Development of a Quadruped Robot with feedback control

Technical Report · November 2015

CITATIONS
0

READS
4,541

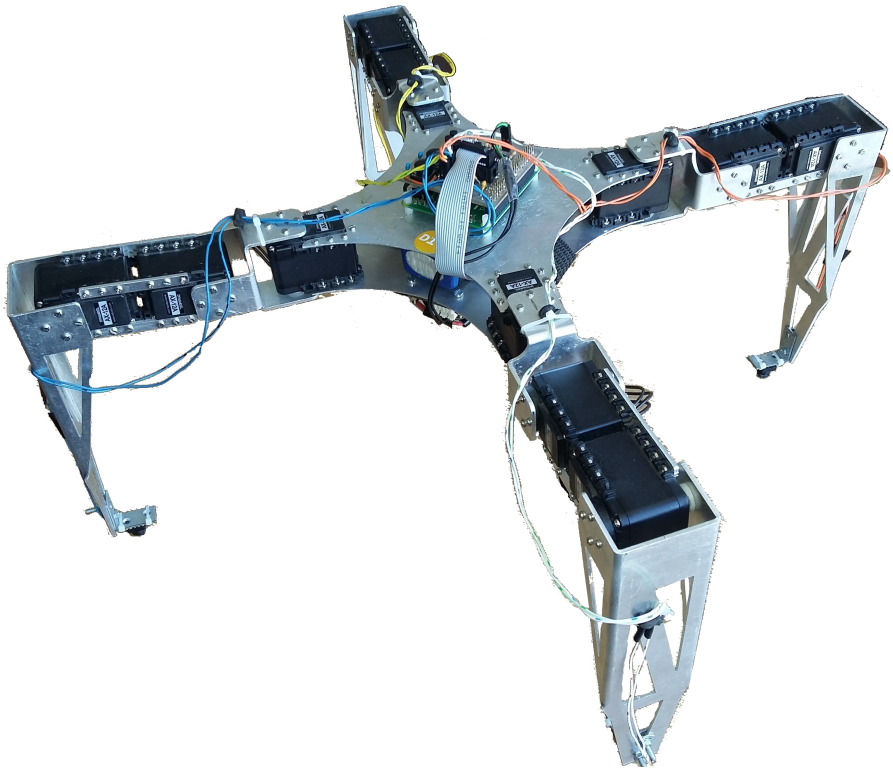
2 authors, including:

Jurie H. Wessels
Stellenbosch University
4 PUBLICATIONS **1** CITATION

SEE PROFILE

Development of a **Quadruped Robot** with feedback control

Jurie Hendrik Wessels
16992482



STUDY LEADER : Prof. Thomas Jones
November 2015

Report submitted in partial fulfillment of the requirements of the module Project (E) 448 for the degree Baccalaureus in Engineering in the Department of Electrical and Electronic Engineering at the University of Stellenbosch.

Declaration of own work

I, the undersigned, hereby declare that the work contained in this report is my own original work unless indicated otherwise.

Signature

Date

Abstract

Legged robots are are highly researched in die modern day and age for a variety of reasons, including military use and to help where its dangerous for people to operate. In this project systems are designed for a already-built four legged robot. These systems include a smart method to control the legs of the quadruped, a device to control the robot remotely, as well as sensors to determine the body's orientation and if the feet is touching the ground. This enables the quadruped to walk over uneven terrain while keeping its balance.

The finished product is successful and all designed systems work as expected. The quadruped successfully keeps its body parallel to the horizon in real time. The included gaits propel the quadruped forward at a reasonable speed, while taking a change of floor height into account.

Abstrak

Baie navorsing is gedoen in die hedendaagse tyd oor benige robotte. Dit word gedoen vir verskeie redes, wat insluit vir die weermag, sowel as om te help in plekke waar dit gevaarlik is vir mense om te beweeg. In hierdie projek is verskeie sisteme ontwerp vir 'n voorafgeboude vier benige robot. Hierdie sisteme sluit in 'n slim sisteem om die bene te beheer, 'n apparaat om die robot draadloos te beheer, sowel as sensors om die robot se orintasie af te skat en te bepaal of die voete die grond raak. Hierdie sisteme stel die robot in staat om oor 'n onewe oppervlak te stap terwyl sy dit sy balans behou.

Die finale produk is suksesvol en al die sisteme werk soos verwag word. Die robot kan sy lyf parallel hou met die horison. Die gemplimenteerde stap patrone kan die robot vorentoe beweeg teen 'n redelike spoed, terwyl 'n variering in vloer hoogte in ag geneem word.

Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviation	**Explanation**
CD	Compact Disc
CPU	Central Processing Unit
FIFO	First In - First Out
GUI	Graphical User Interface
h_{fe}	Transistor current gain
IO	Input/Output
PID	Proportional Integral Derivative
RPi	Raspberry Pi
SPI	Serial Peripheral Interface
TCP	Transmission Control Protocol
UART	Universal Asynchronous Receiver and Transmitter

1 Introduction

Walking machines has always spiked the interest of the world, but was never realized until the past couple of decades. One of the first references to walking machines was by H.G Wells with the 1898 book "The War of the Worlds" [3]. The author talks about big - almost unstoppable - three legged machines wandering the earth and terrorizing humanity. The idea of walking machines however never ventured further than science fiction and horror stories.

Today, however, walking machines are highly researched by the military to help troops transport heavy equipment over rough terrain. This resulted in large quadruped robots being designed, such as BigDog [4], which was designed by Boston Dynamics in 2005. Large competitions are also held to encourage the research into walking robots that can do complex tasks in dangerous, degraded, human-engineered environments. A example of such a challenge is the DARPA Robotics Challenge [5].

Walking robots like BigDog and DRC-HUBO [6], the winner of the DARPA Robotics Challenge, needs a wide variety of feedback systems giving the robot information about its environment. This includes information about its body's current orientation, the terrain around it, its current location and heading, etc. It also needs advanced walking patterns to maximize efficiency, speed, or stability, depending on the current environment and goal.

Research in these fields are still in fairly early stages, and can still be greatly improved upon. The curiosity and passion for robots as familiar and intelligent as any household pet - or even more - will push this research further than most can dream.

1.1 Objective

The objective of this project is to give a quadruped robot the ability to walk over unknown terrain. The walk should adapt to the terrain by using orientation sensors and sensors on the feet to determine when the foot is touching the ground. Another walk should also be implemented in which the robot should at a quicker pace. The robot should be controlled remotely.

This objective can be simplified into a few sub-objectives, which will be discussed throughout this report:

1. Design inverse kinematics algorithms.
2. Design system to control legs in a structured and intuitive way.
3. Determine the quadruped's orientation with the use of sensors.
4. Design system to keep the body's orientation parallel to the ground.
5. Design a walking gait which adapts to it's surroundings.
6. Design and implement a remote control system.

1.2 Scope

This project focuses on hardware and software alike. Some hardware are given as requirements for this project, such as the ADIS16405 Inertial Sensor [7], as well as a pre-built quadruped robot and battery. An processing platform, however, is chosen to control the quadruped. Additional sensors and circuitry needed is designed and built to support the processor.

The software is written from scratch, except for a few available libraries, such as [8]. The software includes a method to control the legs of the quadruped, read relevant sensors to determine information about the quadrupeds environment, adapt to the environment, and communicate with a remote control. The written software is supplied in a CD attached to the report.

1.3 Layout Of Document

In Section 1 a brief overview of the project is supplied. This includes the main objective of this project, the planned steps to be taken, as well as the scope of the project. The literature study in Section 2 includes an in-depth analysis of all the requirements for this project. Section 3 is the detailed design of each aspect of this project. This includes circuits, algorithms and system behavior. In Section 4 the results of this project is shown . It shows if the project met the design requirements, and explain why. An conclusion is written in Section 5, which includes a brief explanation on how the different objectives is achieved, as well as recommendations for improving the design.

2 Literature Study

2.1 Leg Movement

The quadruped has four legs, each with three degrees of freedom. This means each of the four legs has three joints which can each move in one axis. The quadruped will have to control these limbs in unison to move. To achieve this a method known as inverse kinematics will be used. Inverse kinematics is a method used to determine the angle each joint of a leg should rotate to, in order to place the feet at a desired location. To use this method the desired foot co-ordinates, relative to the shoulder of the robot, is needed. It will then implement a set of algorithms to determine how each joint should behave. These algorithms may sometimes return more than one solution. The mechanical limits of the legs results that only one solution will be possible to move the feet to a specific location.

The body should also be able to rotate in all three dimensions, which are roll, pitch and yaw. Thus when the desired co-ordinates of the feet are determined it should take the rotation of the quadruped into consideration. The robot should be able to rotate it's body while standing still, and even walking, without it interfering with where the feet are positioned. This will be used for balance.

To prevent the legs of the quadruped of moving in a jerky manner, and not smoothly, the movements of the servos should be broken down to smaller movements. This will give greater control over the speed of the movement, because the time delay between each small section can be chosen.

A magnetometer can be used to ensure that the quadruped walk in a completely straight line, and potentially adjust the gait if the quadruped starts to walk skew. This will however only have a big effect if the quadruped walks over large distances, which is not what the quadruped is designed for.

2.1.1 Locomotion

The quadruped must be able to propel itself to different locations. This will be done by walking with the quadruped's four legs. There are different walking patterns, or gaits, that the quadruped can use to walk. Each pattern has advantages and disadvantages, including speed, efficiency, and stability.

2.1.2 Creep Gait

The creep gate is a statically stable gait for a quadruped [1]. This gait only lifts up one leg at a time and moves it forward while moving the legs on the ground in a backwards motion, essentially pushing the quadruped forward. To keep the quadruped stable would be mean to keep the center of gravity inside the stability triangle formed by the three feet on the ground, as seen in Figure 1. This can be done by shifting the body from side to side during the gait,

or by changing the way the quadruped moves. This gait is very stable, easy to control, and very energy efficient.

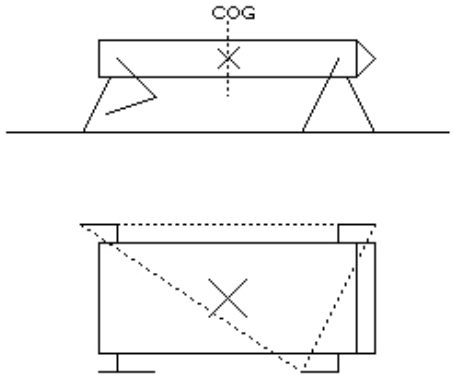


Figure 1: Center of gravity in respect the the stability triangle [1]

Since the creep gait is fairly stable, it's easily possible to increase the walking algorithm to compensate for climbing small steps. To achieve this a sensor will be added on each foot of the quadruped to sense if the foot is touching the ground. The software will then adjust the locomotion of the quadruped to compensate for the foot height.

2.1.3 Trot Gait

The trot gait is a faster moving gait than the creep gait [1]. By using this gait the system is dynamically stable, which means it must be moving to be stable. If moved to slow, it becomes unstable. It works by lifting two diagonal opposite legs simultaneously and moving it forward while pushing the other two legs back.

The downside of this gait is that it's very energy inefficient, and will put a large strain on the frame of the quadruped. The frame of the acquired quadruped is built with aluminum which is quite strong, and should be able to handle the strain of this gait.

2.1.4 Software Implementation

The processing unit will be in control of the quadruped's legs, as discussed in Section 2.3.1. This will be the only form of communication to the quadruped's legs. It will also occupy a large portion of the software needed, and demand a large amount of processing time. This is because of all the complex mathematics needed to describe and manipulate a 3D space. Therefore the software controlling the servo's needs to be efficient and not computationally demanding.

The quadruped will need to execute other operations while controlling the legs - such as communication to the remote control and determining the angle of the quadruped. This means that the software controlling the legs must not halt the execution of the processor until the leg is done moving. One way of avoiding this will be by using threads. This method will be complex to manage and implement. A more effective way will be by using a polling system to move the legs to their destination.

2.2 Processing Unit

The processor is a device that controls the entire robot. It will calculate the inverse kinematics and all similar calculation, as well as controlling the legs accordingly by communicating with the servos, as discussed in Sections 2.1 and 2.3.1. It will retrieve acceleration data from an tri-axial accelerometer and gyroscope, and perform the necessary calculations to accurately determine the quadruped's orientation, as discussed in Section 2.4. It must also be able to connect to a remote control, as discussed in Section 2.3.3.

This means that the processor must be able to handle a complex algorithms at reasonable speeds, have a UART and SPI available, and have the ability to add a method of control. Two options are the Arduino Mega 2560 [9] and a Raspberry Pi A+ [10]. The Raspberry Pi A+ is a smaller model Raspberry Pi which would be a good fit for this project. In Table 1 these two potential processors are compared.

Table 1: Comparison between two processors

Specification	**Arduino Mega 2560**	**Raspberry Pi A+**
CPU Speed	17MHz	700MHz
Input Voltage	7 - 12V	5V
Operating Voltage	5V	3.3V
Number of IOs	54	26
UART	Yes	Yes
SPI	Yes	Yes
WiFi	Shields available	Dongles available
Programming Language	Based on C	Python, Java, C++, etc.
Programmable Memory Available	±246KB	±500MB

It is clear that the Raspberry Pi will be able to handle the large number of calculations much easier than the Arduino due to the faster CPU speed. It also has a much larger memory, which means the the only limit on the software will be processing time, and not memory as well. The Raspberry Pi is thus chosen due to it's speed and available memory.

2.3 Communication

2.3.1 Serial Connection To Leg Servos

The legs of the pre-built quadruped are equipped with servos, which controls the quadruped's movement. The servos used are Dynamixel's AX-12 servos [11]. Most servos use pulse width modulation, or PWM, to control their movement. The AX-12 servos use a different technique. Each servo has a internal microprocessor which can be communicated to with the use of a universal asynchronous receiver/transmitter, or UART [2]. This means the AX-12 servos are more versatile, and can are easily controllable.

The protocol used by the AX-12 is a half duplex asynchronus serial communication (8 bit, 1 stop, no parity). This means that both the sending, and receiving, of data to, and from, the servo will be handled on one wire. The processing unit by which the quadruped will be controlled, as discussed in Section 2.2, will only be equipped with full duplex serial ports [9] [12], which uses two separate wires. This means that a converter will have to be used the convert the control platform's full duplex UART to a half duplex UART. A simple solution is to connect the RX and TX pins of the processing unit to the servo's dataline, each through a small series resistor. This system can be used at low speeds (9600 baud), although the servo's are able to send and receive data to up to 1 Mbaud. The resistor solution will be viable at lower communication speeds, but if faster communication is used a more elegant solution will have to be designed.

The AX-12 servos communicate with a TTL level of 5V. The Raspberry Pi, which has a TTL level of 3.3V, will need a converter circuit to lower the 5V signals from the servo's to 3.3V, and vica versa.

There are a number of software libraries available for the Raspberry Pi [8] to help controlling the AX-12 servos. This will make communication with the AX-12 servos much easier, as it does not need to be written from scratch.

2.3.2 Serial Connection to Inertial Sensor

The inertial sensor which will be used to determine the quadruped's orientation is called the ADIS16405. This device will send it's information back to the processor with the use of a Serial Peripheral Interface (SPI) [7]. The inertial sensor's communication pins can handle both 3.3V and 5V logic, which means it can connect straight to the proposed processor.

Data will have to be retrieved from this device periodically. The device has the ability to let the processor know that the measured data is ready to be retrieved by the use of an interrupt pin. This can ensure that the processor only attempts to read the ADIS16405's data when the data is available.

2.3.3 Remote Control

The quadruped must be controlled wirelessly and therefore some form of wireless communication must be implemented. A method to control the quadruped wirelessly already exists for the Arduino. A WiFi shield is attached to the Arduino, and an application is written for a Android phone. This is a viable and good method to control the quadruped. However, the Raspberry Pi is chosen as an processing unit and an other method will have to be implemented.

WiFi could be implemented on the Raspberry Pi as well. Since the Pi is equipped with USB ports, a simple WiFi dongle can be aquired, and a remote connection will be available. A example of this is the Edimax Wireless Adapter [13]. Phones and computers also has WiFi capabilities, which means the quadruped will be easily controllable with either. The protocol used with WiFi will be a Transmission Control Protocol (TCP), which enables streams of data between two devices. After the processor and the remote (phone or computer) are connected over WiFi, one will start a server and the other would connect as a client. This will then ensure that data can be sent between the two devices.

Bluetooth is also a option to control the quadruped remotely. There are a number of Bluetooth modules available which has serial interfaces, or dongles, which can be easily used by the Raspberry Pi. There are less help available on the internet applicable to Bluetooth communication, relative to the Raspberry Pi. Thus WiFi is chosen as the communication protocol to communicate to a remote control. The device used will be and android based phone, a Samsung Galaxy A3. This is chosen over a computer for mobility.

2.4 Controlling Orientation

The quadruped should be able to determine it's orientation to keep it's body upright and keep it's balance. The ADIS16405 is available which contains a triaxial inertial sensor and magnetometer [7]. There are a number of ways to determine the body's orientation using this information, all varying in accuracy and complexity.

2.4.1 Determine orientation using acceleromter

The most basic way would be to determine the orientation of the quadruped using only the accelerometer. There will always be a constant acceleration of roughly 9.81m/s, or 1g, pointing downwards caused by gravity. The quadruped will not experience other great accelerations, thus it can be idealized that gravity will be the only constant acceleration that the quadruped will experience. This fact can be used to determine the angle by which the accelerometer is titled, by using the following equations:

$$pitch_{angle} = \arctan(\frac{acceleration_x}{\sqrt{acceleration_y^2 + acceleration_z^2}})$$

$$roll_{angle} = \arctan(\frac{acceleration_y}{acceleration_z})$$

The problem with this method that is not very accurate. It's very sensitive to small disturbances of the accelerometer, as well to high frequency movements.

2.4.2 Determine orientation using gyroscope

The current orientation of the quadruped can be determine using the gyroscope, which reads the angular speed. This quadruped's current orientation can be determined by integrating angular speed. This method is very accurate at high frequency movement as a gyroscope is very accurate. The problem with this method it that the gyroscope is not ideal, and will return noise. This will cause the angle to start to drift as time goes on and the noise accumulates, and will result in a inaccurate reading.

2.4.3 Determine orientation using Complimentary Filter

Both methods described in Sections 2.4.1 and 2.4.2 are inaccurate on their own, but when it is combined in the correct way it can result in a accurate measurement. One way of achieving this is by using a Complimentary Filter. It is known that the accelerometer is accurate at low frequencies and the gyroscope at higher frequencies. Therefore a combination of the two will be used.

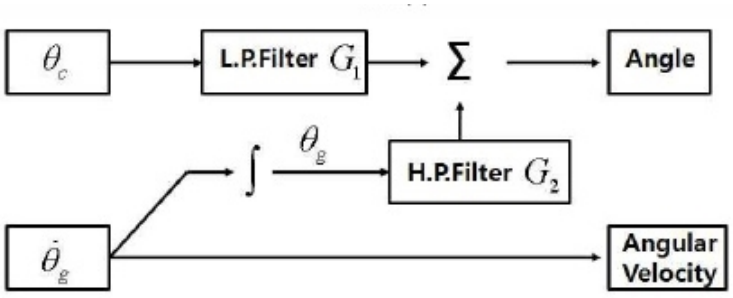


Figure 2: Complimentary Filter

The system shown in the diagram in Figure 2, as found in [14], can be used to estimate the current pitch and roll of the quadruped with good accuracy without the use of long processing time. The accelerometer's data will be put through a low pass filter, and the gyroscope's data - after being integrated - will be put through a high pass filter. This means unwanted noise will be filtered out. The system will also be designed that the two filters have the same cutoff frequency such that $G_1(s) + G_2(2) = 1$. This ensures the filter will always produce a accurate output, regardless of the frequency.

The cutoff frequencies of these filters should be chosen relative to the gyroscope. The gyroscope is more accurate than the accelerometer in design, but it drifts due to a constant small error. To remove this error a high pass filter is used. Thus the cutoff frequency should be

chosen low enough to use the gyroscope as much as possible, without causing to much drift. The amount of acceptable drift will have to be tested to find a suitable value.

The output of this system might still contain high frequency noise from the gyroscope. To filter out the noise, a low pass filter can be implemented. Another, less processing heavy, will be to average the last few angles as determined by the filter.

2.4.4 Determine orientation using Kalman Filter

An more accurate, but processing intensive, method would be to use a Kalman Filter to determine an estimate of the body's orientation by combining data from the accelerometer and gyroscope. This is a widely used method to determine the orientation of objects. An example of this is to measure the movement of human limbs with the use of accelerometers and gyrcopes [15].

A Kalman filter uses an iterative estimation process to process noisy readings into a more accurate reading. It takes into account the probability of an accurate value, which is measured statistically [16].An initial quess is made about the mean and co-variance of the accuracy of the angle. The system will then predict the next angle of the system with the following equations:

$$\hat{\mathbf{x}}_k^- = \mathbf{A}\hat{\mathbf{x}}_{k-1} + \mathbf{B}\mathbf{u}_{k-1}$$

$$\mathbf{P}_k^- = \mathbf{A}\mathbf{P}_{k-1}\mathbf{A}^{\mathrm{T}} + \mathbf{Q}$$

Then the system will then adjust this prediction by using an actual measurement at that time. This will be done with the following equations:

$$\mathbf{K}_k = \mathbf{P}_k^-\mathbf{H}^{\mathrm{T}}(\mathbf{H}\mathbf{P}_k^-\mathbf{H}^{\mathrm{T}} + \mathbf{R})^{-1}$$

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k^- + \mathbf{K}_k(\mathbf{z}_k - \mathbf{H}\hat{\mathbf{x}}_k^-)$$

$$\mathbf{P}_k = (\mathrm{I} - \mathbf{K}_k\mathbf{H})\mathbf{P}_k^-$$

These two steps will be executed sequentially in a loop, and determine an accurate estimate of the required angle.

In these equations x_{k-1} represents the state of the system at the interval $k-1$. $\mathbf{A}$ is an transition matrix that transforms x_{k-1} to x_k in the ideal case. $\mathbf{B}$ is the state control matrix and z_k is the measurement at the interval k. $\mathbf{Q}$ and $\mathbf{R}$ represents the co-variances of the system noise, and $\mathbf{H}$ is the measurement matrix relating z_k to x_k.

2.4.5 Keeping The Body Level

After the current angle of the body is determined the information must be used to keep the body of the quadruped parallel to the horizon. This is done in an attempt to keep the quadruped from toppling over when walking on tilted surfaces. This will be done by using a control system to control the orientation of the body. The system should not have a large overshoot, and should have a short settling time. To ensure the control system can handle a constant changing of the orientation an integrator will be added to the control system. This will also minimize the steady state error, as mentioned in [17]. The system should also not be processing intensive.

The two most basic options would be a proportional integrator (PI) or proportional integrator derivative (PID) controller. Both these controllers contains a integrator, and will thus be able to follow a ramp input. These systems can also be tuned to have a specific transient response. The tuning of this system can be tuned by trail and error, which means a mathematical model of the quadruped is not needed. These controllers are also fairly simple to implement, which means it should not take to much of the processor's time. A diagram of a typical PID controller can be seen in Figure 3.

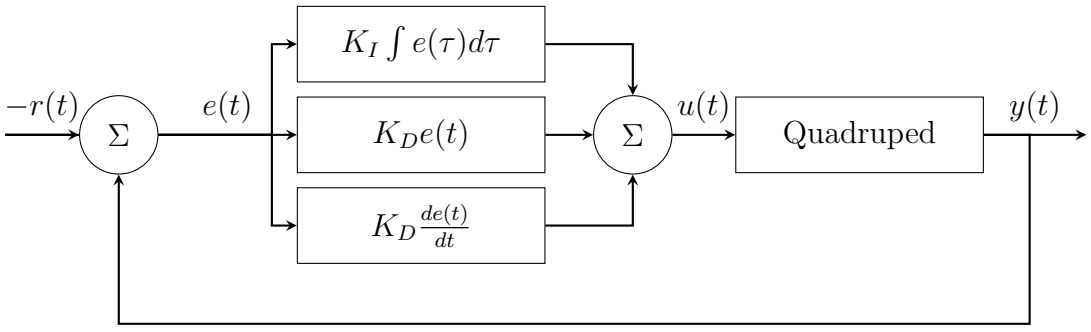


Figure 3: Basic PID contoller layout

More complex controllers can be designed and implemented, which could compensate for non-linear components of the quadruped. By using observers the current state of the quadruped could be estimated to a high accuracy, which means the quadruped will be much more stable. This method, however, would be very processing intensive and would require a large amount of designing beforehand. It would also require a mathematical model of the quadruped, which would be very hard to determine. Since the quadruped's legs are not very accurate - with the servo's mechanical limitations - the accuracy of a complex system is not needed. Thus a simpler solution, such as the PID controller, would be much more effective for this specific task.

3 Detailed Design

3.1 Overview

3.1.1 Hardware

This project has a limited hardware component to it. A variety of circuits will be used to reach the objectives set in Section 1.1. The main things that will need a hardware components can be seen in Figure 4.

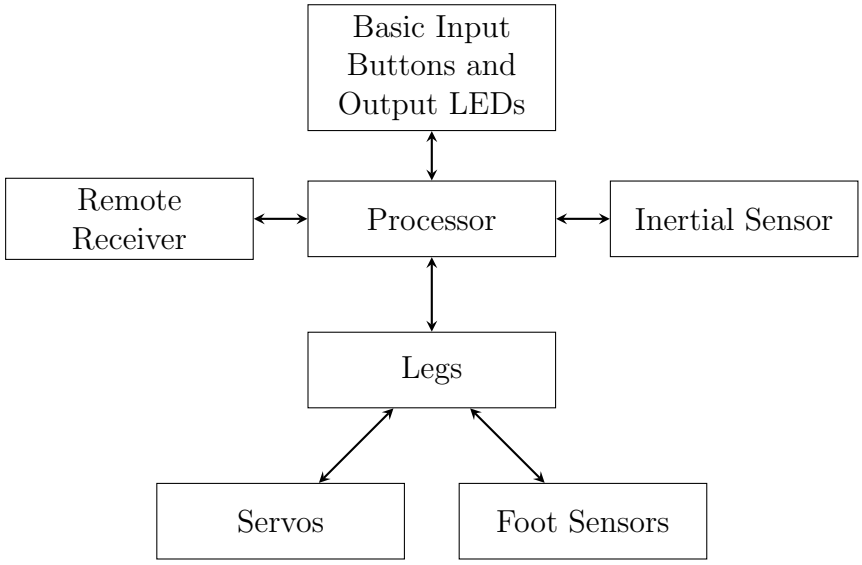


Figure 4: Basic hardware to be used this project

Throughout this section the different hardware components will be designed. All the designed circuits will be then mounted onto a printed circuit board (PCB) and attached to the Raspberry Pi as a shield. This will decrease the number of wires used, as well as putting everything into one compact design.

The final schematic of the shield can be found in Appendix C, and the designed circuit board in Appendix D.

3.1.2 Software

This project will rely heavily on software to control the movement of the quadruped. The Raspberry Pi has a built in Python environment which will be used. To keep the software side modular it will be split into different parts, which will in each have a specific purpose. The basic layout of the software can be seen in Figure 5.

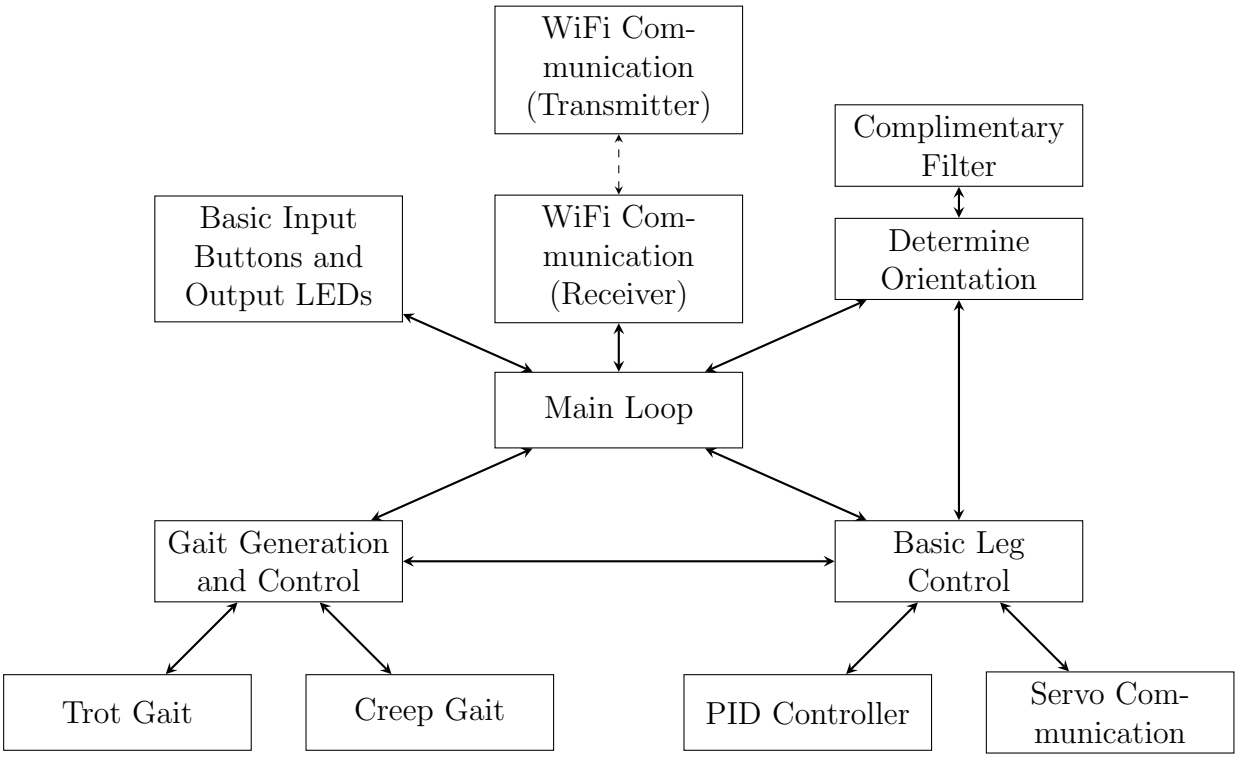


Figure 5: Basic Software Layout

The WiFi communication transmitter won't be programmed on the Raspberry Pi, as it needs to function as a wireless remote control for the Raspberry Pi. Therefore it will be programmed on a Android based cellphone, as discussed in Section 3.4.3.

All the written and used code is available in the supplied CD.

3.2 Controlling Leg Movement

The quadruped has four legs, each with three degrees of freedom. To set the feet of the quadruped at a specific location in a three dimensional space a method called inverse kinematics will be used. To use the inverse kinematics the position of the foot has to be known relative to the body of the quadruped. Since foot positions will be described initially relative to a flat world the co-ordinates will have to be converted and shifted between different axis-systems. This is discussed in the following sub-section.

3.2.1 Describing the 3-Dimensional space

The feet positions will initially be described relative to a flat world. This means the center of quadruped's body will be chosen as the center of the axis, position $[0, 0, 0]$. The x-axis and y-axis will then be parallel to the ground, and the z-axis perpendicular to the ground. This will always be seen as true, even if the ground is angled. This co-ordinate system must then

be converted to a new co-ordinate system This system will still assume that position $[0, 0, 0]$ is at the center of the quadruped's body. The x-axis, however, points to the front of the quadruped, the y-axis to the quadruped's right, and the z-axis perpendicular to the body, irregardless of the quadrupeds current rotation. This is done to ease the calculation for the inverse kinematics. This can be seen in Figure 6. This figure will also show the numbering system chosen for the different legs of the quadruped, as well as the rotation direction chosen as positive.

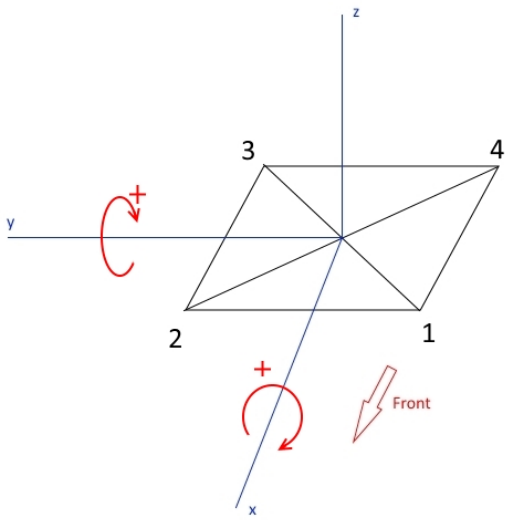


Figure 6: The Desired Co-ordinate System

To determine the co-ordinates of the feet in the desired axis, as seen in Figure 6, the co-ordinates of the feet has to be rotated around the origin. The rotation will be described as:

- ϕ is rotation around the x-axis (roll).
- θ is rotation around the y-axis (pitch).
- α is rotation around the z-axis (yaw).

From this a rotation matrix can be designed. This rotation matrix was derived from [18], and first rotates co-ordinates in the x-axis, then y-axis, and then lastly in the z-axis.

$$R_{xyz}(\phi, \theta, \alpha) =$$
$$\begin{bmatrix} \cos(\theta)\cos(\alpha) & -\cos(\theta)\sin(\alpha) & \sin(\theta) \\ \cos(\phi)\sin(\alpha) + \sin(\phi)\sin(\theta)\cos(\alpha) & \cos(\phi)\cos(\alpha) - \sin(\phi)\sin(\theta)\sin(\alpha) & -\sin(\phi)\cos(\theta) \\ \sin(\phi)\sin(\alpha) - \cos(\phi)\sin(\theta)\cos(\alpha) & \sin(\phi)\cos(\alpha) + \cos(\phi)\sin(\theta)\sin(\alpha) & \cos(\phi)\cos(\theta) \end{bmatrix}$$

These rotated co-ordinates can then be used to determine the angles the servos should turn to with the help of inverse kinematics, as seen in Section 3.2.2.

3.2.2 Inverse Kinematics

Inverse kinematics calculates the angle each joint of each leg should rotate to put the quadruped's foot at a specific location. These algorithms are application specific, so this has to be designed specifically for a quadruped with this project's quadruped characteristics. Although this method may return more than one solution other in applications, this will not happen with this quadruped, as discussed in Section 2.1.

The first step is to determine where the foot has to be placed. After converting the foot's locations to be relative to the body, as described in Section 3.2.1, each foot should be described with the appropriate shoulder as the center of its co-ordinate system. This can be done by adding an offset to the current co-ordinates of the foot. For example, the new co-ordinates of the feet will be, starting from the front left leg (with $bl2 = (bodylength)/2$):

$$newfoot(oldfoot) = \begin{bmatrix} oldfoot.x - bl2 & -oldfoot.y - bl2 & oldfoot.z \\ oldfoot.x - bl2 & oldfoot.y - bl2 & oldfoot.z \\ -oldfoot.x - bl2 & oldfoot.y - bl2 & oldfoot.z \\ -oldfoot.x - bl2 & -oldfoot.y - bl2 & oldfoot.z \end{bmatrix}$$

This will convert each position to a co-ordinate system as in Figure 7. From this the algorithms can be applied to determine the angle each joint should turn. The algorith,s will be as follows.

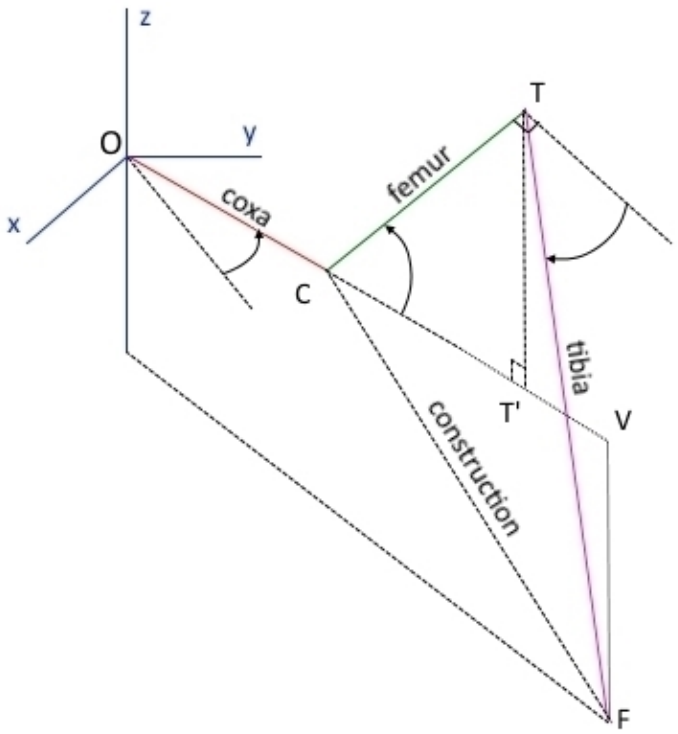


Figure 7: 3D Representation of Leg

The angle the coxa should turn to ($coxa_{angle}$) can be calculated by using the following formula:

$$coxa_{angle} = \arctan(F.y/F.x) - \pi/4$$

The angle the femur should turn to ($femur_{angle}$) can be calculated by first determining position C's co-ordinates:

$$C = [coxa_{lenght} \cos(coxa_{angle} + \pi.4) \qquad coxa_{length} \sin(coxa_{angle} + \pi.4) \qquad 0]$$

Then determining the length of the construction line:

$$construction = \sqrt{(C.x - F.x)^2 + (C.y - F.y)^2 + (C.z - F.z)^2}$$

Next the angles $F\hat{C}T$ and $C\hat{F}V$ should be determined, using the cosines-rule and general trigonometry. The value of $femur_{angle}$ can then be determined, in the most general case, with:

$$femur_{angle} = F\hat{C}T - C\hat{F}V$$

However, when the foot (F) is below or behind point C, the formula for $femur_{angle}$ would

change to:

$$femur_{angle} = F\hat{C}T - C\hat{F}V + \pi/2$$

And when the foot (F) is above point C, or with $F.z > 0$, $femur_{angle}$ should be:

$$femur_{angle} = F\hat{C}T + C\hat{F}V$$

Determining $tibia_{angle}$ can be done by first determining $C\hat{T}F$ by using the cosines-rule, and using the equation:

$$tibia_{angle} = C\hat{T}F - \pi/2$$

vis
$coxa_{angle} = \arctan(F.y/F.x) - \pi/4$
$femur_{angle} = F\hat{C}T - C\hat{F}V$
$tibia_{angle} = C\hat{T}F - \pi/2$

3.2.3 Software Implementation

The software that controls the movement of the legs will be the main part of the software, and should work on a polling system, as discussed in Section 2.1.4. To implement this a system using waypoints will be used. Each waypoint will store the posture (foot locations, translation, and rotation) for the body of an instant - or a duration of time. These waypoints will be stored in a first-in-first-out (FIFO) queue, and each posture will be executed sequentially.

The process will start with main waypoints, which will tell the quadruped what it's posture should be after a certain time duration. The quadruped will then do the necessary moves to reach this posture. To ensure smooth and controlled movement to this posture more waypoints will be needed between the starting waypoint and the main waypoint.

These waypoints will be called straight fractions. They will be created by interpolating the straight path between the starting waypoint and the main waypoint - or destination. This will ensure that feet follow a straight path to their destination in a controlled fashion. The number of waypoints it will create will be decided by taking the duration the feet have to move the destination, and dividing it by a specific period. This period will be tuned to ensure smooth movement, but will be between a few hundred milliseconds and one second, because it will requires a lot of processing time.

To ensure even smoother movement between these straight fractions, each straight frac-

tion will be broken down to more waypoints called angle fractions. With the use of inverse kinematics the required servo angles will be determined for the next straight fraction. Each servo's starting angle and required angle will then be interpolated into a specific number of angle fractions. The angle fractions are the waypoints whose servo angles will be written to the actual servos.

A flow diagram showing the breaking down of waypoints can be seen in Appendix E.2. The polling routine of this system can be seen in Appendix E.1. The polling system will only handle angle fraction waypoints. If the next waypoint retrieved from the queue is not an angle fraction it will break it down, as discussed above.

3.3 Locomotion

3.3.1 Creep Gait

A creep gait is a stable gait for quadruped locomotion where only one leg is lifted at any instance, as discussed in Section 2.1. The pattern decided to be used is derived from the pattern used in [11], and is displayed in Figure 8. This pattern has 16 sections for a entire beat, which is divided into for parts where every leg is lifted once. The total height of the forward offset will be the step size, which will be relative to the foot's default position. The height of the height offset will be how high the legs will be lifted from the default position.

In the case where the quadruped is traveling in a straight line the y co-ordinates of the quadruped will stay constant, and the x co-ordinates will change. Please note co-ordinate system in Figure 6, Section 3.2.1, is used. The x value of each foot will be offset with a value between -(Step Size)/2 and (Step Size)/2, according to the offset given in Figure 8.

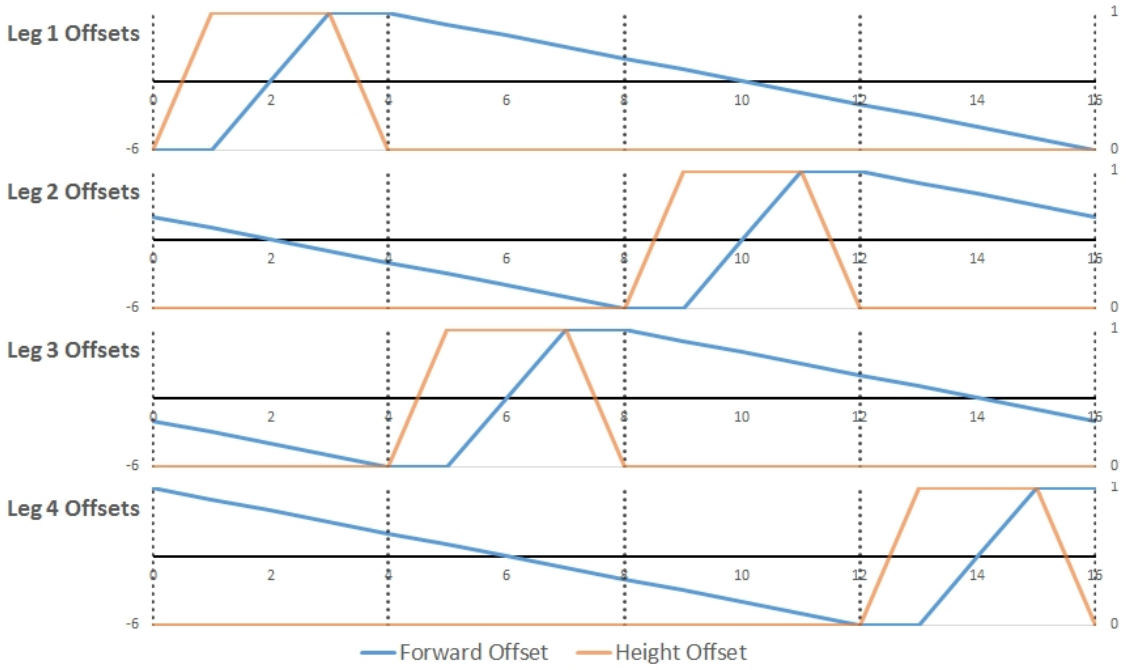


Figure 8: Creep Gait Pattern

The order of the legs lifted up improves the stability of the quadruped. It ensures that when a front leg is lifted, the leg behind it will be close to the leg being lifted - as it was the previous leg lifted. This can be seen in Figure 9 on the left hand side. This creates a bigger stability triangle to keep the center of gravity in. The alternative, on the right of the figure, creates a much smaller stability triangle. When a back leg is shifted it is done diagonally from the previously lifted leg, which means if the previous leg is far from the body, and it will only increase the stability triangle's size.

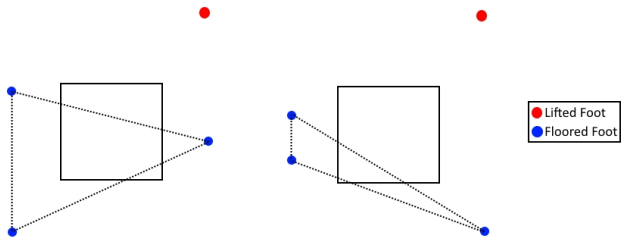


Figure 9: Foot positions to maximize stability

In addition to this the body will be moved side to side to keep the center of gravity inside the stability triangle. The body will be moved to the diagonally opposite side of the leg being lifted. The movement will be relative to the feet. This means the the path the foot will have will stay the same, only the body will change location. The amount of side-to-side movement will be determined by testing. The parameters used for the creep gait can be seen in Appendix F.1.

3.3.2 Stepping Over Obstacles

The quadruped should be able to step on - and over - small obstacles. To achieve this the foot's movement should adapt to the environment. The foot should lower until it reaches the ground, and then push backwards from the position.

Sensors is placed on each foot determine to if it it touching the ground. Each sensor will be a simple micro switch placed on the bottom of the foot. The switch will turn on when the foot pushes on the ground. The circuit of the switch will be exactly as the input switches discussed in Section 3.6.2. The basic configuration of the sensor is displayed in Figure 10. It shows the bottom of the tibia to which a plate is fastened to create a foot. A microswitch is added to the foot with a half sphere as the head of the button. The button head will be constructed out of rubber to ensure traction. The half sphere shape is chosen to direct most of the movement, when pressing against the ground, to the base of the switch.

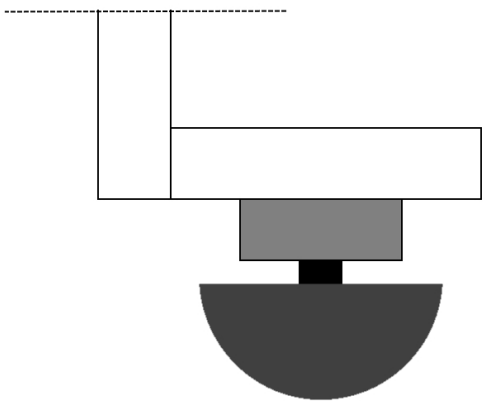


Figure 10: Foot Sensor Configuration

The step pattern, as displayed in Figure 8, can be used, but with a different height offset. The new pattern for the height offset can be seen in Figure 11. First the robot will lift it's leg before forward movement starts. This is done to decrease the chance of the foot hitting any obstacles. The foot is then extended forward, and only after that it's lowered. This will increase the foot's stability, as well as making it easier for the sensors to sense the ground. The foot will still have forward movement while lifting up and lowering, because of the constant forward body movement, but this will be minimal and can be ignored.

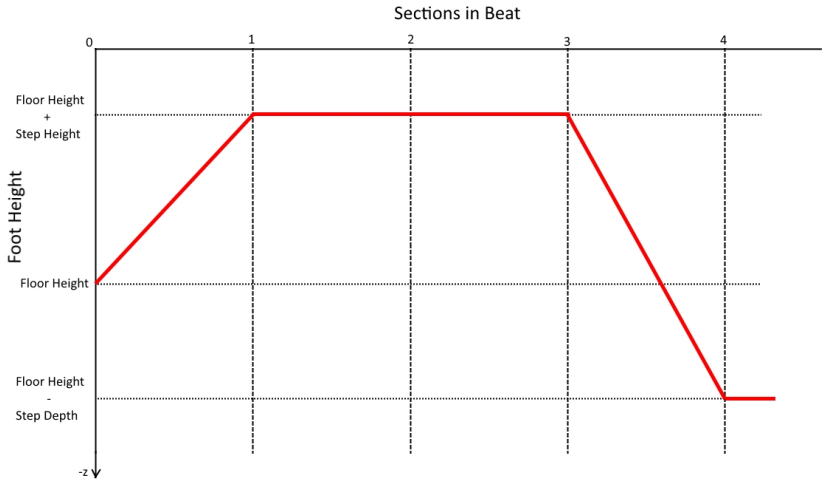


Figure 11: Step Height Pattern

While the foot is lowered it will constantly check if it is touching the ground, and when so it will stop moving. To give the quadruped the ability walk of small steps the quadruped will attempt to lower it's foot lower than it expects the floor to be. When the foot sensor senses that the foot has touched the ground, that height will be stored as the floor height, and the quadruped will push the foot pack for the rest of the beat on that height. The floor height will be reset with step, and will be set separately for each foot.

To determine the height of the floor the servos can be requested to read and return the current angle at which they are rotated. These angles can then be used to determine the co-ordinates where the foot currently resides at, with the specified foot's shoulder as center of the axis. This is done by determining the length of OV in Figure 7, in Section 3.2.2.

First point T is projected onto the line OV, which gives T'. The length from O to T' is then determined by:

$$OT' = coxa_{length} + femur_{length} \cos(|femur_{angle}|)$$

Which means the length OV can be written as:

$$OV = OT' + tibia_{length} \sin(femur_{angle} + tibia_{angle})$$

The height of point T can be determined by:

$$T.z = femur_{length} \sin(femur_{angle})$$

Therefore this results that the foot position, determined by the angles at which the servos are rotated, is:

$$\begin{bmatrix} foot.x \\ foot.y \\ foot.z \end{bmatrix} = \begin{bmatrix} (OV)\cos(coxa_{angle}) \\ (OV)\sin(coxa_{angle}) \\ T.z - tibia_{length}cos(femur_{angle} + tibia_{angle}) \end{bmatrix}$$

If the quadruped's body is rotated while this measurement is taken it will result that the foot co-ordinates will also be rotated. Therefore the foot's position has be rotated in the opposite direction. Since the original rotation was done first in the x-axis, then y and afterwards the z-axis, the opposite rotation has to start with the z-axis and end with the x-axis. It can be assumed that the rotation in the z-axis is zero, as that rotation is never changed with a creep gait. Also, only the z value is needed to determine the height of the foot. Therefore the height of the foot can be written as (with the current rotation of the quadruped as $pitch = \theta$ and $roll = \phi$):

$$unrotated\_foot.z = foot.x(\cos\phi\sin\theta) - foot.y(\sin\phi) + foot.z(\cos\phi\sin\theta)$$

This height will then be used to determine the height at which the foot will push backwards to propel the quadruped forwards.

When the foot touches the ground it will result that the 16 steps of the beat will not always be completed, and the quadruped will not be in the correct position to start with the next beat. Therefore the distance between the foot's current position and the potential position at the end of the beat will be stored, and used as a constant offset throughout the next beat. This offset will be cleared, and set again the next time the leg is lifted.

3.3.3 Curved Creep Gait

The quadruped must be able to turn while walking as well. This means the feet should push back in arcs, instead of straight lines. The same step pattern and offsets will be used as with a straight creep gait, as seen in Section 3.3.1. The difference is that the forward offset will be used to calculate the angle offset at which the foot should be placed down. This is shown in Figure 12.

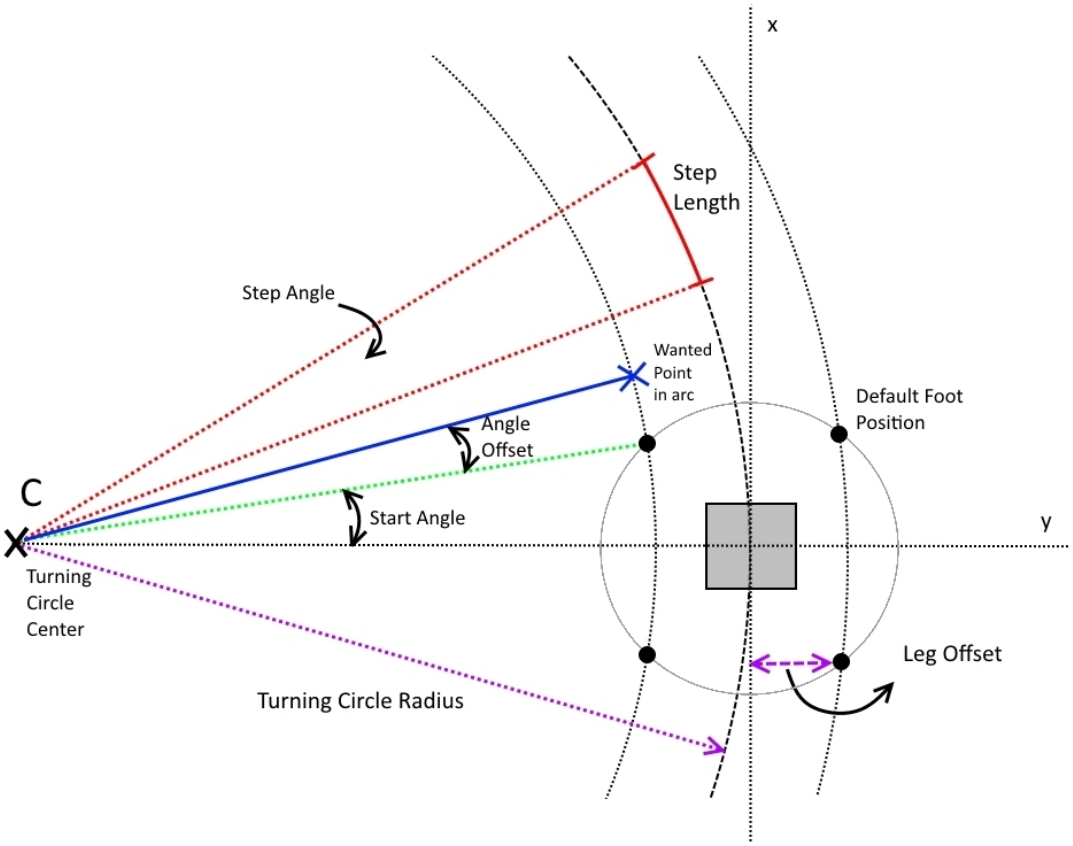


Figure 12: Determining Leg Movement Arcs

The total step angle is therefore calculated by (in radians):

$$\theta_{STEP\_ANGLE} = \frac{length_{STEP}}{radius_{TURNING\_CIRCLE}}$$

This means that each section in the beat, as seen in Figure 8, will move the legs with a angle of:

$$\theta_{STEP\_ANGLE\_SECTION} = \frac{\theta_{STEP\_ANGLE\_SECTION}}{16}$$

The leg offset will be the distance in the y direction from the quadruped's center point and the default position of the feet. These positions of the feet will also act as the center point of the arc. The starting angle can then be calculated by (if calculated for leg 1):

$$\phi_{START\_ANGLE} = \arctan(\frac{LEG\_OFFSET}{radius_{TURNING\_CIRCLE} - LEG\_OFFSET})$$

And the offset angle according to the walking pattern:

$$\phi_{PATTERN\_ANGLE} = \arctan(\frac{PATTERN\_OFFSET}{radius_{TURNING\_CIRCLE} - LEG\_OFFSET})$$

An inner turning circle us defined as a circle with center point C and radius of the turning circle's radius minus the leg offset. Each point on the arc can then be calculated by the formula:

$$x = C.x + radius_{INNER\_TURNING\_CIRCLE} \sin(\phi_{START\_ANGLE} + \phi_{PATTERN\_ANGLE})$$
$$y = C.y + radius_{INNER\_TURNING\_CIRCLE} \cos(\phi_{START\_ANGLE} + \phi_{PATTERN\_ANGLE})$$

The same step height pattern will be ased as in Section 3.3.2. Instead of giving the foot position a constant offset, as with the case of the straight creep, the foot will be given a constant angle offset. The start angle will be offset by angle difference between where the foot stopped, and the full step angle.

3.3.4 Trot Gait

The basic movement of the trot gait is fairly simple. The feet should always move in a straight line. To add extra functionality to the gait it was decided to give the gait the ability to move in any direction. The way the straight line is calculated on which the feet will move can be seen in Figure 13.

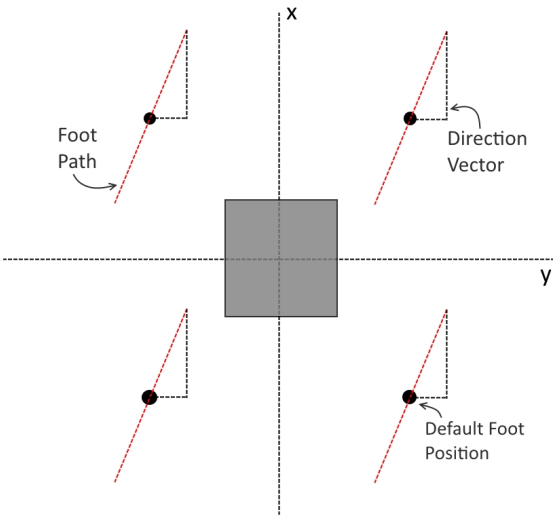


Figure 13: Trot Gait Foot Path Generation

A line is put through the default positions of the feet. This line's length will be the distance each step should reach. The line will point in the direction of which the quadruped will move in, as supplied through a direction vector. This line represents the path the foot will follow on each step. Two of the diagonally opposite feet will push back on that line, while the other two will lift up and move forward. The pattern of how this will happen can be seen in Figure 14.

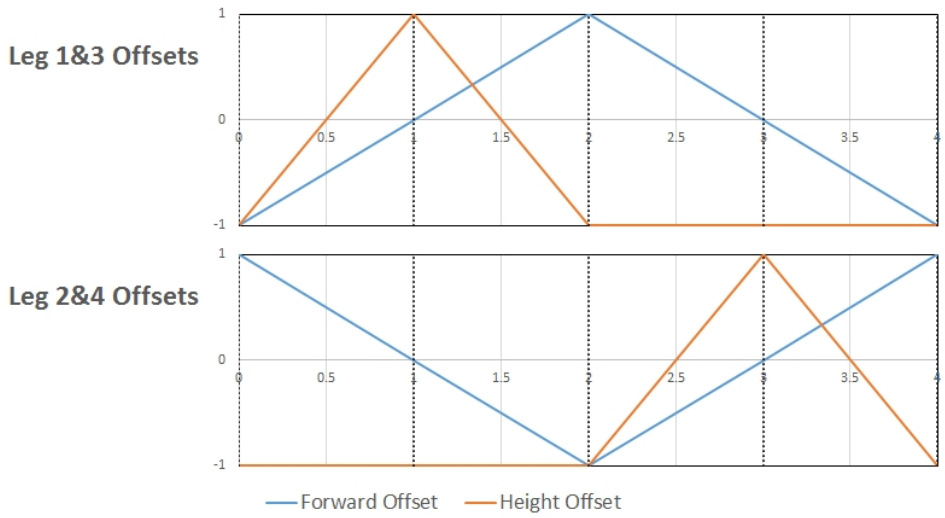


Figure 14: Trot Gait Pattern

The total height of the forward offset will be the distance each step will reach. This offset will be used to determine the point on the red line - in Figure 13 - where given foot should be on any given instant. The total height of the height offset in Figure 14 will be the total height the foot should lift from the floor height.

In this gait there is no stability triangle in which to keep the center of mass, seeing as only two feet touches the ground at any given instant. To keep the body stable the center of mass has to be on the line between the two grounded feet. To keep the body on this line, and keep it from falling to the side, the body is translated to change the location of the center of mass. This is similar to how a human shifts his/her weight to the side of current foot on the ground.

After testing it was found that the following method delivers promising results. The body is shifted forward while the leg is being lifted (sections 0-1 and 2-3 in Figure 14), and then shifts to the side of the grounded foot as the other foot lowers to the ground(sections 1-2 and 3-4 in Figure 14).

3.3.5 Software Implementation

The code that generates the gait patterns should not interfere with the timing of the actual movement of the legs, which means that processing time should be kept to a minimum. Therefore as much as possible calculations are done before the quadruped starts the gait.

The creep gait will need to do most of it's calculation in real time, since it needs to adapt to the environment. The angle of the ground might change, which means the quadruped will have to rotate it's body to compensate. The foot height will also change as the quadruped walks, as it senses when the foot touches the ground.

The creep gait will stay the same throughout the gait. Therefore the entire gait will be generated and saved beforehand, and then just sent to the waypoint queue when needed.

3.4 Communication

3.4.1 Serial Connection To Leg Servos

The AX-12 servos used in this quadruped has an half duplex serial connection, as discussed in Section 2.3.1. The Raspberry Pi only has a full duplex serial port and thus a converter will have to be built to convert the half duplex to full duplex, and vice versa. The Raspberry Pi's pins are not 5V tolerable, thus the servo's 5V data signals will have to be brought down to 3.3V as well.

The recommended circuit to use, according to the datasheet [2], is shown in Figure 15. This uses two buffers which are enabled and disabled separately to control the data flow. The 74HC126 isn't easily available, thus its complimentary package will be used, the 74HC125 [19]. The 74HC125 package has an inverter in front of the enable lines, but this will not be a problem. The Raspberry Pi can easily invert the direction signal internally beforehand, which will effectively turn the 74HC125 into a 74HC126.

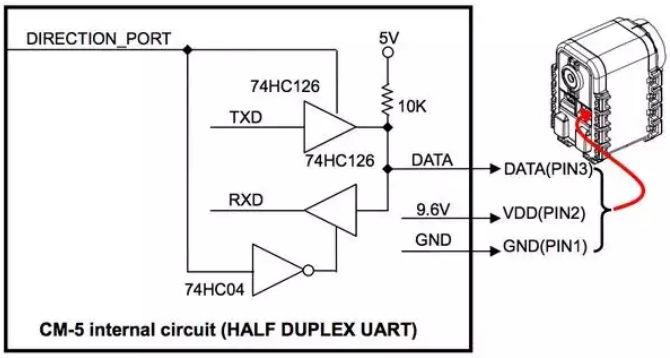


Figure 15: Dynamixel's recommended converter circuitdatasheet [2]

To minimize the amount of packages to be placed on the PCB, the 74HC04 package will not be used. A simple inverter circuit can be built using a NPN transistor and a few resistors, as can be seen in Figure 16. This will also be cheaper than using a 74HC04. The chosen transistor to be used will be the 2N3904 [20].

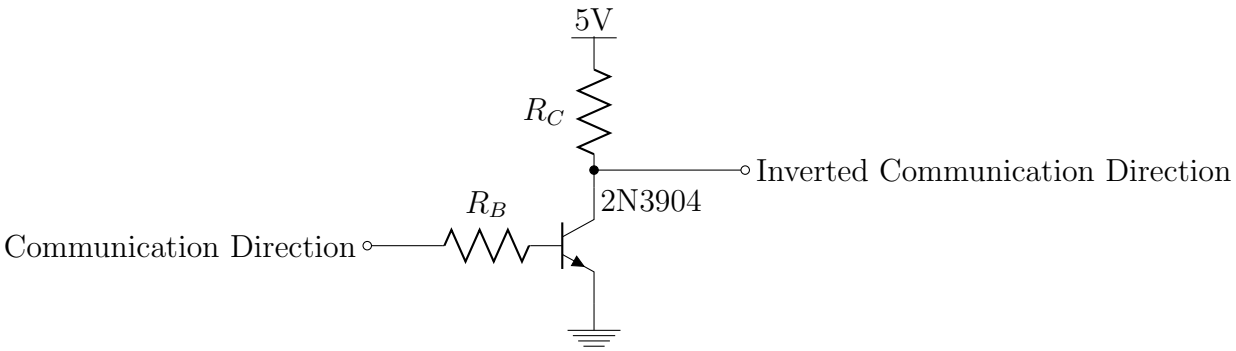


Figure 16: Direction Inverter Circuit

According to the transistor's datasheet a h_{fe} value of 80 can be assumed. The current the inverter has to deliver to the pin is assumed to be 0A. The amount of current the Raspberry Pi can deliver is limited, as stated in [12], thus it's chosen to be 1mA. The current through R_C is also chosen to be 1mA to keep the transistor circuit energy efficient. Also the values $V_{BE(on)} = 0.7V$ and $V_{CE(sat)} = 0.3V$ is taken from the datasheet [20].

This means that:

$$R_B = \frac{3.3 - V_{BE(on)}}{1m} = 2.6k\Omega$$

And:

$$R_C = \frac{5 - V_{CE(sat)}}{1m} = 4.7k\Omega$$

The data sent back from the servos to the Raspberry Pi has to be converted from 5V to 3.3V, and must be able to handle data transfers speeds to up to 1 Mbaud. A simple resistor voltage divider circuit can be used, but the output voltage would change relative to the current the input pin of the Raspberry Pi sinks. The resistor in conjunction with the input capacitance of the Raspberry Pi could also lead long charge up times at high frequencies. Therefore the circuit used to create this converter is shown in Figure 17 [21]. The LED would ensure that the output voltage is 3.3V, and the output resistance of the circuit is low enough to ensure fast charge up times at high frequencies.

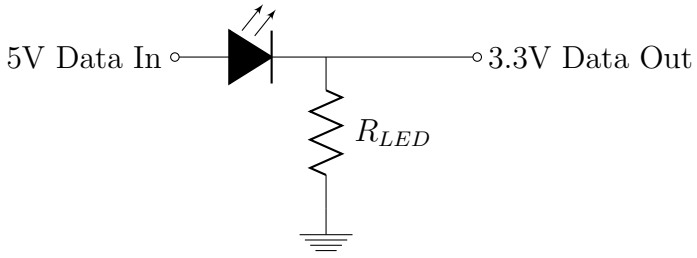


Figure 17: Logic Level Converter Circuit

The schematic created for this logic converters can be seen in Appendix C, with the PCB design in Appendix D.

3.4.2 Serial Connection to Inertial Sensor

The Raspberry Pi's connection to the Inertial Sensor, as discussed in Section 2.3.2, will be a Serial Peripheral Interface (SPI). The proposed circuit, as set in the device's data sheet [7], is as follows:

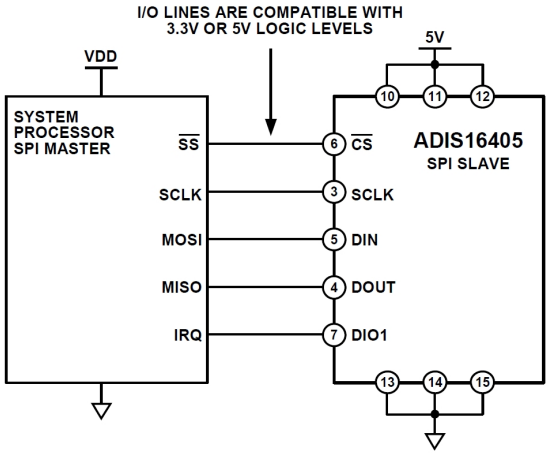


Figure 18: Specified Circuit for SPI

The duration between each of the Raspberry Pi's requests for data from the ADIS16405 will be in the order of milliseconds. The minimum duration between reads, as set by the datasheet [7] is 40 microseconds. This means that the ADIS16405 will always have enough time to generate a output before the Raspberry Pi requests for it again, and therefore the need for a interrupt request pin is not needed. It is added on the used circuit as an fail safe, but is not used. The schematic with the circuit used to communicate with the ADIS16405 can be seen in Appendix C, and the corresponding PCB layout in Appendix D.

3.4.3 Remote Control

The remote control will be used to control the robot remotely. A Wifi module will be used, called a Edimax EW-7811Un [13]. This is a WiFi dongle that can be plugged into the Raspberry Pi's USB port. The Raspberry Pi is also supplied with the dongle's drivers automatically. This was chosen because it's cheaper than Bluetooth, and easy to set up.

First an phone running an android operating system is used to create a WiFi hotspot. The Edimax dongle will then connect to the hotspot, which means the phone and Raspberry Pi will be on the same network. The Raspberry Pi will then create a server, which can communicate to - and from - a client on the android phone through a TCP protocol. This will be written in Python on both platforms, and is taken from [22].

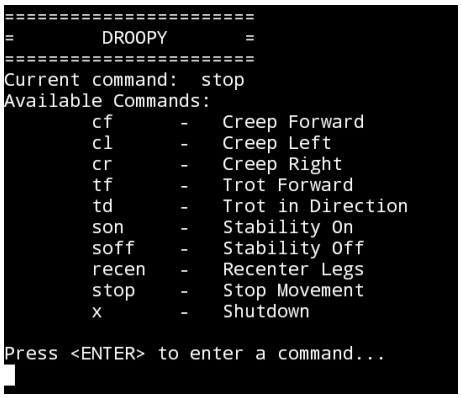


Figure 19: Screenshot of Android Phone's Interface

The remote control should constantly be sending the the current command to ensure the quadruped picks it up as fast as possible, but simultaneously allow input of a new command. The easiest solution would be a graphical user interface (GUI) with the command sending constantly in the background, and the press of a button would create a interrupt to input a new command. The version of Python on the android phone is called QPython, and is still in early stages of development, and this results that the GUI can not be built on this specific android phone. Therefore a simple command line interface is used with the structure as seen in Figure 20. A screenshot of this interface can be seen in Figure 19.

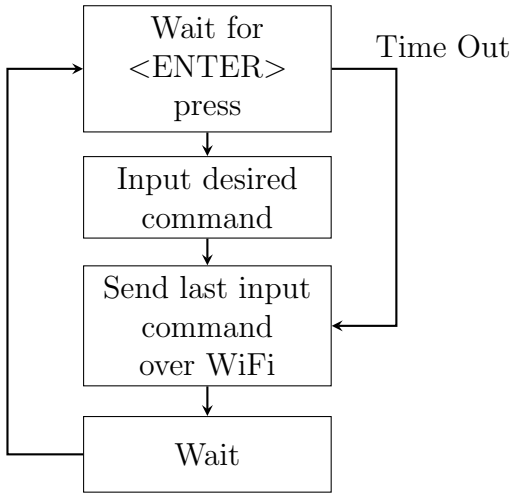


Figure 20: Flowchart showing the WiFi communication of the Android Phone

In Figure 21 the basic working of the server on the Raspberry Pi can be seen. It will run with each iteration of the program's main loop. It attempts to receive data from the android client, but if nothing is received it should not hold up the rest of the program. Therefore the receive method will time out, and then exit the function.

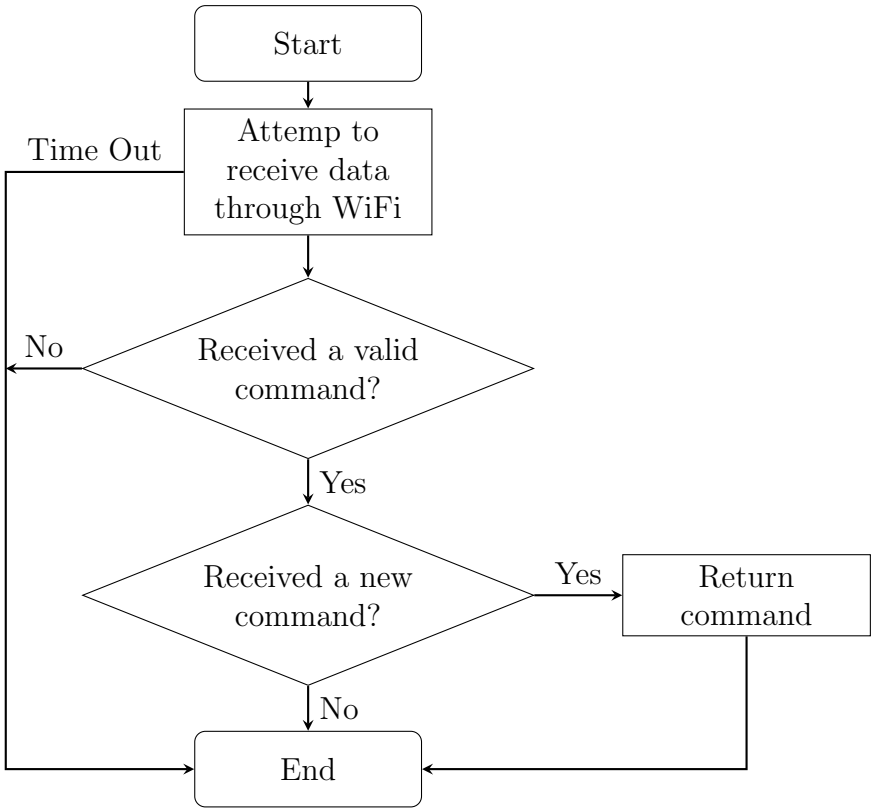


Figure 21: Flowchart showing the WiFi communication of the Raspberry Pi

To ensure that this method transmits data successfully from the android phone to the Raspberry Pi the timings has to be appropriately set. This includes the time duration before a timeout occurs with the receive of data, as well as duration before the pressing of <ENTER> should be timed out. The android should send the data frequently enough that the the Raspberry Pi would always receive data before a it's receive function times out.

3.5 Controlling Orientation

3.5.1 Complimentary Filter

A basic flow diagram of a complimentary filter can be seen in Figure 2 in Section 2.4.3. This system can be expanded to a second order system, as seen in Figure 22, which is taken from [14].

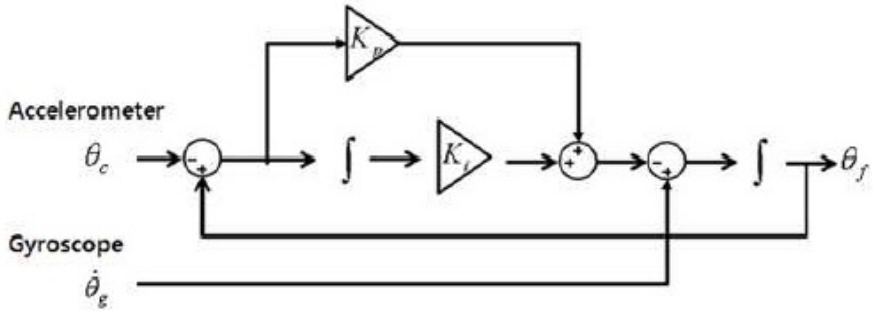


Figure 22: Second Order Complimentary Filter

This can be written with the following equation:

$$\Theta_f = \frac{s^2}{s^2 + K_p s + K_i}(\frac{1}{s}\dot{\Theta}_g) + \frac{K_p s + K_i}{s^2 + K_p s + K_i}\dot{\Theta}_c$$

This equation satisfies the formula $G_1(s) + G_2(2) = 1$. The next step will be to choose K_p and K_i so that the filter works efficiently and accurately. This equation also results that $K_i = \sqrt{\omega_n}$ and $K_p = 2\zeta\omega_n$, where ω_n is the cutoff frequency of the system and ζ the damping of the system. To minimize unwanted warping of the angle ζ is chosen as 0.707, which results in optimal damping.

To determine the cutoff frequency of the high pass filter tests is run to determine a acceptable drift error of the gyroscope. To achieve this 19 tests was run, with each test measuring the drift error after 2 seconds between each cutoff frequency between 0.1Hz and 4Hz. It's not expected for the quadruped to rotate faster than 4Hz, and below 0.1Hz only the accelerometer would suffice. With these tests the gyroscope was kept completely still. The average of the drift error on the y-axis of the gyroscope can be seen in Figure 23. This will be used to determine the pitch of the quadruped.

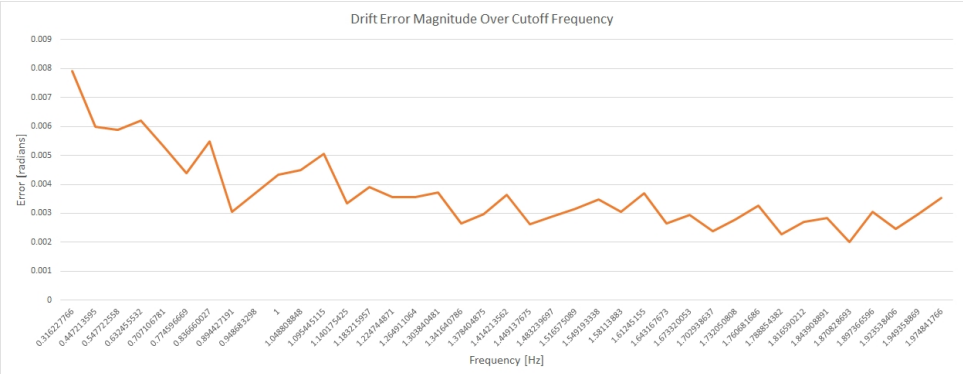


Figure 23: Drift Error vs. Filter Cutoff Frequency for y-axis

From this graph the low cutoff frequency of 1.22Hz was chosen for the y-axis's angular

speed. This will result in a average drift error of 0.2 degrees - or 0.003 radians - over 2 seconds. It is a low cutoff frequency without too much drift, as specified in Section 2.4.3. The result of this filter can be seen in Figure 24. This test is run over 20 seconds while keeping the gyroscope still. It clearly shows that high pass filter filters out the drift, while still letting higher frequencies through.

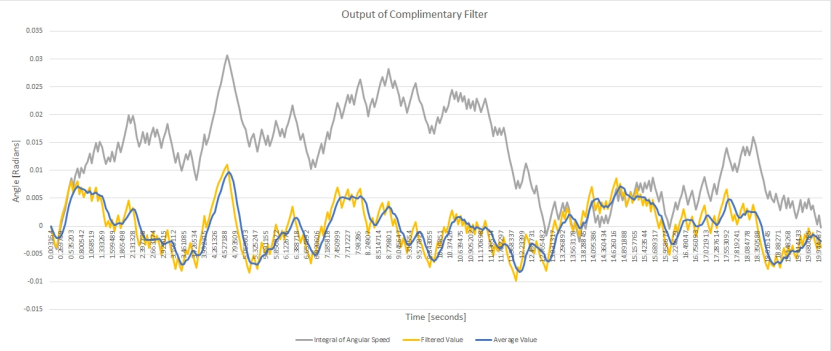


Figure 24: Filtered Integral of Angular Speed

The same cutoff frequency will be used for the low pass filter on the accelerometer, which means it should only pass frequencies through below 1.22Hz.

The gyroscope in the x-axis, which should determine the roll of the quadruped, is faulty and returns unusable data. Therefore only the accelerometer will be used to determine the roll of the quadruped, which will still be put through a low pass filter. This will result in a less accurate measurement at higher frequencies, but since the quadruped would generally move at low frequencies, it should still be sufficient. With testing it is decided to use a cutoff frequency of 1.7Hz.

The constants used for the complimentary filter is displayed in Appendix F.2.

3.5.2 PID System

Once the current angle of the quadruped's body is determined, as discussed in Section 3.5.1, the angle has to be used to rotate the body of the quadruped to be parallel to the horizon. To achieve this a simple proportional integrator derivative (PID) controller will be used. This PID controller has a integrator, to ensure a zero following error, as well as a derivative component which will damp the system and minimize overshoot and decrease settling time. This will be tuned by the use of trial and error for the roll and pitch of the quadruped separately, and by using the methods mentioned in [23].

The determined constants to use for the PID system is displayed in Appendix F.2.

3.5.3 Software Implementation

The quadruped should be able to change it's orientation regardless of where the feet currently resides. This will give the quadruped the ability to enable enable the control system with any gait or posture.

The time interval between each update of the PID system should be kept as constant as possible, to keep the PID system effective. This amount of time should time should also be kept as low possible. The code used for the PID system can be accessed on the supplied CD.

3.6 Basic Input and Output Circuitry

3.6.1 Warning Lights

Although the remote control should be able to notify the user if something goes wrong on the quadruped, as discussed in Section 2.3.3, this method will not always be sufficient. The remote's connection might be faulty, or a small error might occur on the quadruped which is not important enough to send to the remote control. Thus two light emitting diodes - or LEDs - will be placed on the quadruped as status lights. A green LED and an amber LED will be used.

The LEDs must be controlled by the Raspberry Pi's pins, which works with 3.3V [12]. The pins of the Raspberry Pi has a limit on how much current it can supply, which means it's necessary to minimize the current from the Raspberry Pi. To achieve this a transistor circuit will be used to drive the LED, with the 3.3V pin only turning on the transistor. The voltage source to power the LED will then be from it's 5V pin, which is connected directly to the power supply, which means it can handle larger currents. This also means that the LED will be able to shine brighter with more current running through it.

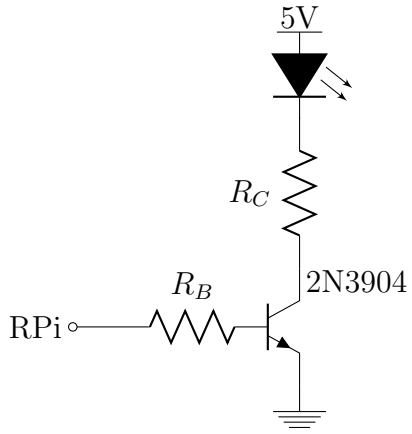


Figure 25: Light Emitting Diode Driver Circuit

The circuit to drive the each LED can be seen in Figure 25. The h_{fe} of the transistor is

estimated at about 80 [20], $V_{BE(on)} = 0.7V$, and $V_{CE(sat)} = 0.3V$. To minimize the current through from the Raspberry Pi R_B is chosen as $10k\Omega$, which means the current pulled from the Raspberry Pi would be around 260μA. The will be the same for both LED driver circuits.

The current through the LED should be at least 15mA to ensure that the LED is bright enough. The green LED has a voltage drop of 3.2V, and the amber LED a voltage drop of 2V. Both of these values was determined by testing the specific LED. Around 10mA is needed to drive each LED. Therefore using the equation $R_C = \frac{5-V_{CE(sat)}-V_{LED}}{1m}$ it's determined that R_C needs to be 100Ω for the green LED and 220Ω for the amber LED. This results in 11mA and 13mA flowing through them respectively.

The schematic created for this LEDs can be seen in Appendix C, with the PCB design in Appendix D.

3.6.2 Input Buttons

Two buttons will also be added to the quadruped in case something goes wrong with the remote control, and for a easier testing environment. These two buttons will be programmed, which is discussed in Section 3.6.3, to accept a short press as well as a long press. This is to maximize the use of the buttons.

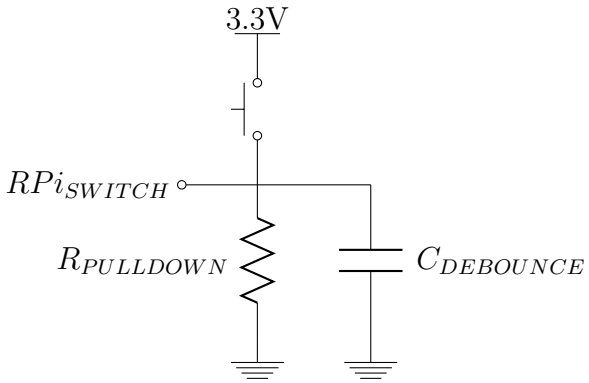


Figure 26: Button Circuit

The circuit that will be used is shown in Figure 26. It is chosen to use a circuit with a pull-down resistor, instead of a pull-up resistor, to save power. It is known that the switch will spend most of it's time unpressed. With the circuit shown in Figure 26 the current will only flow if the button is pressed.

When a button is pressed bouncing occurs on the circuit. This results that the voltage jumps between high and low for a small duration of time. This might confuse the software,

therefore a debouncing capacitor is added to the circuit. The capacitor in conjunction with the pull-down resistor creates a low-pass filter, which will filter out debouncing. To save space on the circuit board the Raspberry Pi's internal pull-down resistors will be used. The pull-down resistors of the Raspberry Pi is between $50k\Omega$ and $60k\Omega$ [12]. If this is used with a 100nF the RC time constant is around 5.5ms. This should be longer than any bouncing that could occur, and would give the Raspberry Pi a clean signal. The schematic created for this buttons can be seen in Appendix C, with the PCB design in Appendix D.

3.6.3 Software Implementation

As mentioned in Section 3.6.2 the Raspberry Pi should be able to distinguish between a long press and a short press to maximize the use of the buttons. The software will be written as seen in the diagram in Appendix E.3. This piece of code will be called in every cycle of the main progam's main loop, to check if a button has been pressed, and what kind of press. The code written for the buttons and LEDs is added on the supplied CD.

4 Results

4.1 Leg Movement

4.1.1 Inverse Kinematics Simulation

The steps shown in Section 3.2.1 and 3.2.2 can be simulated in a program such as Matlab to ensure it's correct. The foot position relative to the world is given to the simulation program, which converts it to be relative to the body. It then calculates the angles the servos should move to. It then displays the foot positions, as well as the calculated leg configuration. In Figure 27 the results are displayed for when the quadruped is at it's default standing posture. This means the body isn't rotated and the legs are in a default position. The red lines represent the legs, and the blue line is a a straight line from the shoulders to the wanted foot positions.

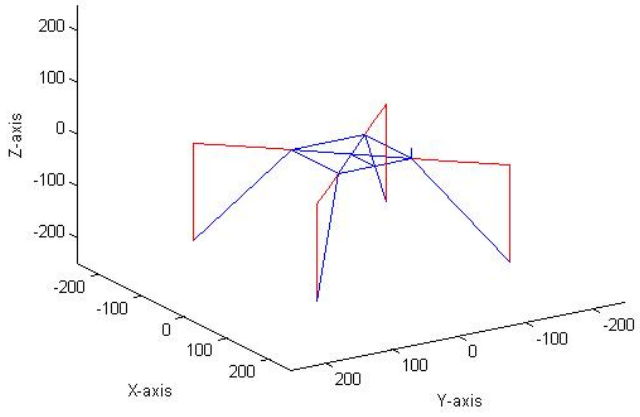


Figure 27: Simulated result for the Quadruped Default Posture

In Figure 28 the quadruped is shown when the body is rolled to $\pi/8$ radians. The four foot positions create a flat surface angled at $-\pi/8$ radians. This shows that the rotation of the quadruped works as expected. Note that the quadruped is drawn in the axis system which is relative to it's body position, and not the feet positions.

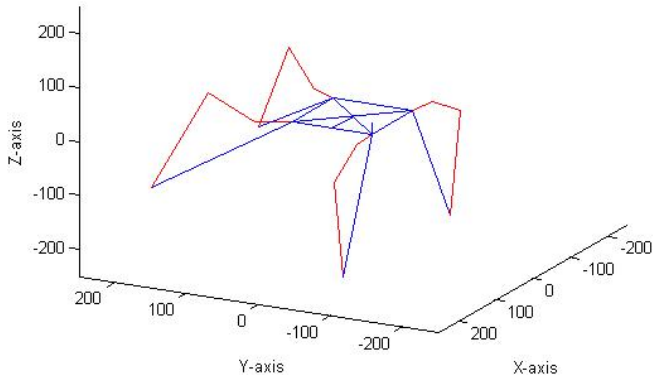


Figure 28: Simulated result for the Quadruped with the body in a roll

This proves that the inverse kinematics algorithm can determine the required angles the servos should turn to to get the quadruped's feet at the desired locations.

4.1.2 Testing Foot Movement

To test if the legs of the quadruped behave as the simulation shows in Section 4.1.1, the quadruped is put down in its default position (1). The the quadruped will then lift up its leg 5cm , and put it down 5cm forward from the original position (positive x direction) (2). After that the foot is lifted and put back to the original location(3). It will then be lifted up again and put down 5cm left the original location (negative y direction) (4). After this the foot will be brought back to it's original position once again(5).

The positions of the foot will plotted on plotted on paper, measured, and put in a graph. The center point of the system will be chosen at point 3, as this will be the first time the quadruped has to place its foot at its default position itself, and will be the closest to the actual default position. The results of this test is plotted in Figure 29.

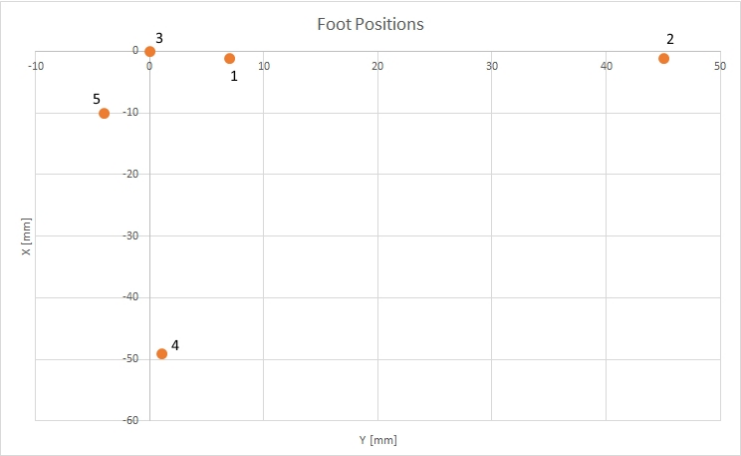


Figure 29: Actual Movement of Quadruped Feet

The foot moved as expected, although with poor accuracy. The foot was placed with only around 1cm accuracy, and the foot was only lifted to 3.5cm. This is not caused by software, but by the servos itself. The servos is not in the same condition as when it was new. If the servos moves to an angle, it does so with precision, but a small load will still move the servo head from side to side a few degrees. Therefore the weight of the legs are pulling the servos down, and the servos does not have a a feedback system to take this into account.

4.1.3 Locomotion

The gaits were tested first by ensuring each part works as expected. The gaits generated the correct locations for the feet over the correct length of time. The inverse kinematics and servo successfully placed the feet at the calculated positions, as seen in Section 4.1.2. The gait was then implemented on the main processor, and the quadruped succesfully propelled itself from one location to another.

4.1.4 Stepping Over Obstacles

The quadruped should be able to step on, and over, small obstacles, as discussed in Section 3.3.2. The quadruped was able to step on small obstacles and keep on moving without a problem. The highest obstacles successfully climbed was 1.5cm lower than the specified step height, which is set in Appendix F.1.

The reason for the 1.5cm offset is due to the old servos, as discussed in the end of Section 4.1.2. The feet does not lift as high as the software specifies.

4.2 Communication

4.2.1 Serial Connection To Leg Servos

The protocol used by the leg servos is half duplex serial communication, and the Raspberry Pi only has a full duplex serial port available, as discussed in Section 3.4.1. The Raspberry Pi also works on 3.3V while the servos communicate at a level of 5V.

In order to convert the servo's 5V data for the Raspberry Pi's 3.3V the circuit in Section 3.4.1, Figure 17, was used. This was tested by requesting data from the servos while running at 1Mbps, with the result in Figure 30. The input is at the top, and the output at the bottom, with block height 2V, and the width 10μs. It is clear that the circuit can handle the speed as the charge up time is not noticeable. The output is very close to 3.3V. The resistor R_{LED} is chosen as 1kΩ, which means the circuit will pull 3.3mA from the battery, which is acceptable.

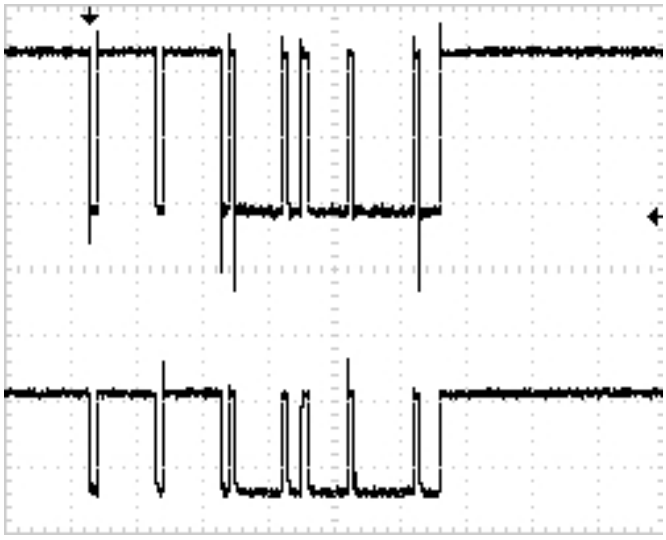


Figure 30: Conversion from 5V to 3.3V

4.3 Controlling Orientation

4.3.1 Calculating Orientation of Quadruped

To test the output of the complimentary filter, as discussed in Section 3.5.1, the inertial sensor was rotation to a known angle and kept there. The output of the complimentary filter is then calculated, and after the value settled it is compared to the known angle. This method won't test the entire complimentary filter, as the filter will work with dynamic movements and not static. To measure and compare dynamic movements, however, would require complex additional hardware, which was not available for this project. The results of this test can is displayed in Table 2.

Table 2: Measured Body Orientation vs. Known Orientation

Set Angle [degrees]	Measured Pitch [degrees]	Measured Roll [degrees]
5	4.3	4.5
10	9.5	9.6
15	14.5	14.4
20	19.8	20.9
25	24.7	24.9
30	29.9	30.2
35	35.1	35.2
40	39.8	40.3
45	45.2	44.7

It's clear from this test that the orientation of the quadruped can be measured with more than 1 degree accuracy in static situations.

4.3.2 PID System

The PID system is designed to keep the body parallel to the horizon, as discussed in Section 3.5.2. To test this system the quadruped was kept stationary while tilting the surface it's standing on. The result for the pitch of the system is displayed in Figure 31. The read angle is the read orientation of the quadruped, the output angle the angle to be sent to the inverse kinematics, and the input angle is the input to the PID system ($e(t)$ in Figure 3, Section 2.4.5). It shows a fast response time of about 1 second, and with a overshoot of about 5 degrees. The overshoot is not ideal, but is needed to insure a fast response. The transient response of the PID system is as expected.

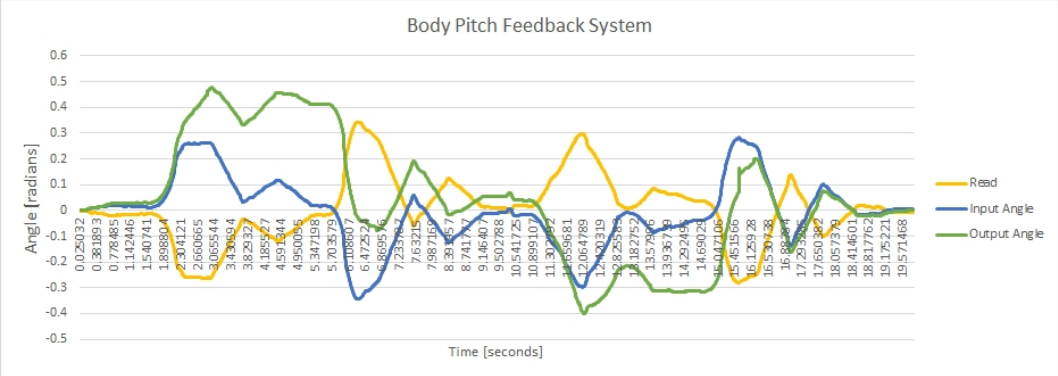


Figure 31: Transient Response of Pitch

The transient response of the quadruped's roll is displayed in Figure 32. Due to the lack of gyroscope, discussed in Section 3.5.1, the measurement of the roll will be inaccurate while moving. This causes the response to be more unstable than the pitch, but can still be

controlled. The response time is around 1.5 seconds, but the settling time is longer than ideal at about 4 seconds. The PID system, however, keeps the body of the quadruped parallel to the ground with fairly good accuracy.

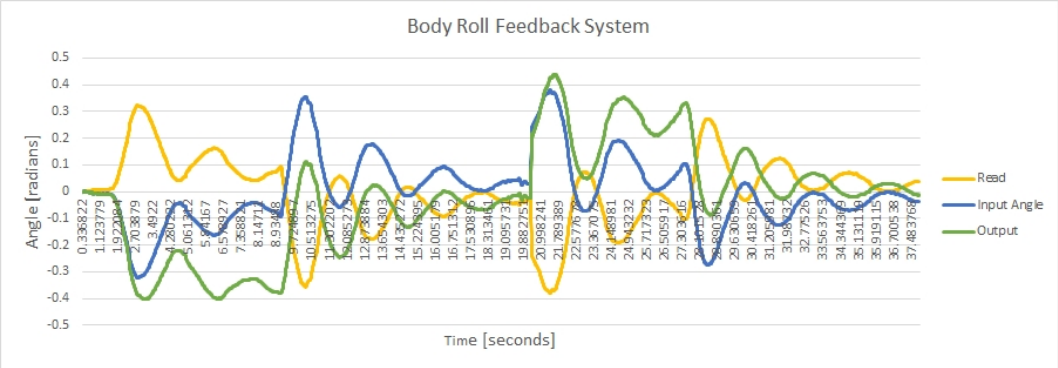


Figure 32: Transient Response of Roll

5 Conclusion

The goal of this project was to develop the required systems for a existing quadruped to enable it to walk over uneven terrain, and to keep it's balance if the surface is angled. After the required research was done multiple systems was designed and built. These systems includes software to control the legs and generate walking algorithms, as well as hardware to control all the different parts of the quadruped. These systems was tested by physically testing the walk of the quadruped over uneven terrain. The test results was successful. The quadruped could walk over uneven terrain while keeping its body parallel to the horizon. Section 5.1 provides a brief explanation of how the different objectives was met, as set in Section 1.1.

5.1 Objectives

The first objective was reached with basic trigonometry, as seen in Section 3.2.2. The second objective was met by designing a system in which waypoints are stored in a buffer, as discussed in Section 3.2.3. Objective 3 is reached with the use of the ADIS16405 in conjunction with a complimentary filter, which is discussed in Section 3.5.1. To keep the body parallel to the ground a PID system is used, which is discussed in Section 3.5.2. Gaits were added, one of which could adapt to the changing floor height, as in compliance with objective 5. This is discussed in Section 3.3. The final objective (6) was to be able to control the quadruped remotely, which was done with the use of WiFi and a TCP protocol. This is discussed in Section 3.4.3.

5.2 Recommendations

There are a few aspects of this project which can be improved upon, or done differently.

The servos itself was a big limiting factor in the accuracy of the leg movement. The servos has a short lifetime and no built-in way to make sure the servo is at the desired position. To improve the accuracy PID controllers can be programmed on the main processor to control the angle of the servos. The servos has the ability to relay its current angle. Another solution would be to acquire servos with built-in PID controllers.

A magnetomer can be used to determine the yaw of the quadruped. This information can be used to ensure that the quadruped moves in a straight line over long distances.

The math used to determine calculate the inverse kinematics and transform co-ordinates are very processing intensive if not done correctly. This project mainly work with radians, which means that trigonometry intensively, which results in complex computations to be done by the processor. A way to minimize computation time would be to make use of direction vectors, instead of angles. Vector algorithms mostly consists of multiplication, which is not very processing intensive. Another way to improve the processing time would be avoid working with floats, and instead do all calculations integer based.

References

[1] Quadruped robot gait study. [Online]. Available: http://blog.oscarliang.net/quadruped-robot-gait-study/

[2] "Dynamixel ax-12 user's manual," June 2006.

[3] H. G. Wells, *The War Of The Worlds*. William Heinemann, 1898.

[4] B. Dynamics. (2013) Bigdog - the most advanced rough-terrain robot on earth. [Online]. Available: http://www.bostondynamics.com/robot_bigdog.html

[5] D. A. R. P. Agency. (2015) Darpa robotics challenge. [Online]. Available: http://www.theroboticschallenge.org/

[6] ——. (2015) Team kaist. [Online]. Available: http://www.theroboticschallenge.org/finalist/kaist

[7] "Adis16405: Triaxial inertial sensor with magnetometer," 2009.

[8] T. Hersan. (2014) Ax-12 python library (for raspberrypi). [Online]. Available: https://github.com/thiagohersan/memememe/tree/master/Python/ax12

[9] Arduino/genuino mega 2560. [Online]. Available: https://www.arduino.cc/en/Main/ArduinoBoardMega2560

[10] Raspberry pi 1 model a+. [Online]. Available: https://www.raspberrypi.org/products/model-a-plus/

[11] D. Smith, "Quadruped robot," November 2013.

[12] Rpi low-level peripherals. [Online]. Available: http://elinux.org/RPi_Low-level_peripherals

[13] 150mbps wireless ieee802.11b/g/n nano usb adapter. [Online]. Available: http://docs-europe.electrocomponents.com/webdocs/10c8/0900766b810c836d.pdf

[14] J. G. Min and E. T. Jeung, "Complementary filter design for angle estimation using mems accelerometer and gyroscope." [Online]. Available: http://www.academia.edu/6261055/Complementary_Filter_Design_for_Angle_Estimation_using_MEMS_Accelerometer_and_Gyroscope

[15] H. J. Luinge and P. H. Veltink, "Measuring orientation of human body segments using miniature gyroscopes and accelerometers."

[16] G. Welch and G. Bishop, "An introduction to the kalman filter," 2015.

[17] Basics of pid control. [Online]. Available: http://www.industrialcontrolsonline.com/training/online/basics-pid-control-proportionalintegralderivative

[18] G. Murray. (2013) Rotation about an arbitrary axis in 3 dimensions. [Online]. Available: http://inside.mines.edu/fshome/gmurray/ArbitraryAxisRotation/

[19] "Quad 3-state noninverting buffers," August 2014.

[20] "Small signal npn transistor," 2003.

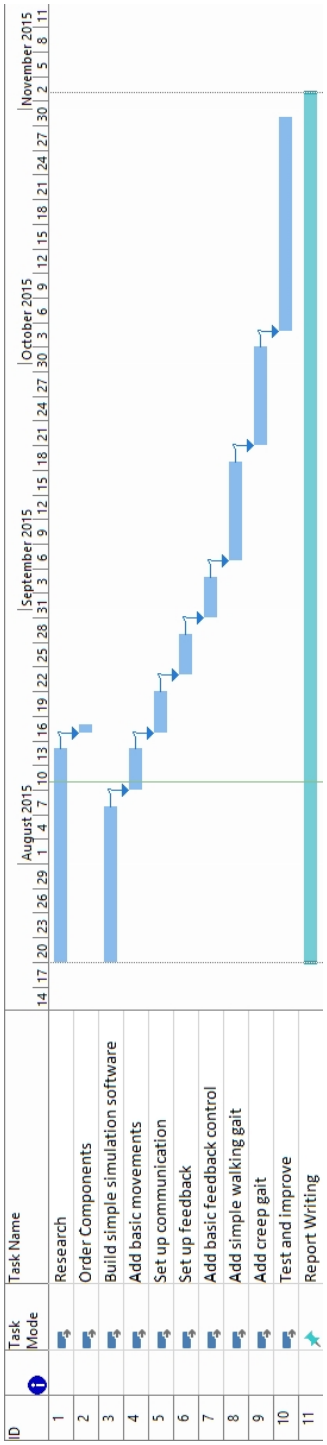
[21] 5v to 3.3v level converter. [Online]. Available: http://openrcforums.com/forum/viewtopic.php?f=84&t=4622

[22] (2012, 09) Tcp communication. [Online]. Available: https://wiki.python.org/moin/TcpCommunication

[23] C. Hardy. (2014, 4) The basics of tuning pid loops. [Online]. Available: http://innovativecontrols.com/blog/basics-tuning-pid-loops

Appendices

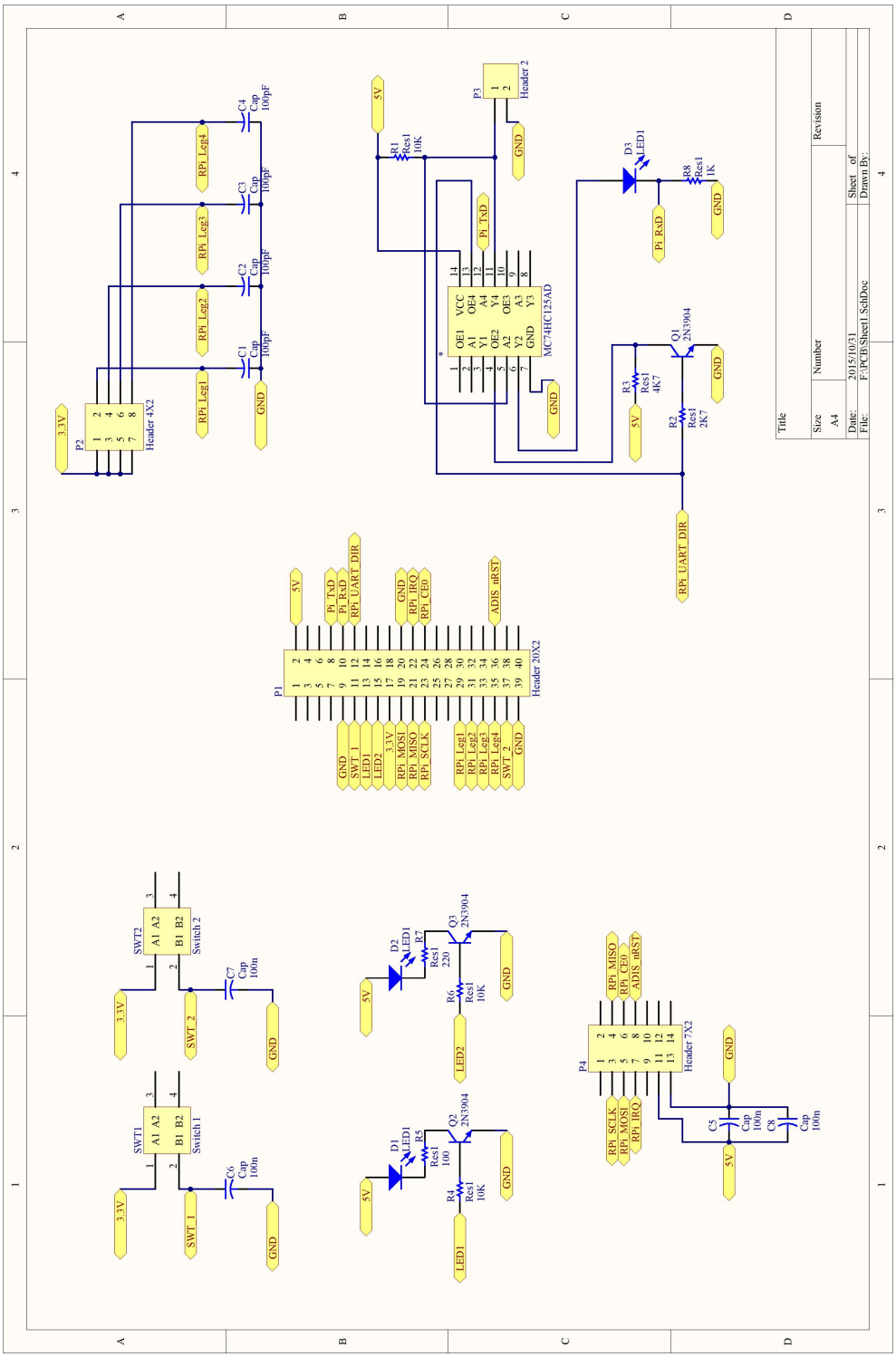
Appendix A Project Planning Schedule



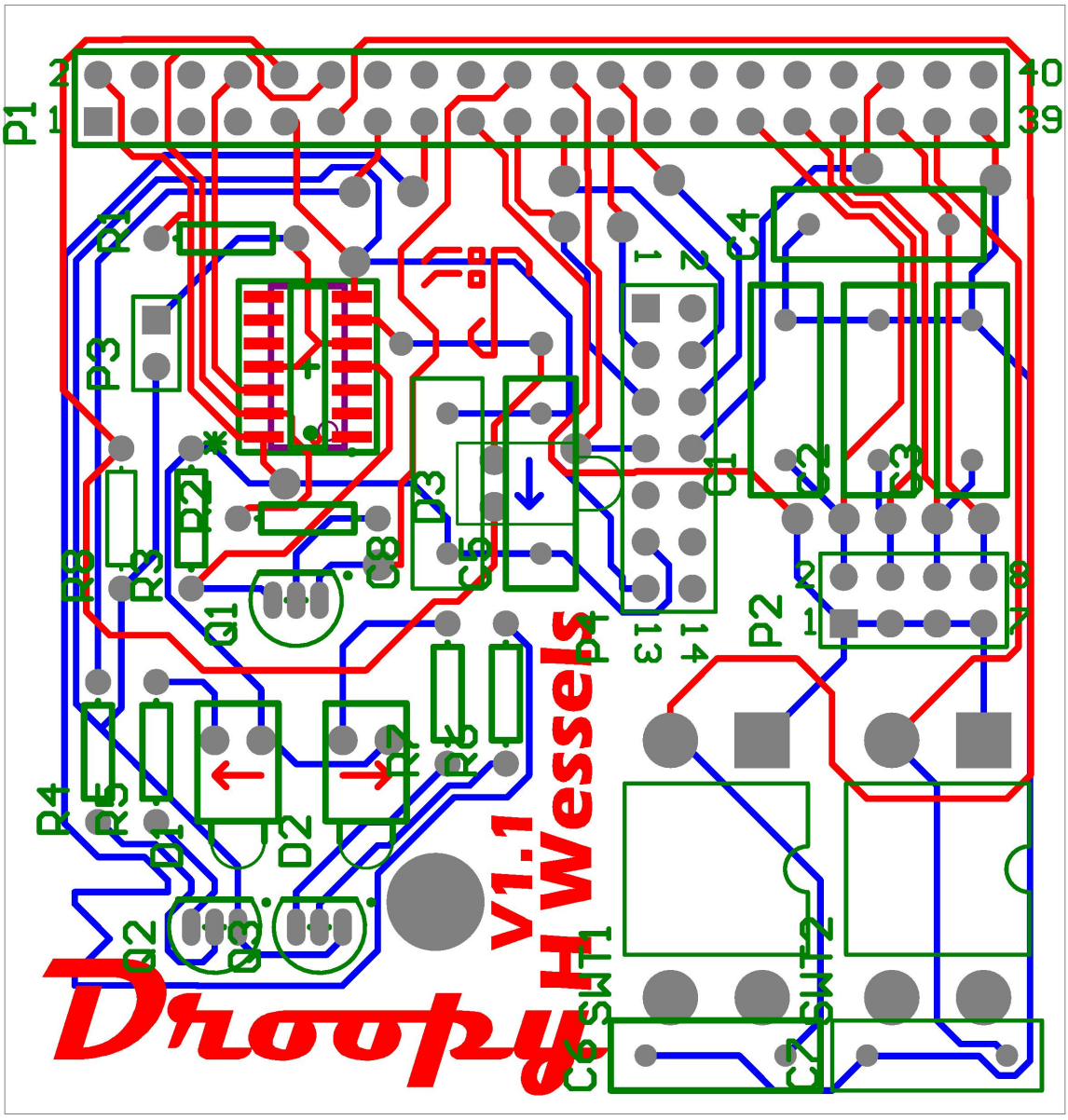
Appendix B ECSA Outcomes Compliance

Outcome	Description	Reference
Problem Solving	A variety of problems was encountered with the designing of this project in relation to circuits, software and mathematics. These problems was accessed and solutions was obtained. The problems and solutions are documented in this report.	2, 3
Application of Scientific and Engineering Knowledge	In the design chapter of this report control systems were created and tuned to a specific response. Various circuits was designed and built as well.	3.4.1, 3.5.2, 3.6
Engineering Design	All designs was thoroughly researched, and the most applicable solutions was implemented.	2, 3
Investigations, Experiments and Data Analysis	The different cutoff frequencies to use with the complimentary filter was determined by testing, retrieving data, and analyzing it for the best possible solution. The same was done for the PID sysem.	3.5.1, 3.5.2
Engineering Methods, Skills and Tools, including Information Technology	This project is mostly software based and a large complex system was designed to control all aspects of the quadruped in real time. Complex mathematics was also used to determine inverse kinematics, and manipulating the axis-systems.	3.2.1, 3.2.2, 3.3.5
Professional and Technical Communication	This report was written according to the regulations as set by the engineering faculty.	All
Independent Learning Ability	Inverse kinematics was mastered in this project, as well as implementing communication over a WiFi network with a TCP protocol.	3.2.2, 3.4.3

Appendix C Circuit Board Schematic

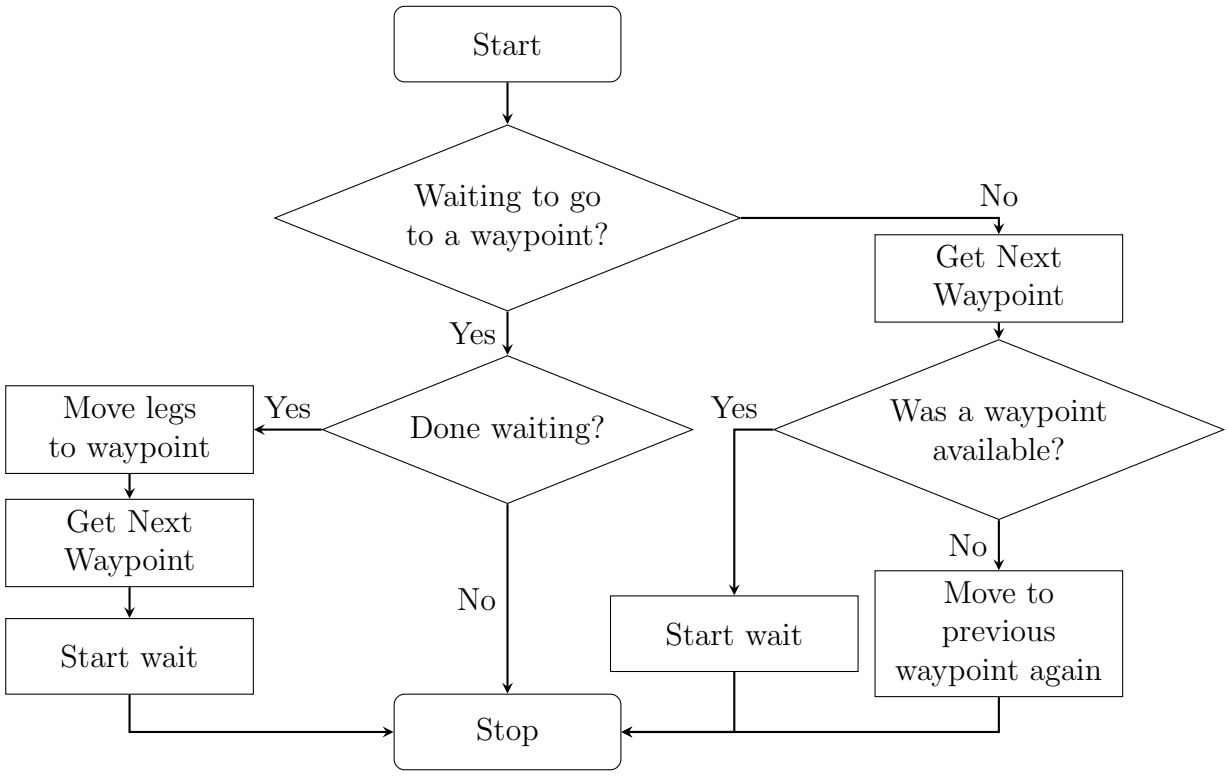


Appendix D Circuit Board Layout

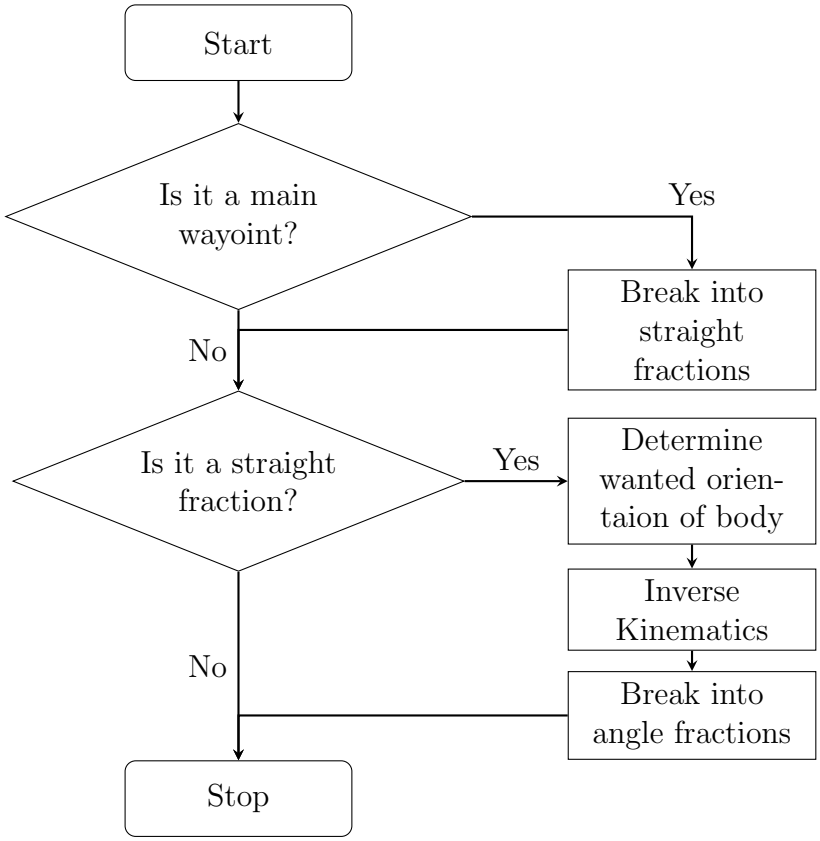


Appendix E Flow Charts

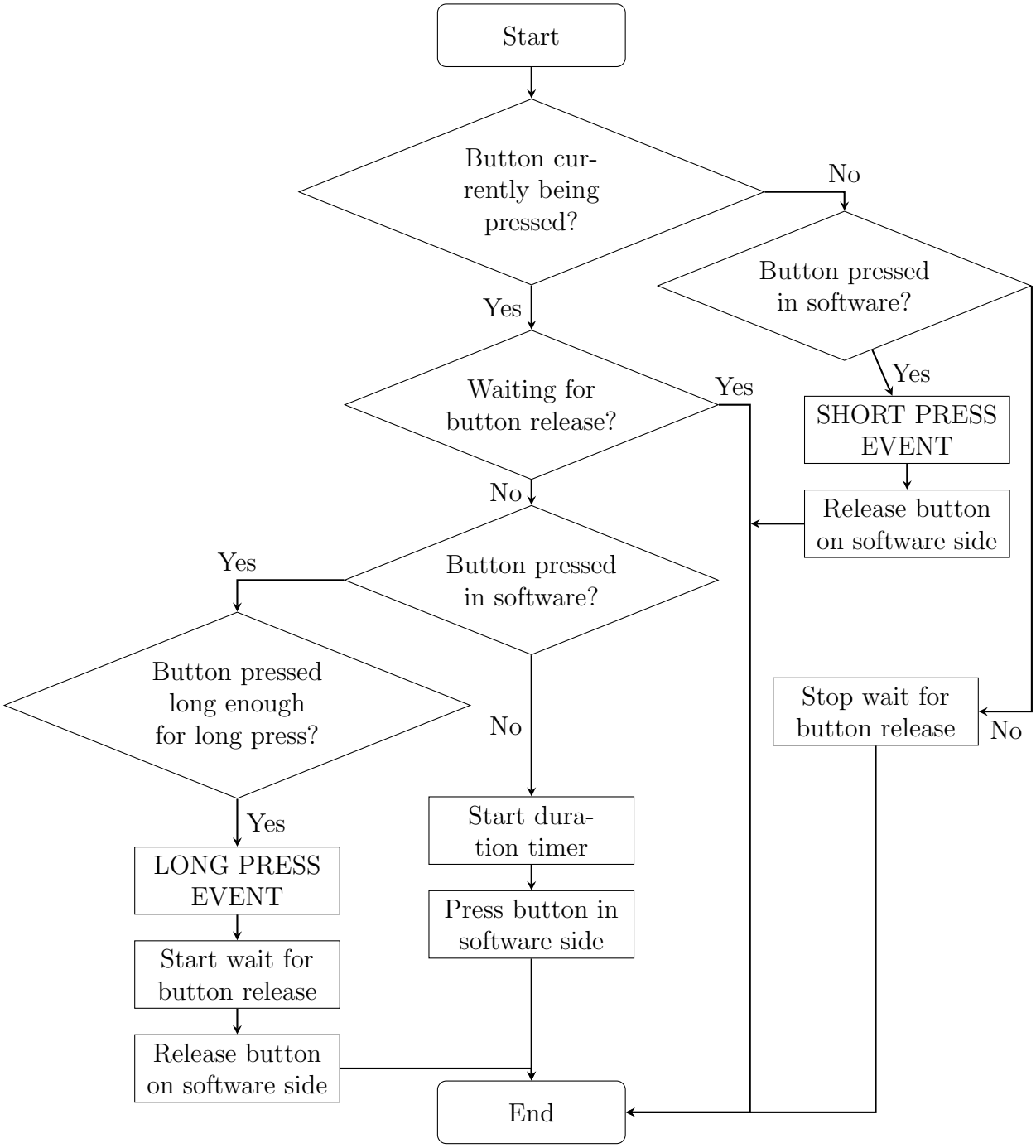
E.1 Flowchart showing polling of waypoints



E.2 Flowchart showing breaking down of waypoints



E.3 Flowchart showing polling of buttons



Appendix F Parameters Used

F.1 Gait Parameters

	Creep Gait	**Trot Gait**
Height Offset	20	50
Step Length	90	50
Step Height	80	60
Step Depth	70	N/A
Forward Translation	60	30
Side Translation	60	15
Height Offset	-40	-10
Number of Sections	128	N/A

F.2 Orientation Parameters

Offset	**Pitch**	**Roll**
Complimentary Filter		
K_I	1.5	0.1732
K_P	3	0.2449
PID System		
K_P	1	1
K_I	1	0.5
K_D	0.2	0.5