Bachelor's project

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Lars-Erik Nes Panengstuen
Edvard Brekke Merkesvik
Stian Johan Olsen
Kristoffer Meggelæ Pedersen

Instrumentation, actuation and software development of Bipedal Robot

Biped Prototype

Bachelor's project in Automation
Supervisor: Pål Holthe Mathisen

May 2021

NTNU
Kunnskap for en bedre verden

Lars-Erik Nes Paenggstuen
Edvard Brekke Merkesvik
Stian Johan Olsen
Kristoffer Meggelæ Pedersen

Instrumentation, actuation and software development of Bipedal Robot

Biped Prototype

Bachelor's project in Automation
Supervisor: Pål Holthe Mathisen
May 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Abstract

Implementing the bipedal walk in robots will allow them to move in rougher terrain and humanoid spaces where other robots might be inadequate. These robots are often explained with quite complicated mathematical functions. The ultimate goal of this project is to create a bipedal robot that is able to walk using a relatively simple mathematical function. The gait will rely heavily on the pendulum effect to increase its energy efficiency and minimise the torque needed.

Previous to this report a master of science thesis has been written about the theory behind the robot and two previous bachelor groups have worked directly on the robot. And we expect other bachelor groups to keep working on the robot. Because of this it is important to make the work as accessible and easy to understand as possible, so that future work will have an easier time at the start of their project.

Various upgrades has been made in the hardware of the robot to add new functionality. New wires were created to connect the servos to the BBB shield, a new bracket was created to secure the new servos to the feet of the biped and a new shield circuit board was created to allow the BBB to read the 5V signals from the encoders as well as send out a 5V PWM signal.

ROS is used to divide the code into packages to make it easier to develop new code or improve existing code. this also makes it easy to implement the code in other systems as the package can simply be added to any ROS workspace. The addition of a message package which all Biped nodes uses also makes it so that any system with this package can easily communicate and interact with the biped.

New IMU-code uses both acceleration measurements and gyroscope measurements and combines them trough kalman filtering along other signal processing to get a steady and accurate measurement of the leg angle. The encoders would have been a great tool to compare the IMU measurements to, but we were not able to do this due to the encoders breaking unexpectedly.

New encoder code takes the angle between the legs and torso measured by the encoders along with the leg angle measured by the IMU´s to produce a estimate of the torso angle. This is useful when controlling the robot and is also used to visualize the robot state in Rviz. The encoders have been incredibly unreliable and we will discuss what might be causing this and potential replacements.

Discrepancies with the motor from previous years have been corrected and documented. The wiring into the servo controllers and the configurations have been altered so it allows for PWM readings. The previous wiring and configurations have damaged one of the motors. The motor code has been rewritten to the current device three overlay, and to allow a second motor. The encoder code has also been implemented to reduce latency.

The servos has been extensively tested/troubleshooted and have been made to function using C++ code with ROS implementation. In addition they have been connected to the main board. One of the four servos has broken, and recommendations on how to proceed with solving this problem has been put forth.

The document also explains the theory used on the different parts of the robot. This is in order to make it easy to understand why certain decisions have been made throughout the project. For this purpose the use of ROS has also been explained.

Both an interface and visualization software has been made. This will make testing easier in the future. While there is room for improvement, it more than makes for good tools to be used in the continued development of the robot.

Lastly a github repository has been put up to be used for further development. Every relevant folder contains README-files with walkthroughs, code explanation and other relevant information for the specific folder. The goal of the Git is that you should be able to understand and launch the entire system as the git as the only resource.

Sammendrag

Tobeinte roboter kan bevege seg i røffere terreng hvor andre roboter ikke kommer til. Disse robotene er ofte avhengige av veldig kompliserte matematiske funksjoner. Det endelige målet med dette prosjektet er å lage en robot som kan gå ved hjelp av en relativt enkel matematisk formel. Roboten vil være veldig avhengig av pendel effekten for å minimalisere momentet som er nødvendig.

Før denne rapporten har en master grad blitt skrevet om teorien bak roboten, og to andre bachelor grupper har arbeidet på roboten direkte. Vi forventer at flere grupper kommer til å fortsette å jobbe på denne roboten fremover. Det var derfor viktig å dokumentere og gjøre alle resursjene så tilgjengelige som mulig slik at frentidige grupper kan bruke mindre tid på å sette seg inn i prosjektet.

Roboten har fått diverse hardware oppgraderinger. Nye ledninger ble lagd for å koble sammen servoene til BBB shieldet. Nye noterings bøyler ble lagd for å feste servoene til beina og ett nytt shield ble lagd for at BBBen skal kunne motta 5V fra enkoderne i tillegg til å kunne sende ut 5V PWM signaler.

ROS blir brukt til å dele opp koden i pakker for å gjøre det lettere å utvikle ny kode og forbedre eksisterende kode. Dette gjør det også lettere å implementere koden i andre systemer siden pakken kan enkelt legges til hvilket som helst ROS workspace. Ved å benytte en egen messages pakke som alle ROS nodene på roboten bruker kan hvilket som helst system kommunisere og kontrollere roboten så lenge systemet også har denne pakken.

Den nye IMU koden bruker målt akselerasjon samt gyroskopiske målinger og kombinerer disse gjennom Kalman filtrering langs signal prosesseringen for å få en stabilt og nøyaktig måling av benvinkelen. Enkoderne hadde vært et veldig godt verktøy å sammenlikne IMU målingene med, men vi fikk ikke gjort dette siden enkoderene ble ødelagt.

Den nye enkoder koden tar vinkelen mellom bena og kroppen som blir målt av enkoderne sammen, med benvinkelen målt av IMUene, for å produsere et estimat på kroppsvinkelen. Dette er nyttig når man skal kontrollere robotoen og det brukes også til å visualisere tilstanden til roboten i Rviz. Enkoderne har vært utrolig upålitelige og det blir diskutert hva som kan være årsaken til dette og hva som kan brukes som potensielle erstatninger.

Avvik som angående motoren som ble oppdaget fra tidligere år har blitt rettet på og dokumentert. Ledningene som går inn i servo kontrollerene og konfigurasjonene rundt har blitt endret på slik at PWM signaler kan bli lest. Det tidligere ledningsoppsettet og konfigurasjonene har skadet en av motorene. Motor koden har blitt skrevet på nytt til det nåværende enhets tre overlegget for å tillate en andre motor. Enkoder koden har også blitt implementert for å redusere ventetid.

Dokumentet forklarer også teorien som er blitt brukt på de forskjellige delene av roboten. Dette er i hovedsak for å gjøre det lett å forstå hvorfor enkelte valg har blitt gjort i løpet av prosjektet. Av samme grunn har bruken av ROS også blitt forklart.

Både et grensesnitt og et visualiserings program blitt lagd. Dette kommer til å gjøre det enklere å teste deler av roboten i framtiden. Selv om det er rom for forbedring, er begge veldig gode verktøy som kan bli brukt videre i utviklingen av roboten.

Til slutt har en Github-side blitt lagd til bruk for fremtidig utvikling. Hver relevante mappe har sin egen README-fil som forklarer hvordan du installerer og utfører mappe-relevante prosesser, forklaringer av kode og annen relevant info til den bestemte mappen. Målet til giten er at du skal forstå og klare å sette i gang hele systemet med bare giten som ressurs.

Preface

This paper documents our work done on the Bipedal Robot Prototype in the sixth and last semester of our bachelor degree in electrical engineering with specialization in automation. The subject itself is worth 20 study points. Working on the project has been both interesting and challenging.

We would like to thank our supervisor, Pål Holte Mathisen, and our employer, Torleif Anstensrud, for guiding us through this project and making time for us even in short notice. We would also like to thank Krisitan Strøm and other employees at the department of Engineering Cybernetics for helping with equipment and giving advice.

Trondheim, May 2021. Group E2110:

[illegible]

[illegible]

[illegible]

[illegible]

Figure 1: Group signatures

Contents

Definitions

BBB "Beaglebone Black": A single board computer.

BBB shield: A circuit board designed to go on top of the BBB to add functionality.

C/C++: General-purpose programming language, often used in embedded systems and real time programming.

Coloumb Friction: Is an approximation of dry friction which is the force two solid objects sliding across each other.

Encoder: A device that converts angular position of a rotating joint to a digital- or analog signal.

GitHub: A code hosting platform for version control and collaboration.

GUI: Graphic User interface, allows a user to input actions using interactive visual components.

I2C: A serial communication protocol between one or several masters and one or several slaves.

IMU "Inertia Measurement Unit": A device capable of measuring angle, angular velocity and its orientation.

Linux: An open source operating system used to run R.O.S. We will be using a distribution called "Ubuntu"

Microsoft Teams: A business communication platform, we use it mostly for video meetings and to store documents.

OS: Operating system.

PCB: Printed circuit board.

PWM: Pulse Width Modulation, mainly used in order to get analog results using digital means. A digital controller creates square wave signals that allows a user to switch between an "on" and "off" state.

Radial Load: The load working perpendicular to the motor shaft.

Restoring torque: The torque which rises to return an object (twisted, rotating, etc.) to its original orientation.

ROS "Robot Operating System": A framework for robot programming to run on.

Rviz: A 3D visualization tool for ROS.

SPI: Short for "Serial Peripheral Interface". Is a communication protocol primarily used in embedded systems.

Technical documentation: Documentation about a product containing most of the relevant information a customer needs.

Underactuated robotics: A type of robot where the number of actuators is lower than the degrees of freedom. E.g. a bipedal with actuators in the hip.

Virtual Machine: A simulation a machine to host another or multiple OSes using the same hardware.

1 Introduction

1.1 Background

An increasingly large part of modern society includes some form of robotics. In most cases this is fixed robots, however as new technologies are developed it is desirable to develop new robotics with higher mobility that can mimic human motion and operate outside the highly controlled environments where they are used today.

In conjunction with the desire to mimic human walk there is also a need to research and develop energy efficient gaits. At NTNU there have been developed several mathematical models for effective walking patterns. These models have been simulated, but has yet to be tested on a physical model. The purpose of this robot is to do exactly that. Our bachelor thesis will look to bring the system closer- and hopefully all the way to completion.

The robot in question is an under-actuated bipedal that is restricted to move in a 2 dimensional plane. The robot consists of 2 stiff legs mounted to a torso at the "hip". With the mathematical models designed to work in two dimensions we have to make sure to prevent lateral tilt, consequently each leg has two points of contact as if you would stretch the robot sideways when you add the third axis.

The structure of the robot is already done, and a variety of electronics and sensors have been implemented. With most of the hardware done, the remaining work is largely software based and will largely consist of measurement and communication as well as making sure the previous work is still functional.[1]

1.2 The outline of the thesis

This thesis starts with introducing the robot and elucidating the problem statement. It then goes trough fundamental theory and concepts as well as the results of early testing before digging into the work done this semester.

The main chapters covers work related to hardware and the embedded system, which contains most of the workload, before finishing of with ROS and visualization. These chapters follows roughly the same model of theory, process, results, challenges.

The thesis ends with a discussion about the project and a conclusion as well as an overview of future work that is required to reach the ultimate goal.

Note that a lot of the documentation is contained within the git and not the appendix as we don't consider it to be very helpful for understanding the thesis.

1.3 Preliminary design and work

1.3.1 Frame

The frame is designed using AutoCAD and was built at the department of Engineering Cybernetics at NTNU[2]. It consist for the most part of square aluminum tubing. The tubing is constructed so the frame has a main body and 2x2 parallel legs. This makes the the balance drastically better than what would be achieved with only two legs. An aluminum plate is mounted on the torso of the frame. This is where the control system for the robot is situated, mainly consisting of the BBB and the power distribution system. The frame has pre-drilled holes on the legs which makes it easy to mount both sensors and servos wherever desired.

1.3.2 Power supply

The BBB does not have enough power to power the servos, actuators and encoders. Hence an external power supply is being used. This is the "QPX600D Dual 600Watt DC Power Supply".

1.3.3 Embedded system

The brain of the embedded system is the BeagleBone Black. A PCB-board is mounted on top of the BeagleBone-pins and every single signal goes through this board. The motors are located at the hip-joint, the encoders is attached to the motors, the IMUs are located on the legs and the servos are located at the bottom of the legs. Neither servos nor IMUs have been implemented properly prior to this year.

1.3.4 Preliminary part list

The bipedal robot itself is made up of a variety of different components. This table includes all hardware and external physical components found connected to the robot. It is worth noting that table 1 are based of last years part list[3], where we have found discrepancies, these have been corrected, but we haven't been able to check all the components.

Description	Product name	Manufacturer
Flexible actuator coupler	4779823	Ruland
Hip bearings	6004-C	Fag
Electric motors for leg actuation	148877	Maxon Motor
1-to-6 Gearbox for motors	Planetary Gearhead	Maxon Motor
Servo controllers for motors	ESCON 70/10 (422969)	Maxon Motor
Servos for retractable feet	TGY-SC340V	Turnigy
Encoders for relative leg angles	2RMHF	Scancon
Inertial measurement unit chip	LSM9DS1	STMicroelectronics
IMU Breakout board	Breakout-LSM9DS1	Sparkfun
Power supply	QPX600D	Aim-TTI
BeagleBone Black	BeagleBone Black	BeagleBoard.org

Table 1: Equipment & Components

1.4 Problem statement

The main problem statement of this bachelor is to further develop code and wiring for the bipedal robot. Ideally the robot would be able to take a couple of steps. The parts of the system that would be the focus was the IMU's, the motor, the servos and the encoders.

In order to improve upon the project it is important to finish modules in order to make continuing easier for the next group. As such making the hardware work properly ended up being most important. This would also require having working code in order to make sure the different parts of the system behaved as planned. In addition this requires making easy to understand documentation to minimize the amount of time spent testing and troubleshooting. Doing this will make reaching the end goal of having a functional bipedal robot easier.

1.4.1 Familiarization with previous work

The robot was built in 2014 and has since then had one master of science student writing his thesis about the theory behind the robot and two bachelor groups working directly on the robot.
This made it necessary to read and become familiar with the three previous theses, and the developed code and documentation in the project folder. Furthermore the group will have to find areas that needs or has potential for improvement.

1.4.2 Learning ROS and C++

In order to advance the development of the robot it is necessary to know how to use basic ROS and C++. The group does not have a lot of experience with either. As such the a prerequisite before starting to work on the robot will be to spend time learning about this. As ROS and C++ are quite vast subjects, a lot of this will be learning by doing throughout the project.

1.4.3 Github & Documentation

This project will have other people working on it later on, hence it is important to have all the documentation neat and clear. It should also be explained in a fashion that is easily understood by the new students. It is most likely other electrical engineering students that will take over. As such the explanations will be aimed towards people with similar educational background.

1.4.4 Implement measurement of upper body angle with IMU, including electronics and software filtering

Currently the robot only knows the upper body's angle relative to the legs measured by the encoders. For it to be able to mimic the gaits mentioned in 1.1 it needs to know its position relative to the ground as well. This will be achieved by IMUs mounted on the legs. Since an IMU uses several sensors filtering these signals and combining them to a reliable angle will be important.

1.4.5 Motor

The actuators have not been tested since the bachelor group in 2019. Even then it was only the inner leg that was tested. This means the group has to find out how the embedded motor system work, troubleshoot the motors and servo-controller and test the developed code. If the developed code doesn't work, changes, or new code will be implemented.

1.4.6 Servo

The servos have not been properly tested for quite some time. As such making sure that both the hardware and the code from previous groups are in working condition will be the first priority. After that any necessary changes will be implemented. If it is discovered that the code or the servos themselves don't work, this will be fixed. The servos also need to be wired to the main board.

1.4.7 ROS

After the dSpace real time system was discarded due to a ransomware infesting the computer a decision was made to control the system using ROS. This transition was started last year and we aim to finish it, creating a complete library to control actuators and read sensors. Ideally the robot should be able to walk once a control algorithm is implemented.

1.4.8 Develop software for (real-time) logging and visualization of robot status based on sensor readings

Being able to visualise and track the robots perceived state can be very helpful when debugging. We aim to implement real time visualization of a robot model as well as tracking these values.

2 Theoretical Framework and Preliminaries

The underlying chapters will elaborate our use of sources and methods to process these. The chapters will also contain fundamental concepts and theory used during the project and in writing this report.

2.1 Sources and Methods

The predominant part of this project is built on knowledge and expertise outside our syllabus, hence it was decided early on that everyone should learn the fundamentals of what we believed was most important to advance the project. The two main subjects to learn was coding in C++ and ROS. Since these are giant subjects on their own, it was important to not go too deep into them and waste time. As this work took place during a pandemic and the campuses was off limits for longer periods of the time. We decided to learn ROS and C++ mostly individually. And since people learn differently the approach to what sources and methods people used varied. The resources we found most useful were, *Ros Wiki*[4] and the comments in the ROS packages code.
During this period we also familiarized ourselves with the previous work. This includes the tree papers, *Stable Gaits for an Underactuated Compass Biped Robot with a Torso*[5], *Instrumentation, actuation and model identification of bipedal robot*[6] and *Biped Robot Prototype - Embedded system integration with PID controller*[7], and the coding, simulations and technical documentation attached to these.
A cumulation of sources was also used when learning the BBB and linux. One of the main sources we depended on was *Exploring Beaglebone - tools and techniques for building with embeded linux*[8].
A lot of work was also put into understanding and changing the wiring to the different components. The main sources here was the hardware references. In a couple of cases we contacted the producers.

2.2 GPIO

GPIO, also known as General Purpose Input/output is a digital pin that can function as either an input or an output pin[9]. This means the pins allow for a lot of flexibility. Depending on what kind of functionality is needed, the pins can be defined and programmed to behave as necessary.

2.3 PWM(Pulse Width Modulation)

Pulse Width Modulation is a powerful technique for controlling analog circuits with a processor's digital outputs[10]. Arguably the main advantage of using PWM is that the power loss created when switching devices is relatively low. In addition an analog signal will vary its output over time. Which isn't ideal when you want to change between a "on" and "off" state. It is also problematic if you are dependant on a certain percentage of input. Digital signals on the other hand can only have a finite amount of values(usually 0V-5V). This makes using a digital signal more predictable as well as more cost effective. Another positive trait is that PWM is less susceptible noise.

In essence PWM is a way of making analog signals digital. The method itself chops the power delivered from a source into discrete parts. This results is a square wave signal which for all intents and purposes can be interpreted to switch between a high and a low state. To put it in even simpler terms a PWM signal simulates a analog signal by applying power in pulses. Which makes controlling an analog device, for example a servo motor, easier.

When it comes to PWM signals the proportion of time when the square signal is in an "on" state is described by the duty cycle. The duty cycle is in other words the percentage of time when the square signal is high. As seen in the figure below.

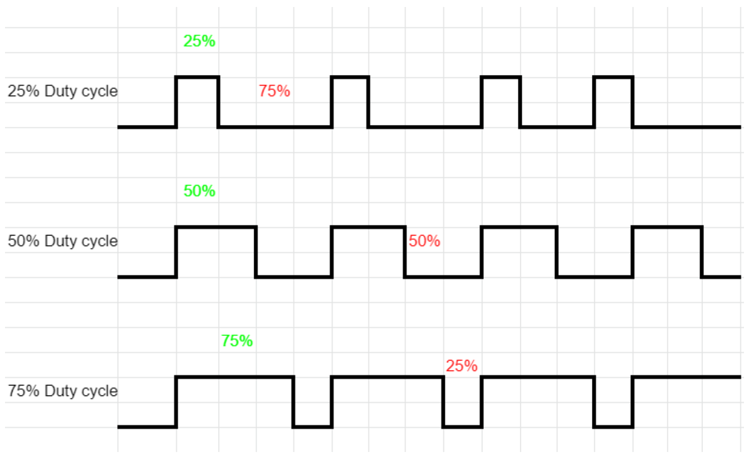


Figure 2: PWM signal duty cycle

2.4 Encoder

A fundamental part of controlling the robot movement is to have an accurate reading of the legs relative to the torso. To achieve this encoders are used. An encoder is a sensor that is used to measure rotation. It can measure both direction of the rotation as well as the angular velocity. The type of encoder used for this project is called an incremental encoder.

Incremental encoders does not measure absolute position, but rather change in rotation via two signals A and B. It is made up by a rotating opaque disc with slits that can pass light through. A light source is placed on one side with a photodetector on the other, The encoder counts the number of flashes to know how far it has rotated. To measure the direction of the rotation another set of slits are introduced with a relative displacement to the original ones. The encoders know which way they are turning by measuring which signal leads the other.

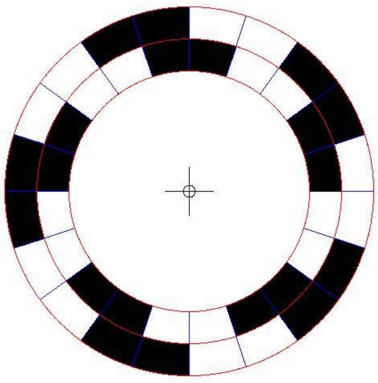


Figure 3: Incremental encoder wheel

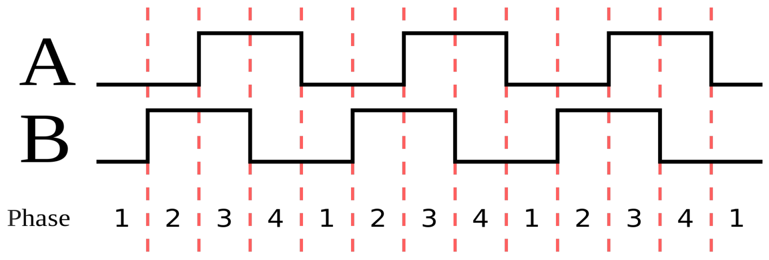


Figure 4: Incremental encoder pulse

As the name suggest an incremental encoder displays it's value in increments. The encoder has 4 states for each hole in the disk, A-on B-on, A-on B-off, A-off B-on and A-off B off. This means that for every slit there is 4 counts. The resolution N is determined by the number of slits on the disc. The angular displacement is then defined as

$$\theta = n\frac{360°}{N * 4} \tag{1}$$

where N is the number of slits each revolution and n is the pulse count.

2.5 Pendulum effect

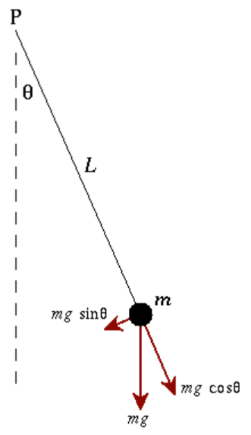


Figure 5: The forces acting on a pendulum in an ideal environment

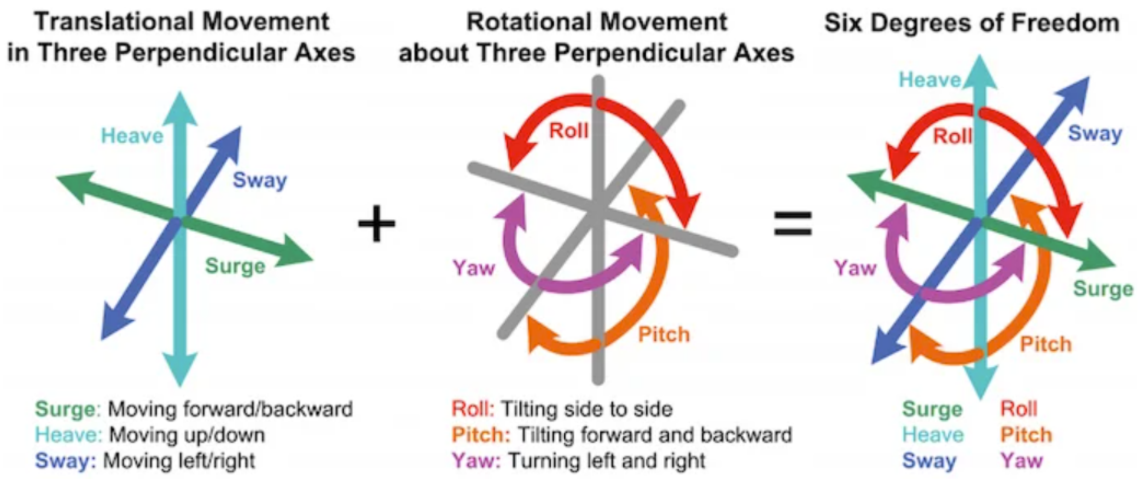
This has been explained in great length and calculated on in the past bachelor reports[11][12]. Thus a short summary should suffice.
A mass **m** hanging a length **l** from a pivot point **P**. Gravity **g** will make a resting point for the mass, this is the so called equilibrium. Gravity will always work with a force equal to $m * g$ on the mass. Changing the angle θ will make the mass oscillate back and forth past equilibrium. The oscillation will have a constant period **T** no other forces apart from gravity acts upon the mass.
In reality frictions at the hip joint, coloumb friction, and air resistance is present. With these two in mind, Patrick T. Squire came up with function 2[13].

$$\frac{d^2\theta}{dt^2} + a|\frac{d\theta}{dt}|\frac{d\theta}{dt} + b\frac{d\theta}{dt} + csgn(\frac{d\theta}{dt}) + \omega_0^2(\theta - \frac{1}{6}\theta^3) = 0 \tag{2}$$

This equation focuses on the dampening effect of the pendulum motion. $\frac{d^2\theta}{dt^2}$ is acceleration. $a|\frac{d\theta}{dt}|\frac{d\theta}{dt}$ is called quadratic dampening and is the dampening from air resistance. Under normal conditions this considered to be proportional to the square of the angular velocity. $b\frac{d\theta}{dt}$ is linear dampening and is derived from low Reynolds number. $csgn(\frac{d\theta}{dt})$ is the dampening from dry friction. $\omega_0^2(\theta - \frac{1}{6}\theta^3)$ is an two term sine-approximation of the restoring torque. This approximation holds good for angels up to 45°. a,b and c are dampening coefficients. sgn is a function representing the positive or negative sign of the value $\frac{d\theta}{dt}$.

2.6 IMU

Inertial measurement unit measures angular rate, force and sometimes magnetic fields[14]. The IMU consists of a gyroscope and an accelerometer.
The accelerometer can sense both dynamic and static acceleration. The static acceleration is almost always gravity and can help to tell the tilt of the accelerometer in respect to earth. This capability is probably most known to be used to tell your phone if it should be in landscape or portrait mode. The dynamic acceleration is for sensing motion.
The gyroscope measures the angular velocity. It tells the so called roll, pitch and yaw. It is not affected by gravity, so its perfect to combine it with an accelerometer



Translational Movement in Three Perpendicular Axes
Heave
Sway
Surge
+
Rotational Movement about Three Perpendicular Axes
Roll
Yaw
Pitch
=
Six Degrees of Freedom
Heave
Roll
Sway
Yaw
Surge
Pitch
Surge: Moving forward/backward
Heave: Moving up/down
Sway: Moving left/right
Roll: Tilting side to side
Pitch: Tilting forward and backward
Yaw: Turning left and right
Surge Roll
Heave Pitch
Sway Yaw

Figure 6: Picture depicting the combination of accelerometer and gyroscope[15].

2.7 I^2C

I^{2}C is a communication protocol. The communication can be between a master and multiple slaves or it could be several masters controlling one or multiple slaves.
To create an I^{2}C-message the data is divided into frames and the slave address is added to the beginning. Before and after every data frame an acknowledgement segment is added. This allows the system to detect if the data is accepted by the receiver and can be very important if there are a lot of noise around the system.

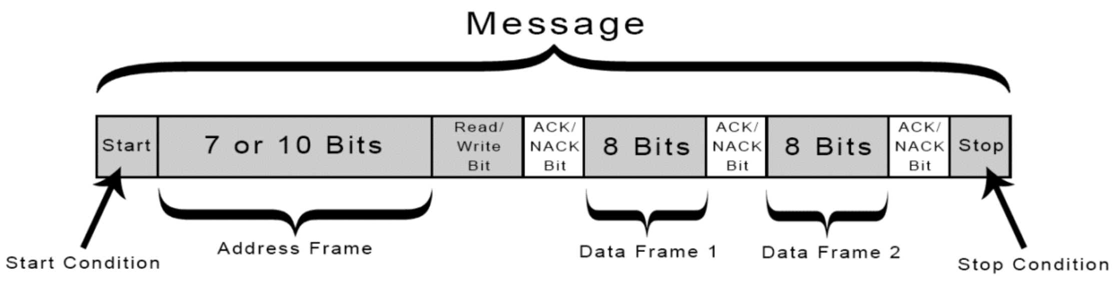


Figure 7: The I^{2}C -message[16]

The downside of I^{2}C compared to the SPI-protocol is the slower rate of data transfer. This does however not affect our system because the IMU can't send data faster than 952Hz.

2.8 SSH

Secure Shell (SSH) is an encrypted network protocol for communication between devices, and what we use to communicate with the BeagleBone. The protocol follows a client-server model, meaning that the connection is established by the SSH client (computer) that connects to the SSH server (BeagleBone). After the client and server are connected the protocol ensures that the information being exchanged is encrypted.

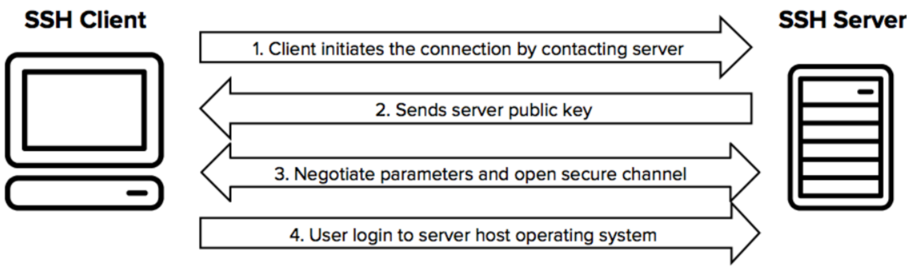

SSH Client
SSH Server
1. Client initiates the connection by contacting server
2. Sends server public key
3. Negotiate parameters and open secure channel
4. User login to server host operating system

Figure 8: SSH setup flow

2.8.1 SCP

Along with SSH we use the Secure Copy Protocol (SCP) to transfer files from the computer to the BeagleBone. SCP is based on the SSH protocol and is included in most Linux distributions. Alternatively one can use (SSH File Transfer Protocol) SFTP the main difference being that it is more secure but slower than SCP.

2.9 Software

- **Oracle VM virtualbox:** Can create and load multiple guest OSes in their own virtual machines. Used in this project to create an linux environment on Windows computers.
- **Winscp:** Used to transfer and edit files on the BBB from a Windows OS.
- **Visual Studio Code:** Used to edit source code.
- **balenaEtcher:** Used when flashing OS image onto the SD-card.
- **ESCON studio:** Used to configure and monitor the Escon Servo controllers.
- **Matlab/Simulink:** Used for system and function simulations.

3 Testing previous work

Continuing on someone else's work brings both benefits and disadvantages. These are some discoveries we made while starting the project initially.

3.1 Hardware

The encoders were not outputting any signal. At first this was believed to be a software flaw from our side, but when nothing seemed to work we realised that both encoders were broken. This is described in greater detail in 5.3.2.

The main hardware improvements made last year was introducing a circuit board to handle all the connections to the BeagleBone. This board did however have several problems.

- The encoder signal which is 5V was led directly into the BeagleBone pins which can only take 3.3V. This ended up frying it. An external level shifter was implemented on top of the PCB to solve this problem prior to us starting this project.
- Upon debugging the board we discovered that 5V was being delivered to the encoder signal output. We are unsure if this is caused by the board itself or because of the level shifter. This is what we believed destroyed the encoders at first.
- The main motor current was being lead through the PCB to the servo controllers.

The combination of these problems led to a decision to design a new PCB.

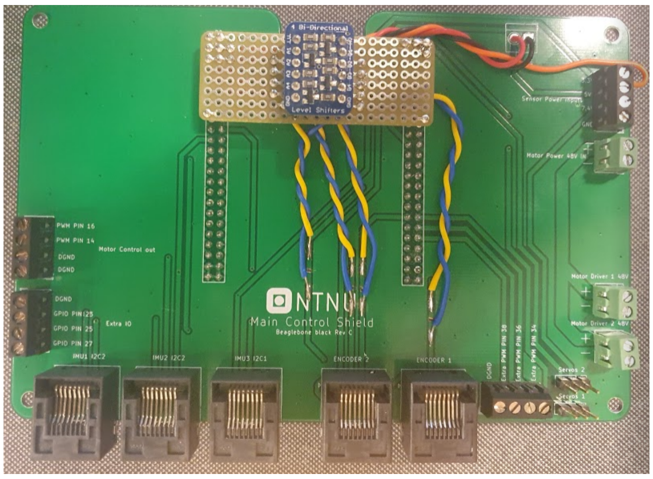


Figure 9: PCB with level shifter

3.2 Code

As the 2020 group mentions in their report[17] the code they developed was meant as groundwork for future development and the ROS code was meant as a "proof of concept". As a consequence none of the previous code was in complete working order from the get go, making testing it difficult. In addition the ROS setup was lost when the BeagleBone was shorted in 2020.

With little documentation as well. The earlier code served as a starting point to work off of.

4 Hardware design

4.1 Printed Circuit Board

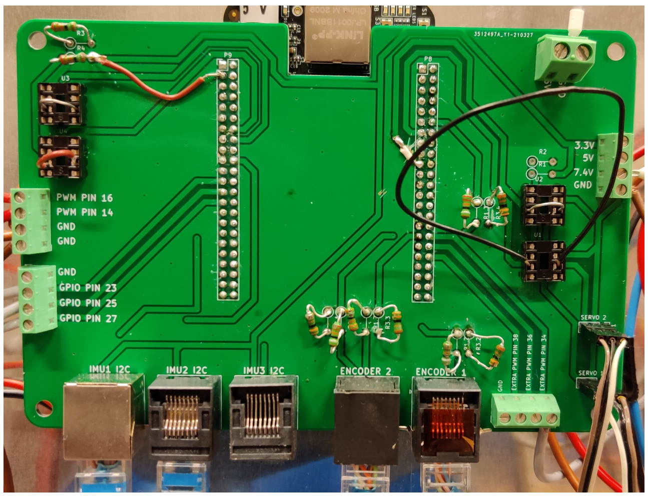


Figure 10: BBB Shield

As mentioned in chapter 3.1 it was decided to design a new PCB early on as we believed it was the cause for the encoders breaking. Unfortunately the CAD files from last year had not been uploaded, meaning we had to start from scratch instead of modifying the existing one.

4.1.1 PWM logic level shifting

The circuit board was made with the assumption that all PWM signals on the BBB, motor controllers and servos worked like the standard servo PWM signal in that 2ms on is interpreted as 100% and 1 ms on is interpreted as 0%. However we now know that the BBB can control the duty-cycle from 0 to 100% and the servo controller interprets 10% duty-cycle as 0 and 90% duty-cycle as 100%.

When designing the circuit board we did not know whether the servo controllers and the 7.4V servos would accept a 3.3V logic level signal. We decided to add a level shifting functionality to the board to change the logic level of the BBB PWM outputs from 3.3V to 5V or potentially, with a few minor changes 7.4V. Due to the relatively small space on the board and the fast switching required, we decided to try to find the simplest solution to minimize the risk of complications. We considered two ways of boosting the signal voltage that each only required three components.

4.1.2 Op-amp circuit

The first was to simply use a op-amp as a comparator to boost the signal. The comparators only needed one op-amp and two resistors to work.

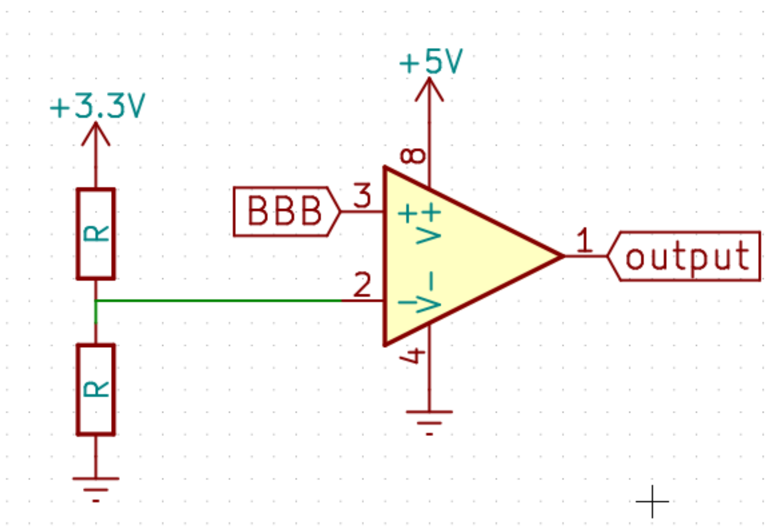


Figure 11: op-amp logic level booster

However the slew-rate and the switching delay would limit the working frequency. We can calculate the effect this would have on the signal.

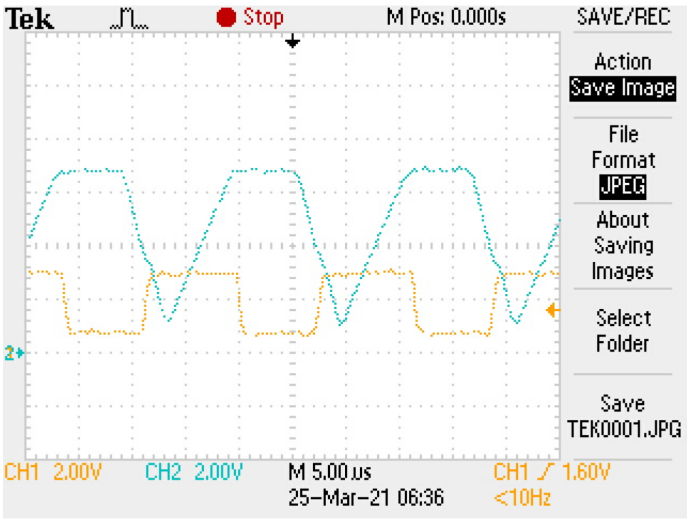


Figure 12: Op-amp test

The test was done using the LM108AJ-8 op-amp, which were easily available to us at the time. We can see from the image that the slew-rate of the op-amp is about 4V/5μs or 0.8V/μs and the delay is about 3 ms. The 3μs delay would not really change the signal only delay it. Which means that the maximum switching delay at 7.4 v is $\frac{7.4V}{0.8V\mu s} = 9.25\mu s$. The Servo signal difference between 0% and 100% is 1ms which means at most this would affect the signal $\frac{9{,}25\mu s}{1000\mu s} = 0.925\%$. This could easily be corrected for in code by simply adding 12,25μs to the duty-cycle. This is also assuming the logic switch of the servo is at 7.4V or 0V. This is not likely going to affect the signal in any meaningful way but it is a downside worth considering.

4.1.3 Transistor circuit

The other way is by using a transistor to boost the signal.

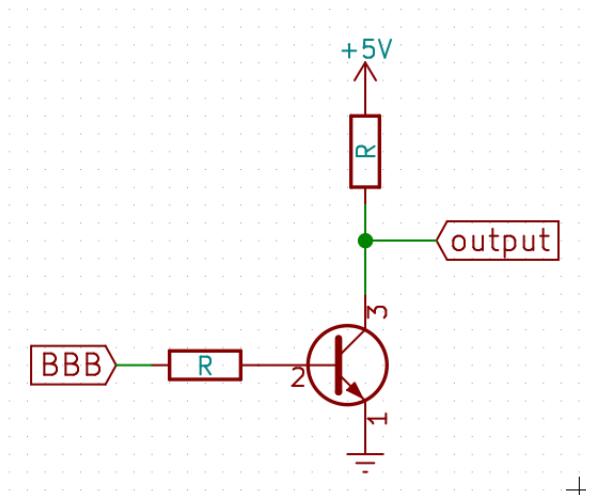


Figure 13: transistor logic level booster

Transistors are way faster, hence does not have the weakness of the op-amp. However in order to use the transistor we would either have to use more parts increasing the complexity of the board, or use an inverted signal to control the servos and motor. Knowing that the BBB can send out any signal this would not be a problem because we can simply send the inverse signal of the signal we actually want.

We decided to use the op-amps to boost the PWM signals and avoid having to invert the signals. This turned out to be unnecessary since both the motor-controllers and the servos accepts 3.3V logic levels. Therefor the op-amps are replaced with a wire allowing the signal to pass from the op-amp input to the op-amp output.

4.1.4 Encoder signal processing

In order for the BBB to read the signals coming from the encoders, the voltage have to be stepped down from 5 to 3.3V. This was done using a simple voltage-divider.

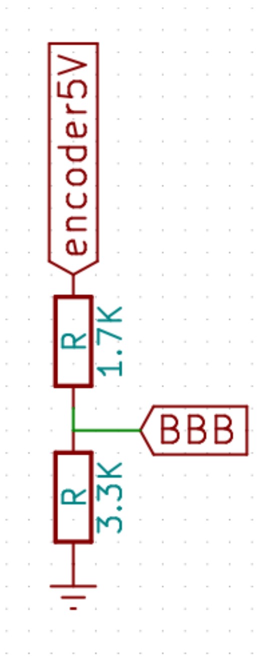


Figure 14: Encoder voltage divider

When deciding what resistor values to use, we decided to use the resistors that would draw around 1mA in its on state. This will make sure we are well under the 35mA rating of the encoders while simultaneously ensuring that any current-draw from the BBB will not pull down the voltage. When the signals from the encoders are 5V we want 1mA current-draw, which means the total resistance to ground will be $\frac{5V}{0.001A} = 5K\Omega$. For the voltage to be 3.3V at the BBB while the current is 1mA, the resistance to ground must be $\frac{3.3V}{0.001A} = 3.3K\Omega$. Which means the resistance between the encoder and the BBB must be $5K\Omega - 3.3K\Omega = 1.7K\Omega$. When considering that we have four signals being processed(two from each encoder) which are on, an average of 50% of the time, we can estimate a total of $4 * 1mA * 5V * \frac{1}{2} = 10mW$ of power lost during operation. We know that 1mA is probably overkill, but we decided that a reliable signal was worth ten milliwatt wasted, which is not significant in this context.

4.1.5 Mistakes

The board was made early in the project to make sure we had a lot of time for testing. A lot of assumptions had to be made to make the board as quickly as possible. We started with the original board as reference and added the functionality we needed. One mistake was that one of the PWM pins that was going to control the servo, was connected to the P8_17 pin. This pin does not have PWM functionality, so we had to scratch of this trace and solder on a jumper from P8_19 instead. Another error was that one of the ground pins of the op-amps on the board was not connected. This was a simple oversight and was easily corrected with a small jumper-wire. While designing the circuit board we forgot to include a 5V connection to the Vcc input of the BBB. This was also solved by simply adding a jumper wire from the nearest 5V pin to the BBB. While soldering one of the Ethernet-sockets on the board, some solder shorted one of the pins between the socket and board which made it very hard to find. Luckily, further testing show that this did not affect the pin that was grounded and it seems to be working fine. The pin in question is P9_18. All of these mistakes have been fixed on the current board as well as in the CAD files in the GitHub.

4.2 Motor wiring

The terminals for the motor power are removed as the circuit board would have been a lot more expensive to manufacture to accommodate the large amount of current the motors are rated for. For instance, according to Advanced circuits circuit-board calculator[18], in order to handle the 16amps the motors are rated for, the trace would have to be 542mm wide with the same copper thickness used on this board. We see no reason why the motor current needs to run trough the board and have decided to wire them directly to the motor-controllers.

4.3 Connecting servos

Although most of the wiring was already finished previously, new wires to connect the servos to the PCB was needed. We soldered two pairs of wires in parallel so that two and two servos gets the same signal and moves in unison. We also mounted 3-pin dupont connectors to make it easy to mount a new circuit board or servos. The wires are also shrink wrapped and color coded.

4.3.1 Servo brackets

Because the Futaba s9254 servos were destroyed in 2019, last years group were tasked with ordering new ones. They ended up with the Turnigy tgy-sc340v which has smaller dimensions and hence does not fit in the original mounting brackets. We were tasked with designing new ones. This was done using the Shapr3D CAD software. The models were then manufactured at the mechanical workshop of the department of cybernetics.

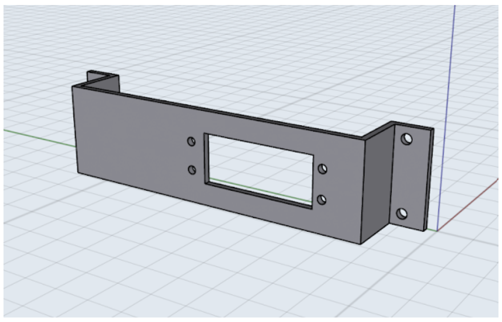


Figure 15: Servo casing

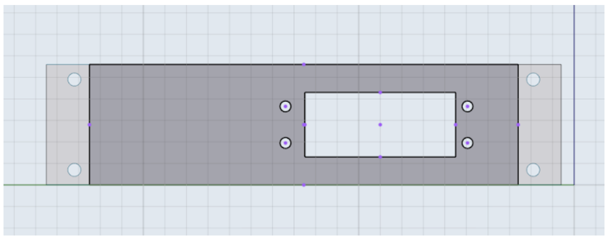


Figure 16: Servo casing

5 Embedded system

The act of making a bipedal robot walk on its own is deceptively complex. It requires processing data from various sensors and using these in stabilizing algorithms. These algorithms do a large amount of mathematical calculations in order to balance the biped. As such it is necessary to control the robot using embedded hardware rather than using an external computer. There are several different components included in this system in order to make the biped walk as desired. It is also necessary for all the hardware to communicate effectively in order to have a complete working system.

The system itself is controlled by a Beaglebone Black, which is a low cost open source single board computer. Both input and output signals are handled by this board, as well as any necessary interim computations.

5.1 Beaglebone Black

The BBB is a relatively low-cost micro computer developed by the BeagleBoard foundation[19]. The computer is one of several community developed embedded devices.

5.1.1 Device Tree Overlay

The device tree overlay describes the hardware on the embedded system, in this case the BBB. While your PC might use BIOS to initialize the hardware when booting up, the BBB will use files to describe the hardware[20]. These files make up the device tree overlay and are decided when flashing the BBB. This made for some frustration early on when the PWM-pins were not available. It is possible to change the overlays by configuring /boot/uEnv.txt, however we found that flashing a new image would be more infallible. We have put up a walkthrough of flashing the new image to the BBB at the Git[21].

Changing the device tree overlay can also change what the user have to do to use the hardware. For example in the first overlay every pin you wanted to use as GPIO had to be exported before use, while in the current overlay all pins (apart from ground) are GPIO by default. This overlay is very important when it comes to coding the BBB. The codes makes recipes on how to write to the different pins. Changing the overlay might make it necessary to make changes to the codes. By making a walkthrough on how we flashed the BBB, we can make sure the same overlay is flashed every time.

5.2 IMU's

For the robot to walk reliably it needs to know the position of itself relative to the world. Two inertial measuring units are used to get the angle of each leg relative to the gravitational field/ ground.

The IMU's used in this project are the LSM9DS1. These come equipped with both an accelerometer, which measures the acceleration, as well as a gyrometer which measures the rotational speed. The accelerometer can measure the pull of gravity and can therefor calculate the angle it is at relative to gravity. However the accelerometer is highly susceptible to noise from things like, mechanical vibrations, centripetal force and impacts. We could use a low-pass filter to smooth out the signal, but this would induce too much delay and would not be suitable for our application. The Gyroscope on the other hand is not very susceptible to noise and measures the angle very accurately. However since the gyroscope only measures the change in angle and the angle is worked out from the integral of the gyroscopes readings. Any error will accumulate and cause the reading to drift over time. Rendering the readings useless in a relatively short amount of time.

Since neither of these readings gives an accurate reading individually, we have to modify the measurements to get a better, more accurate estimate of the angle.

5.2.1 Anti drift

The first and simplest modification to the IMU signal is to reduce the amount of drift on the gyroscope measurement. There are two ways of doing this, the first is to simply utilize the built inn high-pass filter to filter out the static drift. However this causes another problem because it makes the gyroscopes measurement always drift towards zero degrees. This means that if the robot is standing in a stationary position with its legs spread apart, the BeagleBone black will eventually think it´s standing with it´s legs straight down. What makes the drift even harder to handle, is that the amount and direction of drift is dynamically dependent on the actual angle of the leg.

The other way is to measure the drift and try to account for it in code. Upon inspection, it was clear that the drift of the gyroscope was quite consistent, one with around a 1 degree/sec and the other with around 0.5 degrees pr second. We could eliminate most of this drift by simply detecting the rate of the drift at startup and subtracting it from every subsequent measurement. This suppresses the drift from around 1 degree/sec down to less than 0.1 degree pr second. However this would still not make a good enough signal since the measurement could potentially be 5 degrees off in just 50 seconds.

In order to stop the drift completely and get an accurate angle from the IMUs we have to combine the measurement from the accelerometer and the gyroscope. This is known as sensor fusion. There are many ways to combine sensor measurements. We will be focusing on two different ways of filtering the signal.

5.2.2 Sensor fusion using PID

The first is a simple PID loop that controls the adjustment to the gyroscopes drift by comparing it to the output of the accelerometer. The D gain from the measurement is practically useless since the noise of the accelerometer data will cause the derivative of the curve to be all over the place. The P gain will be somewhat useful, but any P-gain will essentially pass the noise straight through. The I-gain however is very useful. This is because the output is an average of the previous measurements and any noise will be reduced by the I gain. However this does come with a lot of delay, since this basically functions like a low-pass filter. This is not too big of an issue since the only thing that gets delayed is the correction of the drift, which is already a very slow process.

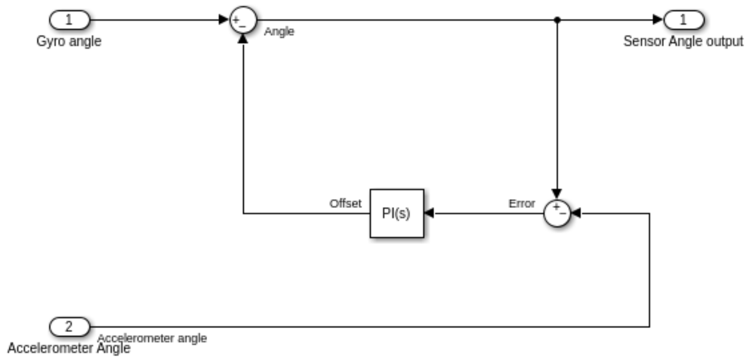


Figure 17: PID Sensor fusion simulink

5.2.3 Sensor fusion using Kalman filtering

The second way is to use a Kalman filter. A Kalman filter uses a series of measurements over time to estimate a probable position of the variable we are trying to measure. Kalman filter works in two stages: Prediction and update. The prediction is a state estimate based on the previous estimated state and the previous measurement.

$$\mathbf{x}_k^- = F\mathbf{x}_{k-1}^+ * B\mathbf{u}_{k-1} \tag{3}$$

We want to measure the angle of the legs and there for let $\mathbf{x} = \theta$. If we approximate that the angular acceleration between measurements is a constant, we can get a formula for the angle in k based on the angle, speed and acceleration in k-1 by taking the double integral of the angular acceleration.

$$\theta(t) = \int\int \alpha \ \mathrm{d}t \ \mathrm{d}t = \int \alpha * t + \omega_0 \ \mathrm{d}t = \frac{1}{2}\alpha * t^2 + \omega_0 * t + \theta_0 \tag{4}$$

If we transform this to a discrete formula we get:

$$\theta_k = \theta_{k-1} + \omega_{k-1} * \Delta t + \frac{1}{2}\alpha * \Delta t^2 \tag{5}$$

Comparing this to formula (1) we can see that $\mathbf{x} = \theta$ and $\mathbf{u} = \omega$ then $F = 1$ and $B = \Delta t$

$$\theta_k^- = \theta_{k-1}^+ * \Delta t \, \tilde{\omega}_{k-1} \tag{6}$$

$\tilde{\omega}$ is the angular speed of the IMU, and the acceleration is handled as noise in the speed measurement. Acceleration is handled as noise because we don't have a sensor that measures the angular acceleration directly. We will discuss ways of prediction the acceleration later in this report.

$$\tilde{\omega} = \omega + \mathbf{e} \tag{7}$$

$\tilde{\omega}$ is the measured angular speed, ω is the actual angular speed and e is the noise of the measurement

The formula for the Error covariance is

$$P_k^- = F\,P_{k-1}^+ F^T + Q \tag{8}$$

We know from the first formula that $F = 1$.

$$P_k^- = P_{k-1}^+ + Q \tag{9}$$

Q is the covariance of the process noise given by the formula

$$Q = F\,\Sigma\,F^T \tag{10}$$

Σ is the mean of the noise of measurement x squared.
Since the Mean of the noise of the measurement is not exactly known and can vary from situation to situation. Q is left as a parameter to be tuned for the specific situation.
The accelerometer is setup to measure the angle to the gravitational field directly.

$$z_k = Hx_k + v_k \tag{11}$$

Where z_k is the measured angle from the accelerometer, x_k is the actual angle and v_k is the noise of the accelerometer measurement. $H = 1$.

Update :
measure residual

$$\tilde{\mathbf{y}}_k = z_k - Hx_k^- \tag{12}$$

kalman gain

$$K_k = P_k^- H^T (R + HP_k^- H^T)^{-1} \tag{13}$$

Here R is the same variable as Q but for the accelerometer instead of the gyroscope. This will also be left as a tuning parameter for the same reasons as Q. We can simplify this equation knowing H = 1:

$$K_k = \frac{P_k^-}{R + P_k^-} \tag{14}$$

Updated state estimate

$$x_k^+ = x_k^- + K_k \tilde{y} \tag{15}$$

Updated error covariance

$$P_k^+ = (I - K_k H) P_k^- \tag{16}$$

Knowing again that H = 1 and that all of the variables are 1x1 matrices we can simplify this down to:

$$P_k^+ = (1 - K_k) P_k^- \tag{17}$$

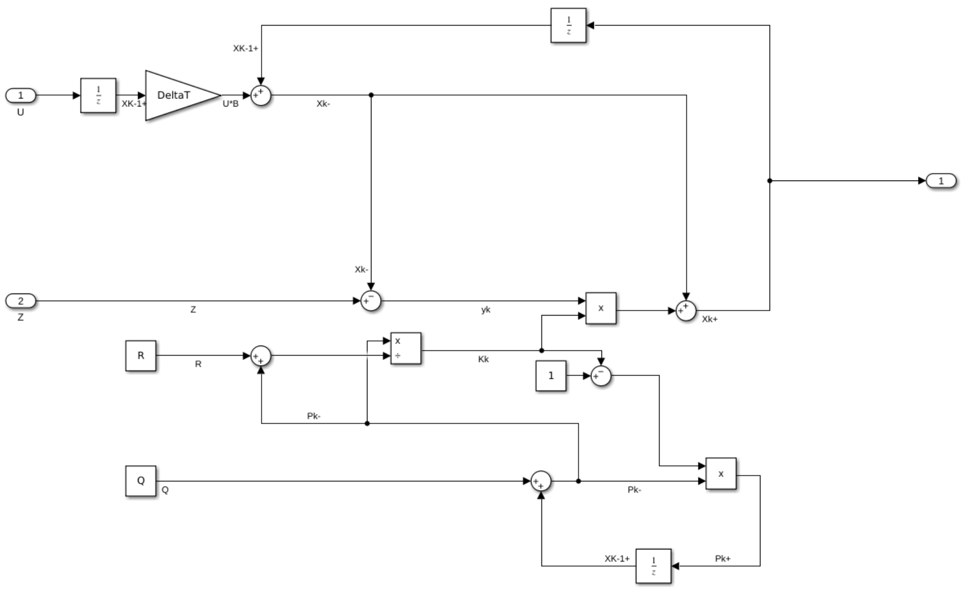


Figure 18: Sensor fusion using Kalman filtering in Simulink

5.2.4 Simulation

The simulation was conducted using two sinus signals. One with a rational period like 1 and the other with an irrational period like $\pi/2$. This made it so that the pattern never repeated itself. In addition we added a custom signal that is meant to simulate a impact on the leg.

Figure 19: Simulink simulations, (Kp = 0, Ki = 1, Kd = 0, Q = 0.01, R = 100)

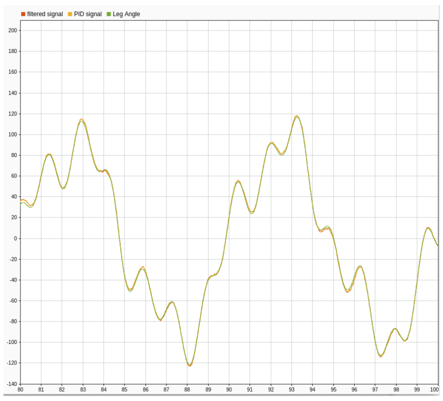


Figure 20: Leg angle, PID, Kalman

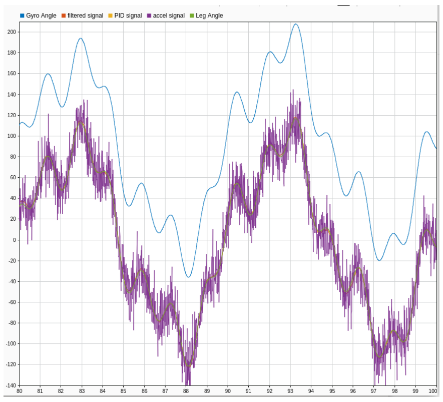


Figure 21: Gyro/acceleration measurement

We can see from the simulated results that the two filtering methods are almost indistinguishable, and both work very well. However this simulation does not factor in things like the centripetal force on the gyro, impact forces on the leg or large vibrations. This might cause the two methods to react differently and further testing on the robot could show a bigger difference between the two filters. The plan was originally to use the encoders to find the real position from a static torso and compare it to the two results from these methods. Since the encoders never worked reliably, we were not able to test them thoroughly.

5.2.5 Results

The best we could do was running the two simultaneously, compare them to each other, and look at the robot to confirm that they were not very far off the real angle.

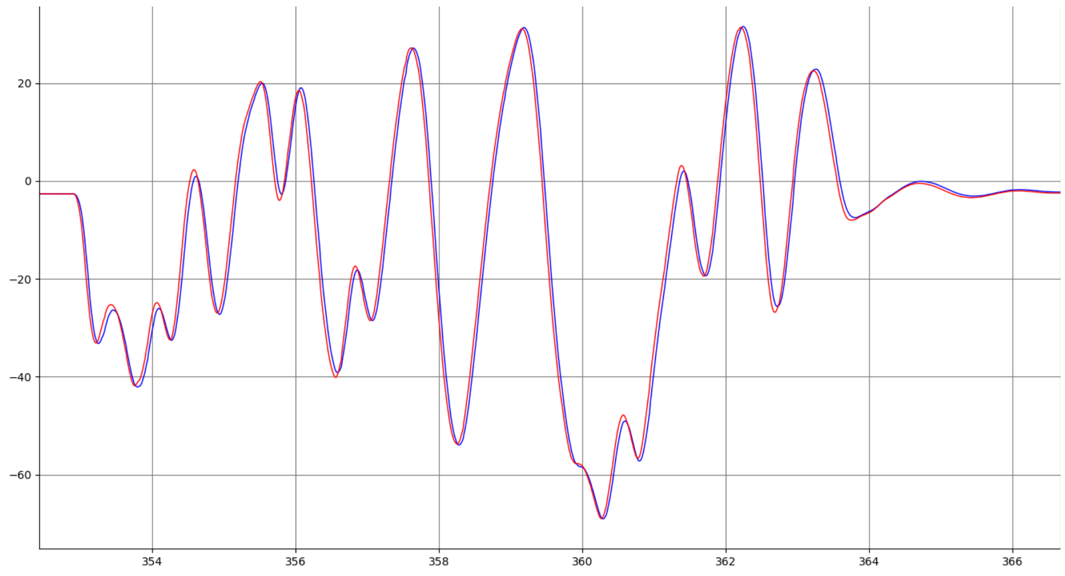


Figure 22: PID(red), Kalman Filter(blue), (Kp = 0, Ki = 1, Kd = 0, Q = 0.01, R = 100)

This was measured while the robot was hanging and the legs were moved back and forth sporadically.
We can see that the two results are quite similar, but not quite as similar as in the simulations. I think this is because the accelerometer has a very noisy signal when moving in this way, which causes the I-gain of the PID to wander slightly off while the leg is swinging. It is hard to know which method is closer to the actual angle without using an encoder.

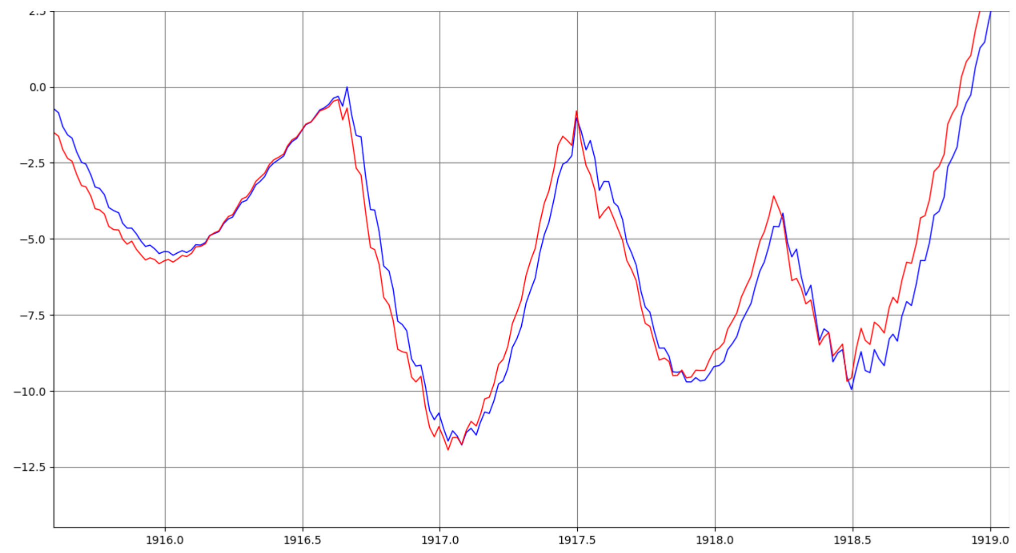


Figure 23: PID(red), Kalman Filter(blue), (Kp = 0, Ki = 1, Kd = 0, Q = 0.01, R = 100)

This was measured while the robot was hanging, by gently kicking the leg of the robot to test the measurements during impacts. The two filters gave quite similar results, although the measurements might seem very noisy. We think this is mostly a result of the mechanical vibrations of the leg itself, not a fault in the measurement or filtering.

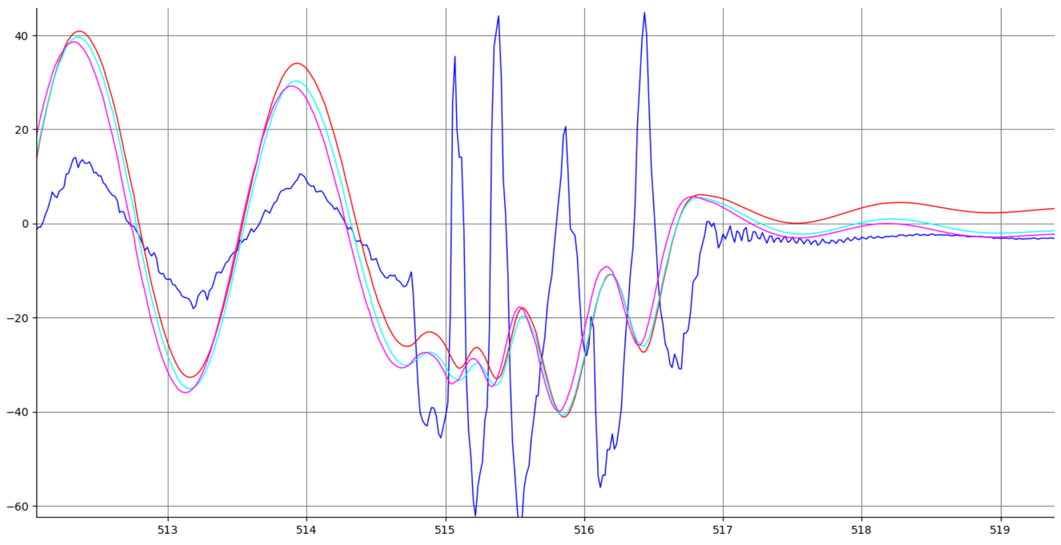


Figure 24: raw accelerometer angle(blue), raw gyrometer angle(red), PID filter angle(pink), Kalman filter angle(turquoise) (Kp = 0, Ki = 1, Kd = 0, Q = 0.01, R = 100)

We can see here that the acceleration measurement does not look like the simulated acceleration measurements. This is because the simulations do not take into account the tangential accelerations and the centripetal force due to the robot hanging. Given how little the gyroscope drifted from the start to the finish of this test, we suspect that the actual value is pretty consistently 7-9 degrees below the raw gyroscope angle. If this assumption is correct we can see that the Kalman and the PID filter was quite far off at certain times. However it is worth considering that this is not taken under normal working conditions and is rather meant as a form of stress testing to see how bad the measurements could get.

5.2.6 IMU code

When the code initiates it configures the IMUs different parameters like filter-bandwidth, power modes, output rate, etc. After this it observes the gyro drift over ten seconds and subtracts the average movement from each subsequent reading. It is very important that the legs does not move during the calibration, as this will make the gyros drift even more than they would have done without the calibration. The code runs 60 times per second, using the Kalman filter to publish the angle of the legs on the leg_ground_angles topic.

Both the Kalman filter and the PID solution are implemented as functions in the code, but only the Kalman filter is called since this is the method that is best suited to implement the improvements suggested in the improvements section.

5.2.7 Improvements

The biggest change needed to improve the leg angle measurements is to modify the Kalman filter to account for accelerations. This can be done in two ways. The first is by knowing what the motors are doing and anticipating the acceleration based on the torque they are asserting on the legs. This method does however require some amount of information from the control-program since the acceleration of the leg depends heavily on whether the leg is on the ground carrying the robot, or swinging forward while the other leg is holding the robot.

The other way is by using the tangential acceleration data from the accelerometer to measure the angular acceleration and implement that into the prediction stage of the Kalman filter. By knowing how far the accelerometer is from the center of rotation, using the linear acceleration in that point, we can calculate the rotational acceleration. This also relies on knowing whether or not the leg is touching the ground since the center of rotation is going to be at the end of the foot when the leg is touching the ground and at the hip joint when the leg is swinging forward.

Another improvement is to try and account for the centripetal acceleration by measuring the angular speed and subtracting the radial acceleration on the radial axis of the accelerometer. Doing this will eliminate the centripetal force´s effect on the accelerometer readings. All of these improvements will improve the Kalman-filters ability to predict the angle of the leg, hence increase the accuracy of the measurement.

The IMU also needs a new housing to hold it securely to the robot and provide support for the cable. This way the BBB will have a stable connection with the IMU.

5.3 Encoder

The biped uses two Scancon-2RMHF encoders, mounted on the backside of the motors.
these encoders have 3000 pulses pr revolution which means 12000 steps per revolution. If the main processor was tasked to keep count of all these pulses it would have to be interrupted thousands of times pr second pr encoder. only to simply add and subtract one from an integer. This i very inefficient and does not make sense. Instead this task is offloaded to a smaller external processor on the BBB board which then updates the processor on the pulse count once the processor requests it. To interface with this chip we use the Quadrature encoder Interface protocol or eQEP for short.

5.3.1 Indexing

In order for the BBB to know the absolute position of the encoder it needs to be able to read the index signal. The index signal is a signal the encoder sends out that pulses once every revolution. This signal lets the encoder know the exact position of the encoder after startup. The BBB does not have an index pin, thus it can not know the exact position after it's turned on. It can only know the position relative to where the encoder was when it started. This means in order for the biped to know where the torso is, it requires a known starting position. This means that every time the encoder code is started the Robot needs to be in a predetermined starting position. This is a huge problem since any error on the initial setup will carry on throughout the run-time of the program.

One option to solve this problem is to install an IMU on the torso of the biped. On startup the robot needs to be perfectly still to make the IMU´s measurements as accurate as possible, and then workout a starting angle from these readings. The problem with this approach is that the encoders will only be as accurate as the initial reading from the IMU´s which is considerably less than that of the encoders.
Another solution is to wire the index pin to one of the IO ports of the BBB and have it start the encoders when it senses that the index pin goes high. The problem with this approach is that the encoders might have moved slightly between the time the index pin goes high and when the eQEP interface starts. In my opinion the second solution sounds better, it requires less setup and does not require any additional parts.

5.3.2 Problems

By far the biggest issue with the encoders are their reliability. Out of the four encoders we had working at the start of this project only one is still functional, and two more was broken before the start of our project. Unfortunately all the encoders broke seemingly when no one was in the lab, thus it is hard to say what caused them to fail.

The first encoder broke while trying to figure out if the encoder code on the BBB was working. The encoder was connected to the small circuit boards on the side of the Robot which connects to the encoder and an Ethernet cable which was then connected to the old BBB shield made by the 2020 bachelor group. This shield used a level shifter to interface with the encoders since the signal from the encoders was 5V and the BBB can only handle 3.3V. When measuring the voltage of the level shifter input where the encoders' signal was wired the voltmeter showed 5V without the encoders connected. This lead us to believe that the encoders had tried to pull down this input on the level shifter which created a 5V difference between the level shifter and the encoder which Fried the encoder.

After designing and producing a new circuit board which used voltage-dividers instead of level shifters. We initially thought we had fixed the problem and installed the two encoders with sockets on the Biped. The encoders were working great for one day. The next day when we arrived at the lab we noticed that the lab bench power supply was drawing around 500mA more than usual. And when probing, the Encoders did not produce a signal. Before we started working on the encoders we asked employees at the institute if they had any similar experiences. It was suggested that encoders with this "ABZ" type connector are very sensitive to being connected to a live connection. However we were aware of this before any of the 3 encoders broke, hence we were very careful not to connect or disconnect the Encoders with the power on, so this was definitely not the cause for any of the three broken encoders.

We have been testing various things on the remaining Encoders to see what might cause them to break. To be able to measure the different signals while the encoder was running we made a custom cable with a connector in the middle to allow us to probe the different signals while the encoder was running.

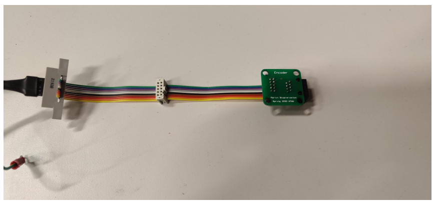


Figure 25: Encoder signal tester cable

We have tried measuring the Power supply voltage on startup to see if there are any spikes, but this does not seem to be the case. We have tried holding the encoders close to the motors while they are running on full force to see if the induced current might have caused issues with the encoders. We have not been able to find anything indication that this is the issue that causes the other encoders to fail. We have also tested to see if there were any ground faults which caused the encoders to touch a live peace of metal somewhere. However we were not able to find anywhere the encoder might have come into contact with that was connected to higher than 5V

One of the encoders have worked throughout the whole project. The only difference between this one and the broken ones is that the broken ones used the small circuit boards made by the 2020 group. We do however not see how these can cause an issue as they are essentially just wires going between the different connectors.

The only abnormality we were able to find on the encoders themselves when testing it on a breadboard was that if we spun the shaft there would be some small spikes on the encoders voltage supply. They were never over 1Volt and should not be able to do damage as the encoders are certified for up to 35V.

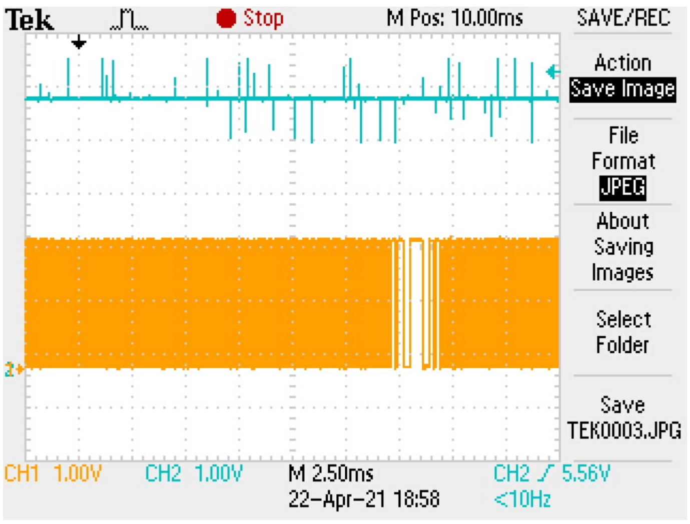


Figure 26: Encoder supply abnormality

5.3.3 Replacement

Since we now only have 1 encoder left, which does not fit on the Biped, we need to find some replacements.We could simply buy the same encoders, but to eliminate the possibility that the encoder models might be faulty we would recommend to try a different option. A great option would be the AEDL series from Broadcom. These encoders come with the same mounting holes as the existing Scancon encoders and can mount straight where the Scancon encoders were. They also have the same connector with the same pin-out and can therefor be connected directly to the existing connector. They come with up to 5000 pulses per revolution which is 66 percent more than the Scancon encoders. Broadcom encoders can be ordered from Digi-key for around 50-60 dollars each. These also have a modular design which lets us just change the code wheel itself if we need a different amount of pulses per revolution or a different shaft diameter to accommodate different motors.

5.3.4 Encoder code

The encoder code is largely unchanged from the one in 2020. We added configuration functionality so that the code will automatically configure the correct pins to eQEP mode and enable the BBB´s eQEP. The encoder code was combined with the motor code to remove the need to communicate through ROS which will remove delay in the feedback loop for the motor controller. The motor node publishes the leg_torso_angle ROS topic.

5.4 Motor

The embedded motor system consists of three parts; the servo controller, the gear and the motor. In the previous reports and documentation there have been found some discrepancies with naming. In the technical documentation and in the 2019 thesis when calculating the power supply[22] the ESCON 50/5 is used as servo controller. The one mounted on the robot is the ESCON 70/10. The differences when calculating the power supply is minimal and to be considered negligible.
In both the 2019 and 2020 thesis the product name for the motor is 14887. This is probably a misspelling from the technical documentations 148877. The part numbers written on the implemented motors are 402890. This number is not mentioned in any previous report, nor the technical documentation. Contacting the producers, Maxon, gave us the answer. The part number 402890 stands for the motor/gearhead combination consisting of:

- DC motor RE40 GB 150W with part number 148877[23].
- Gearhead GP42C with part number 260551[24].

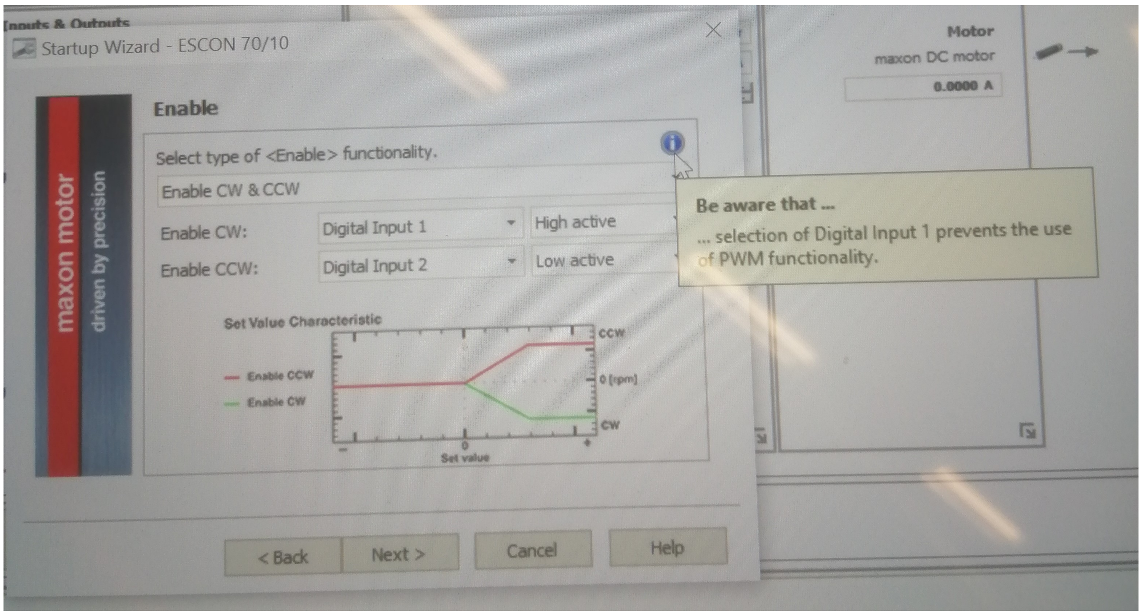
There is also a spare with part number 241318. It consists of the same motor but a different gearhead with part number 203119[25].

5.4.1 Servo controller

When first testing the motor for the inner leg it did not give the power expected. When measuring the motor input with the oscilloscope it showed periodically spikes of voltage instead of a consistent signal. This will make the motor turn on and off really quickly. A consequence of this can be that the brushes of the motor deteriorates, something we believe has happened to the inner leg.

"...the type of operation can significantly deteriorate the brushes if the motor is used for extreme start/stop."[26]

Figure 41 and figure 42 in appendix A shows the connections between the BBB and the servo-controller after the 2019 group and the 2020 group worked on it. You can see that the PWM-signal is connected to the analog input in both instances.
Going through the old configuration there can also be noted that the chosen GPIO inputs prevents the use of PWM functionality, see figure 27. To use the PWM-functionality the PWM-signal has to be connected to Digital I/O 1.


Startup Wizard - ESCON 70/10

Enable

Select type of <Enable> functionality.

Enable CW & CCW

Enable CW: Digital Input 1 High active

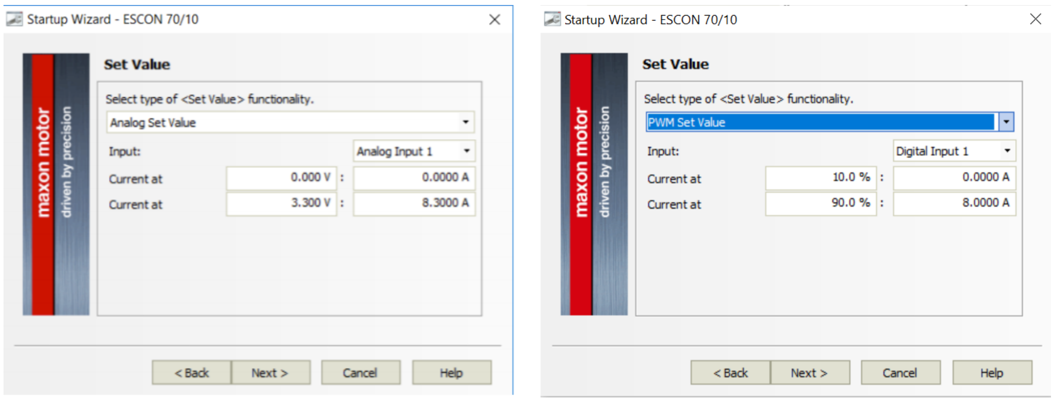
Enable CCW: Digital Input 2 Low active

Be aware that ...

... selection of Digital Input 1 prevents the use of PWM functionality.

Figure 27: The 2019 & 2020 GPIO-inputs.

The "**set value**" part of the configuration decides the mapping of input signal to output signal. In the previous years the input was chosen as voltage, even though the signal from the Beaglebone was PWM. This explains the spikes when measuring the motor input.


Startup Wizard - ESCON 70/10

Set Value

Select type of <Set Value> functionality.

Analog Set Value

Input: Analog Input 1

Current at 0.000 V : 0.0000 A

Current at 3.300 V : 8.3000 A

(a) 2019 & 2020 Configuration

Startup Wizard - ESCON 70/10

Set Value

Select type of <Set Value> functionality.

PWM Set Value

Input: Digital Input 1

Current at 10.0 % : 0.0000 A

Current at 90.0 % : 8.0000 A

(b) 2021 Configuration

Figure 28: Configuration comparison

In the new wiring, figure 43, the GPIO-inputs on the servo-controller are moved to Digital I/O 2 and 3. This way Digital I/O 1 can be used as PWM-input. In the new configuration the input is chosen as PWM. You can see the full servo-controller configuration and wiring in the motor folder at the Github[27].

5.4.2 Motor code

As mentioned in 3.2 the motor code is based on the code developed in 2020[28]. Most of the workload consisted of getting the code to work on the current device tree overlay and to make it work on both motors simultaneously. Another important part was to implement the encoders to enable regulation of the motors.

The purpose of the motor code is to change the direction and the torque of both motors. To control the direction you change the "value" of the GPIO-pins and to manipulate the torque you change the "duty_cycle" of the pwm-pins.

As we currently have no working encoders to be mounted on the motors, the inputs are decided by two sliders on the rqt interface. These can be replaced by encoders without making any big changes to the motor code. In the absence of encoders this system will only function as an open loop. You can see the flowchart for the current motor code in figure 44 in appendix A.

The motor code can be found in the git-folder /Motor/encoder_test/src/ [29]. The file is located here because the folder *encoder_test* contains *CMakeLists.txt* and *package.txt*, both are needed to run the ROS-package. This way the user can upload the whole encoder_test-folder and run the motor code without setting up a new ROS environment. Another motor code can also be found in the src-folder. This code contains a simple PD-controller, but does not contain the slider inputs.

5.4.3 Problems

When it comes to the motor there have been a few errors from previous. Last years PCB had the motor current running through it, see chapter 4.2, the PWM-signal was fed into the analog input and the servo controller configuration was wrong. A combination of the two last errors might have damaged the motor for the inner leg, as this leg currently has a lot less power than the outer leg. The reason why the outer leg is not damaged is probably because none of the previous groups tested it. According to Torleif Anstensrud, the client, the only leg he ran tests on was the inner leg and even then it could only reach about 15°. Accordingly we can probably determine that the motor has been damaged for quite some time.

Then the question arises, does the robot need a new motor or are the current capabilities good enough?
The leg is at most capable of reaching 15° on its own. With the pendulum momentum it can reach a bit farther. From the simulation of the robot's gait we can see that 15° should be enough. A mp4-file of the simulation has been uploaded to the frontpage of the git[21]. However the leg can only reach 15° holding its own weight. Adding the weight of the rest of the robot when the leg is in the foremost position will require a lot more force.
There is also the downside of the two motors acting differently making regulation a lot more complex.

Ending this subchapter on a positive note; the motor code functions the way it is intended. It can both fetch data from ROS and control the motors.

5.5 Servo

5.5.1 General

In order to move the actuators at the end of the four legs Turnigy TGY-SC340V 7.4 volt servos were used. These servos are controlled by the duty cycle of a PWM signal where a higher duty cycle signals a bigger rotation.

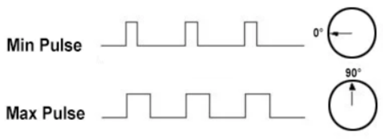


Figure 29: illustration of using PWM to control servo (Note: the duty cycle is not based on real values)

As the TGY-SC340V have gone out of production we were unable to find any data sheet, as a result the information we had to work with was limited. Normally a DC motor actuates the servo itself and a potentiometer is used to designate the position of the servos and a control circuit that controls the motor to get it to the desired position. The control circuit also limits the movement to a predetermined scope, on the TGY-SC340V this limit is 0°-90°.

As the only actuation needed is fully extended and retracted our code only needs to switch between two PWM signals instead of the full range of motion.

5.5.2 Servo code

As we only need the toes to be fully retracted or extended the servo code is developed to simply use a Boolean value as input for each pair of servos. The code, then sets up the necessary parameters and definitions to change the value of the two PWM signals to the corresponding action.

For servos such as these the code needs to be written with the parameters that hobby servos usually work with. The frequency these servos operate after is 50Hz, which is equivalent to a period of 2000000 nanoseconds. Using a PWM signal means the code has to be based on this period. It also means that the maximum period we can operate the servos in is a little bit less than this due to physical limitations.

For our purposes two pins are used as the output for the PWM signals. These pins are the BBB's P8_13 and P8_19. As such the code first sets up and defines these pins as the ones to operate with. Configuring theses pins also include changing the selected pins into PWM mode. After this the chosen pins are initialized.

In order to actually change the positions of the servo the duty cycle of the PWM signal is changed. The previous group that worked on the servos found two duty cycles that seemingly corresponds to fully retracted and fully extended actuators. Fully retracted corresponds to the minimum pulse with of 820000ns, while fully extended requires setting the pulse width to 1900000ns.[30] Testing the code also seemed to show this range to be close to optimal.

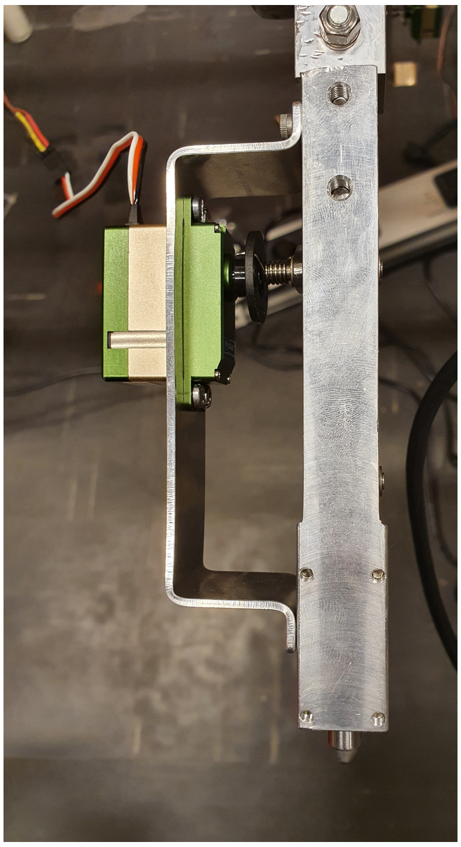


Figure 30: Servo pos "1"(Duty cycle 820000ns)

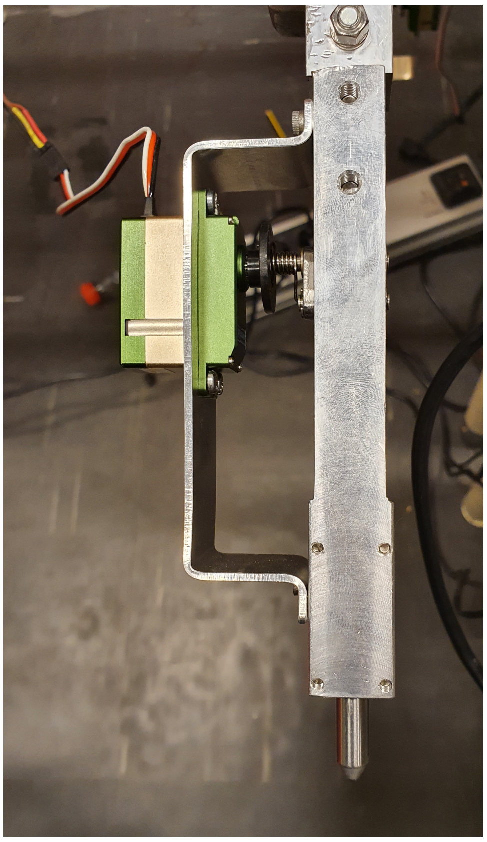


Figure 31: Servo pos "0"(Duty cycle 1900000ns)

Currently we use rqt to decide the input variable as we have no way to decide the gait as of yet. Once the walking algorithms is implemented this can be used to add this feature. The valuer is then used to change the ducy cycle. A value of "0" corresponds to a duty cycle of 1900000ns, if the value is "1" it is set to 820000ns.

5.5.3 Challenges

A lot of the work on the code went to trying to make it function as it should have. Troubleshooting the servos took a lot more time than expected. Much of this time was spent trying to modify the code because it was believed to be a software issue. At some point new code was written in order to try and make the servos work, however this was in the end scrapped.

We eventually got them to work with a simple Python code mainly to make sure the servos would function as expected. However when using the C++ code on the other hand, it took a long time in order for the servos to function properly. Eventually the problem was solved without requiring too much change to the code itself.

We also discovered later on that one of the servos didn't work properly, pulling a large current while spinning slowly. We are unsure what can have caused this and how it has affected our testing, as the servo were still able to spin slowly. This meant that it was not apparent that it was a hardware problem at first glance and that the code might be the problem instead. It wasn't until we tested several at the same time that we realised one was broken.

6 ROS- Robotic Operating System

6.1 What is ROS?

Not to be fooled by the name, Robot Operating System (ROS) is not an operating system, but rather a framework for development of robotics. ROS distinguish itself from most other robot software by being open source and encouraging collaboration; this makes it ideal for education and research which has resulted in a quickly growing user base. This community is key to ROS's success and it has resulted in a massive platform where you can find packages that can handle almost every task[4].

6.1.1 How it works

In a ROS system every process is called a node and each node handles one task whether this is reading sensor values or controlling a servo. Each node is independent from the others. These nodes communicate by using a publish/subscribe model called ROS topics. This means that a particular node will publish any information it wants to output through a ROS topic, other nodes can then choose to subscribe to this topic if its relevant for the process.

With all this information being sent around ROS needs a way to manage the information flow, this is done by ROS master. ROS master provides a service that helps the different nodes locate each other, once this is done they use peer-to-peer communication. The ROS master runs on the main computer, however you can connect other machines communicating through the same core.

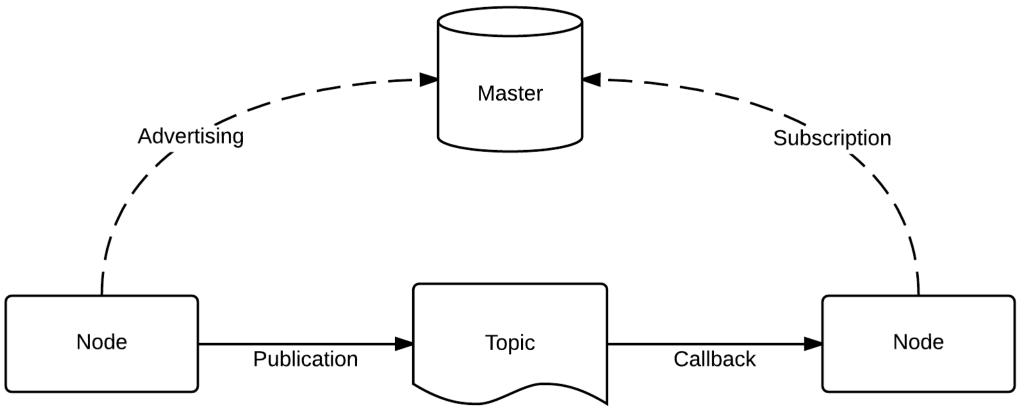


Figure 32: ROS communication

6.2 Why use ROS for this project

There are several advantages of using ROS for this project, the main reasons are:

- **Modularity:** The modular nature of ROS is ideal for projects like this where components might be changed or added later on as the different nodes are independent of each other. This also means that you can use different programming languages for different nodes without problem.
- **Community:** As mentioned before the platform has a massive community meaning you can find packages that will take care of almost any task. The "ROS answers" forum is also a very helpful tool.

- **Visualization:** One of the main priorities for this project is to create a system for logging and real-time visualisation of the robot and ROS has great tools for this in the form of Rviz, rqt and the rosbag packages. Gazebo can also be utilized if simulation of the robot is needed.

6.3 Choosing ROS distribution

Early on it was planned to make the switch from ROS to ROS 2 as it was believed that it would be beneficial in the long run. This however turned out to be challenging as ROS 2 runs on 64 bit while the BeagleBone Black is using a 32 bit processor of the ARMv7-A/armhf architecture. Some sources claimed that using a BeagleBone AI could work, but this was not the case during our testing and we decided to stick to the BeagleBone Black running classic ROS.

The armhf architecture forced some decisions on the software side of the project. We ended up using ROS melodic with Ubuntu 18.04 as our Linux distribution, several factors led to this choice. Ideally we would use Debian as our Linux distribution as it's most commonly used with software development. However the latest versions of ROS is not supported on Debian running on armhf chips. This led the decision to use Ubuntu as our Linux distribution. The choice of ROS distribution was between Melodic (Released 2018 and supported until 2023) and Noetic (Released 2020 and supported until 2025). Melodic ended up being the obvious choice as Noetic was recommended with Ubuntu 20.04 which has no 32 bit image to upload to the BeagleBone. We considered looking for an alternative way to use Noetic, but decided against it as the main trade off of using Melodic is that it doesn't support python 3. However as the system is mainly written in C and C++ this shouldn't cause any problems.

6.4 Packages

To keep all the code organized we chose to develop most of code for the different parts in separate packages. This makes it easy to keep track, as well as to develop the different aspects independently. The exception to this is the encoder code. This has been merged with the motor code to keep possible latency to a minimum.

6.4.1 List of packages:

- **Launch:** This package handles the different files related to launching the system quickly.
- **Messages:** Description of the different messages being sent around. Having this in a separate package makes it very easy to transfer to another system.
- **Servo:** Handles the actuation of the toes via servos. Currently the only function is to extend and retract the toes as an algorithm to decide what toe to retract is still needed to facilitate the robots walking ability completely.
- **Motor:** Interprets the encoder signals and controls the actuation of the motors to adjust the leg angle to the desired position.
- **IMU:** Reads the IMUs sensor values and combines these using sensor fusion by either PID regulation or a Kalman filter.
- **Robot description:** This package combines information from the different sensors to define the robots state.
- **Messenger:** Handles communication between the computer and BeagleBone.

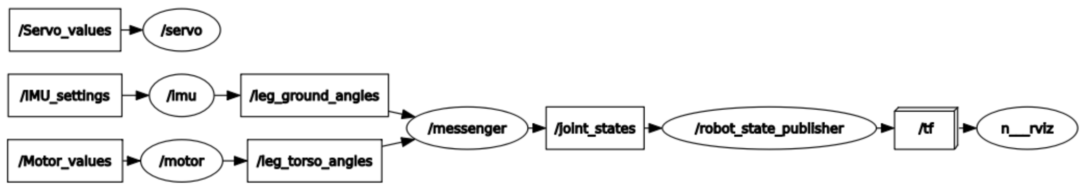


Figure 33: Map of nodes and topics using rqt_graph

Figure 33 shows the communication on the current setup and is made using the rqt_graph package. This feature maps all the running nodes and topics. This allows to show the path from the different inputs to the final visualization in Rviz, however it somehow misses the node that handles user input from the computer. This is shown below using the power of paint.

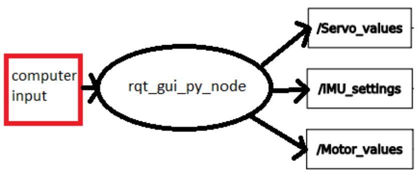


Figure 34: Python node for user input

The rqt_gui_py_node node is used to enable testing from the computer and to change values while the code is running instead of having to recompile.

Once the robot is fully functional this node structure will change as a control algorithm will handle the actuation. That is why we created the control package. As of now this package doesn't do anything, but is configured to take the leg-to-ground and leg-to-torso angles as inputs and output servo- and motor values. Once a control algorithm is ready to be tested it can be easily implemented. Below is a diagram that shows information flow when it is.

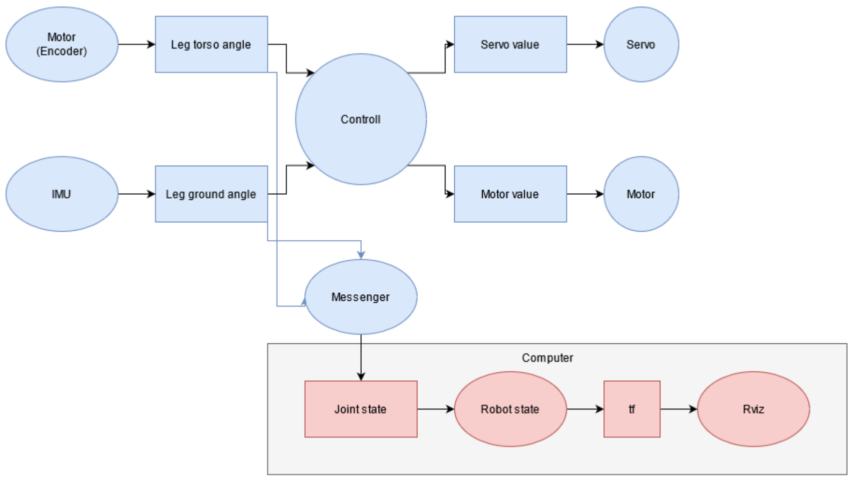


Figure 35: Information flow with control node implemented

The node allowing for user input can of course also be included when the control algorithm has been implemented, but this shouldn't be necessary.

6.5 Launching the system

Usually to start a node you use the rosrun command. However as the amount of nodes increase this process can be confusing as well as time consuming. To tackle this problem we decided to automate as much of the launch procedure as possible, this was done using the roslaunch package as well as bash scripts. The roslaunch package allows you to make files that can launch multiple nodes with a single command.

To run the visualization software it is necessary to run nodes on the computer as well as the BeagleBone, because of this two sparate launch files are needed. In addition we chose to make a separate launch file for Rviz in case someone wants to launch this separately, but this is of no concern if you launch the main computer launch file as this will call the Rviz launch. The launch sequence for the files is described in the flow charts below:

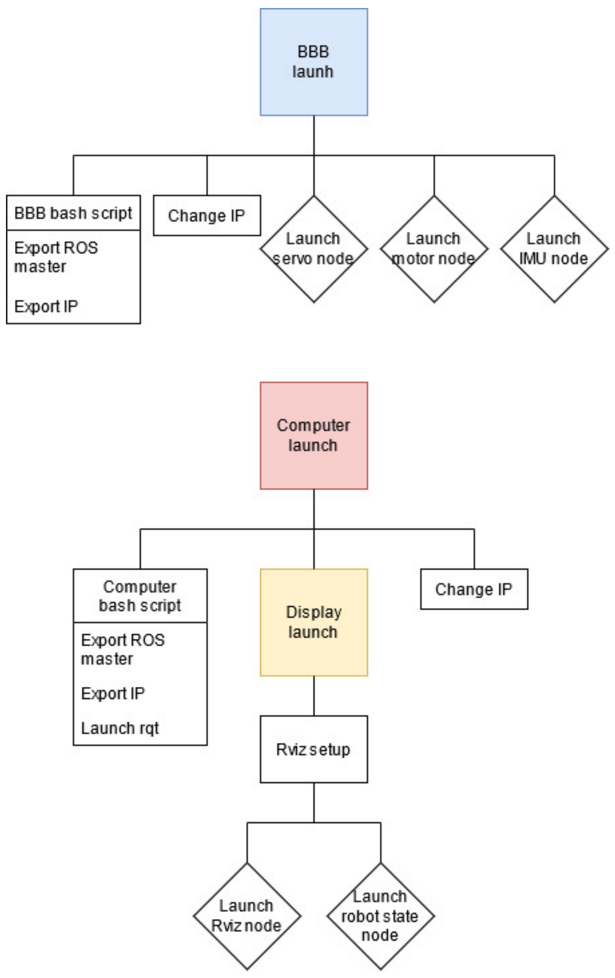


Figure 36: Launch sequence

Even with these launch files there are still some manual commands that are required, this include sourcing setup files for ROS and the catkin workspace as well as making sure the ROS core runs on the BeagleBone and not the computer. This is described in more detail on the GitHub.

6.6 Challenges

ROS has a pretty steep learning curve, and with no prior experience we knew that this would be hard. The good thing is that once you start to get the hang of it familiarizing with new packages and features becomes increasingly easy.

Other than this the only reoccurring problem we had regarding ROS was a lot of debugging when creating packages. Mainly the challenge was to configure the "CMakeList.txt" and "package.xml" files the right way.

7 Real time logging and visualization

After the dSpace system was abandoned in 2020 due to a ransomware infecting the computer that ran dSpace the robot lost all its visualization capabilities. This was a big hit for the project as a good GUI and real time visualization features makes testing and troubleshooting much easier, hence this became an important part of our project.

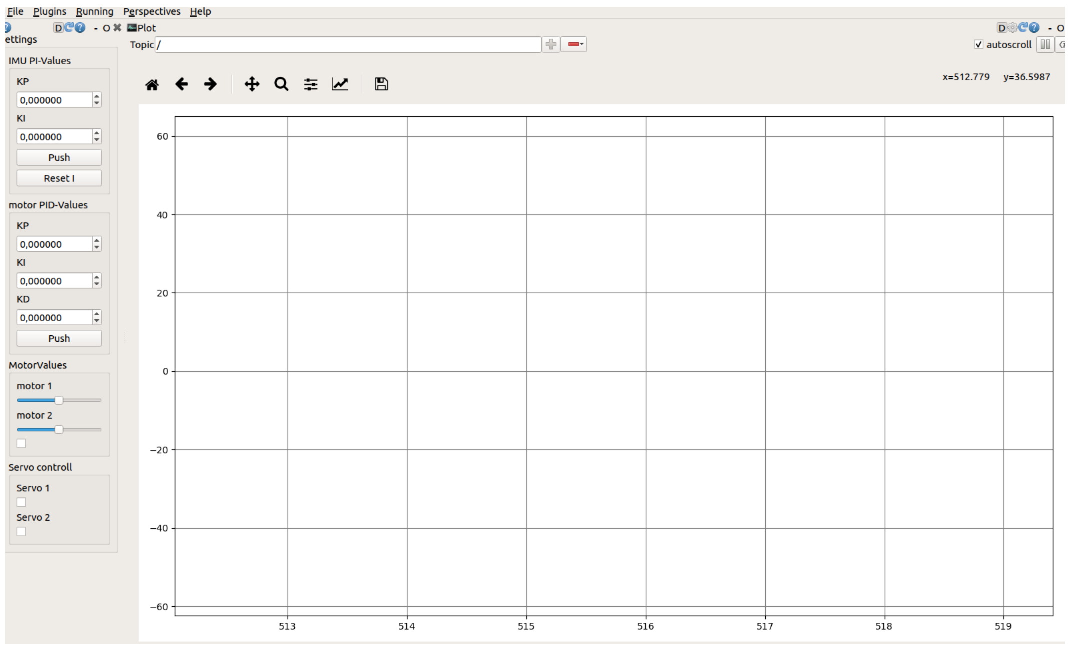
7.1 Rqt

Rqt is a built in framework in ROS that makes it easy to implement various GUI tools as plugins. One of the reasons why rqt is so powerful is its ability to interact with every node and topic in the system, this makes it easy to observe individual parts of the robot, we mainly used it to graph the encoder position.

In addition to graphing values, we used rqt for most of our GUI. This was done by creating a custom rqt package that allows you publish messages to the robot. This allows the user input of different values that includes:

- **IMU:** Values for sensor fusion for both PID- and Kalman values depending on which one is in use.
- **Motor:** The motor controls are implemented as sliders.
- **Servo:** Both sets of servos can easily be extended or retracted.

Having this control panel makes it easy to test different component as well as changing values without having to recompile which can be tedious, especially on the BeagleBone. The package was programmed in python as we found it hard to implement it using C++. We decided on this solution as the code runs on the computer and not the BeagleBone so any speed related issues is negligible.



IMU PI-Values
KP 0,000000
KI 0,000000
Push
Reset I

motor PID-Values
KP 0,000000
KI 0,000000
KD 0,000000
Push

MotorValues
motor 1
motor 2

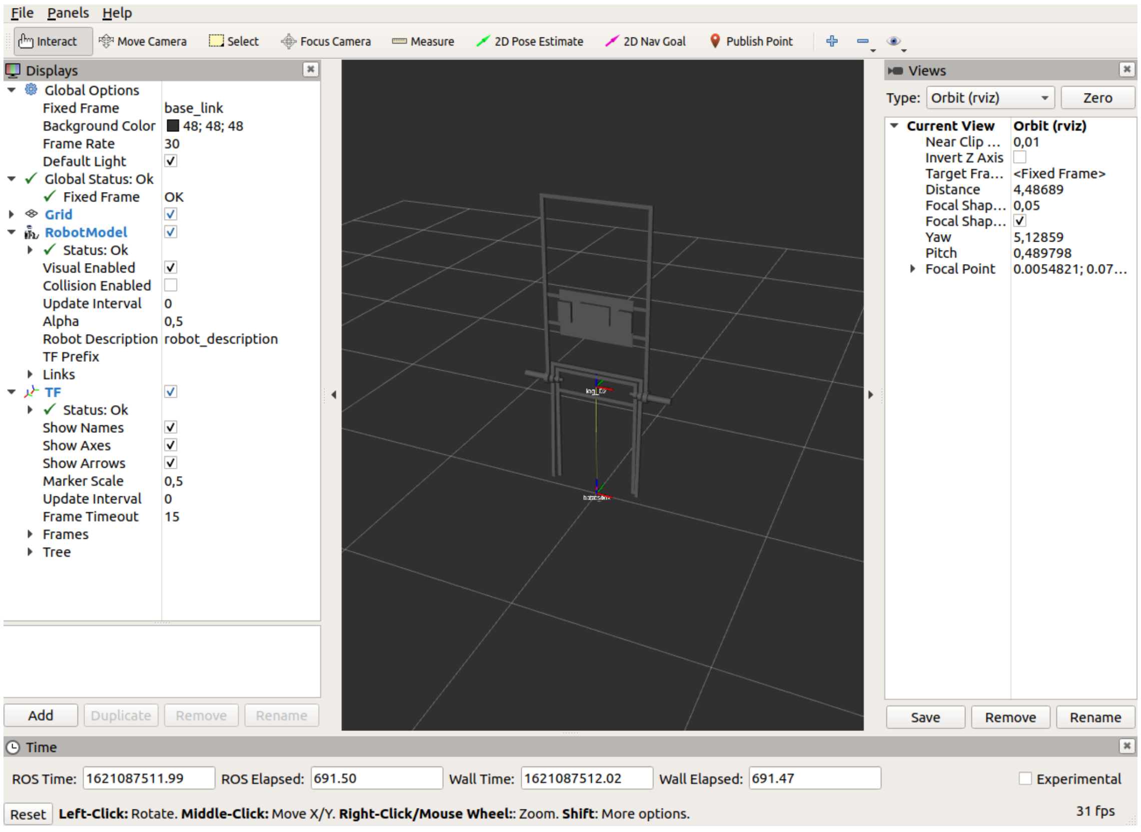
Servo controll
Servo 1
Servo 2

Figure 37: rqt with GUI and encoder graph

7.2 Rviz

Rviz is a built in 3D visualization tool for ROS. Rviz allows for powerful 3d rendering in real time which can provide important insight into how the robot perceive itself and its surroundings.

The robot model used are based on three STL files made in fusion360 to describe the different parts of the robot. These are linked through 2 joints allowing dynamic movement. The robot publishes messages to update this 3D-model in real time. This is useful because it tells you where the robot thinks it is which can help us track down any errors in the measurements.



Global Options	
Fixed Frame	base_link
Background Color	■ 48; 48; 48
Frame Rate	30
Default Light	✓
✓ Global Status: Ok	
✓ Fixed Frame	OK
Grid	✓
RobotModel	✓
✓ Status: Ok	
Visual Enabled	✓
Collision Enabled	
Update Interval	0
Alpha	0,5
Robot Description	robot_description
TF Prefix	
Links	
TF	✓
✓ Status: Ok	
Show Names	✓
Show Axes	✓
Show Arrows	✓
Marker Scale	0,5
Update Interval	0
Frame Timeout	15
Frames	
Tree	

Type: Orbit (rviz)

Current View	Orbit (rviz)
Near Clip ...	0,01
Invert Z Axis	
Target Fra...	<Fixed Frame>
Distance	4,48689
Focal Shap...	0,05
Focal Shap...	✓
Yaw	5,12859
Pitch	0,489798
Focal Point	0.0054821; 0.07...

ROS Time: 1621087511.99 ROS Elapsed: 691.50 Wall Time: 1621087512.02 Wall Elapsed: 691.47 Experimental

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:**: Zoom. **Shift**: More options. 31 fps

Figure 38: Rviz with piped model

Currently the robot are only using the most basic features of Rviz and simply simulating the robot state without considering it's environment. This could however be changed in the future, but would require more data input regarding the robots surroundings.

7.2.1 Gazebo

Gazebo is an open source robot simulation program that could be implemented in the future. Gazebo differentiates from Rviz in several ways, but most of all it is ideal if you don't have a physical model to work with. It would also allow you to simulate the gait algorithms. Using it for this robot however would require a better model of the robot for the physics simulation as the current one is not physically accurate. Instead it is merely a tool for visualizing its perceived state.

7.3 Rosbag

Rosbag is a package that allows the user to record and replay data from topics. The data that is being recorded is stored in bag files, this in turn can then be easily replayed or analysed. Currently we have only done simple testing using the command line tools to run rosbag which is extremely simple. There is also C++- and Python API available that could be implemented[4].

Rqt also has very good integration with rosbag using the rqt_bag package. This package includes a GUI for controlling the recording as well as different replay settings.

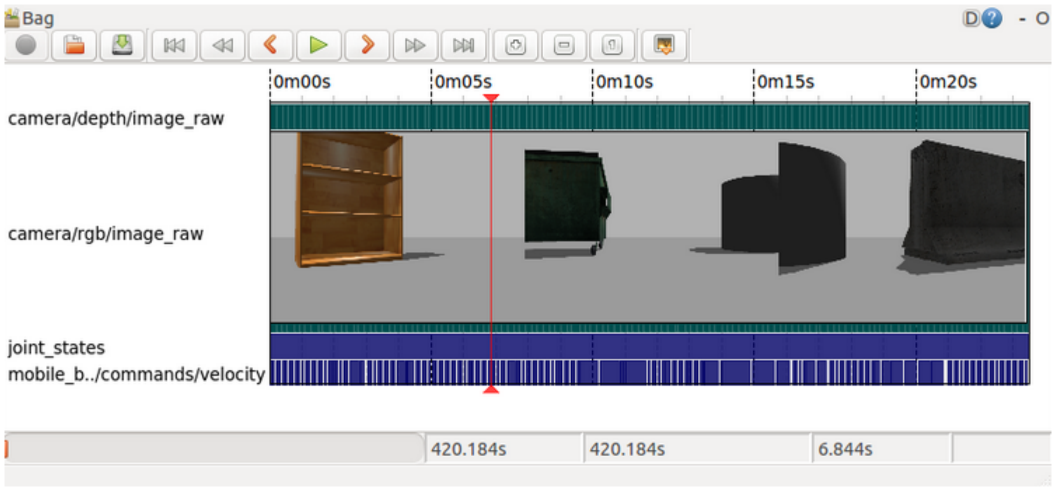


Figure 39: rqt_bag GUI example from ROS wiki

7.4 Challenges

The main challenge was to create the custom python package as we found very little documentation regarding this. As a result a lot of trying and error were needed until we finally got it to work. Setting up Rviz also proved time consuming causing us to create a launch file as mentioned in 6.5.

8 Discussion

8.1 Project management

Early on, we realised that many theoretical prerequisites needed to be met before working with the robot itself. The group members had little to none experience using C++, BeagleBone Black or ROS. This meant that a lot of time early on was spent researching these subjects. Our previous experience was also quite monotone as everyone of us has studied automation. Having experience from electronics or instrumentation would possibly have been more relevant.

As the testing done last year was very limited due to Covid restrictions, it was unclear how much of the robot was in working order. Since we worked mainly from home and had little academic diversity, we decided to split the tasks more or less at random to then merge them later on. In hindsight more collaboration early on would probably have helped in the debugging process, especially of the encoders.

We think experiencing first hand how frustrating it can be to continue badly documented work was a good eye opener early on, and it's something we have been very aware when creating the GitHub and code.

8.1.1 Achievement of early goals

The final goal when starting was to see the robot stand on its own and using a PD regulator to take a few simple steps. It did however become apparent that this was unrealistic as new challenges that we hadn't anticipated became apparent. Discarding this goal, we instead chose to focus on finishing as many of the individual modules as possible. Throughout this process we had a good dialogue with the client about what areas to prioritize and we came to the conclusion that visualization and project organization that would make testing easier would be valuable.

8.2 Hardware

Most of the hardware design was completed in earlier stages of the project, the hardware related challenges were mostly unforeseen problems that was discovered along the way with the exception of the servo brackets.

The biggest unforeseen challenge was the design of a new Circuit board. That debugging would be necessary for the sensors and actuators was expected, but the PCB was believed to have been fixed prior to us starting the project. This proved to have great educational value as we had no prior experience with this. As we needed the board to test the encoders this was done as fast as possible, and hence some minor problems were discovered subsequently. If we had a longer deadline we might have been able to eliminate some of these.

Some hardware malfunctions have been discovered on the motor, servos and encoders. We decided not to order any new parts as these malfunctions did not impact our code development which was our main focus, instead we have documented all our observations to prevent a repeat of this.

8.3 Software

Developing reliable software for different aspects of the robot was the main goal for the project.

8.3.1 Programming

As mentioned in 8.1 we had no experience working in C++. As a result the development was time consuming and had a slow start. However it also resulted in a steep learning curve and development became much more streamlined towards the end of the project. A big focus has been to make the different aspects of the project as modular as possible making future development easier.

As we had more experience using Python, we would probably have been able to do more using this language. We do however believe that the increased speed and reliability that comes with using C++ makes it worthwhile. Learning to develop in an unfamiliar language also has value in itself.

8.3.2 ROS

Using ROS to control the robot seems to be by far the best option. It allows for a very high degree of flexibility and makes expansion very easy. The different parts of the robot can easily be worked on independently and later be set up to communicate with each other. If it becomes necessary to add more components or functionality in the future this can easily be achieved. As much of the groundwork of making packages now is done future development should be fairly easy as well.

Using ROS as our middleware does however mean that for future work the code has to be developed with ROS functionality in mind, and switching to another system would require modifying all the existing code. Another consequence is that all the visualization its based on Rviz and rqt. While all the other code probably could be modified to work on another middleware, all visualization and logging would have to be developed from scratch.

8.4 GIT

One of the goals of this project is to create a well-organized GitHub repository to be used for further development of the robot. As this is a project that builds on the work of several bachelor groups the importance of getting everything well documented can't be stressed enough. It is also important that the explanations are aimed at the right target group, in this instance other electrical engineering students. This is something that has lacked previous years.

We started with uploading the previous documentation and organized it to the best of our ability, focusing especially on making the most relevant information such as data sheets easily accessible. In some instances, we were unable to find desired information (e.g. data sheets for the TGY-SC340V servos who is no longer in production). In these cases, all available information were gathered in a document and saved to the Git.

In addition to the documentation we have also made README-files for most folders. These gives explanations of the contents as well as other relevant information. The GitHub also contains a thorough tutorial on getting started and setting up the BBB and other modules for this project.

We believe that most students should be able to understand and launch the entire system using only the resources on the Git and this report.

9 Conclusion

The primary goal of this thesis is simply to further facilitate the robots walking ability. As the extent of previous work was partly unknown, the road was not clear from the start, but we knew that implementing upper body measurement using IMU and real time visualization would be important. This have been implemented with some restricting factors becoming apparent along the way.

Code for actuation of both servos and motors have been completed, and the system is fully connected using ROS as middleware. A custom GUI for control of the different actuators and other relevant values have been implemented using rqt. Using Rviz a model for real time visualization of the robots perceived state has also been implemented. These features serves as a very good basis for further testing. To allow visualization, tracking of the torso angle using IMU's were needed. We experimented with sensor fusion using Kalman filtering as well as PID regulation with slightly different results, most noticeable when a shock is applied. More testing is required to determine if these results are satisfactory.

Some problems occurred along the way which is to be expected from a project of this size. As the reason for the encoders breaking have not been unveiled, these have yet to be fully implemented even though the code is ready. No encoders also means that a feedback loop can not be fully implemented for motor control. Our goals also deviated slightly from the original plan to include designing a new PCB.

Keeping the project well organized and documented has been an important area of focus as its crucial to keep up productivity. This becomes especially important when different groups work on the same project. To make the transition as easy as possible a GitHub was established with a lot if extra documentation that should provide a very good basis for getting familiar with the work.

Although the robot is not completely ready to walk it has gotten a lot closer with hardware being the main restrictor at this point. We hope and believe that a strong basis for reaching the end goal has been made.

10 Future work

This is an overview of work that needs to be done in order to start testing the gait algorithms it is built for.

10.1 Replacing servos

When connecting all 4 servos we realised that one was broken, we are unsure of what caused this as none of the servos have been exposed to voltage that should have been able to fry it, it might even have been faulty when we started the project. This also means that there was a period where we worked without knowing only three servos were functioning, which could possibly have impacted troubleshooting if the faulty one was used.

After discovering this problem our first thought was to order a replacement, but this proved problematic as we were unable to find any store that sells them. Only one Asian marketplace that might auction them off, but we are unsure of the legitimacy of this site as it contains next to no information. In addition to this we discovered some disadvantages with the current servos.

- The servo is only able to rotate 90° which is problematic as the toe is actuated via a spring that has some backlash. This is only a problem when the toe is extended as it can collapse with pressure. Our solution to this was to put the spring in tension when mounting the servo, but this means the spring will have to be readjusted from time to time. Having servos that can rotate a few degrees more could solve this issue by keeping the spring in tension.
- The servos are made for 6-7,4V. With the BeagleBone and encoders running on 5V and the power supply only having two outlets (the other being used by the motors) you have to use unnecessary electronics simply to get the right voltage, hence having the servos that are certified for 5V would be a lot easier.

After consulting the client it was decided that ordering 4 new servos probably is the best course of action. Our suggestion for new ones are the DSSERVO DS3225. This servo is certified from 4.8-6.8V and can be actuated up to 180° We also believe it would fit in the servo mounts that was made for the Futaba s9254 (Used until 2019) as they have similar dimensions. The only downside is that it has an operational speed of 0.15 sec/60° whereas the Futaba had 0.12/60° We assume this will not affect the functionality, but it should be investigated before a final decision is made.

As the two previous sets of servos have gone out of production we would recommend waiting until necessary to order new ones to prevent this from happening again. We would also recommend to order some in spare in case any more gets destroyed.

10.2 Developing algorithm to extend and retract toes

As meantioned in the toe needs to retract for the leg to be able to swing. Therefore its necessary to develop an algorithm that understands when to extend and retract the servos as the robot is walking. Our thought was to retract the servo of the leg that is being moved forward and extend once it has been moved sufficiently, however we did not have time to expand upon this idea.

10.3 PD-controller

A PD-controller should be implemented and tuned. With currently lack of sufficient power from the motors and that the leg motions relays on the pendulum motion, the integration part is likely redundant. There are two different PD-controllers on the Git. One implemented in a motor-code and one as a standalone ROS-node.

10.4 Feed forward

When all the wires and servos are implemented there should be conducted friction tests and implemented a simple feed forward control to the system. In figure 40 you can see a model of the system with implemented Coulomb Feedforward[31].

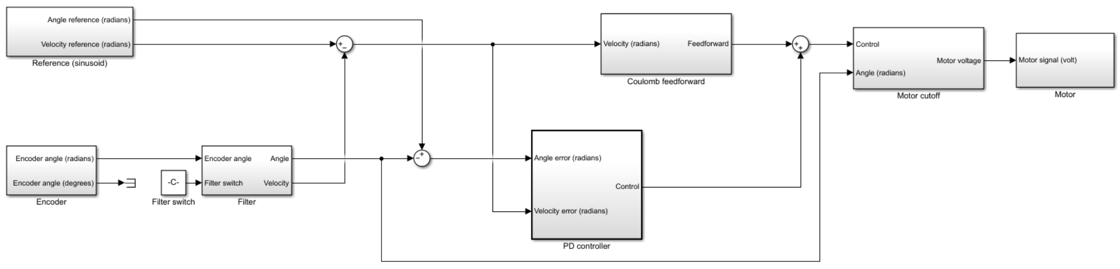


Figure 40: Simulink model by Torleif Anstensrud

10.5 Motor

A decision should be made of whether or not buying a new motor is necessary.

Considering the reserve motor; we can gather from its technical data that it can only handle half the radial load compared to the implemented motor (150 N vs 300 N). Adjustments to the encoder code must be made if the choice to use another gear ratio is ever made. Two different motors will also make regulation a lot more complicated.

With no previous experience of damaged motors we would probably opt for buying a new of the same kind. Buying a new of the same type from Maxongroup, as of writing, will cost just under 600€ and another 40€ for shipping. This is excluding value added tax (mva).

10.6 Further improvements to robot state measurement

To be able to test the different gaits on the robot the estimation of the robot state at any given time needs to be incredibly precise. To do this more information is key, accounting for shock and other disturbances will be important. More advanced sensor fusion algorithm and more sensors like pressure sensors beneath the legs could help with this.

10.7 Improve IMU readings

The thought of having one IMU on each leg was that the anchored leg(the one touching the ground) would provide the most accurate reading. As the robot is not yet ready to walk we have not had a chance to test this, but if this is the desired strategy then an algorithm to chose what leg to read from has to be implemented. This could possibly use input from the code to chose what servo to extend. In addition to this a new case is required to properly secure the IMU to the leg.

Another possibility is to use multiple IMU's at the same time e.g mounting one on the torso that would be less susceptible to the impact from the legs.

IMU improvements is expanded upon further in 5.2.7.

10.8 Debug encoders

We were not able to identify what caused the encoders to break and not having observed them when the problem was caused makes debugging hard. Finding the root of this is important before ordering new ones. Alternatively one would have to order different ones in hopes that these would not have this problem.

References

[1] Stian Johan Olsen Kristoffer Meggelæ Pedersen Lars-Erik Nes Panengstuen, Edvard Merkesvik. *Background*, April.2021. Page iv.

[2] Henrik Baldishol Jakob Karlsen Kristoffer Nordvik Jonas Hjulstad, Endre T. Ellingsen. *Instrumentation, actuation and model identification of bipedal robot*, 2019. Page 1.

[3] Jonas Brunsvik Martin Skårerverket Irfan Ljevo. *Preliminary parts list*, 2020. Page 2.

[4] *ROS.org*, 2020. http://wiki.ros.org/Documentation,accessed16.05.2021.

[5] Christian Fredrik Sætre. *Stable Gaits for an Underactuated Compass Biped Robot with a Torso*, 2016.

[6] Henrik Baldishol Jakob Karlsen Kristoffer Nordvik Jonas Hjulstad, Endre T. Ellingsen. *Instrumentation, actuation and model identification of bipedal robot*, 2019.

[7] Jonas Brunsvik Martin Skårerverket Irfan Ljevo. *Embedded system integration with PID controller*, 2020.

[8] Derek Molloy. *Exploring BeagleBone: Tools and Techniques for Building with Embedded Linux*, 2014.

[9] Sasang Balachandran. *Embedded system integration with PID controller*, 2009. https://www.egr.msu.edu/classes/ece480/capstone/fall09/group03/AN_balachandran.pdf,accessed16.05.2021.

[10] Michael Barr. *Introduction to Pulse Witdt Modulation*, 2001. https://barrgroup.com/Embedded-Systems/How-To/PWM-Pulse-Width-Modulation, accessed 16.05.2021.

[11] Henrik Baldishol Jakob Karlsen Kristoffer Nordvik Jonas Hjulstad, Endre T. Ellingsen. *Instrumentation, actuation and model identification of bipedal robot*, 2019. Page 87-91.

[12] Jonas Brunsvik Martin Skårerverket Irfan Ljevo. *Embedded system integration with PID controller*, 2020. Pages 43-44.

[13] Patrick T.Squire. *Pendulum damping*, 1985. https://aapt.scitation.org/doi/pdf/10.1119/1.14838?class=pdf&,accessed19.05.2021.

[14] Charles Pao. *What is an IMU Sensor?*, 2018. https://www.ceva-dsp.com/ourblog/what-is-an-imu-sensor/#:~:text=An%20IMU%20is%20a%20specific,force%20and%20sometimes%20magnetic%20field.&text=Technically%2C%20the%20term%20%E2%80%9CIMU%E2%80%9D,measures%20of%20orientation%20and%20heading.,accessed16.05.2021.

[15] Robin Mitchell. *MEMs Six Degrees of Freedom Sensors are Built to Cover More ADAS Safety Bases*, 2020. https://www.allaboutcircuits.com/news/mems-six-degrees-of-freedom-sensors-built-cover-safety-bases-of-adas/.

[16] Scott Campbell. *BASICS OF THE I2C COMMUNICATION PROTOCOL*, 2016. https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/#:~:text=I2C%20is%20a%20serial%20communication,the%20master%20and%20the%20slave.,accessed17.05.2021.

[17] Jonas Brunsvik Martin Skårerverket Irfan Ljevo. *Embedded system integration with PID controller*, 2020. Page 57.

[18] Maxongroup. *Advanced circuits PCB trace width calculator*, 20.Mai.202. https://www.4pcb.com/trace-width-calculator.html, accessed 20.05.21.

[19] *BeagleBoneBlack*, 2019. https://beagleboard.org/black, accessed 15.05.2021.

[20] Derek Molloy. *Exploring BeagleBone: Tools and Techniques for Building with Embedded Linux*, 2014. Pages 2019-226.

[21] Stian Johan Olsen Kristoffer Meggelæ Pedersen Lars-Erik Nes Panengstuen, Edvard Merkesvik. *Flashing Ubuntu 18.04*, April.2021. https://github.com/E2110/Biped-Robot-Prototype, accessed 15.05.2021.

[22] Henrik Baldishol Jakob Karlsen Kristoffer Nordvik Jonas Hjulstad, Endre T. Ellingsen. *Instrumentation, actuation and model identification of bipedal robot*, 2019. Pages 98-105.

[23] Maxongroup. *Part number 148877*, 2021. https://www.maxongroup.com/maxon/view/product/148877,accessed16.05.2021.

[24] Maxongroup. *Part number 260551*, 2021. https://www.maxongroup.com/maxon/view/product/260551,accessed16.05.2021.

[25] Maxongroup. *Planetary Gearhead GP 42 C Ø42 mm, 3 - 15 Nm, Ceramic Version*, 14.Mai.2020. https://www.maxongroup.com/maxon/view/product/203119.

[26] Maxongroup. *Why do DC motors fail?*, 28.September 2017. https://www.maxongroup.co.uk/maxon/view/news/Why-do-DC-motors-fail, accessed 14.05.2021.

[27] Stian Johan Olsen Kristoffer Meggelæ Pedersen Lars-Erik Nes Panengstuen, Edvard Merkesvik. *Motor. Configure the Servo-controller*, April.2021. https://github.com/E2110/Biped-Robot-Prototype/tree/main/Motor, accessed 14.05.2021.

[28] Jonas Brunsvik Martin Skårerverket Irfan Ljevo. *Embedded system integration with PID controller*, 2020. Pages 24-25.

[29] Stian Johan Olsen Kristoffer Meggelæ Pedersen Lars-Erik Nes Panengstuen, Edvard Merkesvik. *Motor/encoder_test/src*, April.2021. https://github.com/E2110/Biped-Robot-Prototype/tree/main/Motor/encoder_test/src, accessed 15.05.2021.

[30] Jonas Brunsvik Martin Skårerverket Irfan Ljevo. *Servo duty cycle* , 2020. Page 23.

[31] Johan Schoukens Maarten Steinbuch David Rijlaarsdam, Pieter Nuij. *Frequency Domain Based Friction Compensation - Industrial Application to Transmission Electron Microscopes*, 2011. https://folk.ntnu.no/skoge/prost/proceedings/acc11/data/papers/0166.pdf/,accessed19.05.2021.

List of Figures

List of Tables

A Figures

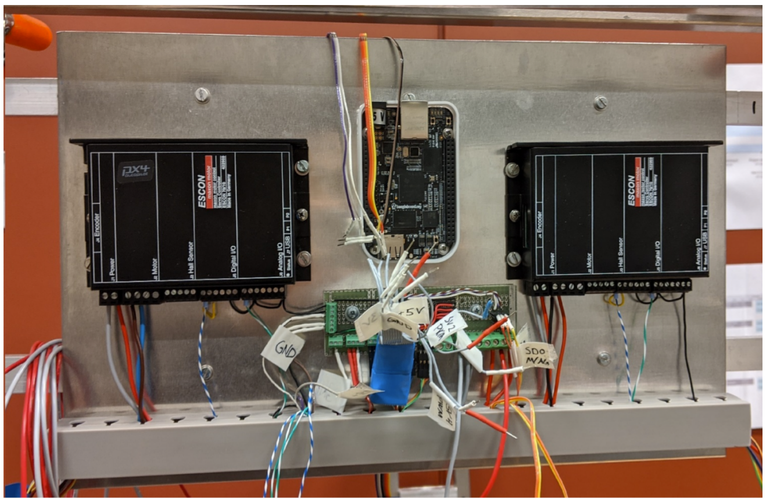


Figure 41: Wiring 2019

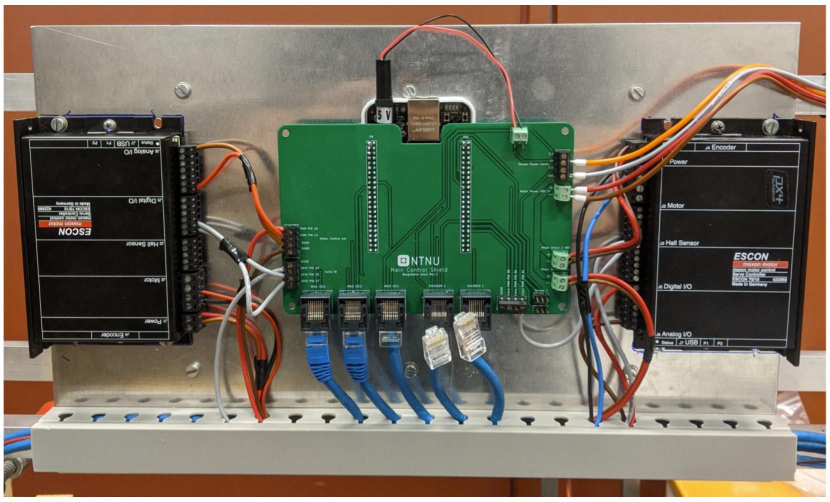


Figure 42: Wiring 2020

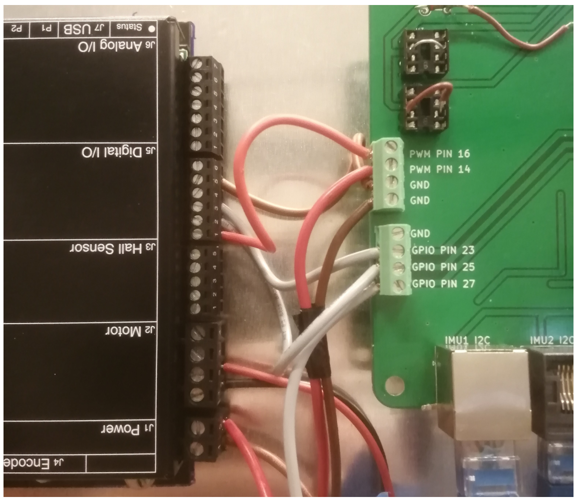


Figure 43: New servo-controller wiring

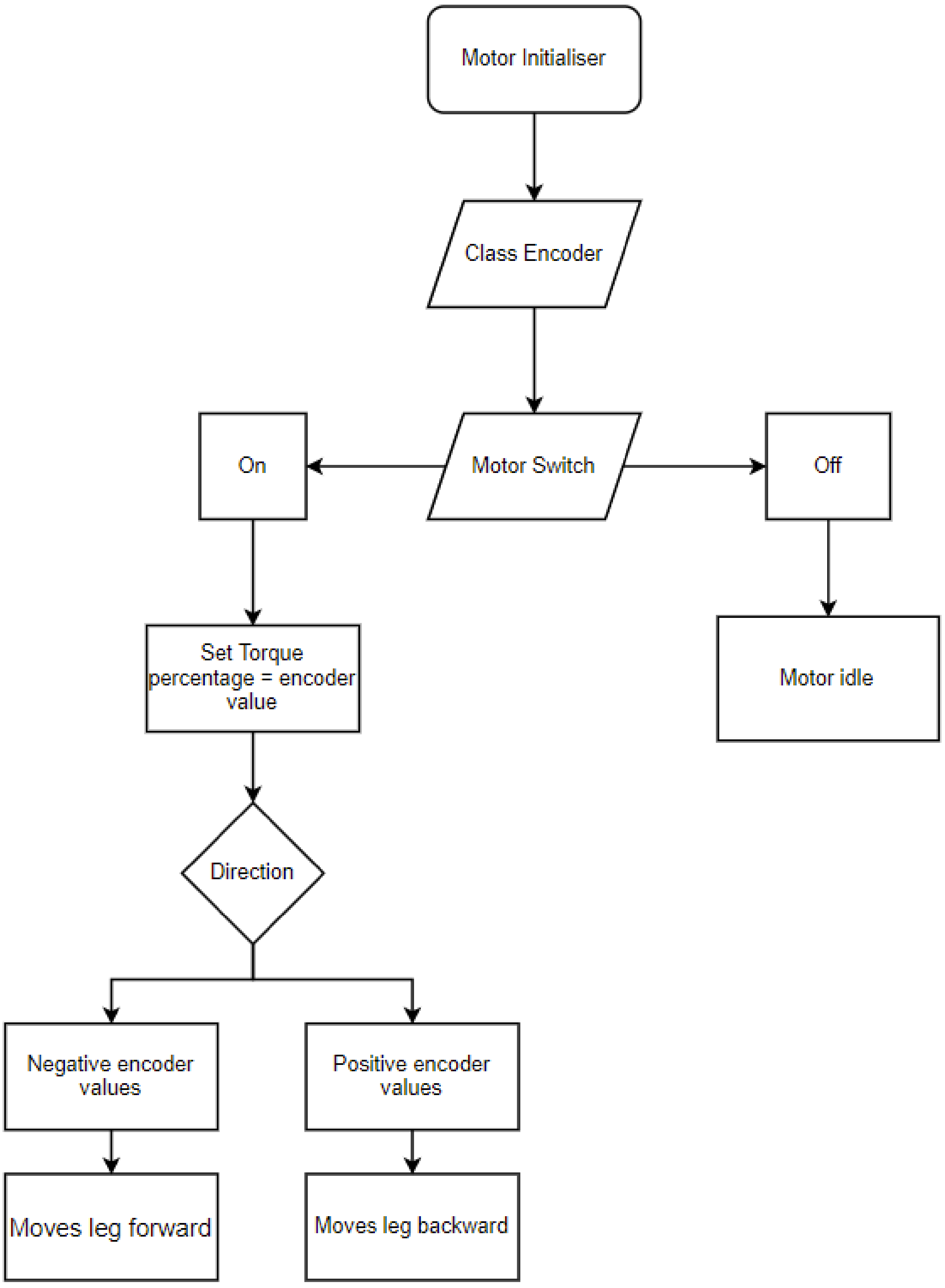


Figure 44: Motor/Encoder flow chart

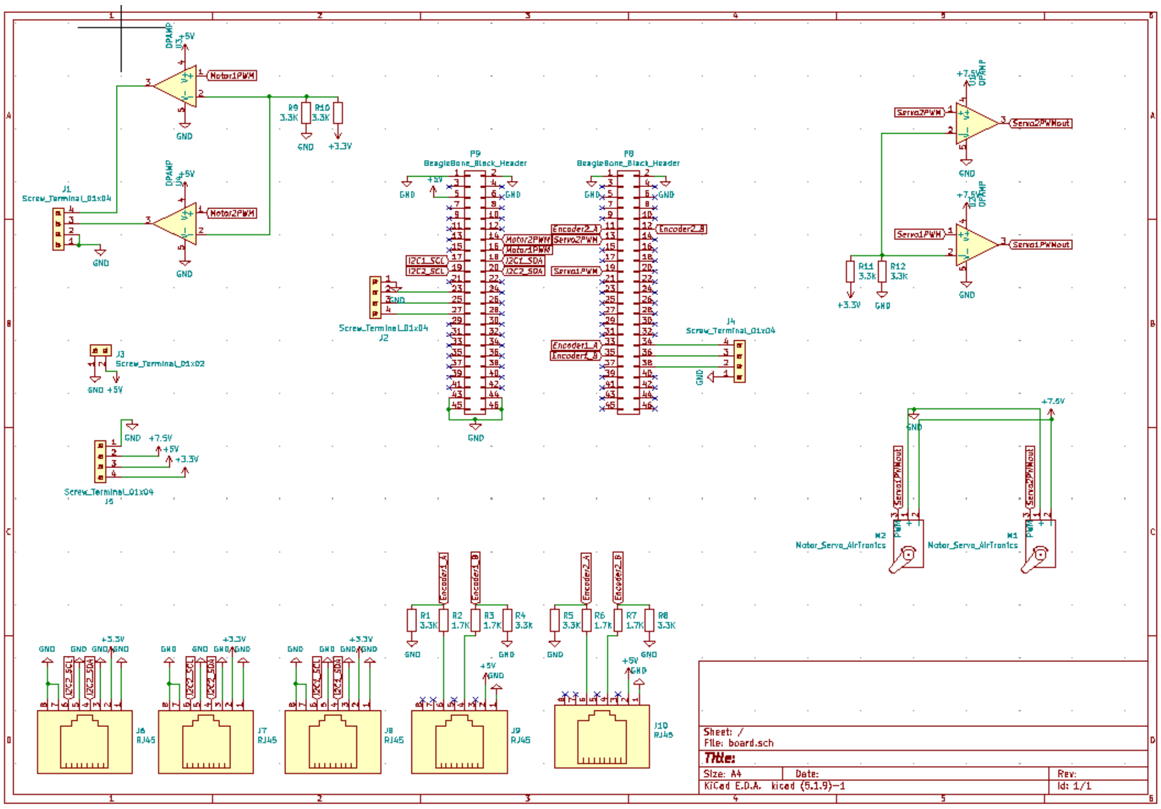


Figure 45: BBB shield schematic

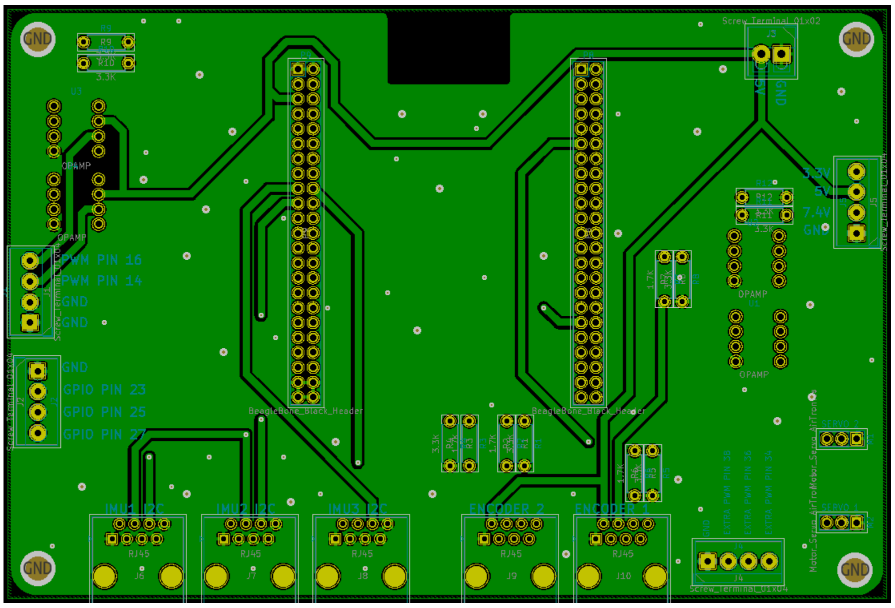


Figure 46: BBB shield back

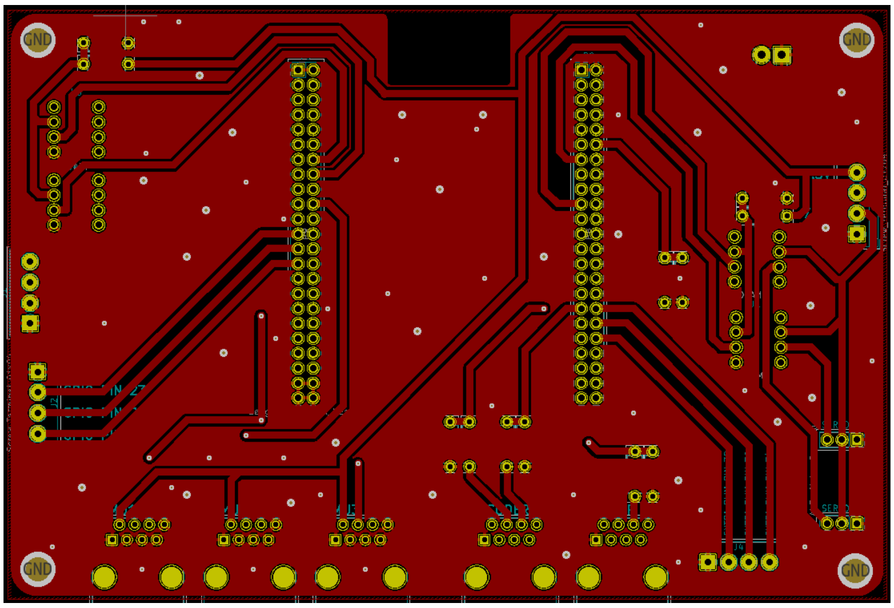
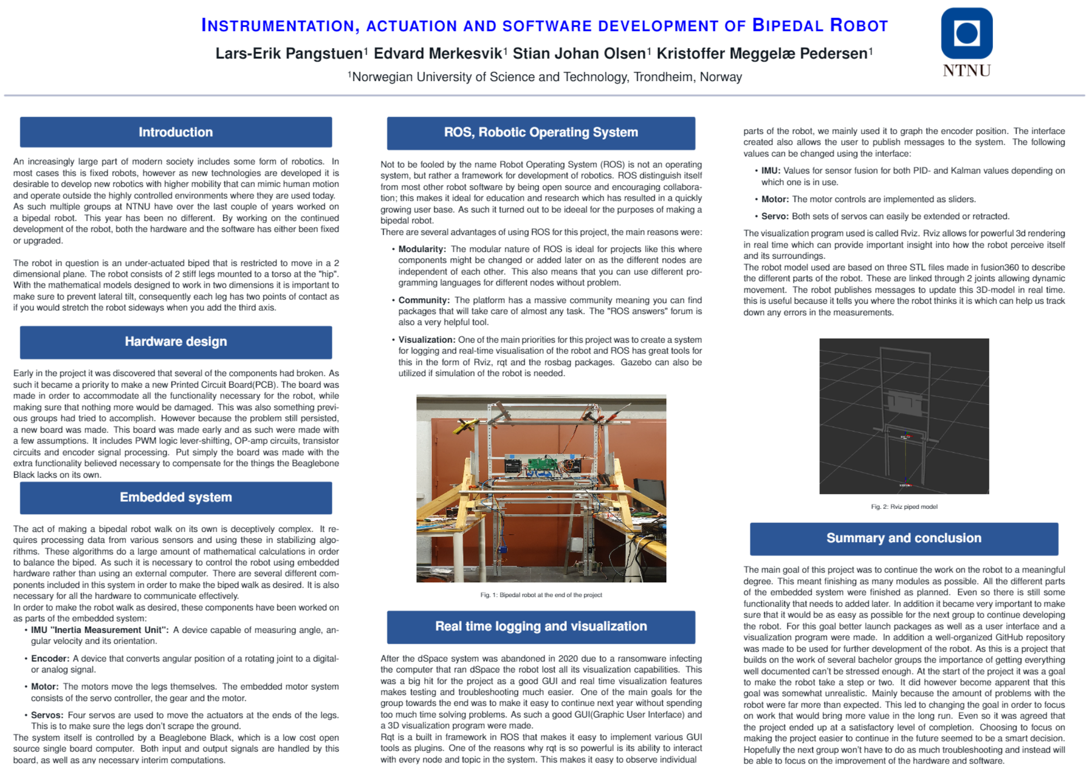


Figure 47: Caption

B Biped poster



INSTRUMENTATION, ACTUATION AND SOFTWARE DEVELOPMENT OF BIPEDAL ROBOT

Lars-Erik Pangstuen[1] Edvard Merkesvik[1] Stian Johan Olsen[1] Kristoffer Meggelæ Pedersen[1]

[1]Norwegian University of Science and Technology, Trondheim, Norway

Introduction

An increasingly large part of modern society includes some form of robotics. In most cases this is fixed robots, however as new technologies are developed it is desirable to develop new robotics with higher mobility that can mimic human motion and operate outside the highly controlled environments where they are used today. As such multiple groups at NTNU have over the last couple of years worked on a bipedal robot. This year has been no different. By working on the continued development of the robot, both the hardware and the software has either been fixed or upgraded.

The robot in question is an under-actuated biped that is restricted to move in a 2 dimensional plane. The robot consists of 2 stiff legs mounted to a torso at the "hip". With the mathematical models designed to work in two dimensions it is important to make sure to prevent lateral tilt, consequently each leg has two points of contact as if you would stretch the robot sideways when you add the third axis.

Hardware design

Early in the project it was discovered that several of the components had broken. As such it became a priority to make a new Printed Circuit Board(PCB). The board was made in order to accommodate all the functionality necessary for the robot, while making sure that nothing more would be damaged. This was also something previous groups had tried to accomplish. However because the problem still persisted, a new board was made. This board was made early and as such were made with a few assumptions. It includes PWM logic lever-shifting, OP-amp circuits, transistor circuits and encoder signal processing. Put simply the board was made with the extra functionality believed necessary to compensate for the things the Beaglebone Black lacks on its own.

Embedded system

The act of making a bipedal robot walk on its own is deceptively complex. It requires processing data from various sensors and using these in stabilizing algorithms. These algorithms do a large amount of mathematical calculations in order to balance the biped. As such it is necessary to control the robot using embedded hardware rather than using an external computer. There are several different components included in this system in order to make the biped walk as desired. It is also necessary for all the hardware to communicate effectively.
In order to make the robot walk as desired, these components have been worked on as parts of the embedded system:

- **IMU "Inertia Measurement Unit":** A device capable of measuring angle, angular velocity and its orientation.
- **Encoder:** A device that converts angular position of a rotating joint to a digital- or analog signal.
- **Motor:** The motors move the legs themselves. The embedded motor system consists of the servo controller, the gear and the motor.
- **Servos:** Four servos are used to move the actuators at the ends of the legs. This is to make sure the legs don't scrape the ground.

The system itself is controlled by a Beaglebone Black, which is a low cost open source single board computer. Both input and output signals are handled by this board, as well as any necessary interim computations.

ROS, Robotic Operating System

Not to be fooled by the name Robot Operating System (ROS) is not an operating system, but rather a framework for development of robotics. ROS distinguish itself from most other robot software by being open source and encouraging collaboration; this makes it ideal for education and research which has resulted in a quickly growing user base. As such it turned out to be ideeal for the purposes of making a bipedal robot.
There are several advantages of using ROS for this project, the main reasons were:

- **Modularity:** The modular nature of ROS is ideal for projects like this where components might be changed or added later on as the different nodes are independent of each other. This also means that you can use different programming languages for different nodes without problem.
- **Community:** The platform has a massive community meaning you can find packages that will take care of almost any task. The "ROS answers" forum is also a very helpful tool.
- **Visualization:** One of the main priorities for this project was to create a system for logging and real-time visualisation of the robot and ROS has great tools for this in the form of Rviz, rqt and the rosbag packages. Gazebo can also be utilized if simulation of the robot is needed.

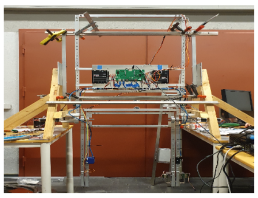


Fig. 1: Bipedal robot at the end of the project

Real time logging and visualization

After the dSpace system was abandoned in 2020 due to a ransomware infecting the computer that ran dSpace the robot lost all its visualization capabilities. This was a big hit for the project as a good GUI and real time visualization features makes testing and troubleshooting much easier. One of the main goals for the group towards the end was to make it easy to continue next year without spending too much time solving problems. As such a good GUI(Graphic User Interface) and a 3D visualization program were made.
Rqt is a built in framework in ROS that makes it easy to implement various GUI tools as plugins. One of the reasons why rqt is so powerful is its ability to interact with every node and topic in the system. This makes it easy to observe individual parts of the robot, we mainly used it to graph the encoder position. The interface created also allows the user to publish messages to the system. The following values can be changed using the interface:

- **IMU:** Values for sensor fusion for both PID- and Kalman values depending on which one is in use.
- **Motor:** The motor controls are implemented as sliders.
- **Servo:** Both sets of servos can easily be extended or retracted.

The visualization program used is called Rviz. Rviz allows for powerful 3d rendering in real time which can provide important insight into how the robot perceive itself and its surroundings.
The robot model used are based on three STL files made in fusion360 to describe the different parts of the robot. These are linked through 2 joints allowing dynamic movement. The robot publishes messages to update this 3D-model in real time. this is useful because it tells you where the robot thinks it is which can help us track down any errors in the measurements.

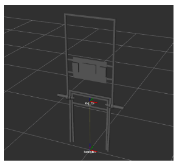


Fig. 2: Rviz piped model

Summary and conclusion

The main goal of this project was to continue the work on the robot to a meaningful degree. This meant finishing as many modules as possible. All the different parts of the embedded system were finished as planned. Even so there is still some functionality that needs to added later. In addition it became very important to make sure that it would be as easy as possible for the next group to continue developing the robot. For this goal better launch packages as well as a user interface and a visualization program were made. In addition a well-organized GitHub repository was made to be used for further development of the robot. As this is a project that builds on the work of several bachelor groups the importance of getting everything well documented can't be stressed enough. At the start of the project it was a goal to make the robot take a step or two. It did however become apparent that this goal was somewhat unrealistic. Mainly because the amount of problems with the robot were far more than expected. This led to changing the goal in order to focus on work that would bring more value in the long run. Even so it was agreed that the project ended up at a satisfactory level of completion. Choosing to focus on making the project easier to continue in the future seemed to be a smart decision. Hopefully the next group won't have to do as much troubleshooting and instead will be able to focus on the improvement of the hardware and software.

Figure 48: Bipedal Robot Poster

C IMU simulations

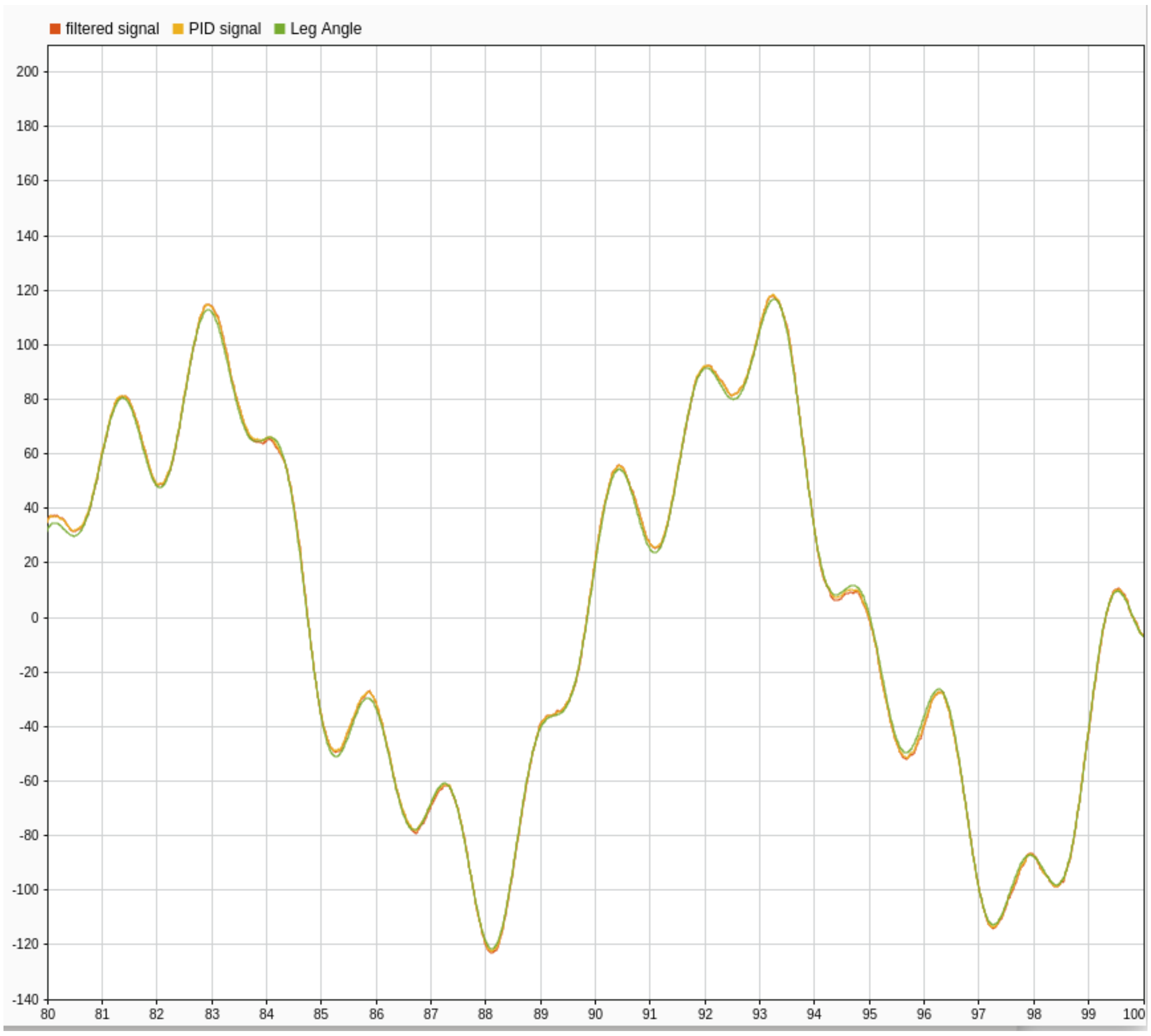


Figure 49: Leg Angle PID kalman simulation

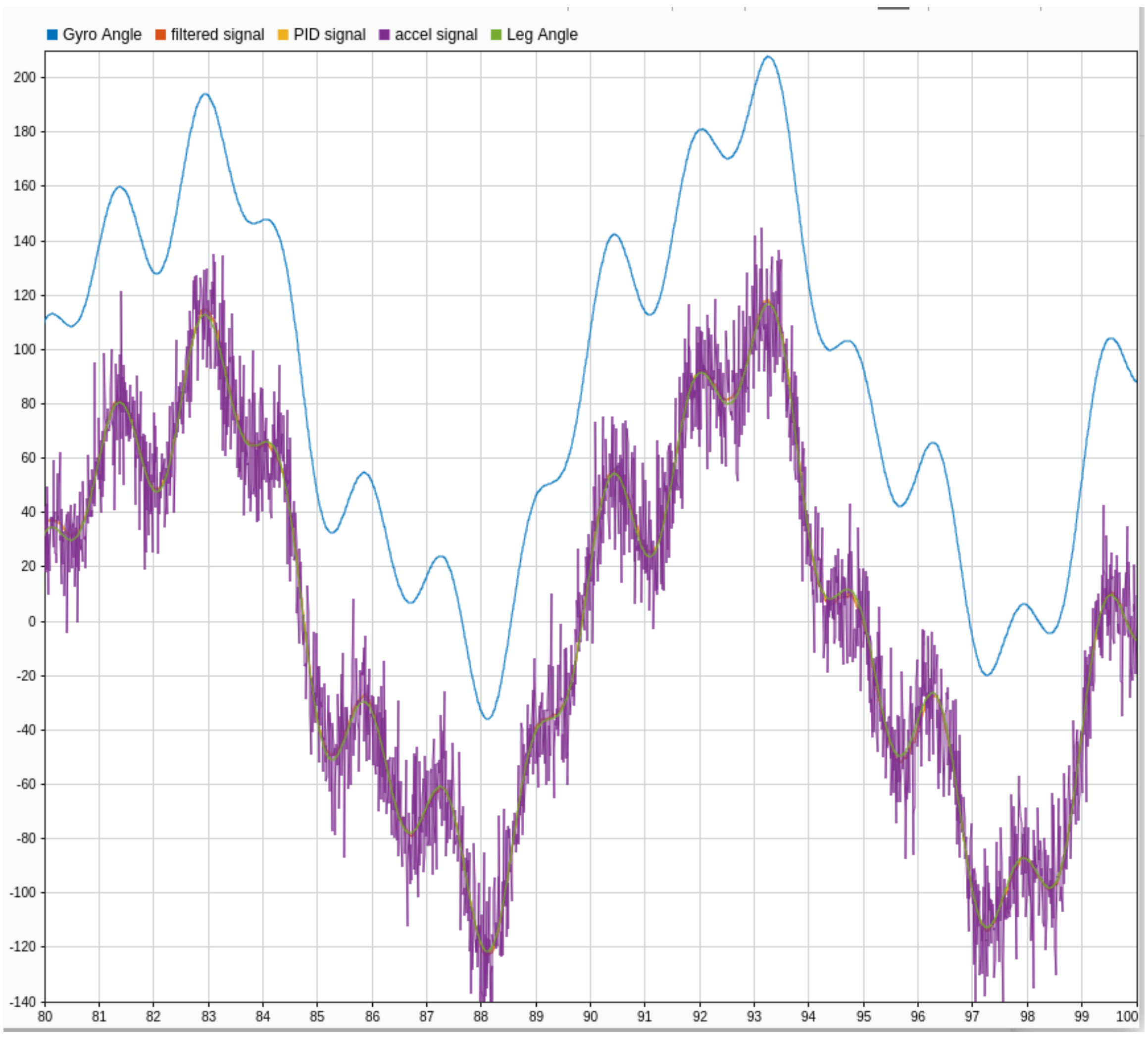


Figure 50: Leg Angle PID kalman simulation

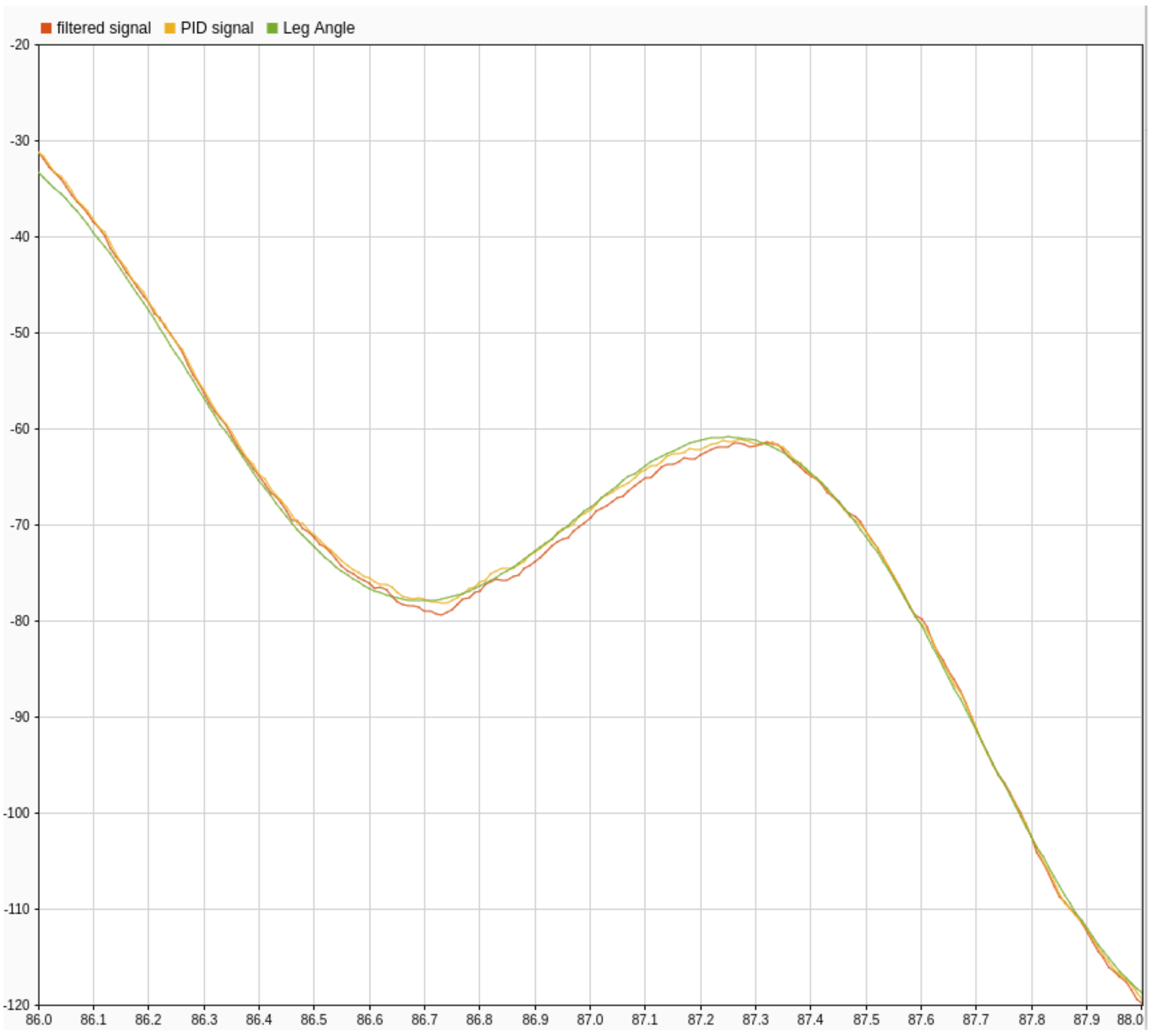


Figure 51: PID, kalman filter simulation

NTNU
Kunnskap for en bedre verden