3-642-16256-5_8. URL: http://dx.doi.org/10.1007/978-3-642-16256-5_8.

BELTER, D. and P. SKRZYPCZYŃSKI (2010). "A biologically inspired approach to feasible gait learning for a hexapod robot." *International Journal of Applied Mathematics and Computer Science* **20**:1, p. 69. ISSN: 1641876X. URL: http://ludwig.lub.lu.se/login?url=http://search.ebscohost.com.ludwig.lub.lu.se/login.aspx?direct=true&db=edb&AN=48797636&site=eds-live&scope=site.

Campos, R., V. Matos, and C. Santos (2010). "Hexapod locomotion: a nonlinear dynamical systems approach." *IECON 2010 - 36th Annual Conference on IEEE Industrial Electronics Society*, p. 1546. ISSN: 9781424452255. URL: http://ludwig.lub.lu.se/login?url=http://search.ebscohost.com.ludwig.lub.lu.se/login.aspx?direct=true&db=edb&AN=81541054&site=eds-live&scope=site.

Cloud9 IDE, Inc. (2015). *Cloud9 ide homepage*. Accessed: 2015-04-28. URL: https://c9.io/.

Coley, G. (2014). *BeagleBone Black System Reference Manual*. Revision C.1. beagleboard.org.

Combine (2013). *Model-based development*. Accessed: 2015-04-23. URL: www.combine.se.

Dürr, V., J. Schmitz, and H. Cruse (2004). "Behaviour-based modelling of hexapod locomotion: linking biology and technical application." *Arthropod Structure and Development* **33**:Arthropod Locomotion Systems: from Biological Materials and Systems to Robotics, pp. 237 –250. ISSN: 1467-8039. URL: http://ludwig.lub.lu.se/login?url=http://search.ebscohost.com.ludwig.lub.lu.se/login.aspx?direct=true&db=edselp&AN=S1467803904000301&site=eds-live&scope=site.

Dustin Kahawita, R (2014). *Development of an ARM/Linux based testbed for rapid control system prototyping via Matlab/Simulink*. Accessed: 2015-04-28. MA thesis. École Polytechnique de Montréal. URL: https://github.com/rdustinkahawita/BLACKlink/blob/master/BLACKlink_Documentation/BLACKlink_Manual_RDK_rev2.pdf.

Embedded360 (2010). *Illustration v-model*. Accessed: 2015-06-09. URL: http://www.embedded360.com/execution-approach/traditional-model.htm#.

Fielding, M. (2002). *Omnidirectional Gait Generating Algorithm for Hexapod Robot*. TJ 211, 415 F459 2002. PhD thesis. Mechanical Engineering, University of Canterbury. URL: http://hdl.handle.net/10092/6027.

García-López, M., E.Gorrostieta-Hurtado, E. Vargas-Soto, J. Ramos-Arreguín, A. Sotomayor-Olmedo, and J. M. Morales (2012). "Kinematic analysis for trajectory generation in one leg of a hexapod robot." *Procedia Technology* **3**:The 2012 Iberoamerican Conference on Electronics Engineering & Computer Science, pp. 342 –350. ISSN: 2212-0173. URL: http://ludwig.lub.lu.se/login?url=http://search.ebscohost.com.ludwig.lub.lu.se/login.aspx?direct=true&db=edselp&AN=S2212017312002666&site=eds-live&scope=site.

Giampiero, C. (2014). *Writing a Simulink Device Driver block: a step by step guide*. Accessed: 2015-04-28. The MathWorks, Inc. URL: http://www.mathworks.com/matlabcentral/fileexchange/39354-device-drivers/content/DriverGuide.zip.

GreenCarCongress (2009). *Dongfeng motor company uses mathworks tools for model-based design of battery management system for hybrid bus*. Accessed: 2015-04-23. URL: http://www.greencarcongress.com/.

Hendricks (2014). *Hexapod crawler robot*. Accessed: 2015-05-06. URL: https://grabcad.com/library/hexapod-crawler-robot-1.

Instituto Superior Técnico, H. R. L. of the Department of Mechanical Engineering at (2011). *Weights of different parts*. Accessed: 2015-05-20. URL: http://humanoids.dem.ist.utl.pt/Iden_external/overview.html.

InvenSense Inc (2012). *Embedded Motion Driver 5.1.1 Tutorial*. Revision 1.0. Accessed: 2015-04-28. URL: http://www.invensense.com/developers/

index.php?_r=downloads&ajax=dlfile&file=MotionDriver_Tutorial_.pdf.

InvenSense Inc (2013). *MPU-9150 Product Specification*. Revision 4.3. Accessed: 2015-04-27. URL: http://www.invensense.com/mems/gyro/documents/PS-MPU-9150A-00v4_3.pdf.

Jensen, J., D. Chang, and E. Lee (2011). "A model-based design methodology for cyber-physical systems." *2011 7th International Wireless Communications & Mobile Computing Conference (IWCMC)*, pp. 1666–1671. ISSN: 9781424495399. URL: http://ludwig.lub.lu.se/login?url=http://search.ebscohost.com.ludwig.lub.lu.se/login.aspx?direct=true&db=edb&AN=80369646&site=eds-live&scope=site.

Kerrisk, M. (2015). *Documentation of termios*. Accessed: 2015-06-09. URL: http://man7.org/linux/man-pages/man3/tcsetattr.3.html.

Lambersky, V. (2012). "Model based design and automated code generation from simulink targeted for tms570 mcu." *2012 5th European DSP Education and Research Conference (EDERC)*. Accessed: 2015-04-28, p. 225. ISSN: 9781467345958. URL: http://ludwig.lub.lu.se/login?url=http://search.ebscohost.com.ludwig.lub.lu.se/login.aspx?direct=true&db=edb&AN=89784746&site=eds-live&scope=site.

Leifsson, L. ., H. Sævarsdóttir, S. . Sigurðsson, and A. Vésteinsson (2008). "Grey-box modeling of an ocean vessel for operational optimization." *Simulation Modelling Practice and Theory* **16**:EUROSIM 2007. Accessed: 2015-05-21, pp. 923–932. ISSN: 1569-190X. URL: http://ludwig.lub.lu.se/login?url=http://search.ebscohost.com.ludwig.lub.lu.se/login.aspx?direct=true&db=edselp&AN=S1569190X08000488&site=eds-live&scope=site.

Lin, P.-C., H. Komsuoglu, and D. Koditschek (2006). "Sensor data fusion for body state estimation in a hexapod robot with dynamical gaits". *Robotics, IEEE Transactions on* **22**:5, pp. 932–943. ISSN: 1552-3098. DOI: 10.1109/TRO.2006.878954.

MathWorks (2015a). *Beaglebone black support from embedded coder*. Accessed: 2015-05-11. URL: http://se.mathworks.com/hardware-support/beaglebone-black.html.

MathWorks (2015b). *Embedded coder user's guide 2015a*. Accessed: 2015-05-11. URL: http://se.mathworks.com/help/releases/R2015a/pdf_doc/ecoder/ecoder_ug.pdf.

MathWorks (2015c). *Simulink coder™ user's guide 2015a*. Accessed: 2015-05-11. URL: http://se.mathworks.com/help/releases/R2015a/pdf_doc/rtw/rtw_ug.pdf.

Miller, S. (2014). *Simmechanics contact forces library*. Accessed: 2015-05-20. URL: http://www.mathworks.com/matlabcentral/fileexchange/47417-simmechanics-contact-forces-library.

Murray, C. J. (2010). *Automakers opting for model-based design*. Accessed: 2015-04-24. URL: http://www.designnews.com/.

NXP Semiconductors N.V (2014). *UM10204: I2C-bus specification and user manual*. Revision 6. Accessed: 2015-04-28. URL: http://www.nxp.com/documents/user_manual/UM10204.pdf.

Ohlsson, N. and M. Ståhl (2013). *Model-Based Approach to Computer Vision and Automatic Control using MATLAB Simulink for an Autonomous Indoor Multirotor System*. EX014/2013. MA thesis. Department of Signals and Systems, Chalmers University of Technology, Sweden. URL: http://publications.lib.chalmers.se/records/fulltext/179662/179662.pdf.

Pansenti, LLC (2012). *Github project linux-mpu9150*. Accessed: 2015-04-27. URL: https://github.com/richards-tech/linux-mpu9150.

Ridderström, C. (2003). *Legged locomotion: Balance, control and tools — from equation to action*. Stockholm, Trita-MMK: 2003:19. PhD thesis. Department of Machine Design, Royal Institute of Technology, Sweden. URL: http://ludwig.lub.lu.se/login?url=http://search.ebscohost.com.ludwig.lub.lu.se/login.aspx?direct=true&db=cat01310a&AN=lovisa.001493015&site=eds-live&scope=site.

Robotis (2006). *Dynamixel AX-12A manual*. Accessed: 2015-05-20. URL: http://support.robotis.com/en/product/dynamixel/ax_series/dxl_ax_actuator.htm.

ROBOTIS INC (2015). *CAD drawings robotis*. Accessed: 2015-05-06. URL: http://en.robotis.com/BlueAD/board.php?bbs_id=downloads&mode=view&bbs_no=26324&page=1&key=&keyword=&sort=&scate=DRAWING.

SparkFun Electronics® (2015a). *Sparkfun 9 degrees of freedom breakout - mpu-9150*. Accessed: 2015-04-27. URL: https://www.sparkfun.com/products/11486.

SparkFun Electronics® (2015b). *Sparkfun 9 degrees of freedom imu breakout - lsm9ds0*. Accessed: 2015-04-27. URL: https://www.sparkfun.com/products/12636.

Spong, M. W., S. Hutchinson, and M. Vidyasagar (2006). *Robot modeling and control*. Hoboken, N.J. : Wiley, cop. 2006. ISBN: 0471649902. URL: http://ludwig.lub.lu.se/login?url=http://search.ebscohost.com.ludwig.lub.lu.se/login.aspx?direct=true&db=cat01310a&AN=lovisa.001645177&site=eds-live&scope=site.

Trossen Robotics. *PhantomX AX hexapod mark II kit*. Accessed: 2015-06-09. URL: http://www.trossenrobotics.com/phantomx-ax-hexapod.aspx.

Trossen Robotics. *PhantomX hexapod assembly guide*. Accessed: 2015-02-12. URL: http://learn.trossenrobotics.com/10-interbotix/crawlers/phantomx-hexapod/133-phantomx-hexapod-assembly-guide.html.

Wei, S., C. Kyungeun, U. Kyhyun, W. Chee Sun, and S. Sungdae (2012). "Intuitive terrain reconstruction using height observation-based ground segmentation and 3D object boundary estimation." *Sensors (14248220)* **12**:12, pp. 17186 –17207. ISSN: 14248220. URL: http://ludwig.lub.lu.se/login?url=http://search.ebscohost.com.ludwig.lub.lu.se/login.aspx?direct=true&db=a9h&AN=84496918&site=eds-live&scope=site.

A

Model parameters

Results of parameter estimation for the discrete transfer function of the PID regulator (4.1) is showed in Table A.1. T_s is the sample time of the PID regulator.

Name	Value
P	4.1814
I	-0.001736
D	-0.18253
N	33.234
satvel	336.88
T_s	0.01 s

Table A.1 Parameters from servo identification.

After weighing of different parts and combination of data found at [Instituto Superior Técnico, 2011]the final weight used for the different parts of the hexapod is showed in Table A.2.

Part	Weight [g]
Complete system	1801
Main body (small battery)	631
Coxa	24
Femur	72
Tibia	99

Table A.2 Weight used for different parts of the hexapod.

B

Visual results

Here are some links presented for the reader to be able to get visual representation of the resulting locomotion.

- A video showing the model and a basic walking pattern can be found here: https://www.youtube.com/watch?v=cNsxK2tae8k&feature=youtu.be.
- For the unsatisfactory result of contact forces refer to: https://www.youtube.com/watch?v=CZE3yZiU938&feature=youtu.be.
- View of the movement achieved when using the ArbotiX-M, use the link: https://www.youtube.com/watch?v=U6y1bwKrU7k.
- Better performance was achieved with more advanced control and the BeagleBone Black as processor, see: https://www.youtube.com/watch?v=vGBNpEx8doc.
- The implemented balancing working on a stationary hexapod can be seen using the following link: https://youtu.be/dCN-lKQaNCw.

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

Document name
MASTER´S THESIS

Date of issue
June 2015

Document Number
ISRN LUTFD2/TFRT--5971--SE

Author(s)
Dan Thilderkvist
Sebastian Svensson

Supervisor
Simon Yngve, Combine Control Systems AB
Anders Robertsson, Dept. of Automatic Control, Lund University, Sweden
Rolf Johansson, Dept. of Automatic Control, Lund University, Sweden (examiner)

Sponsoring organization

Title and subtitle
Motion Control of Hexapod Robot Using Model-Based Design

Abstract

Six-legged robots, also referred to as hexapods, can have very complex locomotion patterns and provide the means of moving on terrain where wheeled robots might fail. This thesis demonstrates the approach of using Model-Based Design to create control of such a hexapod. The project comprises the whole range from choosing of hardware, creating CAD models, development in MATLAB/Simulink and code generation. By having a computer model of the robot, development of locomotion patterns can be done in a virtual environment before tested on the hardware.

Leg movement is implemented as algorithms to determine leg movement order, swing trajectories, body height alteration and balancing. Feedback from the environment is implemented as a internal measurement unit that measures body angles using sensor fusion.

The thesis has resulted in successful creation of a hexapod platform for locomotion development through Model-Based Design. Both a virtual hexapod in Sim-Mechanics and a hardware hexapod is created and code generation to the hardware from the development environment is fully supported. Results include successful implementation of hexapod movement and the walking algorithm has the ability to walk on a flat surface, rotate and alter the body height. Implementation also contains a successful balancing mode for the hexapod whereas it is able to keep the main body level while the floor angle is altered.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
1-96

Recipient's notes

Security classification