Instrumentation, actuation and model identification of bipedal robot

Bachelor Thesis

Jonas Hjulstad, Endre T. Ellingsen, Henrik Baldishol,
Jakob Karlsen, Kristoffer Nordvik

Bachelor Thesis report for TELE3001 Bachelor Thesis
Supervisor: Torleif Anstensrud, ITK
Submission date: 20 May 2019

Trondheim, January 2019

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

NTNU

Abstract

This project is dedicated to facilitating a robot's walking abilities, approached using instrumentation, programming and physical modeling.

Object-oriented programming is used to separate the code written for communication between devices. In order to preserve the functionality of the Inertial Measurement Unit (IMU), the manufacturers library has been ported to a new platform.

The robot is given a Single Board Computer (SBC) as the control unit, expanding its abilities with Linux, which supports multiple programming languages. Board specifications and software is introduced to facilitate future real time projects. The resulting solutions presents a high-performance platform with real time capabilities within- and outside the Linux operating system.

Movement capabilities and position awareness is achieved with suitable solutions for motor control and joint position feedback. The joint position feedback mainly consists of obtaining an accurate measurement of the relative position between the robot's leg and torso using an IMU.

The position measurements have been analyzed both theoretically and in practice in order to find a optimal configuration of the measurement unit. Simulations and experiments concluded with leg mounting, placed close to the hip as the optimal IMU placement, aswell as operating with two IMUs. But currently it is torso-mounted due to cable noise. This complication requires further research for an optimal solution, which is discussed in this report.

Modeling the dynamics of the unactuated swing of each of the robot's legs was done through the use of harmonic balance. The differential equation describing the swing of the robot's leg is not solvable, so several approximations and assumptions were made when linearizing it. Using harmonic balance, solvable equations for all unknown parameters were acquired and solved for said parameters.

Simulations from the final measurement model showed deviations up to 7°from the real system. In the future it is recommended that a new model is made using an optimization algorithm. This will allow for a more accurate model since the governing differential equation does not have to be altered.

The final system is capable of precise angle measurement, motor- and servo-actuation, both controlled from embedded hardware.

Sammendrag

Denne prosjektoppgaven handler om tilrettelegging av en tobeinet robots evne til å gå, som gjøres ved hjelp av instrumentering, programmering og modellidentifikasjon.

Objektorientert programmering brukes til å separere kode brukt til kommunikasjon mellom de forskjellige enhetene på roboten. For å bevare funksjonaliteten til robotens posisjonssensor har tilhørende biblioteker blitt tilpasset en ny platform. Roboten har fått en Single Board Computer (SBC) som styreenhet. Med Linux, som støtter flere programmeringsspråk, utvider dette mulighetene for styring av roboten. Teknisk beskrivelse av styreenheten og kombinert spesifikk programvare tilrettelegger for framtidige sanntidsløsninger. Dette gir en ytelsessterk platform med mulighet for sanntidsløsninger innenfor og utenfor Linux-operativsystemet.

Bevegelsesevne og robotens posisjonsbevissthet finnes med gode løsninger for motorstyring og sensoravlesninger. Hovedproblemet med posisjonsbevissthet lå i relativ posisjonsavlesning fra treghetsmålingsenheter. (Inertia Measurement Unit, IMU)

Posisjonsmålingene har blitt analysert i teori og praksis for å finne en opimal konfigurasjon for måleintrumentet. Videre har dette blitt implementert i praksis, med en begrensing på plassering av IMUen til overkoppen. Dette strider imot plasseringen funnet i analysen som tyder på at IMUen plassert nærme hofteleddet er å foretrekke. Samt opere med to IMUer, en på hvert bein. Dette problemet krever videre arbeid for en ideel løsning som blir diskutert i denne rapporten.

Modellering av dynamikken i robotens bein ved frie svingninger ble gjort ved hjelp av «harmonic balance». Differensiallikningen som beskriver svingningen i robotens ben er uløselig i konvensjonell matematikk, så det ble tatt i bruk flere forenklinger og antakelser ved linearisering av modellen. Ved hjelp av denne lineariseringen og harmonic balance ble det laget likninger for hver av de ukjente parameterne som beskriver dynamikken i systemet.

Den ferdige modellen ble simulert og sammenliknet mot data fra det reelle systemet. Man kan se at simuleringen bommer med opp mot 7°i løpet av det fulle svinget. Det anbefales at det legges vekt på en optimaliseringsalgoritme i framtiden, da dette trolig vil gi mer nøyaktige resultater uten at man må gjøre endringer på den originale differansiallikningen.

Det komplette systemet er kapabelt til presise vinkelmål, samt motor- og servo-aktuering, alt styrt fra en integrert styringsplattform.

Preface

The main project of this thesis is the finalizing endeavour of our third year and Bachelors degree within electrical engineering, specialized in automation. It comprises 20 study credits, which in this project corresponds to about 500 work hours per student. The assignment is given by Department of Engineering Cybernetics, NTNU. This is the conclusive report from mentioned project. In order to fully understand the content of this report, a minor expertise in the field of elecrical engineering is preferred.

We would like to acknowledge our supervisor Torlief Anstensrud, as well as the services provided by employees at the Department of Engineering Cybernetics.

20/05-2019
Date

Jonas Hjulstad (JH)

Endre T. Ellingsen (EE)

Henrik Baldishol (HB)

Jakob Karlsen (JK)

Kristoffer Nordvik (KN)

Table of content

List of Figures

List of Tables

Listings

Glossary

Glossaries	
Sagittal plane	Two-dimensional surface set by the longitudinal axis of the robot, able to represent all angular changes and freedoms of movement
AutoCAD	Computer-Aided Design software used for modeling
dSpace RTI 1103	Computer based realtime system with analog and digital ports.
ControlDesk (Next Generation)	Software to control the Dspace rti1103 board
MPU	MicroProcessor Unit, incorporates the CPUs functions with an integrated circuit
SBC	Single Board Computer with storage outside of the MPU, allowing the chip to function as a general computer
IMU	Inertial Measuring Unit, measuring linear accelerations, angular velocities and magnetic fields.
PWM	Pulse Width Modulation, used to approximate analog output signals
Biped	Two-legged
Algorithm	Step-by-step procedure to solve logical/mathematical problems
Embedded System	System specialized to perform a specific task
Embedded Hardware	Hardware designed specifically to perform tasks required by Embedded Systems
Realtime	The actual time of a process event
RTOS	RealTime Operating System, capable of measuring and responding to process events ∼instantaneously
SPI	Serial Peripheral Interface Bus (Communication Protocol)
I2C	Inter-Integrated Circuit (Communication Protocol)

Glossaries	
GPIO	General Purpose Input Output
RAM	Random Access Memory, fast computer data storage
Microcontroller	Single-board computer using a CPU with integrated storage
Arduino	Microcontroller distributor
Arduino Mega	Powerful Arduino distribution with a large GPIO-interface
BeagleBone Black	High-performance SBC
Raspberry Pi Model B	Quad-core SBC distribution
Servo	A rotary actuator used for controlling angular or linear position
EPROM	Erasable, Programmable Read Only Memory
EPROM Programmer	Device to write instructions and data into EPROM
IDE	Integrated Development Environment, software with tools to simplify and automate code development and processing
Eclipse	Open-source IDE software
Debian	Free software-based Linux-distribution
Proprietary software	Company owned, restricted software
Ubuntu	Debian-based linux-distribution with proprietary software
TI	Texas Instruments
APT	Advanced Packaging Tool
Repository	Storage location for software packages
SSH	Secure Shell, network protocol for secure connection
IPv4-address	Unique, 32-bit Internet Protocol address to identify conneced devices
Shell	Command-line user interface
Terminal	Program interface for running shells
PRU	Programmable Realtime Unit
Compiler	Code translation software
Binary file	File with content in a computer-readable format
Bash	Command language used by shells

Glossaries	
Bash script	Executable file programmed with shell-commands
Processor architecture	Design describing the how the processor behaves, and machine code is handled by the processor.
GNU	Open-source operating system (Recursive acronym "GNU's Not Unix"
GCC	GNU Compiler Collection
Assembler	Low-level to machine code translator
Linker	Combines code files mid-translation
Loadable kernel module	Loadable code used to extend the functionality of the kernel while running
Abstraction	Reducing complexity of the program by reducing interface through use of module oriented programming
Class	Data structure containing both data and functions
Objects	A single instance of a class
Sample Time	The time interval between each discrete measurement
Map	A standard storage structure which contains key-value pair with unique keys
Segmentation Fault	A failure condition raised by hardware with memory protection, notifying the operating system that software has attempted to access restricted memory.[20]
Restricted Memory	Memory that has not been allocated to the interacting process
Pointer	A variable whose value is the address of another variable. [21]
Dereference	In C++, to "dereference" a pointer, means accessing the value whose memory cell is being pointed to
Null-Pointer	A pointer which has not been assigned a valid memory address
Allocate	In C++, to "allocate" memory, means assigning memory to the process to be used to store variables and instances of objects.
Memory Block	A continous chunk of allocated memory

Glossaries	
Deep copy	A copy of an instance where values contained in dynamically allocated memory is copied into a new collection of allocated memory.
Constructor	A function being called upon the instantiating of an object
Destructor	A function being called when an instance of an object runs out of its scope
SPIDEV	Loadable kernel module driver, responsible for interfacing with the SPI hardware modules
Static	Type definition declaring data/functions as unique, forcing all related functions to use the same variable
Level Converter	Circuit which converts voltage level, mainly used to convert logic levels between 3.3V and 5V
Simulink	Graphical programming enviroment tightly integrated with Matlab enviroment, developed by mathworks.
Baud rate	Defines transmissionspeed in a communications system, in bit per seconds (bps).
FreeCAD	Open Source 3D Parametric Modelling Software.
Fusion 360	3D CAD software
CURA	Software to slice 3D models and generate G-code (machine code) for printing the model with a 3D-printer.
PLA filament	Material used in 3D-printing
PCB	Printed Circuit Board
Sensor fusion	Strategy combining the measurement from an accelerometer and a gyroscope in order to improve the measurement.

Glossaries	
Accelerometer	Unit measuring linear accelerations
atan2(y,x)	Trigonometric function returning the angle between the x-axis and the ray to the point (x,y) in a cartesian coordinate system.
Gyroscope	Unit measuring angular velocity
Bias	Offset,non-random amount a measuring device is off the actual
MATLAB	Advance mathematics software with its own language based on C developed by mathworks.
θ	Actual angle of the IMU.
θ_a	Unfiltered angle calculated by accelerometer.
measurements θ_g	Unfiltered angle calculated by gyroscope measurements.
ω_g	Angular velocity measured by gyroscope.
θ_c	Sensor fusioned filtered angle.
θ_{enc}	Angle measured by encoder.
IMU-position,Torso	Disance from hip-joint to the IMU.
IMU-position,Legs	Distance from the tip of the leg the IMU is placed on to the IMU.
Jerk	Quick, sudden movement, more accurate it is the derivative of accelerations.
HyperTerminal	Software to print and save data from a communication port on the computer, for example an USB port.
eQEP	Enhanced Quadrature Encoder Pulse module
eCAP	Enhanced Capture module [6, p.2465], (Primarily used for readings, output for one PWM)
EHRPWM	Enhanced High Resolution Pulse Width Modulator
Joint	Point where two parts of the robot are fitted together
Transfer function	Matemathical function describing output for a given input
Harmonic oscillation	Periodic motion where force is directly proportional to displacement

Table 1: Glossaries

1 Introduction

1.1 Background

In modern industry, robotic solutions comprises an increasingly larger share of production. While these are essentially mounted operations in predictable environments, it is highly desirable to extend this technology to be applied to challenges outside the factory floor. This would require development of robotics with a better capability for mobility. A suitable proposal would be machines that mimics the gait of humans.

Since McGeer's seminal paper[22] on planar passive walkers, research has been inspired to further examine the possibility of near passive walking machines. At NTNU, advanced mathematical models [23] have been develeoped to plan the motions of such gaits. A physical prototype has been assembled to test the feasibility of these models. Our Bachelor thesis concerns the instrumentation and completion of this prototype.

The current mathematical model concerns a bipedal machine, which is constricted to move in the sagittal plane. It is equipped with two pairs of legs, each pair powered by a motor rotating parallel to each other. A torso mounted at the hip constitutes a third degree of freedom and connects the assembly together. The torso is a passive degree of freedom affected by gravity and motor torque. Since the orientation of the torso is not directly controlled, the robot is underactuated.

With that being said, the main structure of the prototype is already completed. There is a considerable amount of work within measurements, mounting components, cabling, system identification and documentation. The prior work and design is described in greater detail in Christian F. Sætres paper[23].

1.2 Preliminary design and work

1.2.1 Frame

The frame is designed through AutoCAD and built at the Department of Engineering Cybernetics, NTNU. It is constructed using square aluminum profiles, and custom-machined aluminum parts such as the housings for the bearings, the flexible actuator couplings, the motor mounts, and the retractable point feet. The aluminum profiles are connected through custom-machine angle brackets. The robot is fully disassembleable with the exception of the bearings which are press-fitted into the construction.

1.2.2 Actuators

Each leg is separately actuated with an electric motor, which has a max torque of approximately 3 Nm. The motors operates optimally at 48V, and are powered with a power supply on 600 W through two servo controllers. The controllers receive a [-10V,10V] signal from a dSpace Controller Board, and send a [-3A,3A] signal to the motors. Each leg has two feet equipped with retractable pointers, which are separately powered by a servo motor for extension and retraction. To function

properly, the servos require a PWM signal delivered at 4.8V. Furthermore, the feet are designed such that the weight of the robot rests on a lockable joint.

1.2.3 Measurements

Each of the hip actuators has an encoder attached for measuring the angle of rotation between its rotor and stator. In addition, the encoder's measurements are directly transferred to the dSpace Control Panel. The measurements can then be read through dSpace Control Desk. In each gearbox there is a slack which is noticeable when switching torque direction.

1.2.4 Power and Control

The main components that need external power are the micro controller, the two motors and the four servos at the feets. The current power supply can deliver 600 W across two separate channels. One is reserved for the two motors, while the other is reserved for the electronics. Currently all the I/O is handled by the dSpace Controller Board via a Connector Panel.

1.2.5 Preliminary parts list

The following table includes all the parts that are currently mounted on the robot or disposable for use.

Description	Product name	Manufacturer
Flexible actuator coupler	4779823	Ruland
Hip bearings	6004-C	Fag
Electric motors for leg actuation	14887	Maxon Motor
1-to-6 Gearbox for motors	Planetary Gearhead	Maxon Motor
Servo controllers for motors	ESCON 50/5 (409510)	Maxon Motor
Servos for retractable feet	S9254	Futaba Corporation
Encoders for relative leg angles	2RMHF	Seancon
IMUs for absolute leg angles	Breakout-LSM9DS1	Sparkfun
Arduino for processing IMU	Arduino MEGA ADK	Arduino
600W power supply	QPX600D	Aim-TTI
Connector Panel for I/O	CLP1103	dSpace
Controller Board	DS1103 PPC	dSpace
BeagleBone Black	BeagleBone Black	BeagleBone
ControlDesk	ControlDesk	dSpace

Table 2: Parts list

1.3 Problem statement

The main problem statement of the thesis is to apply instrumentation to a two-legged robot prototype, which we can divide into five different categories as mentioned below.

An additional problem was model identification, which was intended as a bonus objective dependant of available time and personnel.

1.3.1 Voltage supply

The robot has two electric motors with their own driver, requiring a specific voltage. Also, the servos and the single board computer (SBC) require some specific voltage, so the robot needs a power supply delivering the right amount of power for each part.

Some components require lower voltage than others, which is achievable with voltage dividers, or DC choppers/converters. One of these designs will be used to create a stable power supply, power efficient and unaffected by variable current drain.

1.3.2 Angular position measurement

A sensor is needed to know the torso and leg positions, according to the surroundings. This will be accomplished by utilizing an IMU mounted on the torso. From the IMU's gyroscope and accelerometer, we get the angular velocity and the torso's orientation relative to the world frame. With encoders measuring angles between the torso and each leg, this should be sufficient information for finding the robots orientation and position. There is, however, a known issue regarding slack in the transmission for the leg's actuators, which could potentially result in an erroneous bias in the measurement.

1.3.3 Servo control

There are in total four servos to be controlled, one mounted on each leg. PWM input signal range for each servo need to be identified for equal extension/retraction.

1.3.4 Wiring, mounting and communication

All the parts are to be mounted and wired together in a functional and organized way, thus making them able to communicate through a single board computer.

1.3.5 Documentation

By documenting our work, people working on the robot in the future will find what they need from our work without any inconvenience.

1.4 Structure

Each subject of the report are presented in its own section. Each section follows the same structure (in general), starting with a introduction of subject-relevant theory. This is followed by method describing the procedure, and lastly results are presented. In addition, each section contains discussions and conclusions where it is relevant.

Embedded system This section starts with a complete software setup, followed by the functionality and setup of toolchains. With a basic understanding of code translation, the projects code development can be explained. Lastly, information about the relevant features of Linux are included.

BeagleBone realtime capabiliites and subsystems: This subsection is mainly theoretical, intended to give an overview of how interrupts work in the boards processor subsystems. A minimal amount of examples are used to give a foundation for programming.

IMU: This section contains the solution of implementating the IMU. Furthermore reasoning of the implementation obtained through a comprehensive simulation, followed by experiments in order to verify the results of the simulation.

Encoder: This section contains the theory of the encoder. Also the implementation of the encoder with the Beagle Bone Black.

Actuation: This section contains the theory behind the actuation ... osv In addition to the method and results to find the control signal span for the servos.

Model identification: This section conains the theory, method and results of the model idenification.

Discussion: This section contains a summary discussion of the results from each part of the project.

Conclusion: This section contains a short conclusive summary of the total result of the project. In addition to explanations for suggested further work.

Words appearing in *italic* are terms explained in the glossary chapter. The terms are explained in a chronological order.

When referred to the bachelors folder, [24], every file referred to lies in the folder which is reffered. The folder ("Simulation" in this example) will be included in the reference as follows: [24, Simulation]. Due to size constraints, some of the stored gaits of the Simulaion folder have been removed.

2 Embedded system

As the biped is intended to be able to move independently on its own, it cannot be controlled with direct connection to an external computer. In addition, the stabilizing algorithm consists of a considerable amount of complex mathematics, whose execution is imperative to its balance. This makes it a hard real time system. Therefore control through *embedded hardware* with a powerful processor is desirable.

Embedded hardware is computer-based controllers which are implemented in larger devices and are usually dedicated to a specific task regarding monitoring or control[25]. These systems are programmed and being run by a real-time operating system (RTOS) which is designed to serve real time applications. In other words, These systems are designed to react fast to a trigger and are capable of preemptively adjust system resources to prioritize tasks of higher priority during run-time[26].

There is a vast selection of embedded devices on the market, but the most capable are the *single board computers* (SBC). SBCs are complete computers built on single circuit boards, where the user can run applications found on personal computers, such a word editing software or even games[25][27]. But more importantly, they have built-in connectivity and signaling features ready for embedded use such as *SPI* and *I2C buses*, (*GPIO*) pins, *PWM* outputs, analog inputs and timers. In comparison to *micro controller*s, which are boards with an integrated microprocessor and RAM [28], SBCs have higher power usage, but offer superior computational resources, averaging clock speeds around 1 Ghz[29]. This makes the familiar Arduino microcontroller family less desirable as their strongest board, the Arduino Mega, only runs at 16 MHz.

By the beginning of the project, two Arduino Mega's and the SBC BeagleBone Black were already at disposal, albeit the task at hand was to identify a better alternative SBC. As of 2019, most SBCs are being sold for between 80 and 100 USD. Due to the fact that a SBC already was acquired, the only price-appealing alternative worth considering was the Rasberry Pi 3 Model B for 40 USD. It has the quadratic core ARM Cortex-A53 processor running at marginally better 1.2 GHz. In addition to having inferior connectivity options, the group opted for the BeagleBone Black.

Equipped with 2x 46 pins, its digital pins deliver 3.3V which makes it suitable in combination with the LSM9DS1, see chapter 4. It features 8 PWM outputs which makes it desirable for controlling the biped's multiple servos. In addition, it also features a Quadrature Encoder interface (eQEP) module which makes it ideal for reading encoders. BeagleBone Black uses Linux Debian 9.5 as operating system and is able to run programs coded in several different languages for example java, python, C and C++.

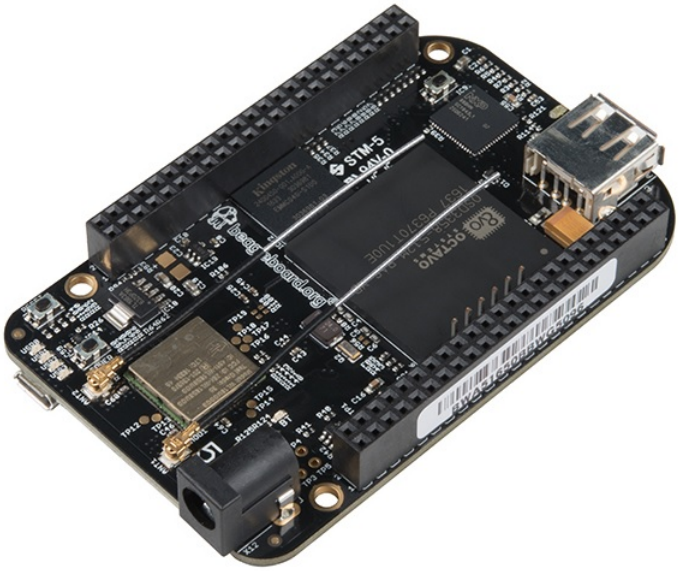


Figure 2.1: A BeagleBone Black [1]

2.1 BeagleBone Setup

The BeagleBone should be setup according to the getting started guide[30], which involves flashing the board with a new image using an SD-card.

After successful flash, the BeagleBone can be connected to using Secure Shell. (subsection 2.7.1) To download additional packages, internet can be bridged from the connected computer. There are multiple guides available for Windows[31] and Linux [32]. Alternatively, packages can be downloaded on a internet-connected computer, and shipped over using the scp-command.

The Build-essentials package is neccessary for on-board development. This package includes GNU compilers, libraries and package development tools. Listing 1 updates package list and installs the newest build-essential with root privileges.

```
root@beaglebone:~# apt-get update
root@beaglebone:~# apt-get install build-essential
```

Listing 1: APT package installation

Device tree overlays needs to be enabled in order to use the board's Programmable Realtime Units. This is done by uncommenting lines in the /boot/uEnv.txt-file:

```
nano /boot/uEnv.txt
******************/boot/uEnv.txt*******************
...
###pru_rproc (4.4.x-ti kernel)
uboot_overlay_pru=/lib/firmware/AM335X-PRU-RPROC-4-4-TI-00A0.dtbo
###pru_rproc (4.14.x-ti kernel)
uboot_overlay_pru=/lib/firmware/AM335X-PRU-RPROC-4-14-TI-00A0.dtbo
###pru_uio (4.4.x-ti, 4.14.x-ti & mainline/bone kernel)
uboot_overlay_pru=/lib/firmware/AM335X-PRU-UIO-00A0.dtbo
...
```

Listing 2: Enable device tree overlays

Listing 2 shows overlays which have been enabled by uncommenting the *uboot_ overlay**-lines. This enables the drivers for *remoteproc* and the *AM335x Application Loader* after reboot. (Subsection 3.2.4)

2.2 Code Composer Studio

Texas Instruments has its own Eclipse IDE-based development platform. CCS adds compilers for the Programmable Realtime Units, and can easily be integrated with the Cortex A8 cross-compiler toolchain.

2.2.1 Installation

This installation is done on Ubuntu 18.04.2 LTS, but can be installed on Windows and MacOS (64-bit)

Download the latest CCS version from TIs wiki[33] (CCS 9.0.0.00018 used in this installation) Decompress file and run the installation executable:

```
1 /home/user/Downloads# tar -xvzf CCS<version>_linux-x64.tar.gz
2 /home/user/Downloads# cd CCS<version>_linux-x64
3 ../Downloads/CCS<version>_linux-x64# ./ccs_setup_linux...
```

Listing 3: GCC example commands

The setup wizard may require additional libraries preinstalled. These should be downloadable from the default sources using the Advanced Package Manager. (Commands: apt-cache search 'keyword' and apt-get install 'package')

In the setup wizard, select Sitara AMx Processors under processor support, then complete the installation.

After completion, CCS can be opened from the optional Desktop shortcut, or an alias can be addded to run the executable. (See subsection 2.7.3)

```
1 alias ccs='/opt/ti/ccs900/ccs/eclipse/.ccstudio'
```

Listing 4: Alias for CCS

The current version of CCS only supports Eclipse 3.8 (Juno). Packages by Eclipse needs to be installed from the Juno repository, which can be configured under 'Install new software'(Figure 2.2).

By adding a repository-link, the Remote Systems packages can be accessed under 'General purpose tools' (Figure 2.3/2.4)

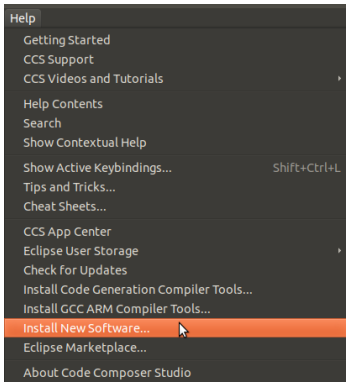


Figure 2.2: Install new software

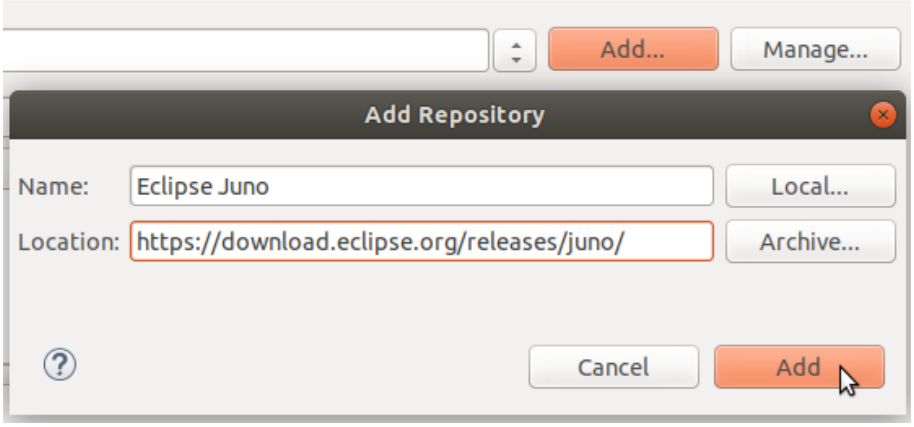


Figure 2.3: Adding a new repository

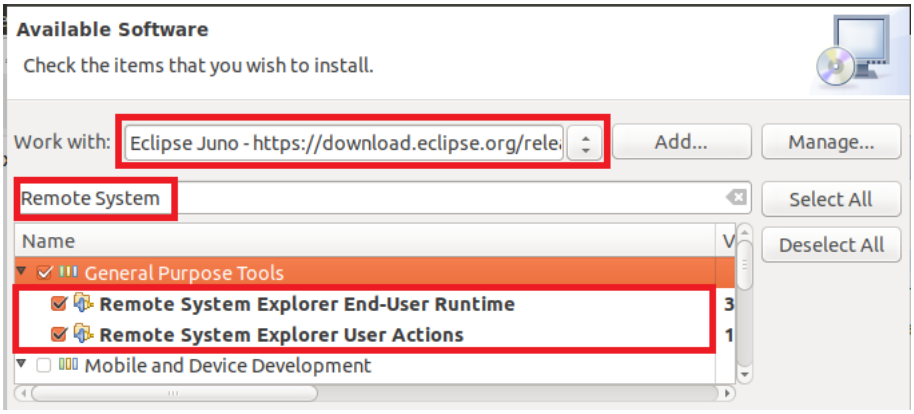
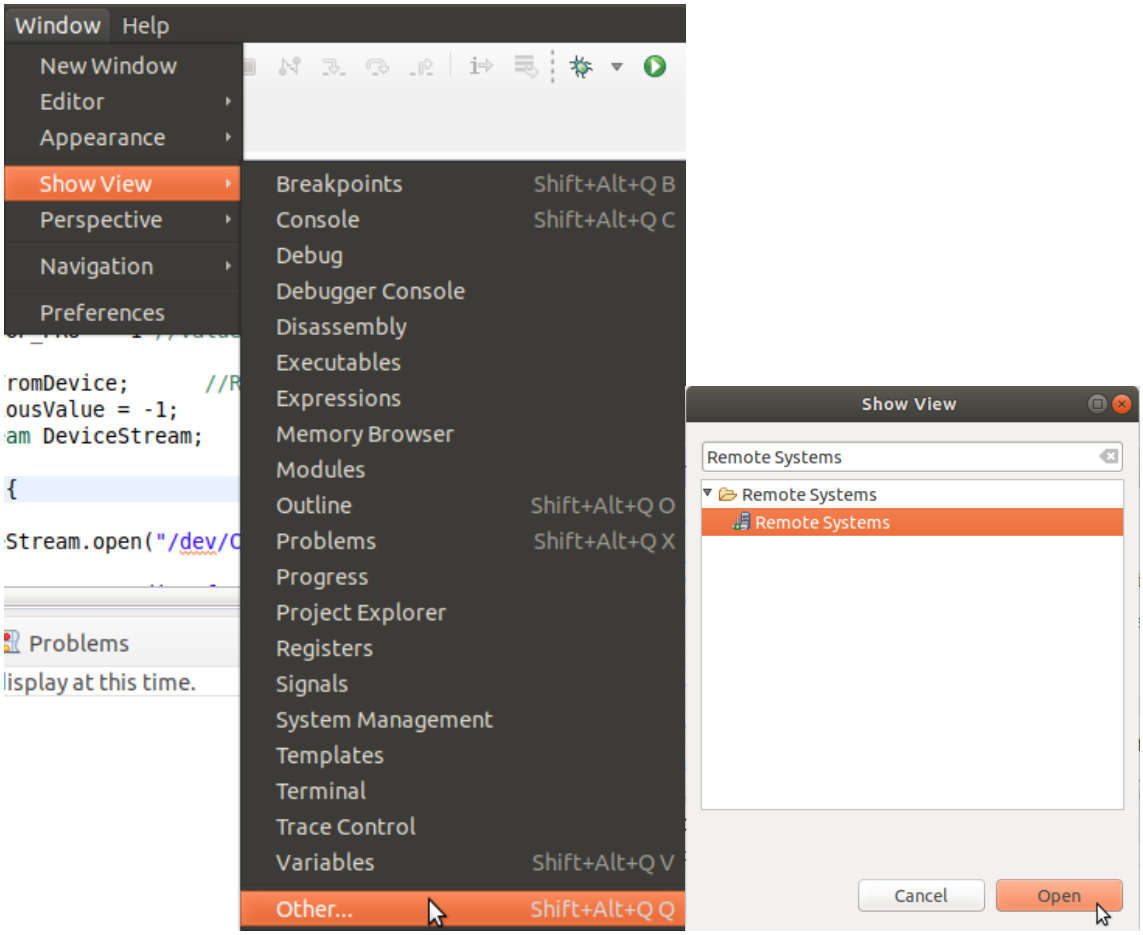


Figure 2.4: Remote System packages

2.2.2 Remote System Explorer

A connection can be made directly from CCS to BeagleBone with SSH in order to transfer files and access directories. The Remote Systems view can be opened from the 'Window'-options list. (Figure 2.5)

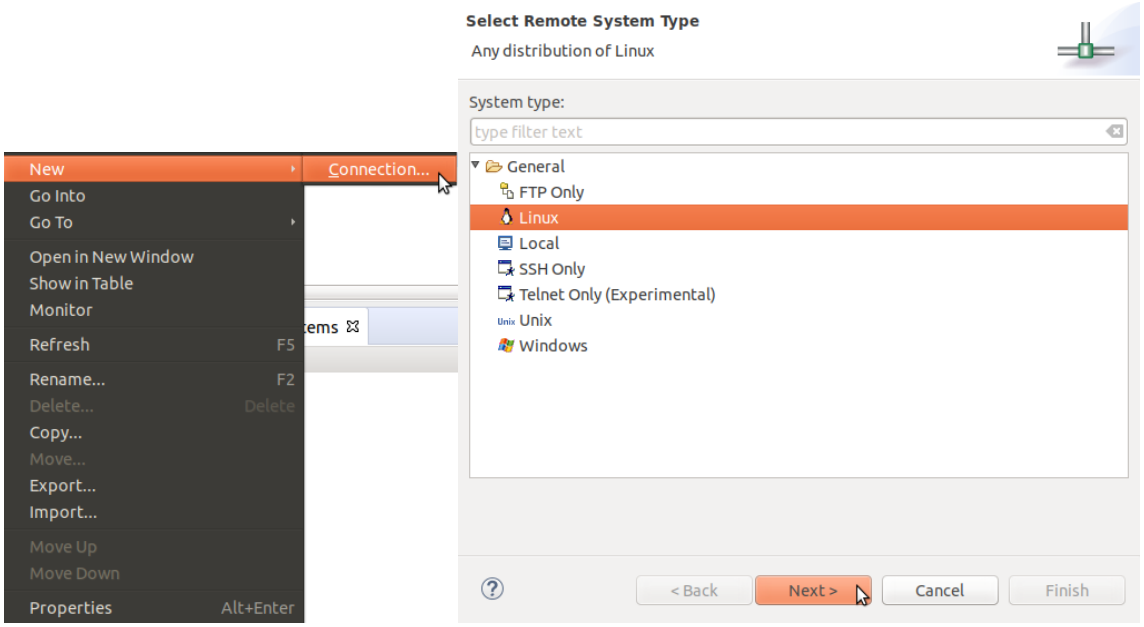


(a) (b)

Figure 2.5: Opening remote system explorer

A new Linux connection can be made from this view (Figure 2.6)

The following SSH-settings needs to be set, including BeagleBones default ip (Figure 2.8a and 2.8)



(a) (b)

Figure 2.6: New Remote System connection 1

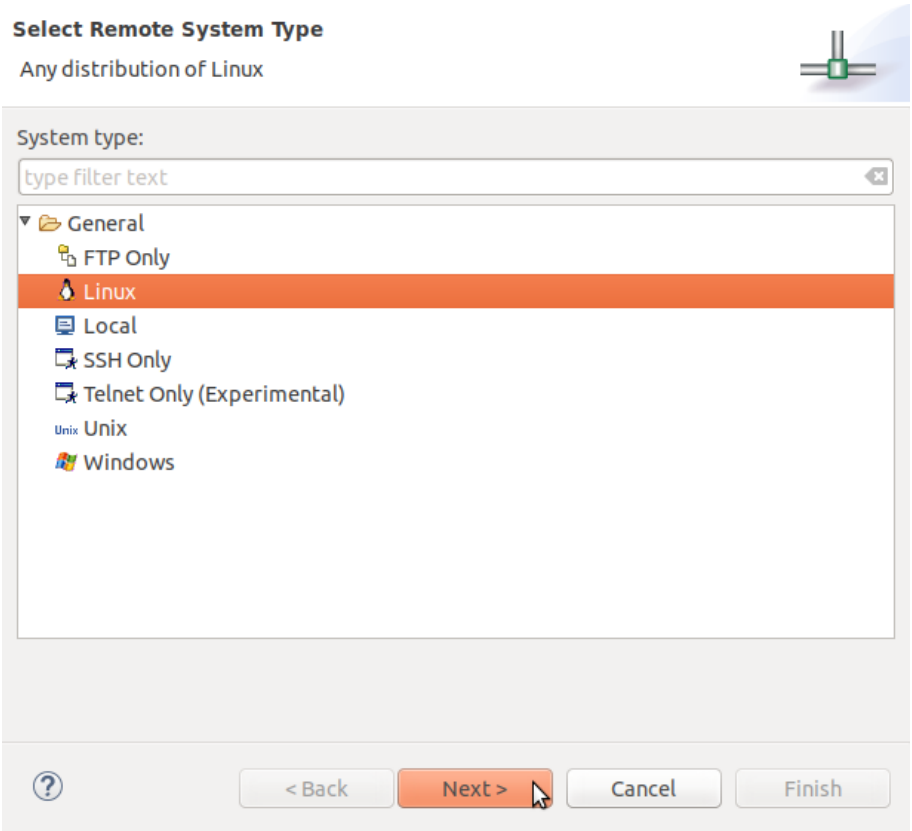
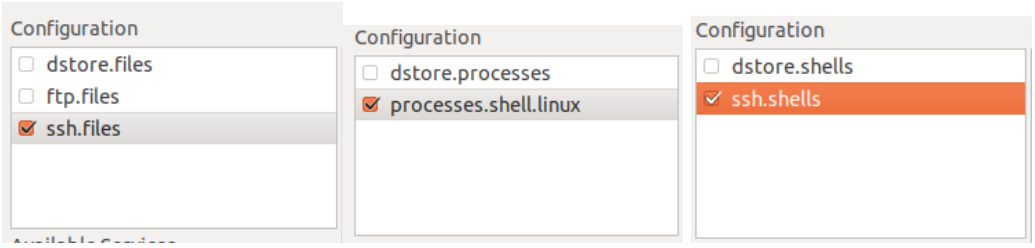


Figure 2.7: Remote System connection type

Login will be promted when trying to access directories within the connection. This explorer allows file transfer directly to/from the workspace.



(a) (b) (c)

Figure 2.8: New Remote System connection settings

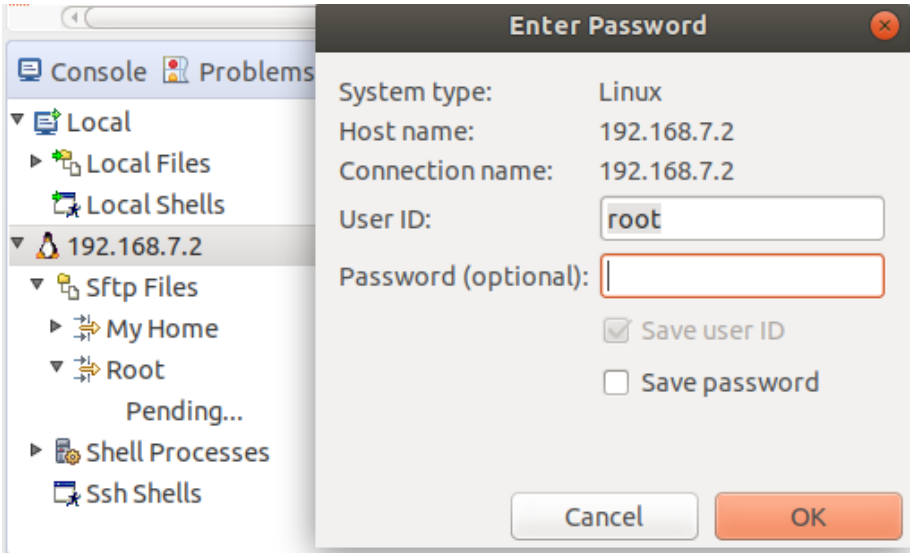


Figure 2.9: Remote System login

2.2.3 PRU projects

In order to create PRU-targeted projects in CCS, the PRU Compiler must be added from the CCS App Center. (View → CCS App Center) After restarting and installing the new compiler, CCS projects can be made specifically for the BeagleBone Black AM335x-processor in C/Assembly.

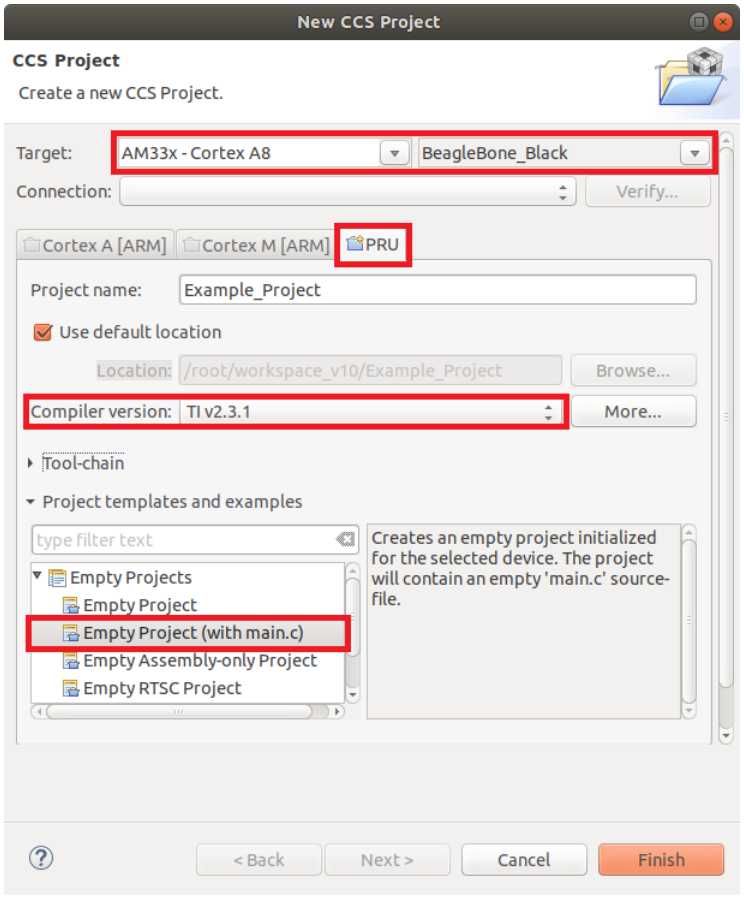


Figure 2.10: CCS project example

To convert the resulting *.obj-file to binary, a CCS conversion utility can be configured in post-build steps in project settings (Figure 2.11)

Listing 5 shows the post-build step instruction

```
"${CCS_INSTALL_ROOT}/utils/tiobj2bin/tiobj2bin" "${BuildArtifactFileName}"
"${BuildArtifactFileBaseName}.bin" "${CG_TOOL_ROOT}/bin/ofd2000"
"${CG_TOOL_ROOT}/bin/hex2000" "${CCS_INSTALL_ROOT}/utils/tiobj2bin/mkhex4bin"
```

Listing 5: PRU Project post build steps

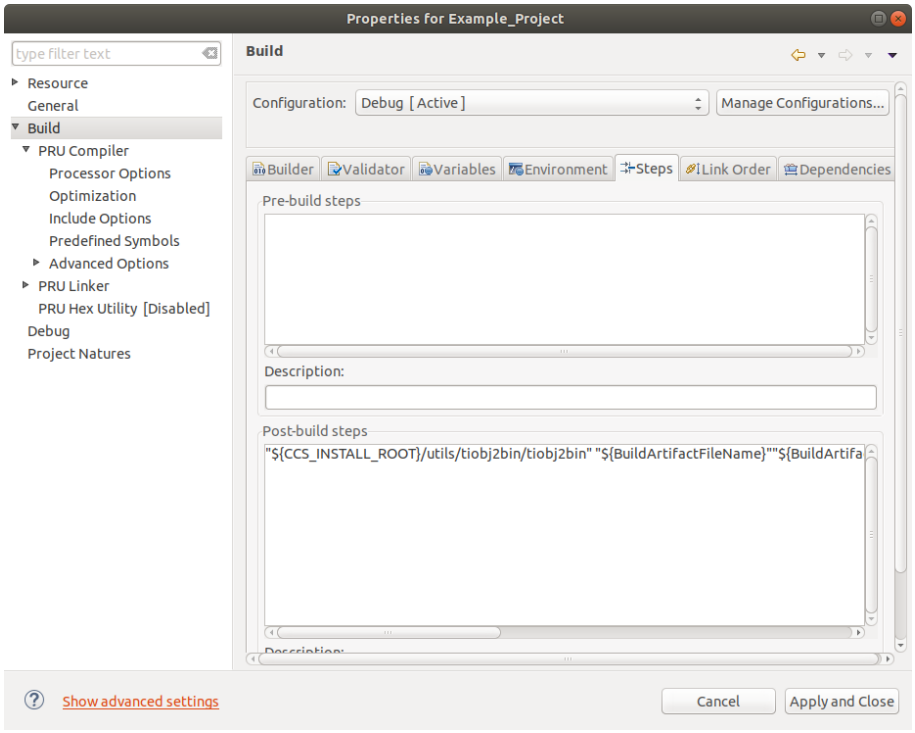


Figure 2.11: Object to binary conversion settings

2.2.4 Launch configuration

A custom bash script can be used to transfer, execute and retrieve console log from projects created in the workspace:

```
#!/bin/bash
BINARY=$"BinaryFileName"
EXECUTABLE=${PWD##*/}
TARGET_PATH=$"/root/DesiredProjectFolder/"$EXECUTABLE"/"
BINARY_PATH=$(cd ..;find $BINARY | grep .bin)
WORKSPACE_PATH=${PWD%/*}
EXECUTABLE_PATH=$PWD"/Debug/"$EXECUTABLE
ssh root@192.168.7.2 "mkdir -p $TARGET_PATH;cd $TARGET_PATH;"
scp $EXECUTABLE_PATH root@192.168.7.2:$TARGET_PATH
scp $WORKSPACE_PATH"/"$BINARY_PATH root@192.168.7.2:$TARGET_PATH
ssh -tt root@192.168.7.2 "(cd $TARGET_PATH;chmod a+x $EXECUTABLE;./
    $EXECUTABLE)"
```

Listing 6: Project deploy and execute script

Listing 6 identifies current project name by directory and finds its executable file in /Debug/. An additional binary file, from a separate project in the same workspace is searched for using the *BINARY*-variable.

With the all executable paths identified, the project can be transfered to desired *TARGET_PATH* using the scp-command. The connection is closed by the end of program execution. (-tt option)

By including a copy of a deploy-script inside a project, the execution can be automated using external tools configuration:

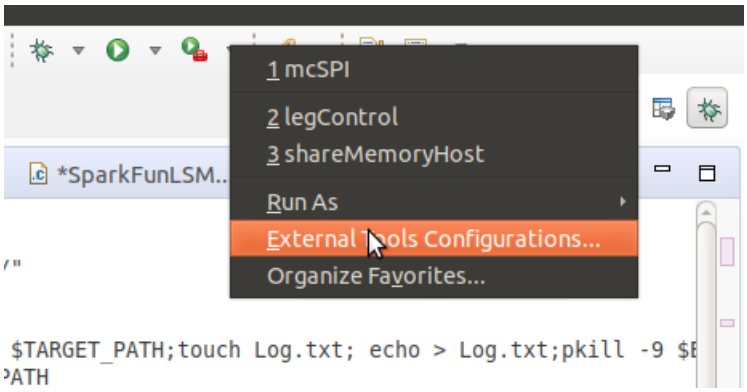
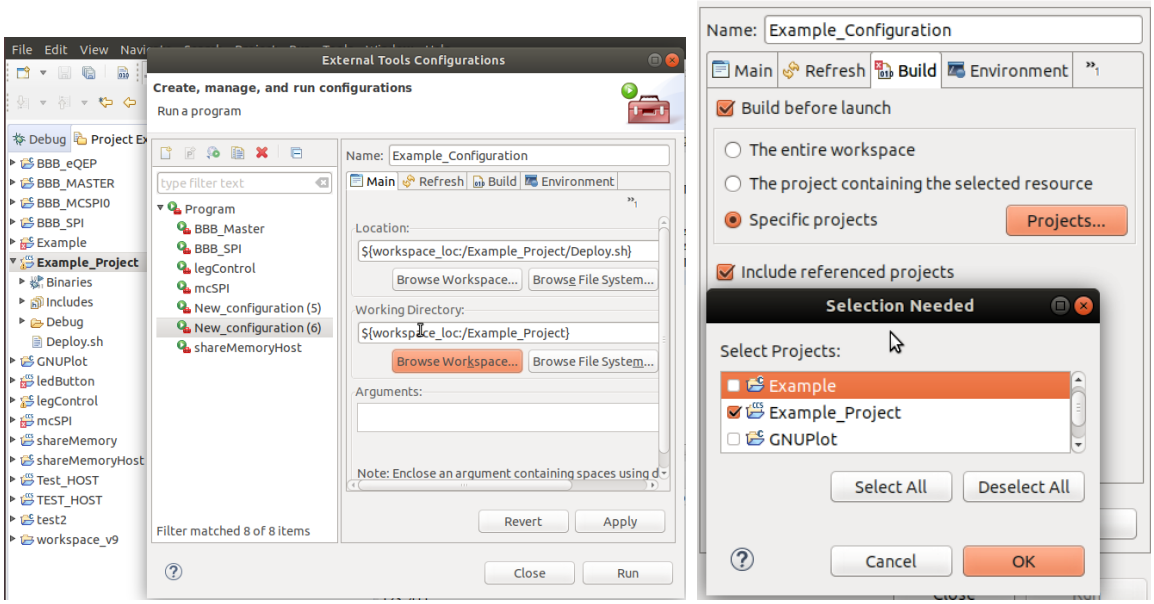


Figure 2.12: External Tools location



(a) Launch path configuration (b) Launch build configuration

Figure 2.13: Launch configuration settings

Figure 2.13b configures multiple project-build which can be used with a custom bash file to transfer additional PRU-projects. (Included in documentation)

2.3 Compilers and toolchains

BeagleBone Black has two different processor architectures, which requires different software to translate C/C++ code into readable machine code. To be able to setup code translation correctly with IDEs (and for debugging purposes), a basic understanding of the translation process is useful.

2.3.1 Overview

Compiling C/C++ code is (in general) the same process regardless of toolchain, and follows the same steps as the GCC-compiler:

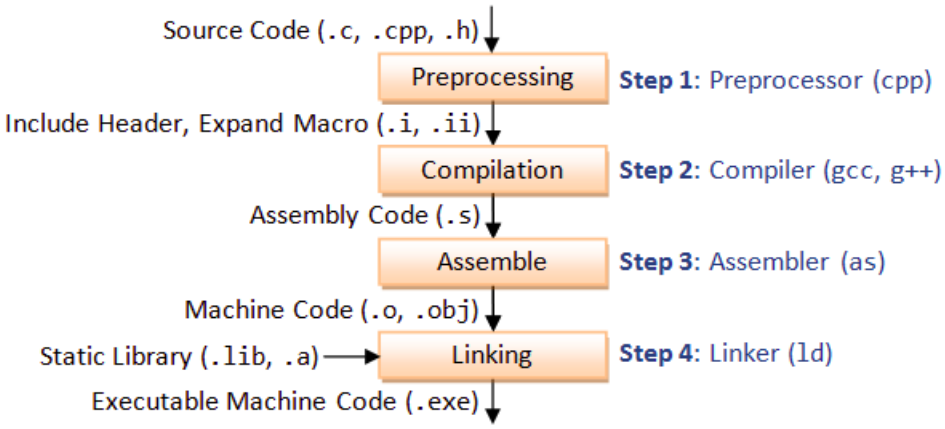


Figure 2.14: GCC compilation process[2]

During reprocessing, all #includes are replaced by corresponding headers, and all defined macros are replaced with their definitions. All comments and unnecessary text is removed.

The *compiler* translates all resolvable code with included headers to assembly code. (some functions stay untouched, requiring linking)

The *assembler* translates the result to machine code, still leaving unresolved code from the libraries untouched.

Finally, the *linker* fills in the necessary machine code compiled from the libraries source files, which results in an executable file. The default format is an Executable, Linkable Format (ELF)

The ELF-format uses page tables to locate the required content of the executable, allowing the information to be stored more flexibly

2.3.2 GNU Arm Embedded Toolchain

Compiling code for the Arm Cortex A8 follows the process in fig.2.14, using the *GNU Arm Embedded Toolchain*, consisting of:

- GNU C/C++ compiler - Responsible for preprocessing/translation to assembly
- Binutils - Responsible for Assembly to machine code translation and linking
- GDB - Additional debugging tools (Not used in this project)
- NewLib - Standard libraries practical for embedded systems

The GNU C/C++ compiler is configured to use the binary utilities to compile executables in a single command. In the Linux shell, the compiling can be stopped at any step using -E, -S and -C ,in order to produce pre-processed (*.i), assembly (*.s) and unlinked (*.o) files. The example in listing 7 shows translation from c to assembly code, and dynamical linking of shared libraries, with custom output names.

```
1 root@beaglebone:~# gcc -S main.c -o main.s
2 root@beaglebone:~# gcc main.c -o main.o -lSharedLibrary
```

Listing 7: GCC example commands

Note: This toolchain is included in the latest Debian images for BeagleBone Black

2.3.3 PRU software development tools

The programmable realtime units (see chapter 3.2), runs compiled assembly instructions defined by Texas Instruments. This requires a custom c/c++ compiler and binary utilities. Tools and usage is described in TI's users guide[34], which provides an overview of development flow:

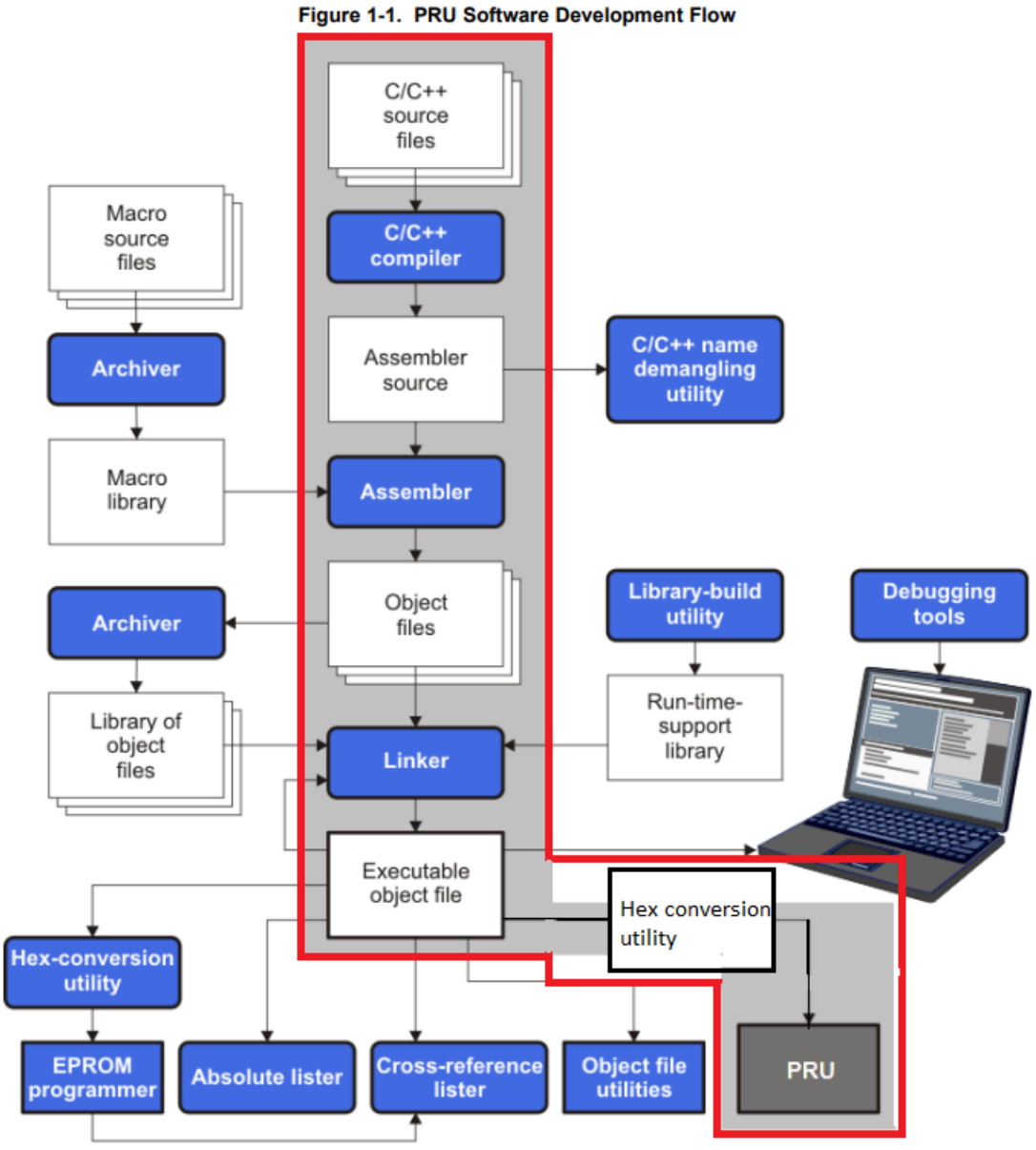


Figure 2.15: PRU software development flow [3]

This toolchain follows the same flow as GCC, with the exception that the executable file needs to be in one complete file. Memory page tables and dynamic direction is run by the linux kernel and can only be used by the main processor. The EPROM-programmer, which is responsible for loading the application, requires guidelines. The format is resolved by using the included Hex conversion utility[35, p. 296]. Listing 8 converts a compiled object file to binary by specifying output format and desired name.

```
1 ~/exampleProject# ls
2 PRUProgram.obj  hostProgram
3 ~/exampleProject# hexpru PRUProgram.obj -b -o PRUProgram.bin
```

Listing 8: Hexpru object conversion

Note: The software development tools are preinstalled in the latest Debian images for BeagleBone Black. This conversion can also be done with a conversion tool in CCS (section 2.2.3)

The executable binary file can be loaded with a userspace driver by TI named libprussdrv, included in the AM335x support package. (See subsection 3.2.4)

2.3.4 Loadable Kernel Modules

Loadable Kernel Modules (LKMs) are used to extend the functionality of the kernel (subsection 2.7.5), making it possible to change how hardware is interfaced while the operating system is running. Programming drivers to run in kernel space requires appropriate header files, which can be installed using the Advanced Packaging Tool (Listing 9)

```
root@beaglebone:~# apt-get install linux-headers-$(uname -r)
```

Listing 9: Linux header files installation

$(uname -r) refers to current kernel version.The header files will be installed under /usr/src/linux-headers-$(uname -r)/. *.c-files can be compiled by redirecting it to a *Makefile*(Subsection 2.7.2), located in the linux-header folder.

Listing 10 configures the GNU toolchain to compile kernel-compatible code. On first line, a target is defined. Directory of operation is changed to the linux-headers Makefile, where it will be invoked with optional arguments (modules/clean)

```
obj-m+=hello.o
all:
make -C /lib/modules/$(shell uname -r)/build/ M=$(PWD) modules
clean:
make -C /lib/modules/$(shell uname -r)/build/ M=$(PWD) clean
```

Listing 10: Linux-header Makefile example

Listing 11 shows exampleCode.c compiled with using *make*

```
root@beaglebone:~/example# ls
exampleCode.c  Makefile
root@beaglebone:~/example# make
make -C /lib/modules/4.14.71-ti-r80/build...
..Leaving directory '/usr/src/linux-headers-4.14.71-ti..'
root@beaglebone:~/example# ls
exampleCode.c   exampleCode.mod.c  exampleCode.o  m..
exampleCode.ko  exampleCode.mod.o  Makefile       M..
root@beaglebone:~/example#
```

Listing 11: Linux-header Makefile example[19]

This results in many outputs, where exampleCode.ko is the loadable kernel module. *.ko modules can be inserted/removed into/from the kernel using commands *ins*-and *rmmod*.

Note: The author of "*Exploring BeagleBone:* .."[36] has introductory examples available online[19].

2.4 Main program

The main idea for the program was going to be a framework; a collection of module-based libraries for the peripheral devices of the robot. The purpose was to create simplistic interface to main, furthering easy intuition for the future work. This was mainly done through use of Object-Oriented Programming, allowing introduction of *abstraction* and hide the complexity of controlling the robot.

2.4.1 Abstraction, Modularity and Object-Oriented programming

As a program grows, the maintainability becomes gradually more difficult. Every programmer has a definite threshold of how many lines of code they are able to efficiently maintain when developing. This threshold may vary, but as it is passed, development becomes exponentially more difficult. In order for the program to be able to grow past this threshold, one has to use abstraction, reducing one's interface to the complexity of the program. To achieve this, modularity have to be implemented.

A module is a enclosed portition of the program which is designed to serve a specific purpose. Its interface to the rest of the program is strictly limited to a set of type-predetermined inputs and outputs. Hence, development of intended behaviour narrows down to the various circumstances created by the given inputs. When utilizing such modules at a later stage of development, one is able to think of the module as a black box, creating an intuition of what it does, not how it is being done. By doing so, one is able to reduce the train of thought necessary to develop larger algorithms. Modules are usually defined as, but not limited to, objects, libraries, functions and headers.

In this project, modularity is mainly achieved through Object-Oriented Programming. Objects are data structures that contains both data and functions. In addition, they are eligible to have exclusive relations to other objects. The only type of data structure used in this projects is called classes. A class contains member variables and functions which are available to a predetermined environment, which in turn creates modularity. Among a class' member functions, there is always at least one constructor and destructor. When an instance of a class is created, the constructor runs, ensuring the contents of the class is being managed correctly. When the object has reached the end of its lifetime (when program sequencing runs out of the scope), the destructor is called, resulting in used computer resources being freed in a proper manner. During the instance lifetime, it used as an abstraction, handling complex operations through its various member functions.

2.4.2 robotLeg Class

By implementing a larger object called "robotLeg", an unified abstraction of all the implemented libraries were achieved. The class contains an instance of every peripheral class in addition to functionality which handles their respective interfaces. This results in reduced complexity when measuring and actuating the biped robot.

```
class robotLeg {
private:
  LEG leg;
  static SPIBus SPIDEV;
  LSM9DS1 IMU;
  encoder enc;
  motor mot;
  servoPair ser;

   ...
public:
  robotLeg();          //  Default constructor creates an instance of type INNER.
  robotLeg(LEG leg);
  //feedback
  int readParameters();   //  Updates all feedback variables (encoder and pitch
    angle).
  std::string getLeg();   //  Returns which leg currently instanciated. Useful for
    error handling.
  double getEncAngle();   //  Returns measured angle between body and leg
  double getPitch();      //  Returns legs current pitch relative to world.
  double getMotAct();     //  Returns current motor actuation.
  bool getSerState();     //  Returns servo state, true = fully extended, false =
    retracted.
  //actuators
  void setMotAct(double act); /*  setMotorActuation, act : {-100.0 : +100.0} ->
    ~(-3 : 3) Nm. Maximum coninous signal: 38 % = 3.17 Amper ~ 1.147 Nm */
  void setServo(bool act);  /*act: True = Fully extended. False = Retracted*/

};
```

Listing 12: The "robotLeg" class, centralizing use of the peripheral libraries.

To control either of the robots legs, one only need to create an instance of "robotLeg" and use its member functions such as demonstrated bellow.

```
innerLeg robotLeg(INNER);
outerLeg robotLeg(OUTER);
//feedback
innerLeg.readParameters();
std::cout << "Inner Leg's encoder angle: " << innerLeg.getEncAngle();
std::cout << ",   Pitch: " << innerLeg.getPitch();

  ...
//Actuation
outerLeg.setMotAct(40.0);
outerLeg.setServo(true);
```

Listing 13: Demonstration of using the "robotLeg" class.

2.4.3 File interaction and Structure

Utilizing peripheral devices with the BeagleBone Black is mainly done like any other task on a Linux/Unix OS; it is just a matter of handling a bunch of file streams as "*Everything is a File*"[37]. In programming, both writing and reading files are tedious tasks and takes a relatively long time. To demonstrate, by running the following snippet of code on the BeagleBone Black, one could measure the required runtime of doing the simplest of file interaction in C++.

```
1  //Writing
2  timePoint timeEnd, timeStart = Time::now();
3  std::ofstream writeStream("test.txt");
4  writeStream << "a";
5  writeStream.close();
6  timeEnd = Time::now();
7  duration = timeEnd - timeStart;
8  std::cout << "Writing: " << duration.count() << "ms";
9  //Reading
10 char c;
11 timeStart = Time::now();
12 std::ifstream readStream("test.txt");
13 readStream >> c;
14 readStream.close();
15 timeEnd = Time::now();
16 duration = timeEnd - timeStart;
17 std::cout << ", Reading: " << duration.count() << "ms" << std::endl;
18 std::cout << "char: " << c << std::endl;
```

Listing 14: Timing basic file interaction on the BeagleBone Black

This would on average have the following output:

```
Writing: 0.0017ms, Reading: 0.00018ms
char: a
```

Listing 15: Results of listing 14

Since these operations will be executed several tens of times through a single run of the loop (see figure 2.16), the runtime dedicated to file interaction will have a major impact on the overall *sample time*. Given how crucial it is for the stabilizing algorithm to be able to calculate and execute corrective action within a certain deadline, it is imperative to make effort to reduce the time dedicated to sampling and logging. Therefore, all file interaction should be as limited as possible. This led to the decision of having the inputs read once in the beginning and log the state variables in the end of the running loop. Thus, the running course of the main thread would be the following figure 2.16.

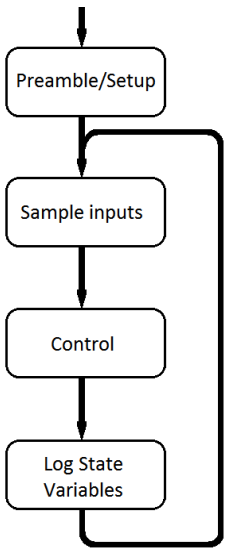


Figure 2.16: Flowchart of the main thread

In figure 2.16, the general contents of "Preamble/Setup" and "Log State Variables" are described in section 2.4.4. "Sample Inputs" would simply be calling the "readParameters" member function of each instance the "robotLeg" class. By doing a timing test similar

to listing 14, one would on average measure a time consumption of $1.8ms$. The "Control" segment is intended for a future implementation of a trajectory following and stabilizing controller.

2.4.4 Utilities

In addition to the peripheral libraries, functionality handling the file interaction were necessary.

2.4.4.1 Importing Configuration Parameters

In the flowchart (figure 2.16), "Preamble/Setup" would consist of instantiating the "robotLeg" class and reading configuration parameters from a predisposed text file called "config.txt". Achieving the latter would require more effort than presumed as the cross compiler, at the time of development, would not resolve usage of the standard container: *map*. Instead, a custom map container class was created, named "strstrMap". It would have the same properties as a standard map, instantiated with both keys and values declared as strings. Maps are very useful to the extent of reading parameters from a configuration file, as one wishes to extract whatever is in it into pairs of keys and values.

```
logSampleTime= 50
testChoice= SINE
SineAmp= 30.0
Kp= 1.0
Ti= 10.0
Td= 1.0
Ts= 0.001
```

Listing 16: An example of what config.txt could look like

After reading through the config.txt file, the configuration parameters may be assigned to their declared variables by accessing the map by intuitively stating their name as a key. The values are returned as strings and need to be converted to the appropriate type depending on the declaration of the variables.

```
1  int logSampleTime;
2  string testChoice;
3  double SineAmp, Kp, Ti, Td, Ts;
4
5  strstrMap configStore;
6  mapConfig(configStore, "/root/config.txt"); //Inserts the contents of "/root/config
       .txt" into "configStore" in pairs of keys and values
7
8  logSampleTime = stoi(configStore["logSampleTime"]); //stoi() - Convert string to
       integer
9  testChoice = configStore["testChoice"];
10 SineAmp = stod(configStore["SineAmp"]);    //stod() - Convert string to double
11 ...
12 Ts = stod(configStore["Ts"]);
```

Listing 17: An example of how to assign the configuration parameters

During development, *segmentation faults* would be produced at seemingly random after implementing the "strstrMap" class. This is an error that is induced when the software tries to access

restricted memory. It is commonly produced after trying to *dereference* a *null-pointer*[20]. One would thereafter troubleshoot all dereferencing within the class, which yielded no success. It was eventually noticed that the segmentation fault would only be induced if the number of configuration parameters surpassed ten. For every tenth configuration parameter, strstrMap would try to *allocate* ten more bytes of memory by replacing its *memory block* with a *deep copy* of itself which was ten bytes larger. This became a moment of realization as the deallocation showed to be wrongfully handled; it would attempt to deallocate memory outside the defined size of the memory block.

```
68 void strstrMap::allocMore() {
69   asize += 10;
70   std::string **newMap = new std::string*[asize];
71   for (int i = 0; i < asize; i++) {
72     newMap[i] = new std::string[2];
73     if (i < size) {
74       newMap[i][0] = dynMap[i][0];
75       newMap[i][1] = dynMap[i][1];
76     }
77     delete[] dynMap[i]; //<---- wrong!
78   }
79   delete[] dynMap;
80   dynMap = newMap;
81 }
```

Listing 18: Wrongfully deallocating of memory

The fix was rather easy, but proves how detrimental improper handling of memory allocation can be.

```
71   for (int i = 0; i < asize; i++) {
72     newMap[i] = new std::string[2];
73     if (i < size) {
74       newMap[i][0] = dynMap[i][0];
75       newMap[i][1] = dynMap[i][1];
76       delete[] dynMap[i]; //<---- ok!
77     }
78   }
```

Listing 19: Correct deallocating of memory

2.4.4.2 Logging State Variables

Logging state variables is done by using the implemented logSample function:

```
1 int logSample(double par[], int size, const char* path, bool reset)
```

Listing 20: The prototype of the logging feature; "logSample"

The function will create or overwrite a text file called "measurementLog" located at a given file path: "path". Furthermore, it will write a single line with the data passed to it in "par", where each element is divided by a semicolon. If the "reset" parameter is passed as false, logSample will append the line to the existing data.

By timing the function in the same manner as in listing 14, one would average a time consumption around 0.4ms. If this would prove to be inadequate, one could reduce average sample time by reducing the frequency of logging, sacrificing data resolution.

```
127     double par[] = { sampleTimeStampMS * 1000,
128             innerLeg.getEncAngle(),
129             outerLeg.getEncAngle(),
130             innerLeg.getPitch(),
131             outerLeg.getPitch(),
132             innerLeg.getMotAct(),
133             outerLeg.getMotAct(),
134             innerLeg.getSerState(),
135             outerLeg.getSerState()
136     };
137     if  ((int(sampleTimeStampMS) % logSampleTimeMS) == 0){  //Initiates logging
    every [logSampleTimeMS] milliseconds.
138       if (logSample(par, 9, logPath, false) < 0) {
139         std::cout << "logSample failed." << std::endl;
140         break;
141       }
142     }
```

Listing 21: A demonstration of logging at a reduced frequency

2.4.5 Sample Time Considerations

Thus far, it was measured that the average time consumption of the "Sample Inputs" and "Log State Variables" segments in figure 2.16 were 1.8ms and 0.4 ms. This makes up 2.2ms of active run time, which is ideal. As the current solution utilizes the standard IMU configuration, which is run with a static sample time of 25 ms, the sequencing of the loop has to be halted for the remaining 17.8ms. Static sample time is achieved by having the sequencing enter a while loop where the condition is a timer being lower than the desired sample time.

```
while (1) {
  //Sample Inputs..
  //Control..
  //Log..

  while (desiredStaticSampleTime > loopTimer){
    //Update loopTimer..
  }
}
```

Listing 22: A demonstration of inducing a static sample time

2.5 IMU Library

Instead of recreating software to be able to read measurements from the IMU, a library can be ported. The SparkFun LSM9DS1 comes with library support for Arduino. In order to use its functions, all information transmitted through the library, needs to be redirected.

2.5.1 Overview

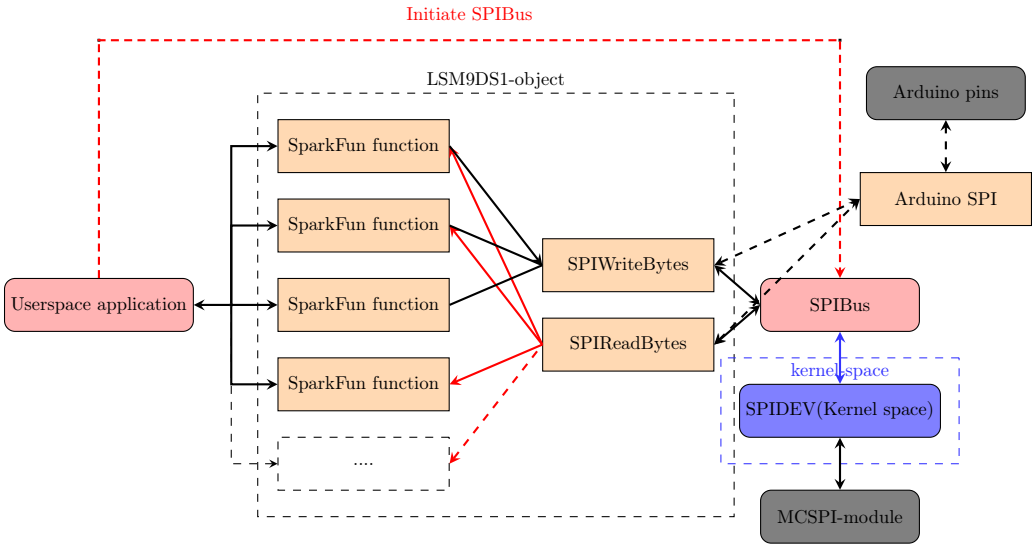


Figure 2.17: IMU Library Flowchart

Figure 2.17 illustrates the information flow from the original library, and the redirection for BeagleBone. The original library supported both I2C and SPI-communication. I2C has been removed, and all information passed through SPI, is passed onward through a custom SPI-module object.

2.5.2 SPIBus objects

The first object to be initiated is from SPIBus class. This class configures one of the MCSPI-modules on the board from userspace, through the SPIDEV driver. SPIDEV is a loadable kernel module which configures a character device (subsection 2.7.6), allowing userspace applications to interact with hardware peripherals. BeagleBone Black has two MCSPI-modules[6, p.4883] associated with SPIDEV1.x and 2.x in the /dev/ -directory. Creating an object without arguments results in initialization of SPIDEV1.x.

2.5.3 LSM9DS1 objects

Objects for the IMUs require a SPIBus object as input. The class has been modified to call a member function from the SPIBus object in order to send/read data. The class supports two default initializations, which configures each IMU device to be identified with different pairs of CS-Pins (For pin definitions and communication, see subsection 4.1). Additional devices can be added by adding additional path-parameters for CS-Pins. This will initialize a different constructor.

A calibration member function has been made in the LSM9DS1-object. By sampling multiple stationary values, it is able to find start value and offset for angle calculations. (Appendix B.1)

2.5.4 Custom functions library

An additional library has been added for functions too specific to be contained within the objects. These functions configures default parameters, initializes calibration and checks communication lines.

2.5.5 Abstraction

The library is a part of of the legControl-object (Subsection 2.4.1 and 2.4.2). A static SPIBus-object for SPIDEV0 is initialized in legControls constructor. The object reads clockwise around the IMUs z-axis by default, which is the relevant axis when leg-mounted.

```
int main()
{
    robotLeg innerLeg(INNER);
    robotLeg outerLeg(OUTER);
    while(1)
    {
        innerLeg.readParameters();
        outerLeg.readParameters();
        std::cout << innerLeg.getPitch() << ", ";
        std::cout << outerLeg.getPitch()<< std::endl;
        usleep(DELAY);
    }
}
```

Listing 23: Reading leg angles

Listing 23 shows two legControl-objects, which configures IMUs to default CS-Pins. Functions from the custom library are called to setup default parameters and calibration.

Note: The custom functions library (LSM9DS1_Custom_Functions) is based on examples from the LSM9DS1 github repository[38]

2.6 Result

The ported library is able to transmit and receive the same data as the SparkFun-Library would for Arduino.

2.7 Linux

This subsection introduces the features of Linux relevant to this project. It is intended to summarize the features, since in-depth documentation is easily accessible online.

2.7.1 Secure Shell

SSH can be used to communicate safely between devices, through wired and wireless connections. When logging in to another device, encryption data is sent prior to the actual data transmission. This opens for a temporary ssh-session. A keypair is distributed between the communicating devices, telling both how to encrypt/decrypt the data.

Permanent host and private keys should be exchanged for devices frequently connected. By permanently storing a private/public-keypair between the devices, it is possible to perform passwordless logins. Procedure for generating and exchanging keys are shown in listing 24:

```
root@Ubuntu:~# ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key ...id_rsa): /root/.ssh/
    id_example_BBB
...
root@Ubuntu:~# ssh-copy-id -i /root/.ssh/id_example_BBB root@192
    .168.7.2
root@Ubuntu:~# ssh -i /root/.ssh/id_example_BBB root@192.168.7.2
```

Listing 24: SSH key exchange

SSH-keygen and SSH-copy-id generates a new, custom keypair and sends the public key to the BeagleBone. Passwordless connection can now be made by specifying which keypair to use when using the ssh-command. This only applies for the logged in user (root) at the ipv4-address specified (192.168.7.2).

2.7.2 Makefile

A makefile is used to automate command processes by defining targets and recipes. The makefile is usually structured with subtargets and subrecipes, which must be fulfilled in order to approach parent targets.

```
----------------Makefile-------------------
Main_Target: SubTarget0 SubTarget1
        cat file0.txt  file1.txt
SubTarget1: file0.txt
        touch file1.txt
        echo "Hello" > file1.txt
SubTarget0:
        touch file0.txt
        echo "World " > file0.txt
```

```
10 ----------------------------------------
11 root@Ubuntu:~# make
12 ...
13 Hello
14 World
```

Listing 25: Makefile example

Listing 25 is a Makefile which creates two text files, fills them with text and prints the content on two lines. The entrypoint is Main_Target, which requires both subtargets to be executed before proceeding with its own routine.

Makefiles usually contains more advanced procedures. All syntax is available in the GNU Make manual[39]

2.7.3 Executables and Aliases

In order to simplify execution of applications, the executable file should be accessible from any directory. Other applications (e.g. gcc) are accessible everywhere because of the PATH-environment variable. By exporting a directory path to the PATH-variable, all executables within this directory will be visible anywhere. This export should be performed for every shell. By writing it in a users .bashrc-file, it will be executed upon shell creation:

```
1 user@Ubuntu:~$mkdir DirToExport
2 user@Ubuntu:~$ nano ~./bashrc
3 ------------------------/home/user/.bashrc----------------------
4 ....
5 export PATH="/home/user/DirToExport:$PATH"
6 ------------------------------------------------------------------
7 (CTRL+X, Y)
8 user@Ubuntu:~$ source ~./bashrc
```

Listing 26: Permanent PATH modification example

Listing 26 Shows how folder DirToExport can be exported to PATH. All executables within DirToExport will be visible in all directories.

Alternatively, an alias could be created. Aliases can be made for any shell command. By declaring aliases in a users .bashrc-file, they will be defined in every shell created by the user.

Listing 27 shows an alias added to the bottom of the .bashrc-file. The file is saved and loaded using the source-command.

```
1 user@Ubuntu:~$ nano ~./bashrc
2 ------------------------/home/user/.bashrc----------------------
3 ....
4 alias ccs='<command>'
5 ------------------------------------------------------------------
6 (CTRL+X, Y)
```

```
user@Ubuntu:~$ source ~/.bashrc
```

Listing 27: Permanent alias example

2.7.4 Device trees and overlays

Device trees are used to describe the computers hardware and its capabilities. Having all hardware capabilities described in one, unified way makes it possible to add additional hardware without installing specific, one-purpose drivers.

Many hardware components have common properties. It is possible to describe these using the same device tree, while adding individual properties in a *device tree overlay*. Newer versions of linux support dynamic loading of device tree overlays. *Das U-Boot* Boot Loader is responsible for loading the operating system. Overlays can be loaded at boot by configuring /boot/uEnv.txt.

The latest recommended Debian images have overlays loaded by default. These configures interaction with pins, modules and ports on the board. For specific purposes, some overlays may have to be disabled/overwritten (e.g PRU-communication, HDMI-Pins blocking module access).

The universal-io overlay describes the different configurations available for each pin and makes them directly configurable from the shell using the config-pin command[40]

Overlays can be associated with kernel drivers and Loadable Kernel Modules (section 2.3.4), allowing drivers to reserve memory upon boot. Disabling overlays or adding overlays in the uEnv.txt-file will overwrite reservations.

2.7.5 Kernel

The kernel is the core of the operating system, the part which has complete control of everything running.

All communication between hardware and user-space software passes through the kernel. This makes the kernel responsible for controlling which processes to run, requiring user applications and external peripherals to request runtime and hardware access.

Having a system for scheduling tasks is crucial to make different hardware work together. The default Linux kernel takes care of this, and puts its own most critical tasks at highest priority.

2.7.6 Character Devices

Character devices can be used as a link between kernel and userspace. By creating and specifying a device location in a Loadable Kernel Module (section 2.3.4), information can be passed through it by using standard stream functions from userspace. (Example listing 28, section 3.1.1)

The /linux/fs.h-header file includes file_operations, which makes it possible to run functions when the device is interacted with from userspace. These devices can map memory with kernel-space authority, allowing most memory to be read from/written to through the device.

Both /dev/mem and spidev are character devices created with own file_operations-functions, making it possible to interact with memory and the mcSPI-modules registers

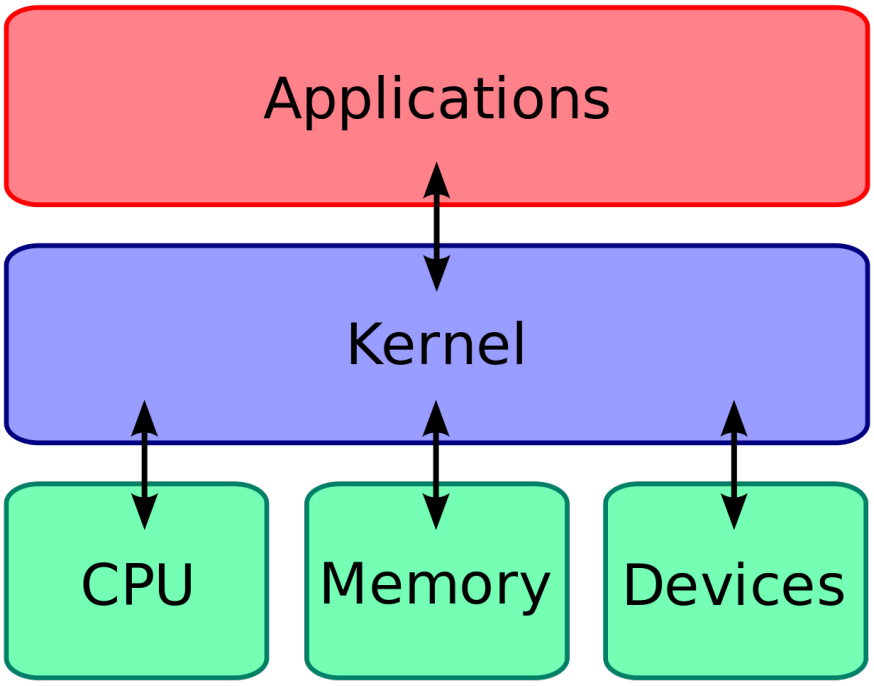


Figure 2.18: Kernel role in application-hardware communication[4]

Note: *Linux Device Drivers, third edition*[41] explains programming and usage of device drivers in detail.

2.7.7 Shell commands

The following tables lists the shell commands required to perform all operations done during the project

Navigation and basic file manipulation	
cd <path>, "..", "/", "∼/"	Enter path directory, parent folder, root directory, root home directory
ls (-a) (-r)	List files and directories in current directory including hidden ones (-a) and subfiles/directories(-r)
find -name "<name>"	Searches for <name>recursively from the current directory
sudo <command>	Execute commands with sudo privilegies. (User must be in sudo-group)
sudo visudo	Open sudoers settings
mkdir <dirname>	Create a new directory. (To create parent folders, use (-p))
rm (-r) <target>	Remove <target>. (Recursively with (-r))
mv <target><NewLocation>	Move <target>to a new location. mv renames the file if <NewLocation>contains a new filename.
cp (-r) <target><path>	Copy target file to new path. (Recursively with (-r))
su <user>	Change user. (Sudo su for root login on Ubuntu)
Package manager tools	
apt-get <option> update upgrade install <package> remove <package>	Avdanced Package manager Update database of available packages Install available package updates Install new package Remove <package>
apt-cache search <string>	Search for package name with <string>
dpkg -i <package.deb>	Install a downloaded package
git clone <url>	Clone GitHub respository. (Requires git-package from APT)
wget/curl <url>	Download file using HTTP(s)/FTP
tar xzf <compressedFile>	Extract (x) and uncompress (z) specified file (f) (Optional (v) to show process)
unzip <zippedFile>	Unzip file
nano text editor	
nano (-c) <target>	Edit <target>with optional line numbers (-c)
CTRL+X	Exit with a save prompt
CTRL+W, <string>	search for <string >
CTRL+\, <string>,<newString>	Find and replace strings
CTRL+SHIFT+- <LineNumber>	Go to line number
CTRL+6	Set mark
CTRL+K	Cut Line/marked text
CTRL+U	Paste copied text

Table 3: Terminal commands 1

SSH configuration	
ssh-keygen	Generate new RSA-keypair. (Example in section REFF RSA)
ssh-copy-id <user>@<ip-address>	Copy RSA public key to device with <ip-address>logged in as <user>.
ssh user>@<ip-address>	Login as <user>at device with <ip-address>
ifconfig	List network interfaces
Compiling and execution	
chmod a+x <target>	Add permissions to execute (x) <target >for all (a) users
make	Run Makefile in the current directory
source <target>	Execute <target>in the **current shell**
./<target>	Execute <target>in current directory, creating a **new shell**
<ins/rm/ls>mod <target>	Insert/remove/list loadable kernel module(s)
gcc/g++ <target> (-E) (-S) (-C) (-l<library>) (-O)	Compile C (gcc) or C++ (g++) with additional options: Output preprocessed code Output assembly code Output unlinked *.obj-file link shared libraries Specify ouput name
clpru <target(s)> –run_linker <cmd-file> (-O)	Compile <target>to *.obj-file. (PRU compiler, C/C++) Run linker with specified command file Specify output name
hexpru <target> (-B) (-O)	Run hex conversion tool on <target> Convert to binary format specify output name

Table 4: Terminal commands 2

Miscellaneous	
shutdown (+delay)	Shutdown after (+delay) minutes
reboot	Reboot instantly
ln (-s) <source><location>	Create symbolic (-s) link to <source>directory or file, put in <location>

Table 5: Terminal commands 3

3 BeagleBone realtime capabilities and subsystems

This section is intended to be an introduction to information flow in the BeagleBones subsystems. Some hardware documentation is included where necessary in order to give an overview of how interrupts are controlled on the main processor and the PRU-ICSS subsystem.

3.1 Main system

3.1.1 Memory

To exchange information efficiently between hardware modules on the board, the variety of physical memory and hardware registers is translated to virtual memory by the kernel. The kernel keeps a page table describing the connection between each virtual and physical page address. Physical

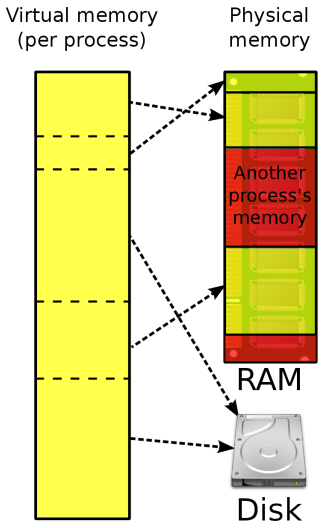


Figure 3.1: Physical to virtual memory translation[5]

memory is given a virtual representation in kernel space, while userspace access is given in the /dev/mem-character device. Memory can be accessed using the mmap()-function to map the desired area of the physical memory. [6, p.177]

```c
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <stdio.h>

#define SHARED_MEMORY_OFFSET 0x4a310000
#define SHARED_MEMORY_SIZE 0x10000
int main()
{
  int fd = open("/dev/mem", O_RDWR | O_SYNC);
  void* voidMemPointer = mmap(NULL, SHARED_MEMORY_SIZE, PROT_READ, MAP_SHARED, fd,
    SHARED_MEMORY_OFFSET);
  int*  intMemPointer = (int*) voidMemPointer;
  printf("Value read from address: %d\n", intMemPointer[0]);
}
```

Listing 28: mmap()-example

Listing 28 opens the /dev/mem char device, and maps the PRU-ICSS shared memory with a void pointer. A recasting makes it possible to print memory content. **Note:** For offsets and memory addresses, see the AM335x technical reference manual. [6, p.177-186]

The prussdrv application loader library[42] (Subsection 3.2.4) simplifies this process with its prussdrv_map_prumem()-function. The library defines sizes and offsets for the different memory areas, requiring only a pointer and a memory name-argument to perform the mapping.

3.1.2 Interrupt controller

Figure 3.2 shows the interrupt controllers signal interface in the Microprocessor Unit Subsystem (MPU). The controller supports up to 128 interrupts with different priorities, which can be mapped to modules and pins on the board. The host interrupt controller is used to control the Arm Cortex A8 processor.

3.1.2.1 Interrupt request

When a system event signal is received by the controller, an interrupt request (IRQ) will be sent to the main processor. If the priority of the request is greater than the current running thread, the current running process is halted and relevant data is tempoarily unloaded. The processor will find the corresponding interrupt handler memory address from a table. After running the handler, the previous process and its data will be loaded.

3.1.2.2 Interrupt handlers

Interrupt handlers are the instructions initiated by the kernel once an interrupt request has been received (Also called Interrupt Service Routine, ISR).

Because of the Linux kernels priories, realtime capabilities are restricted. The kernel will always prioritize its own critical processes, which are treated with top priority upon interrupt request.

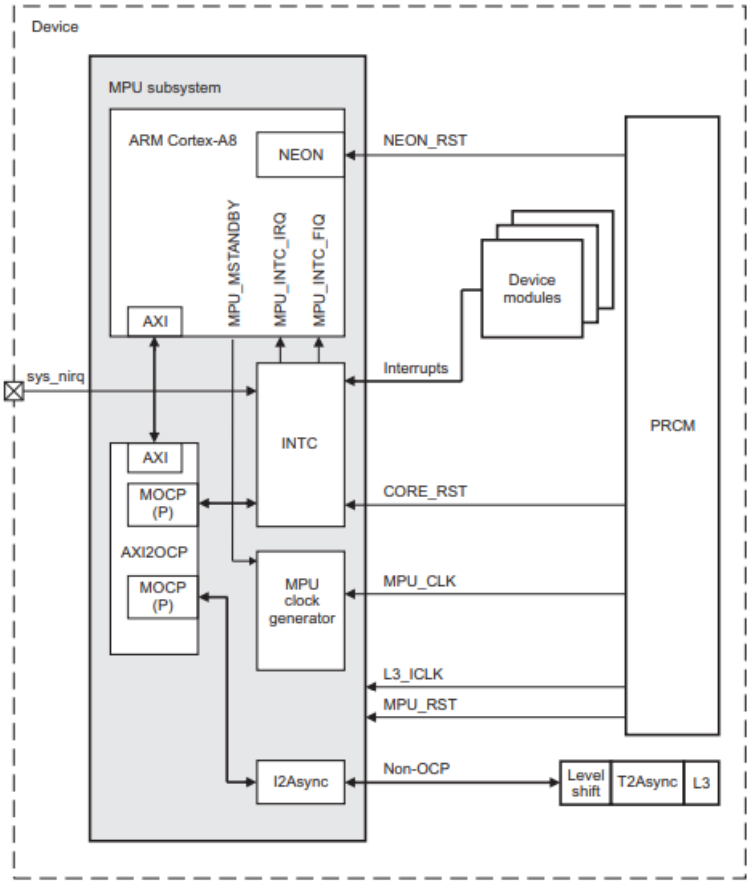


Figure 3.2: MPU subsystem

3.2 PRU_ICSS

BeagleBone Black has a Programmable Realtime Units and Industrial Communication SubSystem (PRU-ICSS). The PRU-ICSS contains two PRUs combined with internal memory and peripherals[43]

The 200MHz processors are useful for offloading the main processor for low-frequency tasks. The subsystems modules could be programmed to handle desired communication protocols, while the PRUs manages data and interrupts. The PRU-ICSS by itself is a hard realtime subsystem specifically made with respect to latencies and timing.

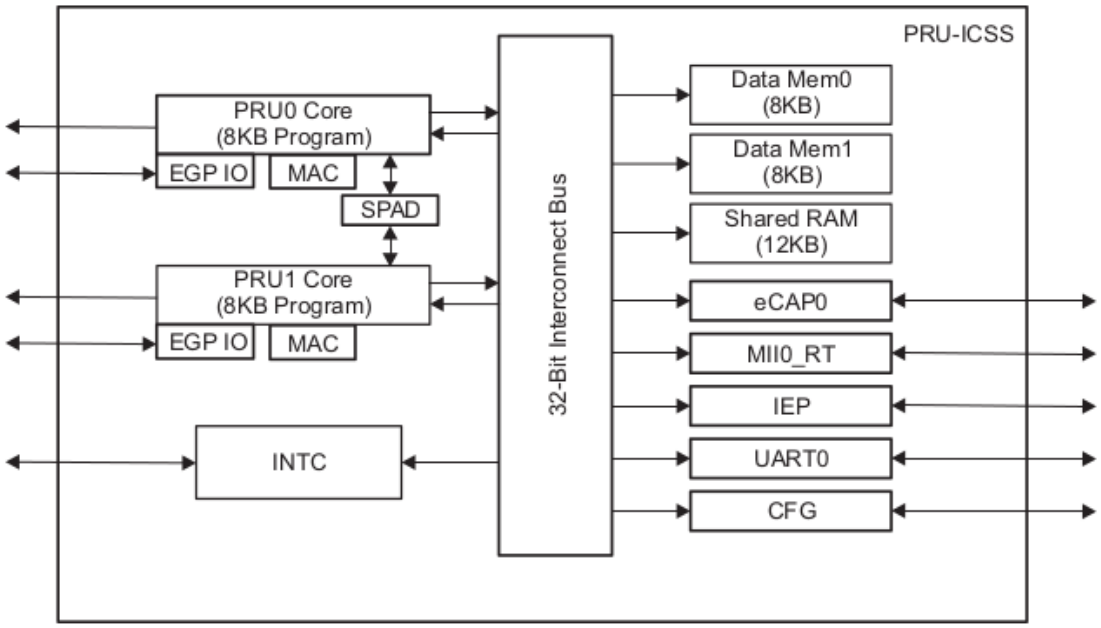


Figure 3.3: PRU-ICSS Block Diagram[6, p.199]

3.2.1 Latencies

The PRUs also offers a lower latency to I/O-interfacing with direct access to the device pins. Due to latency from the interconnects needed to route signals from the main processor, the PRU subsystem is able to toggle pins 40x faster[7]

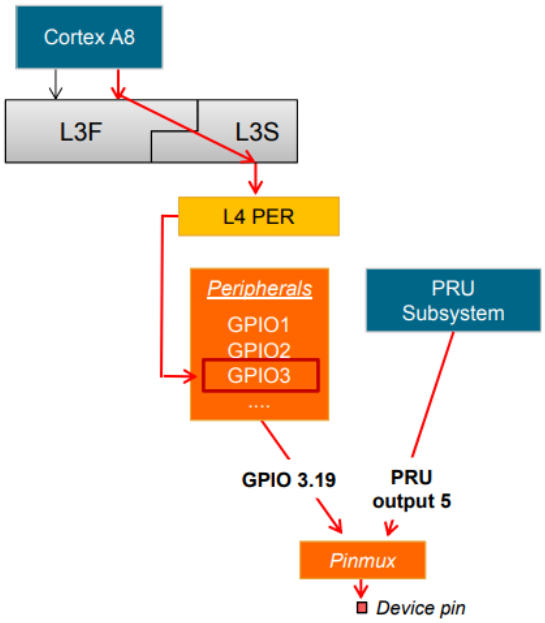


Figure 3.4: GPIO Latency [7]

Along with the well-documented memory read/write-latencies[44], the PRU-ICSS-subsystem is deterministic and well-suited for hard deadlines.

MMRs	Read Latency (PRU cycles @ 200MHz)
PRU CTRL	4
PRU CFG	3
PRU INTC	3
PRU DRAM	3
PRU Shared DRAM	3
PRU ECAP	4
PRU UART	14
PRU IEP	12
PRU R31 (GPI)	1

Figure 3.5: PRU local subsystem read latencies

3.2.2 Memory mapping

Start Address	PRU0	PRU1
0x0000_0000	Data 8KB RAM 0	Data 8KB RAM 1
0x0000_2000	Data 8KB RAM 1	Data 8KB RAM 0
0x0001_0000	Data 12KB RAM 2 (Shared)	Data 12KB RAM 2 (Shared)
0x0002_0000	INTC	INTC
0x0002_2000	PRU0 Control Registers	PRU0 Control Registers
0x0002_2400	Reserved	Reserved
0x0002_4000	PRU1 Control Registers	PRU1 Control Registers
0x0002_4400	Reserved	Reserved
0x0002_6000	CFG	CFG
0x0002_8000	UART 0	UART 0
0x0002_A000	Reserved	Reserved
0x0002_C000	Reserved	Reserved
0x0002_E000	IEP	IEP
0x0003_0000	eCAP 0	eCAP 0
0x0003_2000	MII_RT_CFG	MII_RT_CFG
0x0003_2400	MII_MDIO	MII_MDIO
0x0003_4000	Reserved	Reserved
0x0003_8000	Reserved	Reserved
0x0004_0000	Reserved	Reserved
0x0008_0000	System OCP_HP0	System OCP_HP1

Figure 3.6: PRU_ICSS memory map[45]

The PRUs can access the PRU_ICSS-memory using Load Byte Burst (LBBO) and Store Byte Burst (SBBO)-instructions (More available from the Texas Instruments assembly instruction manual[46]).

3.2.3 Interrupt Controller

The PRU_ICSS-subsystems interrupt controller[47] can be used to control thread priority with respect to external devices and the PRU-ICSS' modules.

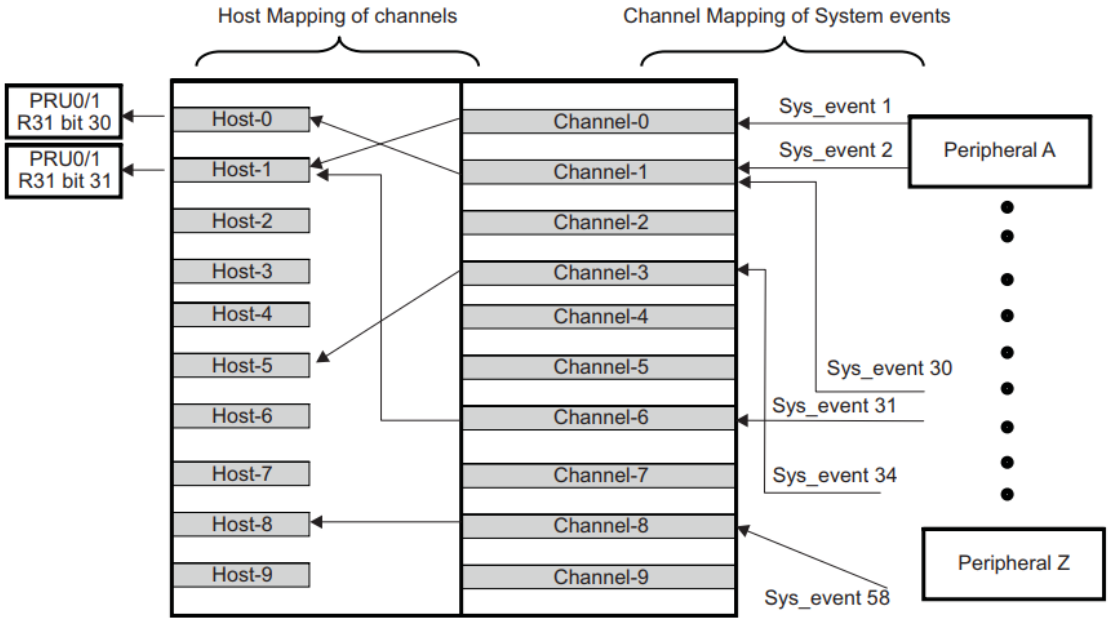


Figure 3.7: PRU Interrupt Controller

This interrupt controller supports 10 channels, which in total supports 64 system events. Priority is descending from channel 0 to 9 with own priorities for the system events mapped to the channel. System events from outside peripherals can be mapped to any channel, which is directed to a mapped host (Arm Cortex A8, PRU0 or PRU1). Due to the PRUs deterministic design, they can not be asynchronously interrupted. Instead, they poll the interrupt controller for changes.

This controller allows interrupts to be forwarded to the interrupt controller in the MPU-subsystem (Chapter 3.1.2). This allows modules of the PRU-ICSS subsystem to interrupt a program running on the boards main processor given the right priority configurations. Mapping is simplified by using a data structure in the prussdrv-library, which uses /dev/uiox character devices (subsection 2.7.6) to interact with hardware registers.

Since these interrupts are passed through character devices, their priorities are handled by the Linux kernel. This prevents applications from userspace from achieving hard realtime capabilities, since Linux will prioritize its own critical processes.

3.2.3.1 Interrupt example using threads

Threads_And_PRU_Interrupt[24] is an example made to demonstrate the inconsitency of POSIX-threads created in userspace, combined with demonstration of interrupt handling of PRU interrupts. One thread is incrementing a variable, while the other is awaiting for the interrupt signal. The resulting sum from the first thread changes each time. This could be caused by varying memory latency and thread prioritzation.

This code uses the application loaders skeleton code[42], combined with thread-creation and priority functions from <pthread.h>.

Note: Including <prussdrv.h>while using the cross-compiler resulted in erros, as the compiler did not recognise keywords required by the library. (__far) This program was compiled with the on-board gcc. (Example listing

3.2.4 Application loading and communication

There are different approaches used to load code into the PRUs instruction RAM. The PRU application loader from the subsystems support package[48] solves this by taking a binary file as input and writing it directly into the instruction RAM, followed by execution. All PRU-projects made during this project is based on this approach.

Note: This application loader is outdated, but has been sufficient to demonstrate the usage of the PRUs. Software support by TI has been moved to Processor Software Development Kit[49], which unfortunately was incompatible with the Linux Distributions used in this project. (Ubuntu 18.04.2 LTS is supposed to run it smoothly, but did not.)

Another option is to use the remoteproc-framework[50]. This is a Linux-filesystem based approach, where binary code is placed in a directory (/lib/firmware). The state of the PRUs can be controlled from /sys/class/remoteproc/remoteprocx. The PRU_rproc LKM-driver will load and execute the binary when given a 'start'-signal.

Note: This solution has issues with the kernel in the newest recommended debian images.

The only standard for communication between the cores are RPMSG, which is associated with the remoteproc framework. Its intention is to abstract the communication, to the point where sending/receiving information is done with two functions. This standard is well-developed and supported by Texas Instruments.

Because of incompatibility, writing directly to shared memory has been the solution in this project.

3.2.5 Hard Realtime Capabilities

The BeagleBone will experience more demanding calculations in future projects. This could create situations where several processes needs to run at the same time. If the processes are handled with wrong priorities it could result in insufficient timespans for calculation or measurement processing. If the completion of a process ends up being fatal for the robots purpose, a hard Realtime Operating System (RTOS) is needed.

The boards hardware described in the MPU-subsystem and in the PRU-ICSS shows how the interrupt signals are processed and how Linux is restricting the priorities of interrupts. It is possible to modify/replace the layer responsible for configuring the interrupt controllers and handlers.

3.2.5.1 RTLinux

RealTime Linux is a kernel modification which adds a micro-kernel to schedule tasks. The orignal kernel now runs on a lower priority, surpassed by custom processes configured in kernel space.

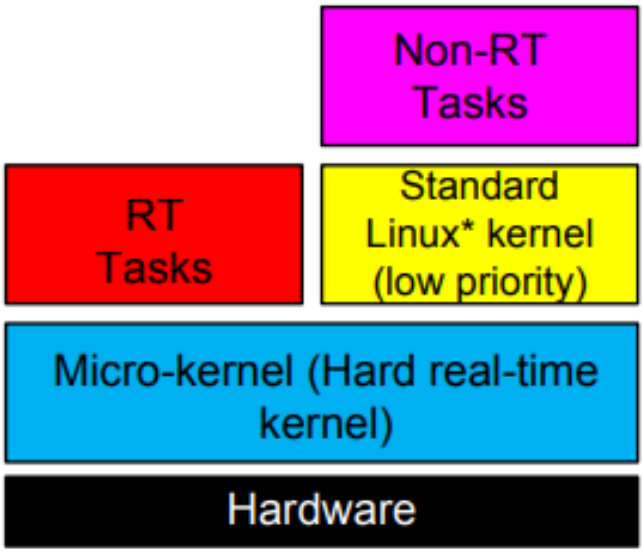


Figure 3.8: RTLinux layer overview[8]

This modification is specifically designed for the linux kernel. In order to have a thread surpass kernel priority, it must be programmed in kernel space. (LKM, subsection 2.3.4)

3.2.5.2 Xenomai

Xenomai comes with a nano-kernel responsible for interrupts (Hardware Abstraction Layer (HAL) and AEDOS). This kernel is overlayed with the abstract xenomai RTOS-core, which is compatible with several RTOS by using Xenomai skins.

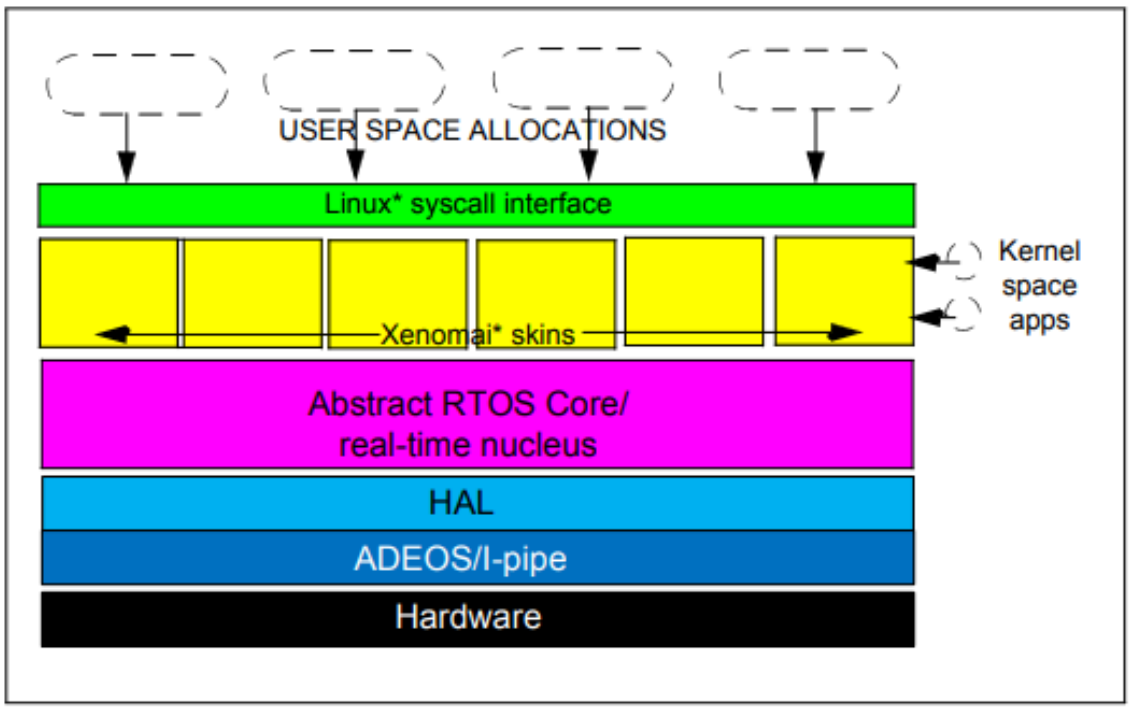


Figure 3.9: Xenomai layer overview[8]

The skins on top of the nucleus (RTOS-core) mimics the interface, which custom RTOS requires to communicate with the boards hardware. With multiple skins, it is possible to schedule several operating systems to run on the same board. This allows user-space applications to be run directly through the nucleus with its own skin without using the Linux kernel. C/C++ applications are ported using the POSIX-skin, allowing threads to be passed using the pthread library.

4 IMU

The SparkFun IMU - LSM9DS1 used in this project is constructed with an accelerometer, gyroscope and magnetometer. It measures rate of angular rotation, acceleration and magnetic fields in three dimensions.

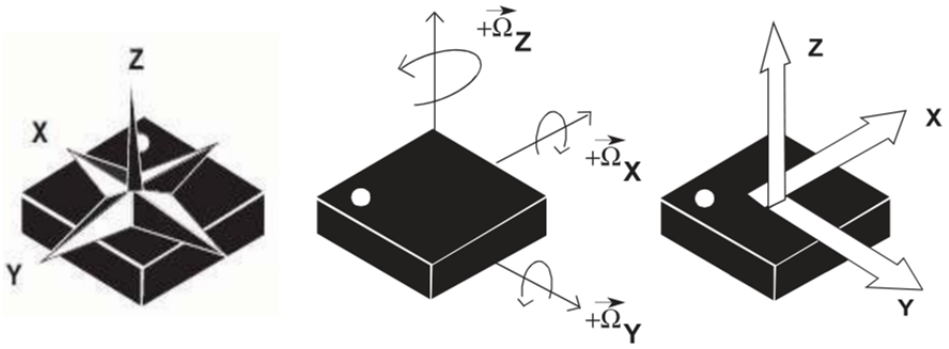


Figure 4.1: LSM9DS1 axes [9]

It requires a voltage supply of 3.3V. A voltage supply greater than 3.6V can permanently damage the *IMU*. If the chosen device supply more than 3.3V it is necessary to use level shifting. LSM9DS1 is constructed with specific magnetometer-, accelerometer- and gyroscope interrupts, DRY, INTM, INT1 and INT2. The interrupts can be triggered if, for example, the value of the object's acceleration exceed a certain threshold. The LSM9DS1 supports both I2C and SPI communication protocols. (subsection 4.1). Figure 4.2 goes into more detail on how each pin operates.

Pin Label	Pin Function	Notes
GND	Ground	0V voltage supply
VDD	Power Supply	Supply voltage to the chip. Should be regulated between **2.4V and 3.6V.**
SDA	SPI: MOSI I^2C: Serial Data	SPI: Device data in (MOSI) I^2C: Serial data (bi-directional)
SCL	Serial Clock	I^2C and SPI serial clock.

The remaining pins are broken out on the other side. These pins break out SPI functionality and interrupt outputs:

Pin Label	Pin Function	Notes
DEN	Gyroscope Data Enable	Mostly unknown. The LSM9DS1 datasheet doesn't have much to say about this pin.
INT2	Accel/Gyro Interrupt 2	INT1 and INT2 are programmable interrupts for the accelerometer and gyroscope. They can be set to alert on over/under thresholds, data ready, or FIFO overruns.
INT1	Accel/Gyro Interrupt 1	
INTM	Magnetometer Interrupt	A programmable interrupt for the magnetometer. Can be set to alert on over-under thresholds.
RDY	Magnetometer Data Ready	An interrupt indicating new magnetometer data is available. Non-programmable.
CS M	Magnetometer Chip Select	This pin selects between I^2C and SPI on the magnetometer. Keep it HIGH for I^2C, or use it as an (active-low) chip select for SPI. **HIGH (1): SPI idle mode / I^2C enabled** **LOW (0): SPI enabled / I^2C disabled.**
CS AG	Accel/Gyro Chip Select	This pin selects between I^2C and SPI on the accel/gyro. Keep it HIGH for I^2C, or use it as an (active-low) chip select for SPI. **HIGH (1): SPI idle mode / I^2C enabled** **LOW (0): SPI enabled / I^2C disabled.**
SDO M	SPI: Magnetometer MISO I^2C: Magnetometer Address Select	In SPI mode, this is the magnetometer data output (SDO_M). In I^2C mode, this selects the LSb of the I^2C address (SA0_M)
SDO AG	SPI: Accel/Gyro MISO I^2C: Accel/Gryo Address Select	In SPI mode, this is the accel/gryo data output (SDO_AG). In I^2C mode, this selects the LSb of the I^2C address (SA0_AG)

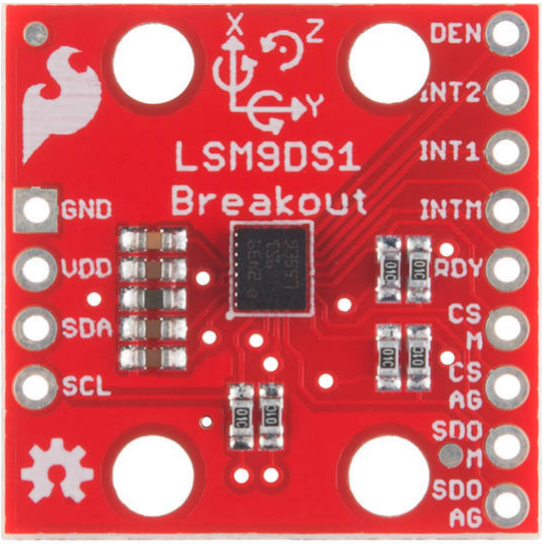


Figure 4.2: LSM9DS1 pin specification [9]

4.1 SPI

SPI is originally designed to work for short-distance communication in embedded systems. Its area of application has expanded and is now used between microcontrollers and, for example, sensors. SPI is not limited to 8-bit words. In addition, it has a simple software implementation thanks to guides on the internet.

4.1.0.1 Setup

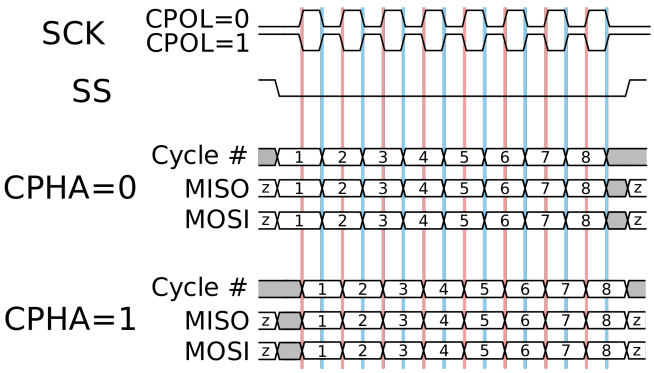
As a synchronous data bus, SPI uses separate lines for data transmission between a master and one or several slaves. To synchronize all lines, it requires another line that contains a clock signal, which is generated by the master. The data and the clock signal are sent simultaneously from the master to the slave and vice versa.

The master sends signals to the slave through a line called *MOSI*(Master Output, Slave Input). When the slave detects an edge on the clock signal, it will read the next bit. If the slave is to send data in return, it will place the data in a third line called *MISO*(Master Input, Slave Output).

For the master to receive this data, it must know the size of the data package and when it is sent. In some cases this could be problematic. With SPI there is no way to send variable sized data packages in any direction without having it predefined in slave/master. However, implementing SPI with an IMU allows one to neglect this issue entirely. Thanks to the IMU's specific command structure, the master knows in advance what the incoming data contains and the size of the package. This information can be found in the data sheet of the IMU [51].

4.1.0.2 Data flow

In order to get the communication to flow, the master sends a logic 0 to its slave through the *CS*-line. The transmission of data is a full-duplex data transmission and takes place every clock cycle. On the edge of every clock cycle the master sends a bit to the slave through *MOSI*. The slave synchronously sends a bit to the master through *MISO*. With a shift register in each end, the most significant bit is sent in both cases. The bit that is received is then put as the least significant bit in each shift register. When every single bit has been transferred to the counterpart, the procedure is done if no more data is to be exchanged. If more data is to be exchanged, the registers will reload and the process repeats. SPI has four different modes for data transfer called SPI Mode 0, 1, 2 and 3. Each mode differs from the others in its timing to send data.



(a) [10]

SPI Mode	CPOL	CPHA	Clock Polarity in Idle State	Clock Phase Used to Sample and/or Shift the Data
0	0	0	Logic low	Data sampled on rising edge and shifted out on the falling edge
1	0	1	Logic low	Data sampled on the falling edge and shifted out on the rising edge
2	1	1	Logic high	Data sampled on the falling edge and shifted out on the rising edge
3	1	0	Logic high	Data sampled on the rising edge and shifted out on the falling edge

(b) [10]

Figure 4.3: CPOL and CPHA cycle and configurations[10][10]

figure 4.3 illustrates how the timing is determined by CPOL and CPHA logical values. CPOL determines whether the clock is idle on 0 or 1 and if the cycle pulses with 1's or 0's. If CPOL is set to 0, the clock will idle at 0 and pulse with 1's. If CPOL is set to 1, the clock will idle at 1 and pulse with 0's. CPHA determines whether data is to be transmitted on the leading or trailing edge of a clock cycle. If CPHA is set to 0, the data is sent on the trailing edge of a clock cycle and received on the leading edge. If CPHA is set to 1, the data is sent on the leading edge of a clock cycle and received on the trailing edge. Reading the data sheet of a product is necessary to determine which SPI mode results in optimal communication.

4.2 Implemenation

4.2.1 Functionality

The LSM9DS1 IMU operates with 3.3V and the Arduino operates with 5V logic. To avoid any unnecessary damage, the signals from the Arduino to the IMU needs to be converted from 5V down to 3.3V. A logic converter chip from Adafruit solved this issue, mounted together with the IMU on a protoyping circuit board(see appendix C.3). The IMU can use SPI or I2C to communicate with the Arduino. Because of its speed, SPI was chosen as communication protocol. SparkFun already has hookup guides for the IMU to Arduino, which was used to figure out how to set it up. When using BeagleBone Black, the logic converter is uneccessary since it already operates with 3.3V logic. Then the logic converter needs to be removed and the CHANNEL-connections on the level shifter circuit needs to be short circuited.

4.2.1.1 RS-232

Initially, communication between Arduino and dSPACE was implemented in order to read IMU values to the dSPACE software called *ControlDeskNG*. Motor control and encoder readings were already functional, with the IMU being the missing piece. In order to read data from Arduino, a communcation bus is needed. The *RTI 1103* board is compatible with RS 232, RS 422 and

CAN bus. The RS 232 prototcol is used in this project. This protocol has a maximum *baud rate* of 115200 bps. The protocol uses 10 bits per byte resulting in around 10 kByte per second. By using 1 byte resolution, it will match the update-rate of IMU-values. However, if we want greater resolution the update-speed will decrease since we need several bytes for each value. In order to use float values (IEEE 754), a minimum of 16bits are needed.

The RS-232 connector on the dSPACE RTI1103 board operates with ± 3-25V. To make this compatible with Arduino it needs to be converted to 0-5V. To achieve this we use a specific chip (Max 233). See appendix C.2.

To read the values on the ConrolDeskNG software, a C-code file generated by *Simulink* is needed. The C-code file is then imported into ControlDesk. The dSPACE platform has its own Simulink-library with blocks designed for the RTI1103 connectors. To read and send values through the UART RS-232 port, serial blocks TX and RX and the Serial Setup Blocks are used. The baud-rate needs to match the baud-rate of the other end. After transitioning into BeagleBone Black, this setup is no longer needed to read the IMU values. However, it is definitely relevant in further work regarding real-time user-interface. See section 10.1.8. This setup should work with any device using the RS-232 protocol. The simplicity of this setup makes it great for initial prototyping, even tough faster and more complex protocols are probably more suited for the final configuration of the robot.

4.2.1.2 Mounting

In order to mount the IMU (with the level converting circuit) on the robot, a mounting bracket is needed. Regarding the IMU mount, there are two alternatives; either on a leg or the torso. Hence, two models were designed, see figure 4.4 and 4.5. The difference between them being the design for mounting the bracket onto the joint. This was designed in *FreeCAD*, even though *Fusion360* was used for finalizing the models. The reason for the change of software was based on some poor experiences with FreeCad. Also, the amount of tutorials and help on the internet is in favor of *Fusion360*. The transition into Fusion360 was simple. In addition to the software advantages, the transition seemed like a good choice. The brackets were 3D-printed with a Ultimaker 2+ printer using *CURA* to slice the STL model, printing with *PLA filament*.

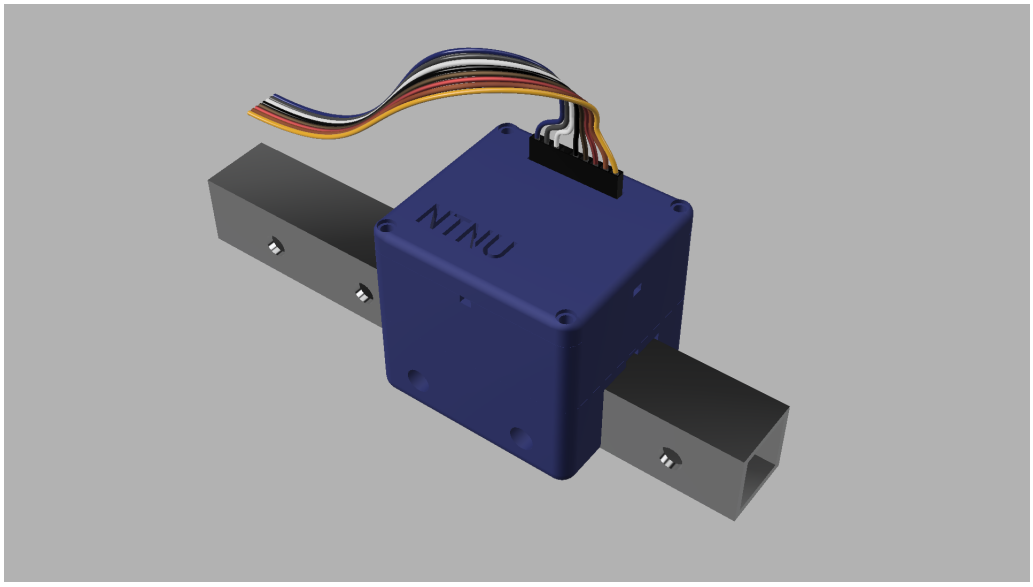


Figure 4.4: IMU Casing for mounting on Leg

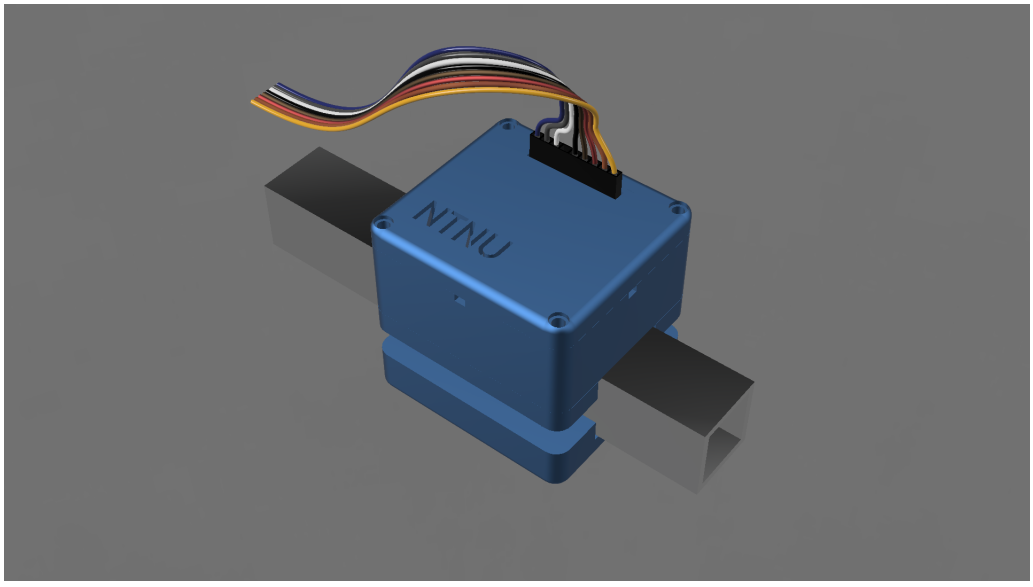


Figure 4.5: IMU Casing for mounting on Torso

4.2.1.3 Wiring

The wires from the IMU goes to the power distribution board, appendix C.6, and to the beagle bone. SPI is mainly designed to be used on-*PCB* communication. It does not work over longer distances. Since the IMU-placement is not set, cables up to about 2.5 meters may be needed to reach the IMU when placed on the legs. This was first tested on Arduino and seemed to work. However, when testing on the BeagleBone, it did not work. The main reason for this is suspected to be the logic voltage level of the BeagleBone, which is 3.3V compared to the 5V of the Arduino. When run over some distance, the noise level induced in the cable will become considerably larger in comparison. Thus, the BeagleBone struggles to separate the 3.3V logic levels from the noise, resulting in miscommunication. In this setup, the BeagleBone is only capable of running the IMU

on the torso until the SPI-cable length issue is solved. Luckily there are several ways to work around this which are discussed later on in section 10.1.1 .

4.3 Calculation

In order get an angle as close to reality as possible from the IMU, numerous measures should be made. Especially a strategy called *sensor fusion* will be crucial in regards to the IMU calculation.

4.3.1 Accelerometer

The *accelereometer* measures linear accelerations along the x, y and z axis, meaning it also measures gravity in every axis. This is the key to measuring angle relative to the world using an accelerometer. The equations describing the accelerometer are based on mathworks implementation of their accelerometer-block [52].

$$\bar{A}_{imeas} = \bar{A}_b + \bar{\omega}_b \times (\bar{\omega}_b \times \bar{d}) + \bar{\dot{\omega}}_b \times \bar{d} + \bar{g} \tag{4.1}$$

$\bar{\omega}_b$ are body-fixed angular rates, $\bar{\dot{\omega}}_b$ are body-fixed angular accelerations and $\bar{d}$ is the lever arm. The lever arm $\bar{d}$ is defined as the distances that the accelerometer group is forward, right and below the center of gravity. In this case the lever arm will be 0 except for the z-axis, where the $d_z = radius$ where $radius$ is the distance from rotational center to the IMU mounted on the torso or leg.

$$\bar{d} = \begin{bmatrix} d_x \\ d_y \\ d_z \end{bmatrix} = \begin{bmatrix} -(x_{acc} - x_{CG}) \\ y_{acc} - y_{CG} \\ -(z_{acc} - z_{CG}) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ radius \end{bmatrix} \tag{4.2}$$

$$\bar{A}_{meas} = \bar{A}_{imeas} \times \bar{A}_{SFCC} + \bar{A}_{bias} + noise \tag{4.3}$$

$\bar{A}_{SFCC}$ is a 3-by-3 matrix of scaling factors on the diagonal and misalignment terms in the nondiagonal and $\bar{A}_{bias}$ are the biases.

To calculate the torso angle from the accelerometer, the ratio between the y- and z-acceleration caused by gravity is used with the trigonometric function *atan2*. This works great if there is no bias, noise or body accelerations.

$$\theta_a = atan2(\frac{a_y}{a_z}), \frac{a_y = g_y}{a_z = g_z} \tag{4.4}$$

As mentioned, this implies the IMU is not accelerating in any direction, meaning it is held perfectly still or moving with constant velocity. If this is not the case, the additional accelerations will affect the angle calculation.

This also makes it rather sensitive to physical noise like vibrations and impulses. This makes the accelerometer quite efficient at detecting slow angular motion (low frequencies) and getting a

correct position relative to the world frame. Faster motion, causing noise and greater accelerations, will result in distorted and offset measurements.

4.3.2 Gyroscope

The *gyroscope* measures angular velocity ω in three axes, x, y and z. The measured value depends on several *biases*, scaling factors and noise.

$$\bar{\omega}_{meas} = \bar{\omega}_b \times \bar{\omega}_{SFCC} + \bar{\omega}_{bias} + Gs \times \bar{\omega}_{gsens} + noise \tag{4.5}$$

ω_{SFCC} is a 3-by-3 matrix of scaling factors on the diagonal, and misalignment terms in the non-diagonal, ω_{bias} are the biases, (Gs) are the Gs on the gyroscope and ω_{sens} are the g-sensitive biases. The equations describing the accelerometer are based on MathWorks implementation of their accelerometer-block [53].

From this, the angle θ is calculated by integrating ω

$$\theta_g(k) = T_s\omega(k) + \omega(k-1) \tag{4.6}$$

$$\theta_g(z) = \frac{T_s z}{z-1}\omega(z) \tag{4.7}$$

The gyroscope usually has an approximately constant bias value. When integrated to get the angle it will drift off from the real value. This can be reduced by simply subtracting an estimate of the bias value before integrating, but it does not remove it completely. Also, the only reference to the angle is the previous angle, meaning a correct initial value when starting calculations is needed. However, since the gyroscope only measures angular velocity, any linear movement will not affect the measurement. Because of this, the gyro is great for detecting rapid changes(high frequencies) without getting to distorted by noise. It is not ideally for stationary values, due to the bias drift and the lack of reference.

4.3.3 Sensor fusion

From the sections 4.3.1 and 4.3.2, both the accelerometer and gyroscopes unique advantages and drawbacks are mentioned. The best option is to combine the strengths from each unit.

In order to calculate the angle of the torso, the measurements from the gyroscope and the accelerometer is combined. This is called *sensor fusion*. The implementation used here is based on the equation for this described in [54]

As mentioned in [54], the effect of the sensor fusion on both the gyro and accelerometer measurements appears in 4.6. The accelerometer-filter removes the high frequency noise. The gyroscope-filter removes the low-frequency drift. These filters share the same resonance frequency. This resonance frequency will be a function of the weighting constant α and the sampling period T. In

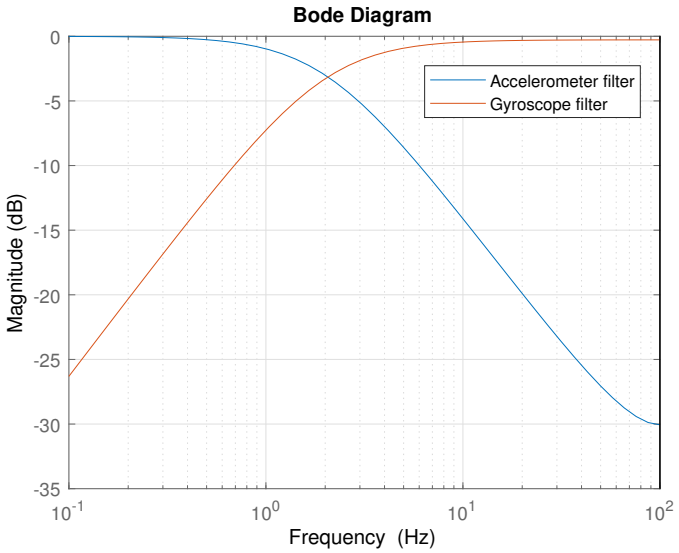


Figure 4.6: Sensor fusion, $f_c = 2Hz$

order to decide the resonance frequency of our filters, $\alpha(\omega_c T)$ is needed.

$$\alpha = e^{-\omega_c T_s} \tag{4.8}$$

4.3.4 IMU errors

In both the gyroscope and the accelerometer, there are several factors that cause errors in the measurements, some of them more relevant than others. In this project, the main problem is likely to be bias in the gyroscope. The ones discussed here are based on information from an article from Novatel [55]:

- Bias
  - Stability
  - Repeatability
- Scalefactor
- G-dependency
- Sensor Non-orthogonoality
- Noise

A standard equation describing a measurement system:

$$y = Sf(x) + b \tag{4.9}$$

Where x is the actual value, S is the scale factor, f(x) is the function which includes any nonlinearities, b is the bias and y is the measured value.

The scale factor is found in the datasheets of the manufacturer[51]. If the input and output in percentages does not correlate in a 1:1 scale, there is a scale factor error. In this case, this is most likely small enough to be neglected. Both the gyroscope and accelerometer will be considered linear. The part that really matters in this case is the bias, specifically the stability of the bias in the gyroscope. As mentioned, if there is a bias in the gyroscope, this will cause a drift of the angle measurement. In order to prevent this, identifying and subtracting this bias prior to the integration process is crucial. This is usually done in a startup procedure or a calibration of the gyroscope. The stability of this bias may cause problems since there is not a good way to identify changes in the bias if the gyro is running and the system is not stationary. In this case, the robot is not supposed to run for a longer period of time. Therefore an initial bias identification should be sufficient.

G-dependency is a change in bias due to accelerations experienced along the sensing axis. Regarding this setup, the gyro is going to measure along the axis without motion (robot motion is 2 dimensional), which means this is most likely safe to neglect as well. Sensor Non-orthogonality may cause error, specifically on the accelerometer. If the measurement axis are not perpendicular to each other, the angle calculation based on gravity will not be correct. It is also sensible to assume that this is tolerable as well and rather examine the matter closer if it turns out to be a problem. Noise in the sensor is unavoidable, but the sensor fusion should handle the noise from the accelerometer in the filter and from the gyroscope in the integration process.

4.4 Simulation

The intention of the simulation is to find the optimal configuration of the IMU. To accomplish this, the effects of the following points will be examined further:

- Robot gait-pattern
- IMU-position
- Sensor fusion
- Sample time
- Bias removal
- Noise

This may service to identify limitations and requirements regarding the IMU-implementation, positioning and the sensor fusion configuration.

4.4.1 Simulink-model

In order to simulate the effect of these strategies, a model in *Simulink* was built. In addition, the toolbox *Aerospace blockset* was used. Aerospace blockset includes blocks for both accelerometer and gyroscope with three degrees of freedom each. These blocks grants the ability to add dynamics, bias, noise etc to the measured value of the IMU. Most of the calculations are run in *Matlab* in advance. The Simulink model takes in the forces experienced by the IMU and run these through the IMU-block and the sensor fusion algorithm. See figure 4.8 and 4.9. The most challenging

part of the simulation is calculating the states of the IMU based on the states of the robot and the placement of the IMU. This is done in Matlab and the mathematics behind the simulation are shown below. The complete Simulink models with associated matlab scripts and a brief instruction ("README.txt") can be found at [24, Simulation].

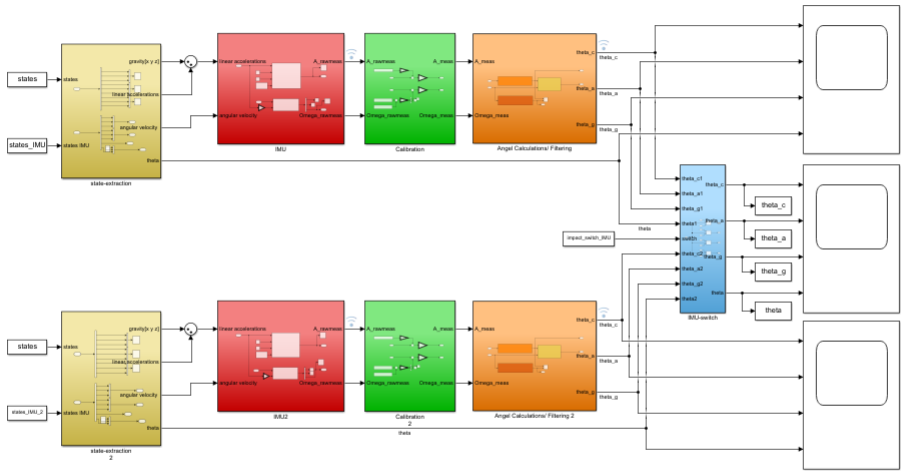


Figure 4.7: Simulink model with IMU on each leg

4.4.1.1 Physical variables

The physical values the IMU will experience are gravity and accelerations due to motion. From a set of Matlab scripts provided by Torleif Anstensrud we are able to simulate different gait patterns. From this script the states $\begin{bmatrix} q & \dot{q} & \ddot{q} \end{bmatrix}^T$ of each body part based on the chosen gait are extracted. Gravity will be a function of the absolute angle of the body, q, the IMU is placed on.

$$\bar{g} = \begin{bmatrix} g_x & g_y & g_z \end{bmatrix}^T = g \begin{bmatrix} cos(q_{abs}) & sin(q_{abs}) & 0) \end{bmatrix}^T$$

In order to find the accelerations caused by motion, the first step is calculating the position of the IMU $\vec{s}$ in reference to the tip of the stanceleg. $\vec{s}$ is differentiated to the acceleration vector. $\vec{a} = \vec{v} = \vec{\dot{s}}$. The states for each joint affecting the IMU, including the angular acceleration $\ddot{q}$, is needed to achieve this.

There are three different equations based on the IMU placement, one for each body part. Since the robot is theoretically 2D, the z-axis can be ignored and the equations can be run in 2D for the linear accelerations:

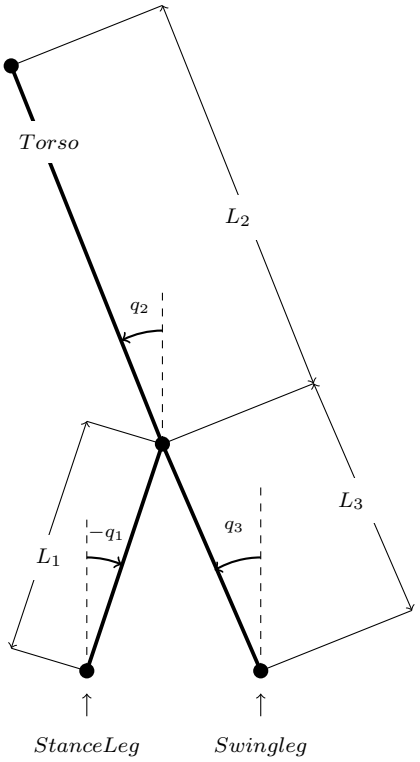


Figure 4.8: Definitions robot

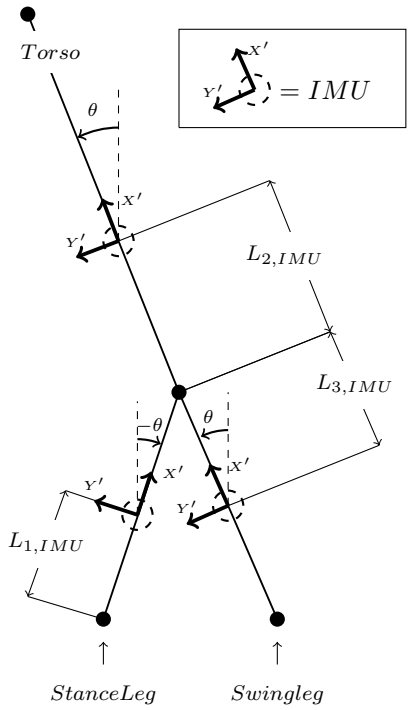


Figure 4.9: Definitions IMU

Stance Leg

$$\vec{s} = \begin{bmatrix} L_{1,IMU}cos(q_1) \\ L_{1,IMU}sin(q_1) \end{bmatrix} \tag{4.10}$$

$$\vec{a} = \begin{bmatrix} -L_{1,IMU}sin(q_1)\dot{q_1^2} \\ -L_{1,IMU}cos(q_1)\ddot{q_1} \end{bmatrix} \tag{4.11}$$

Torso

$$\vec{s} = \begin{bmatrix} L_1cos(q_1) + L_{2,IMU}cos(q_2) \\ L_1sin(q_1) + L_{2,IMU}sin(q_2) \end{bmatrix} \tag{4.12}$$

$$\vec{a} = \begin{bmatrix} -L_1(cos(q_1)\dot{q_1^2} + sin(q_1)\ddot{q_1}) - L_{2,IMU}(cos(q_2)\dot{q_2^2} + sin(q_1)\ddot{q_2}) \\ -L_1(sin(q_1)\dot{q_1^2} + cos(q_1)\ddot{q_1}) - L_{2,IMU}(sin(q_2)\dot{q_2^2} + L_{2,IMU}cos(q_1)\ddot{q_2}) \end{bmatrix} \tag{4.13}$$

Swing Leg

$$\vec{s} = \begin{bmatrix} L_1cos(q_1) + L_{3,IMU}cos(\pi + q_3) \\ L_1sin(q_1) + L_{3,IMU}sin(\pi + q_3) \end{bmatrix} = \begin{bmatrix} L_1cos(q_1) - L_{3,IMU}cos(q_3) \\ L_1sin(q_1) - L_{3,IMU}sin(q_3) \end{bmatrix} \tag{4.14}$$

$$\vec{a} = \begin{bmatrix} -L_1(cos(q_1)\dot{q_1^2} + sin(q_1)\ddot{q_1}) + L_{3,IMU}(cos(q_3)\dot{q_3^2} + sin(q_3)\ddot{q_3}) \\ -L_1(sin(q_1)\dot{q_1^2} - cos(q_1)\ddot{q_1}) + L_{3,IMU}(sin(q_2)\dot{q_3^2} - cos(q_1)\ddot{q_3}) \end{bmatrix} \tag{4.15}$$

This is in reference to the world frame. to get this in reference to the IMU-frame, it must be rotated to the reference frame to match the orientation of the IMU.

$$R_0^{IMU} = \begin{bmatrix} cos(q) & sin(q) \\ -sin(q) & cos(q) \end{bmatrix} \tag{4.16}$$

R^{imu_0} is defined by the given matix where q is the absolute angle of the body the IMU is placed on. This returns the equation for the linear acceleration:

$$a^{I\vec{M}U} = R_0^{IMU} \begin{bmatrix} \vec{a_x} \\ \vec{a_y} \end{bmatrix} \tag{4.17}$$

This is a 2D vector. When translated to the 3D IMU, a constant zero to the z axis is added.

4.4.1.2 IMU

The accelerometer takes in the linear accelerations which were calculated, $\bar{g}$ and $a^{I\vec{M}U}$. It also takes in angular velocity $\bar{\omega}$ and accelerations $\bar{\alpha} = \dot{\bar{\omega}}$. This is to simulate the additional accelerations based on the placement of the IMU. Since they are already calculated, they are not used.

The Gyroscope takes in angular velocity $\bar{\omega}$ and G's. G's is the accelerations in G's. The G's will affect the measurements based on the G-sensitive bias.

Both units will update their measurement value based on the set update rate of the block, which will be the sample time of the IMU-calculation in this case.

4.4.1.3 Calibration

This block is used to remove bias from the measured signal and convert the scaled signal back to SI-units. Since the signals are to be derived and integrated, it is more practical to operate with SI-units.

$$\bar{A}_{calmeas} = (\bar{A}_{rawmeas} - \bar{A}_{measbias})\bar{A}_{SFCC}^{-1} \tag{4.18}$$

In the simulation the bias removal might seem artificial as it is inserted a known bias to the IMU. It is included to make the model a good visualization of the entire process of angle calculation. In addition, the effect of bias removal are definitely relevant in regards to the sensor fusion configuration. In the simulation, this is done by removing the set bias by a scaled factor of the input bias. The input bias is based on a measured value from the real IMU. In order to measure the bias from the gyroscope it is placed in a stationary condition. The value from the gyroscope at this point is considered bias. There is noise within this value, as well as several different biases. Nevertheless,

for simplicity's sake it is operated with only one common bias. To keep noise from affecting the bias value, the final value is based on several measurements.

4.4.1.4 Angle calculation/ Filtering

In the angle calculation/filtering block, the algorithms for angle calculation and sensor fusion are implemented.

Angle calculation:
This block takes in the measured acceleration and calculates the angle θ_a based on the measurement being only gravity.

Sensor fusion /w complementary filter:
As mentioned, this fuses the two measurements with a scaling constant α.

$$\theta_c(z) = (1-\alpha)\frac{1}{1-\alpha z^{-1}}\ \theta_a(z) + \alpha\frac{T_s}{1-\alpha z^{-1}}\ \omega_g(z) \tag{4.19}$$

This is the calculated angle θ_c, which is to be used by the gait-planning algorithm.

4.4.1.5 IMU-switching

When simulating with two IMUs on both legs, the measurement from the stanceleg-IMU is wanted. To accomplish this, but still keep the dynamics from the swingleg-measurements, a function to switch between IMUs at the exact moment of impact time is needed. Luckily, the moment of each impact is given by the gait simulating script. In Simulink, this is realized with a logical switch, which is triggered by a logical data set, switching a boolean value at each impact.

From appendix B.2 the complete functionality of the simulation is illustrated. The impact switches flips the signal routing on each impact. From this, the solution with two IMUs appears, either measuring while being the stanceleg or swingleg. If it is preferred to simulate with only one IMU on a single leg, reading the output from the sensor fusion before the second impact switch is enough.

4.4.2 Results

The described model was simulated, attempting to answer the mentioned key questions. Since there are several variables preferred to run with different values, a set off "Standard values" is set, which will be used unless specified differently.

Gait Pattern	test poly 7 004 .xml
IMU-position	0.3 [m]
Resonant frequency, fusion filter	0.2 [Hz]
Sample time	5 [ms]
Bias removal	90%
Noise	Off

Table 6: Standard Values Simulation

The physical parameters used in the simulation are the ones provided in the set of scripts provided. They are not accurate to the physical robot, but in lack of better parameters they were the ones to be used. Nevertheless the simulation should provide a better insight in configuring the IMU.

4.4.2.1 Robot gait-pattern

Running the simulation with different gaits yields different results. The more "violent" the gait, the more unwanted acceleration will be experienced by the IMU, resulting in a less precise measurement. This is quite clear in figure 4.10, where the result from three different gaits are illustrated. "test poly7 004 .xml (which is the gait that ran as default throughout the simulation) gives the best result. It is also the mildest of those three, with the lowest frequency and amplitude. It is also worth mentioning the difference between stanceleg and swingleg. In figure 4.10 the IMU starts as stanceleg, resulting in the other half as swingleg. There is a striking difference in how close θ_c is to θ when operating on stanceleg or swingleg. This is a good indication operating with two IMUs and using the values from the stanceleg at all times might seem like a good choice.

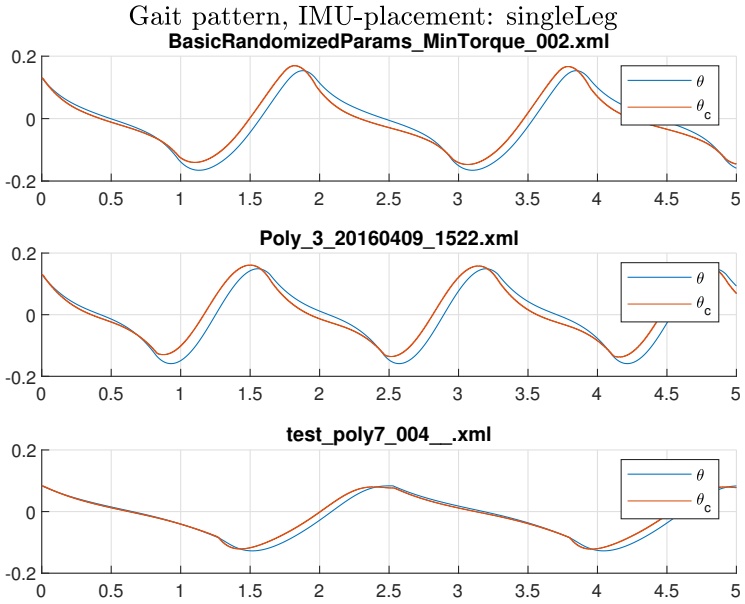


Figure 4.10: Different gaits

4.4.2.2 IMU-placement

When it comes to placement of the IMU, answering both which body part (stanceleg, swingleg or torso) and were on that part the IMU should be placed is preferable. It is natural to first decide on the preferred body part. After running simulations on each alternative (figure 4.11), it looks majorly beneficial to operate with two IMUs, one on each leg. It is also worth mentioning the seemingly poor performance of having the IMU on the torso.

Next up is finding the exact placement of the IMU on each leg, most likely the same distance from the rotary axis. The results are actually best illustrated when simulating with a single IMU on one leg. One of the key elements in this matter is actually what happens with the IMU-measurement on the swingleg. Figure 4.12 shows the IMU starting on stanceleg, then at around T = 1.25s, the other

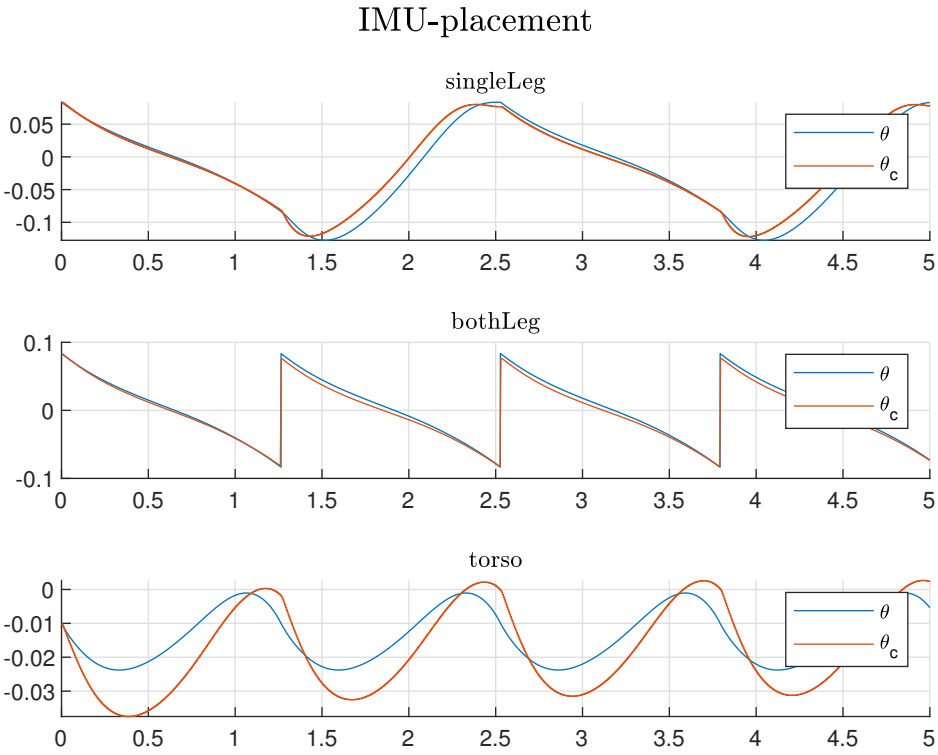


Figure 4.11: IMU placed on different bodyparts

leg impacts and this leg swings forward, making an impacts at around T = 2.5s. When operating with two IMUs it is only the part as stanceleg which is useful. Therefore the measurements do not need to be accurate when perating as swingleg. In this plot the measurement is quite accurate for stanceleg, but not for swingleg. On the other hand, as the IMU is moved up towards the hip, it is quite accurate for swingleg and slightly less accurate for stanceleg. The important part is that during the transition from swingleg to stanceleg, the measurement is less accurate the closer the IMU is placed to the legtip. This will also affect the measurement after the transition. For this reason, it is actually not preferable to place the IMU as close to the tip of the stanceleg as possible, but rather a compromise somewhere in between. In the simulation, a distance of 0.7meters from the legtip is what actually gives the best results. The measurement is also satisfying while the leg is swingleg, resulting in the correct starting angle as it transition into stanceleg. However, this simulation does not take into account the effect of the leg's impact on the ground. These will most likely cause some issues with the accelerometer and might change the suitable solution. In addition, the parameters of the simulation operates with a leglength of 1m. However the physical legs of the robot is approximately 0.63m. Hence solution with 0.7meter does not apply for the actual robot. However it is rational to suggest a placement close to the hip center as ideal. Therefore the solution with 0.5m will be provisional considered the best.

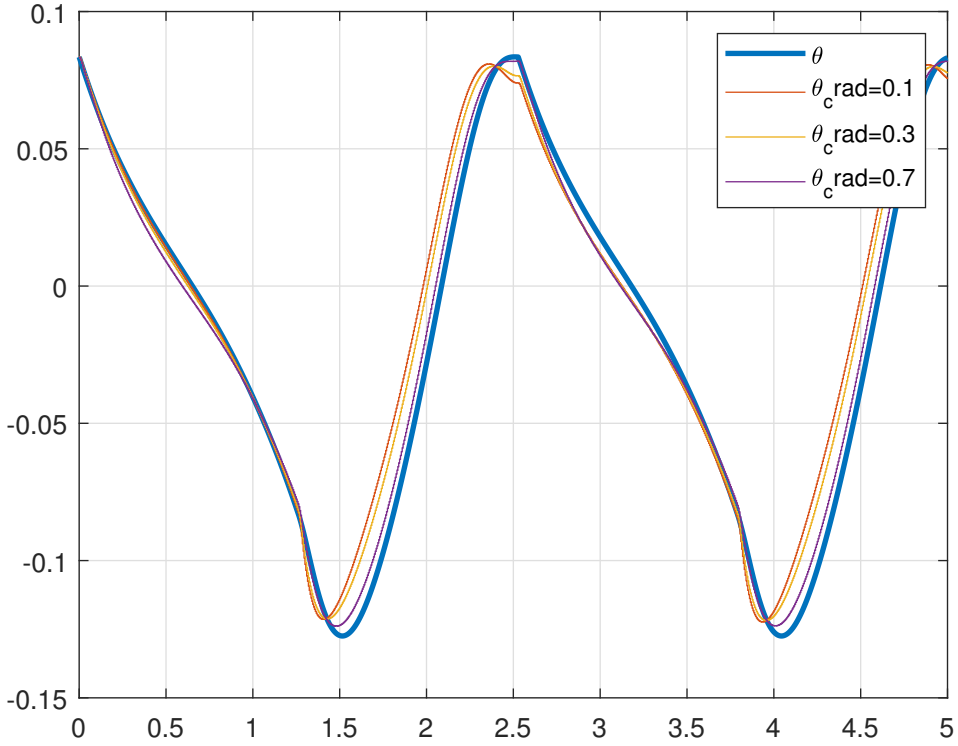


Figure 4.12: IMU placed on different places on single leg

4.4.2.3 Sensor fusion

To find the ideal configuration for the sensor fusion, information about the real system and the exact bias drift from the gyro, the amount of noise and the effect of impacts when walking are needed. These will all influence how to configure the sensor fusion. A look at figure 4.13 will illustrate the effect of the sensor fusion. Here there are three plots. The "Accel filtered" shows a linear simulation of the transfer function from the contribution of the accelerometer, with the unfiltered contribution being the gray curves. The Gyro filtered plot shows the same for the gyro. To summarize, this plot illustrates the contribution from both the gyroscope and the accelerometer and the complete sensor fused value, all with different resonance frequencies for the sensor fusion algorhithm. This plot is made with the IMU placed on the torso. The reason for that is to best illustrate the effect of the sensor fusion, particularly the drift removal of the gyro.

When taking a closer look at the accelerometer filtering, the measurements are quite similar to the unfiltered measurement(gray), with a resonance frequency of 2Hz. As we lower the frequency, the measurement gets filtered towards a stationary value. This might seem odd, but the "overfiltered 0.02Hz" measurement is actually the best. The important part of the accelerometer contribution is to be a periodic function with the same mean value as the actual angle. The amplitude of the function should be as small as possible, except for when the accelerometer is able to measure correct angles (only gravity affecting the accelerometer). In reality, this is just when the IMU is stationary.

The key part of the sensor fusion lies in the contribution from the gyroscope. The drift of the unfiltered gyro is too substantial to ignore and most of the filtered measurements do not drift at all. However, the shape of the curves change as we increase the resonance frequency. This part

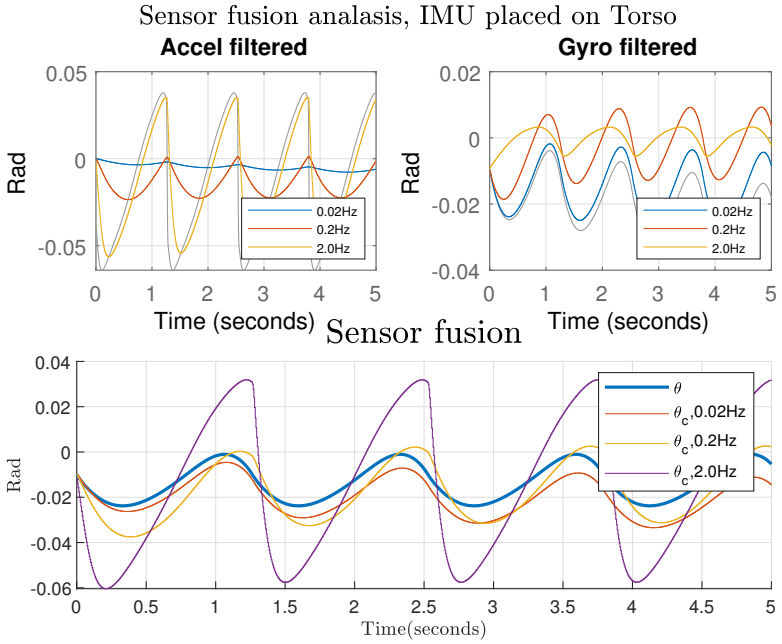


Figure 4.13: Sensor Fusion analysis

gets filtered towards zero. When using a resonance frequency that is to small, the drift is not removed. It is desirable to maintain the shape of the unfiltered curve, but without the drift. Hence a resonance frequency of 0.2Hz seems to work quite well.

A closer look at the sensor fusion plot reveals that a resonance frequency of 0.2Hz gives the best results out of the three. It is not perfect, but this may be improved with different strategies. For example, changing IMU-placement to the legs. It is also worth mentioning that these results depend on the amount of gyro bias one is able to remove. The desired configuration is to use as low resonance frequency as possible without the drift. If more of the gyro bias drift are removed, a lower resonance frequency could be used.

4.4.2.4 Sample time

The effects of running with different sample times within reasonable limits should not affect the measurements, except for how the different samples results in different resolutions. However, there is one important variable that needs to be changed with the sample time. That is the constant of the sensor fusion algorithm, α. As illustrated in figure 4.14, running with different sample times without updating α yields different results. The simulation with sample time of 1ms return the least accurate result. The plot below with α adjusted according to the sample time shows how this fixes the problem. When setting up the real system, α can not be set as an initial value, but must be continuously updated according to the sample time if the system runs without a constant sample time.

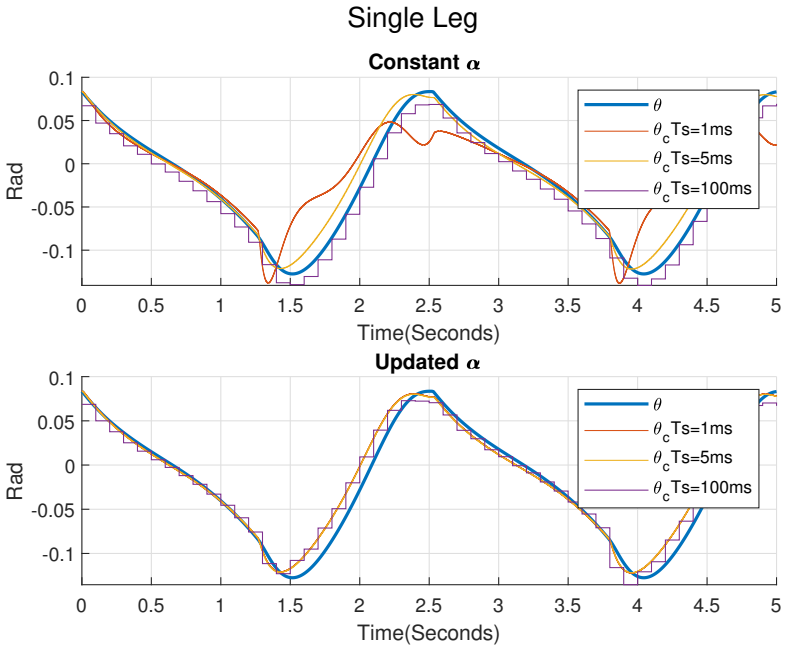


Figure 4.14: Sample time analysis

4.4.2.5 Bias Removal

From the simulation shown i figure 4.15, one can see that the bias drift from the gyro measurement results in a stationary bias value in the measured sensor fusion. This error gets reduced the more the gyro drift is removed. It looks as though removing at least 90% (bRFg = 0.9) of the bias is needed to get acceptable results. This fact makes it a priority to figure out a satisfactory method to identify and remove the bias. One solution that comes to mind is to solve this in software with a calibration algorithm at startup. The algorithm needs to identify if the gyro is stationary and then save the gyro value as the bias. The IMUs are going to need a calibration algorhitm at startup anyways in order to get the correct starting angles, both for the IMUs and the encoders.

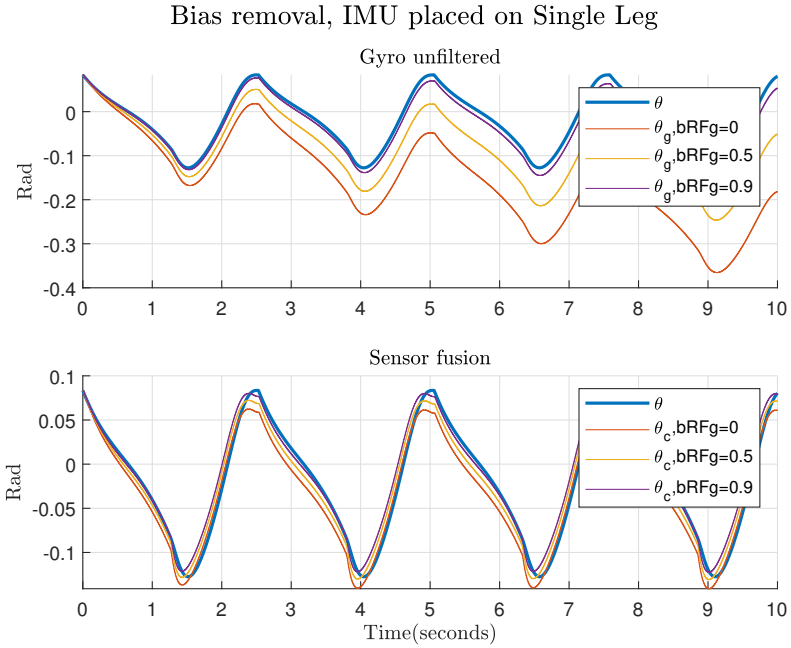


Figure 4.15: Bias removal plot

4.4.2.6 Noise

This simulation verified the reason why the complementary filter is needed. The accelerometer is very sensitive to noise. The gyroscope however, is significantly more sustainable to noise since the raw measurements are integrated to get the angle. In figure 4.16 the numbers describing each curve (0, 400, 4000) are just the factors the noise is scaled up with. These numbers are loosely based on some noise measurements were the scalefactor of 400 are the most realistic and the scalefactor of 4000 gives more noise than is to be expected . This is most certainly not confirmed values. The main reason for this simulation is to see how the system responds as more noise is added. The unfiltered accelerometer gets very distorted as the noise increases, but the sensor fused value seems to only get affected by the gyroscope noise.

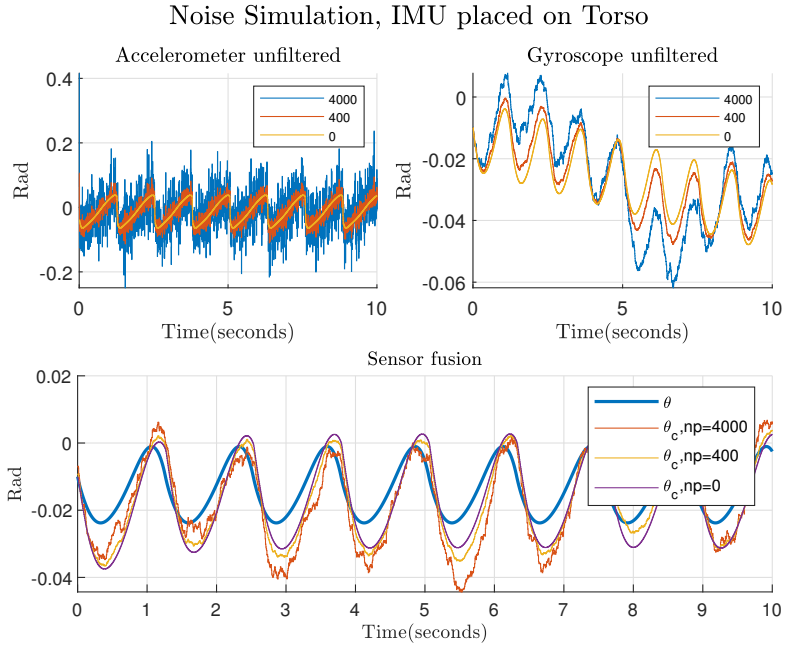


Figure 4.16: Noise plot

In section 4.3.3 it was mentioned that the low frequencies get blocked out from the gyroscope and the high frequencies from the accelerometer. While the noise frequency is above the chosen resonance frequency of the complementary filter, it will be blocked from the accelerometer contribution, while the noise from the gyroscope will be let through. In figure 4.17 this effect is shown. In addition, from the Fourier transform of the two signals the difference in amount of noise on the filter inputs is clear. This is as mentioned because the gyroscope angle is filtered by the integration process, resulting in a descent filtered output.

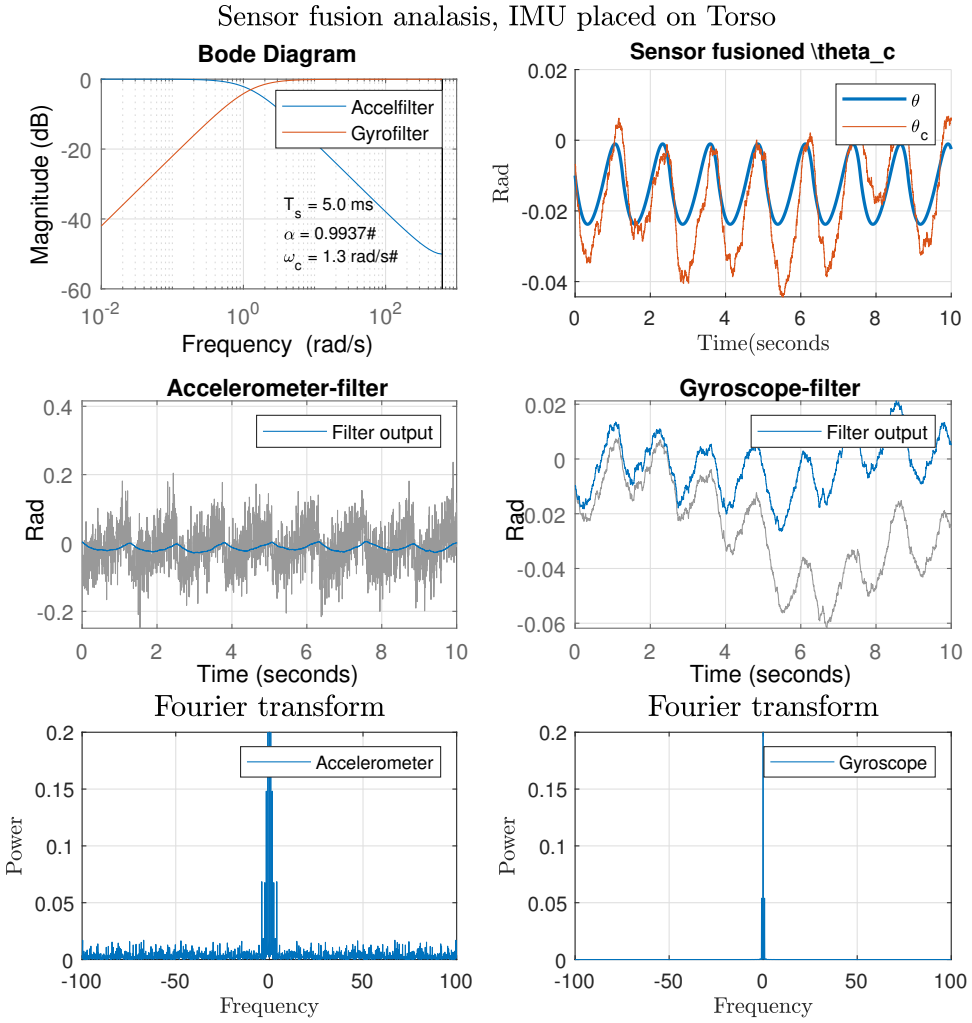


Figure 4.17: Sensor fusion analysis, noise factor = 4000

4.4.2.7 Conclusion

The intention of the simulation was to find the optimal configuration of the IMU. This may have been partly accomplished, but is not possible to verify without testing with the actual robot. Also the flaws of the physical paramters used in the simulation makes the results differ some from reality. A more important aspect of this simulation is to identify the effects different configurations and disturbances have on the system. For example, the improvements of removing the gyroscope bias, resulting in less stationary bias. Some of these findings could be useful during the initial setup of the system, troubleshooting and optimizing the angle measurement if it is not found sufficient. Also, the importance of updating the α is significant since a wrong value could give some unpleasant results. More importantly, identifying the reason could be a vexing practice.

The final optimized result of the simulation is shown i figure 4.18.

These are ran with the same default parameters mentioned earlier, with the exception of the IMU placement, which is 0.5m from the legtip. It is not perfect, but fairly decent with a maximum error of around 0.008 radians which equals around 0.45 degrees. The complementary filter resonance frequency of 0.2Hz might need to be changed. It is desired to keep this as low as possible using only

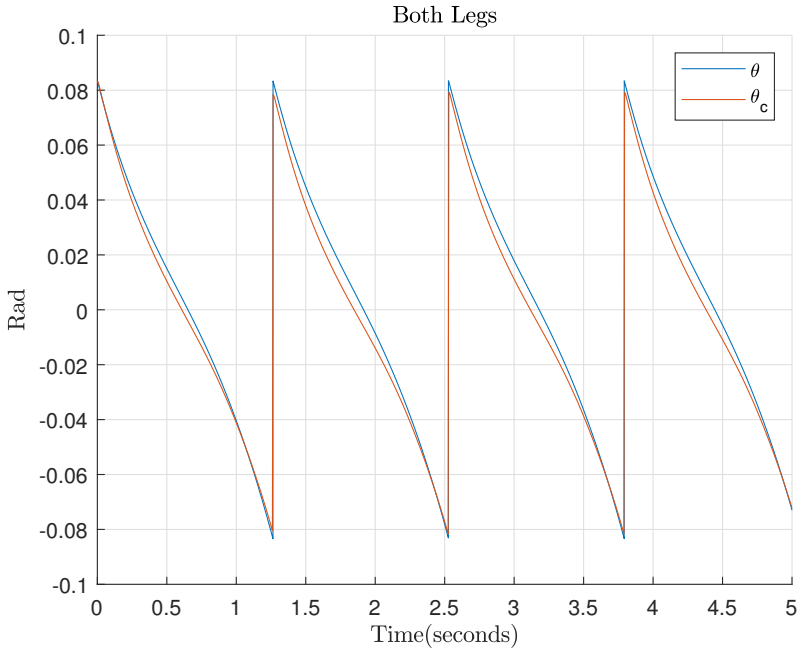


Figure 4.18: Final result simulation

the gyroscope. However, the amount of bias removed dictates how low it can be set without getting a drift in the measurement. After the effect of impacts are taken into account the configuration will also most likely need to be altered.

4.5 Verification

In order to verify the conclusions drawn from the simulation regarding the sensor fusion, several experiments were ran. Also, the effect of *jerks* when the legs hit the ground needs to be answered, since the simulation did not take this into account at all. In total, four different experiments were ran.

- Calibration
- Sine
- Step
- Impulse leg/torso

4.5.1 Setup

The calibration experiments are just running with and without the calibration algorithm.

The swing experiments are ran with an IMU placed on a leg (2nd and 3rd mounting holes from legtip using legmounting case). The motors drive the leg in different motions, while the torso is kept in place. Since the angle between the torso and the leg is measured from the encoder, this can be used as a reference and compare the angle-measurement from the IMU to the encoder angle.

The impulse experiments are ran with an IMU placed on a leg (2nd and 3rd mounting holes from legtip using legmounting case), or an IMU placed on the torso (1st mounting hole from hip joint using torsomounting case). In this experiment the encoder values are useless since the torso can move. This means there are no referance angles to compare with the three angles from the IMU, only gyroscope or accelerometer and the sensor fusioned angle. As discovered from the results, the gyroscope is barely affected by the jerks, therefore the gyroscope is used as a reference. Keeping in mind the bias drift (which is nearly nullified by the calibration), the curve shapes are the interesting part. Any offset values (within reasonable restraints) are most likely caused by the bias drift of the gyro.

In the sine-experiment the leg-motors are fed with a sine-input, driving the leg with a sine-motion within certain limits of the frequency. If the sine input frequency is too low, the result will look like an alternating step. Due to some faults in the wiring the actuation of the motor did not function optimally at the time these experiments were ran. The wiring was fixed later, but not in time to run the experiments again. Nevertheless, this has no effect of the results with the exception of getting a clean sine wave motion from the legs.

In the step-experiment the leg-motors are fed with a step-signal, holding it for 1 second then the input-signal is set back to zero. This results in a swinging motion as if someone lifted up one leg and released it, but in a more controlled setting.

In the impulse-experiment the robot legs are constrained, see figure 4.19, making the angle between them a set constant. One set of legs is dropped from a certain height, causing a jerk to the robot when the legs hit the ground. The IMU is placed either on the swingleg or on the torso as illustrated. For practical reasons, this experiment had to be ran with a rather "unrepeatable setup" using a cardboard plate to achieve approximately the same distance for the swingleg to fall. There were no tools to measure the timing of the fall accurately since the fall was triggered by a human removing the cardboard plate. The key part here is not the exact results from exact inputs, but rather to illustrate and identify the effect of the sensor fusion when experiencing jerks.

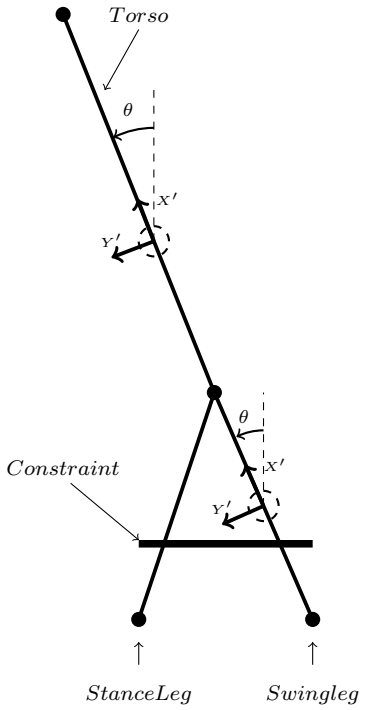


Figure 4.19: Impulse experiment, IMU placed on swingleg OR torso

4.5.2 Arduino code

In order to extract the data from the Arduino, a program called *HyperTerminal* was used. It communicates with the Arduino's serial ports and prints the result to a .TXT file. These files are read by Matlab in order to plot the results. The reading of the encoders are based on the implementation found in [56].

There were some issues with these experiments. Since there was no functional BeagleBone setup at the time, the code to run this experiment was made with an Arduino Mega ADK. It has a processor running at 16 MHz, which is low compared to the 1 Ghz on the BeagleBone. Printing values in particular is a rather time consuming process and should be kept at a minimum. This causes some constraints regarding sampling period. There seemed to be some kind of drift on the encoder measurements as well. The reason for this was not identified, but it is suspected the slow processing speed of the Arduino could be the problem. After discovering a fault in the Arduino code regarding the sampling of the IMU, the drift stopped. The sampling period of the IMU did not match the sampling period of the runcode in Arduino. This may have caused the runcode to stop and wait for a new set of values from the IMU when called upon. If the encoder moves to a new "tick" while waiting for a new IMU-value, it is plausible it will not be registered and this will result in the encoder measurement drifting off of the correct value. When this was fixed, and the runcode had the same sampling period as the IMU, the encoder measurement no longer drifted.

A calibration algorithm was made in order to remove most of the bias of the gyro and also for making sure the correct starting angle, both for the sensor fusioned angle and the encoder, was found. See appendix B.1. The algorithm uses the gyroscope to check if the IMU is stationary.

After it has been stationary for a set amount of time, it saves the gyrovalue and calculates the starting angle based on the accelerometer measurement. In order to remove some influence by noise, this is done several times and the average of these values are the ones to be used.

4.5.3 Results

4.5.3.1 Calibration

To test the effect of the calibration, the IMU is just left stationary, with and without calibration (see figure 4.20). Without any bias removal the gyroscope angle already drifted off by more than 100°in 15 seconds. In the sensor fusioned angle this results in a major offset of over 6°. With the calibration, this drift is reduced to around 0.3°in the same period of time, resulting in a offset of around 0.04°to the sensor fusioned value. Since the bias drift is preventing any further reduction of the resonance frequency of the complementary filter, this looks very promising. These results are unfortunately not completely repeatable, since the exact effect of the calibration will vary every time due to noise. Also, the bias is not some constant value. This is why the averaging part of the calibration is needed.

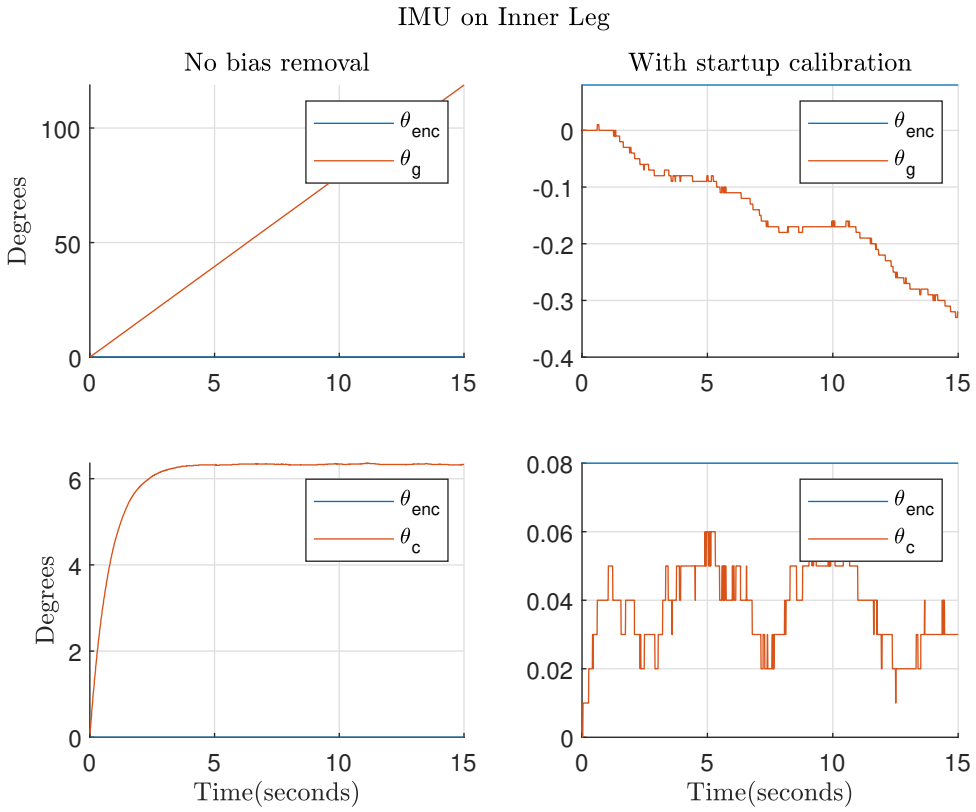


Figure 4.20: Calibration experiment

4.5.3.2 Swing experiment

From the swing (t=4s), figure 4.21, the IMU is off by around 3°from the encoder. This is also by far the "easiest" test, with almost no additional accelerations.

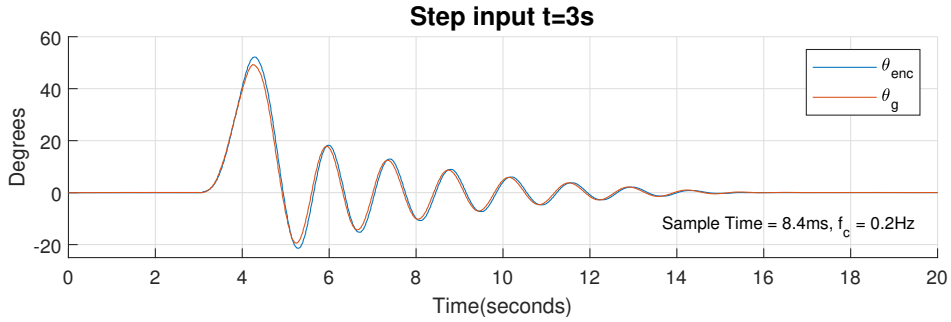


Figure 4.21: Step actuation experiment

In the sine experiment there are additional accelerations that will affect the accelerometer. From figure 4.22 this is quite clear. The angle from the accelerometer is actually phase shifted almost 180°. This causes a small phase shift in the sensor fusioned angle compared to the gyro angle. In regards to noise, the superiority of the gyroscope is undeniable.

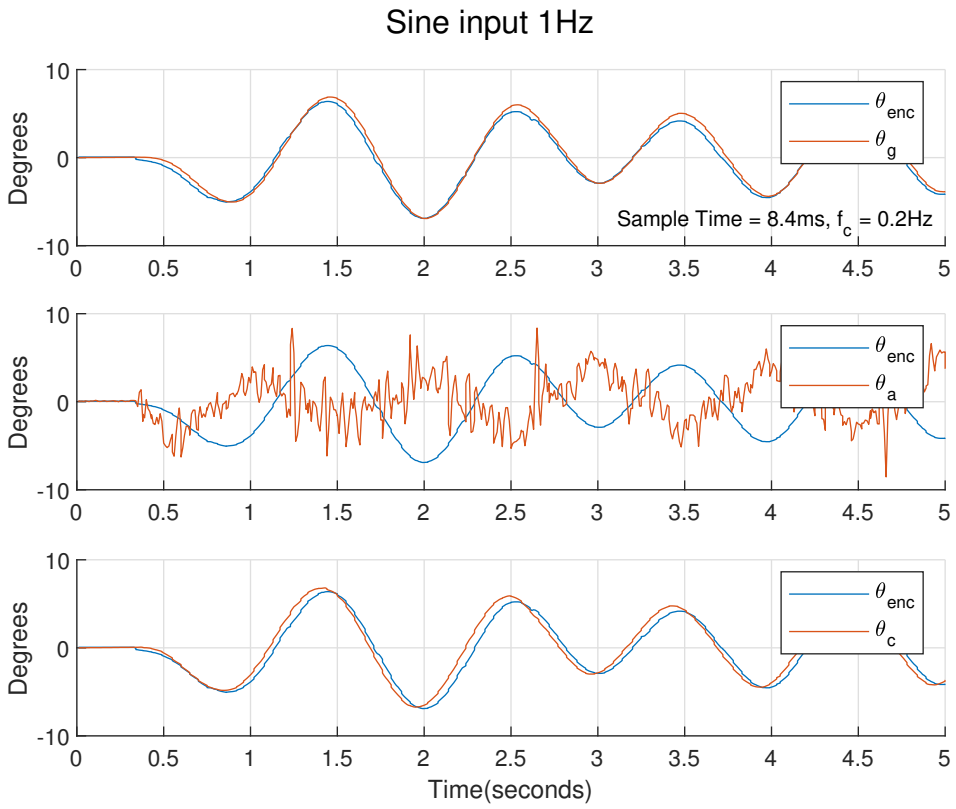


Figure 4.22: Sine actuation experiment

4.5.3.3 Impulse

The most important question with these experiments is how the jerks from the leg impacts will influence the sensor fusion measurement. The initial configuration and a resonance frequency of 0.2Hz yielded rather poor results (see figure 4.23). The accelerometer gets quite large spikes of almost 200°from the impact. The sensor fusioned value gets influence by this, resulting in a big overshoot of almost 15°and a total of 3 seconds longer before it reaches a stationary value compared to the gyro.

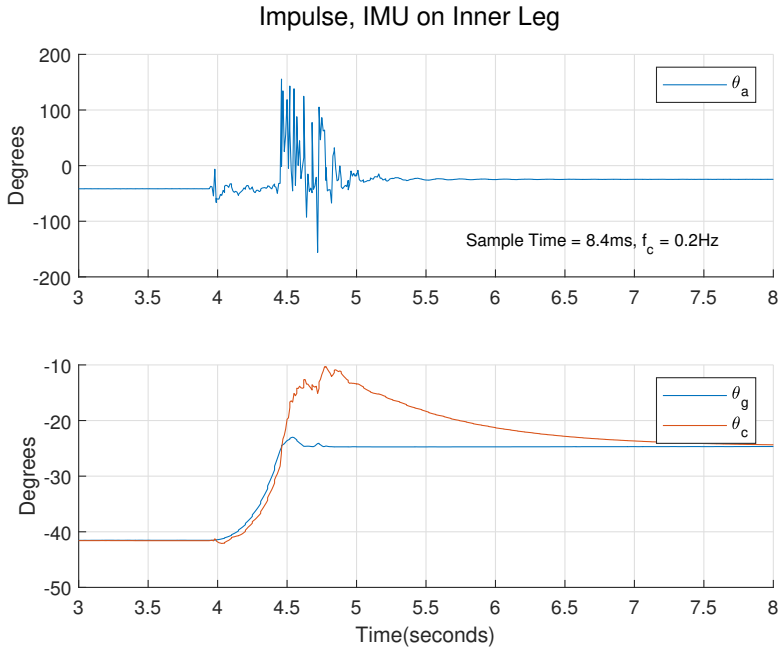


Figure 4.23: Impulse experiment Inner Leg, $f_c = 0.2Hz$

Because of the great success of the calibration, reducing the complementary filter resonance frequency by some margin should work. Reducing it with a factor of ten, to 0.02Hz, seems to give some promising results (see figure 4.24). Nonetheless, after the impulse there is a difference of around 0.5°between the gyro value and the sensor fusioned value. This does not necessarily mean the sensor fusioned value is wrong, but potentially the gyroscope. One possible explanation is G-dependent bias as mentioned in section 4.3.4.

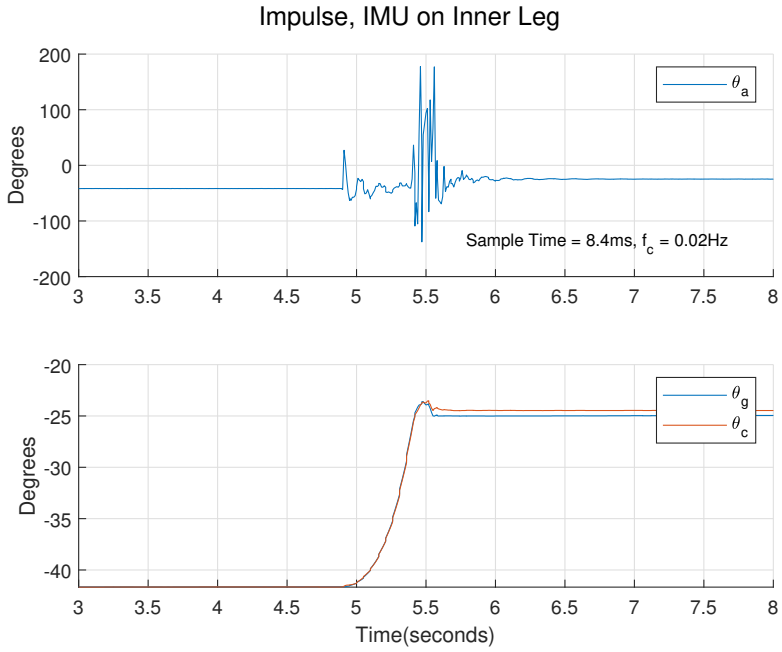


Figure 4.24: Impulse experiment Inner Leg, $f_c = 0.02$ Hz

With the IMU placed on the torso (figure 4.25), seemingly similar results from the sensor fusion was obtained. The spikes on the accelerometer measurement is somewhat different though. With the IMU on the torso it seems like the noise from the impact lasts considerably longer. This might be another indication that the solution with IMU placed on the legs will be a better option.

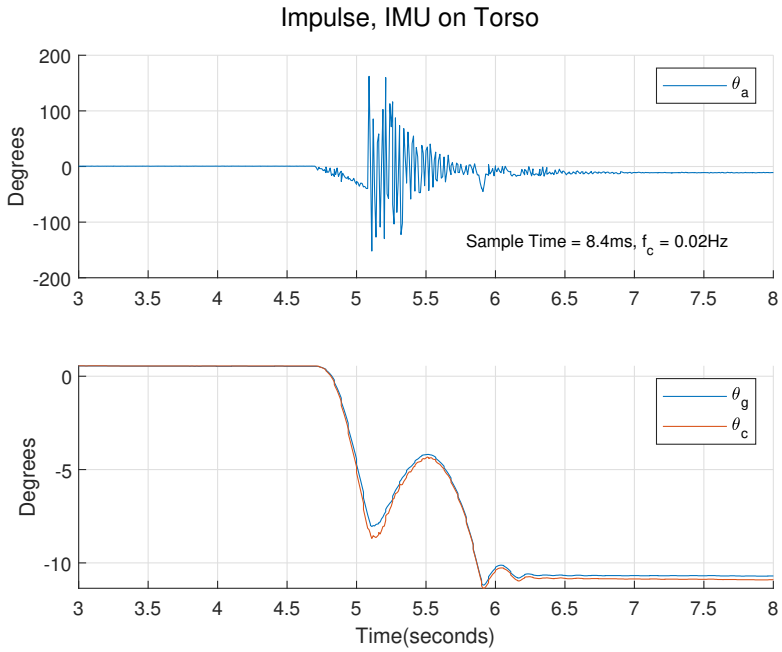


Figure 4.25: Impulse experiment Torso, $f_c = 0.02Hz$

4.5.4 Conclusion

In conclusion, there is plenty of useful data gathered from these experiments. The calibration results in an extended lower limit of a functional resonance frequency, which was already is discovered to possibly be a necessity. From both swing experiments, but especially the impulse experiments, this is most certainly the case. With a resonance frequency of 0.02 Hz, the accelerometer noise from the impacts are suppressed completely, but seem to get a tiny offset. The configuration of the sensor fusion is getting to a point were it almost neglects the accelerometer completely, even tough that is not necessarily for the worse. The accelerometer is needed for both the calibration and the sensor fusion while there is still any bias drift on the gyroscope.

5 Encoder

5.1 Incremental Encoders

In order to acquire a precise and reliable measurement of the angles between the legs and body, angular displacement *transducers*, called encoders, have been utilized. These are instruments that create an electrical signal that is proportional to the angular displacement of the shaft it is mounted on [57]. There are two types of encoders; incremental and absolute, where incremental encoders have been applied on the biped. An incremental encoder is made up of an opaque, rotatable disc with several transparent openings in two circular patterns. The disc is mounted on the shaft to

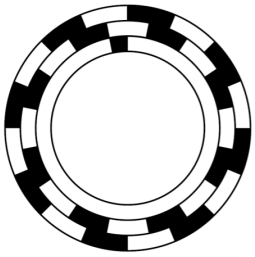


Figure 5.1: An illustration of an incremental encoder's disc. [57]

be measured and is equipped with a light source and a photodetector on either side of the disc. When the shaft is revolving, the photodetector will register pulses as the light is let through. By counting the pulses, one could estimate the developed angular displacement from an arbitrary zero point reference.

$$\theta = n\frac{360^\circ}{N} \tag{5.1}$$

N is the total number of pulses in a revolution and n is the current pulse count. One can decide the direction of rotation by employing two sets of pulses, each from their own circular pattern. The patterns are identical, but are phase shifted by 90°. The direction of rotation can then be decided by which pulse series have the first rising edge.

In order for the incremental encoder to produce absolute angles, there needs to be a defined zero point reference. For this reason, most incremental encoders feature a third sector on the disc with a single transparent opening which is to serve as an absolute, mechanical zero point reference.

In control systems, utilization of incremental encoders is dependent on a device whose purpose is to keep count of pulses and handle data concerning developed angle relative to a calibrated zero point reference. These would usually be connected to the encoder by two channels, each transmitting a pulse series often referred to as "A" and "B" signals.

5.2 Enhanced Quadrature Pulse Module (eQEP)

In order to efficiently read encoder pulses without using the main processor, a module of the PRU_ICSS-subsystem could be used. The-eQEP module is a *cryptographic accelerator*, a co-processor designed to perform one specific set of operations, in this case incrementing a value. For

each pulse received by the encoder, rising and falling edges on signal A and B are received on corresponding eQEP-pins (Figure 5.2)

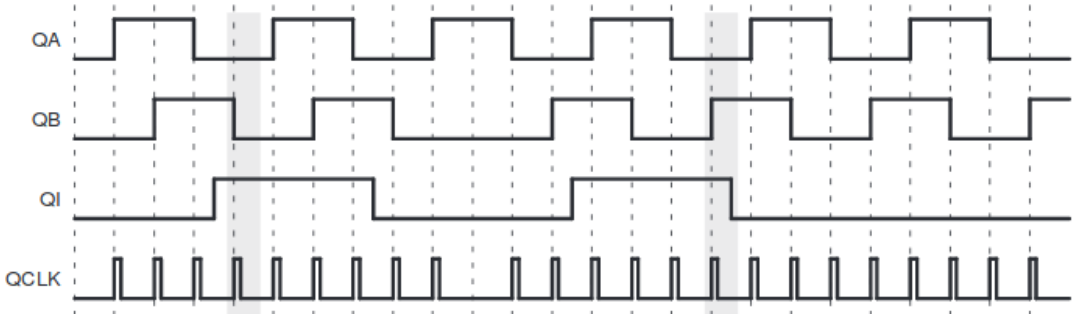


Figure 5.2: Quadrature Pulse readings[11]

5.3 Angle calculation

For every clock period, all edges on both signals are read, resulting in an incrementation four times greater than the encoders resolution. Its angle can be calculated from the measured number of pulses (Equation 5.2)

$$\theta = \frac{\text{quadrature reading}}{4 \times 3000} \times 360 \tag{5.2}$$

The encoders mounted on the robot measures the angular position of the motor shafts. Since the desired measurement is the exact angle of the legs, one has to consider the associated gear reduction, which is 6.

$$\theta_{\text{leg}} = \frac{\theta}{6} = \frac{\text{quadrature reading}}{6 \times 4 \times 3000} \times 360 \tag{5.3}$$

5.4 Implementation

5.4.1 Using eQEP modules on BeagleBone Black

To the extent of measuring the encoders with the BeagleBone Black, one would have to utilize its eQEP-modules; eQEP0, eQEP1 and eQEP2. The module is to be fed the A and B signals of the encoder. The associated pins were found by looking up a header chart for the BeagleBone Black online[17].

Channel	Pin A	Pin B	Notes
eQEP0	P9.27	P9.92	
eQEP1	P8.33	P8.35	Only available with video disabled
eQEP2	P8.11	P8.12	Only available with eQEP2b unused (same channel)
eQEP2b	P8.41	P8.42	Only available with video disabled and eQEP2 unused

Table 7: The BeagleBone Blacks eQEP modules and its associated pins [17].

As mentioned in section 2.4.3, utilizing one of the eQEP modules on a Linux/Unix OS is simply done by managing a few files. If one wanted to use the EQEP2, one would need to start by

configuring its associated pins; P8.11 and P8.12. This is manually demonstrated as the following:

```
root@beaglebone:~# cd /sys/devices/platform/ocp/ocp:P8_11_pinmux/
root@beaglebone:/sys/devices/platform/ocp/ocp:P8_11_pinmux# echo
    qep > state
root@beaglebone:/sys/devices/platform/ocp/ocp:P8_11_pinmux# cd /sys
    /devices/platform/ocp/ocp:P8_12_pinmux/
root@beaglebone:/sys/devices/platform/ocp/ocp:P8_12_pinmux# echo
    qep > state
```

Listing 29: Manually configuring the P8.11 and P8.12 pins to eQEP mode

As a result, one would need to set the poll frequency for the associated character device driver(2.7.6) and read the current pulse count. These files are found in the eQEP module folder which is located within the associated "PWM Subsystem" directory. By referring to *AM335x and AMIC110 Sitara^{TM} Processors Technical Reference Manual*[18], page 184, one will find the following. The

Device Name	Start Address
PWM Subsystem 0	0x4380_0000
eQEP0	0x4380_0180
PWM Subsystem 1	0x4380_2000
eQEP1	0x4380_2180
PWM Subsystem 2	0x4380_4000
eQEP2	0x4380_4180

Table 8: The starting memory address of the PWM Subsystems and the eQEP modules[18]

Start Address of the PWM Subsystem's associated memory is featured in the name of its directories, which are located in the following path: */sys/devices/platform/ocp*. Setting poll frequency and reading pulse count for eQEP2 would thus be demonstrated as the following:

```
root@beaglebone:~# cd /sys/devices/platform/ocp
root@beaglebone:/sys/devices/platform/ocp# find -name "*epwmss"
./48300000.epwmss
./48304000.epwmss
./48302000.epwmss
root@beaglebone:/sys/devices/platform/ocp# cd 48304000.epwmss/
root@beaglebone:/sys/devices/platform/ocp/48304000.epwmss# ls
48304100.ecap  48304180.eqep  48304200.pwm  driver  driver_override
     modalias  of_node  power  subsystem  uevent
root@beaglebone:/sys/devices/platform/ocp/48304000.epwmss# cd
    48304180.eqep
root@beaglebone:/sys/devices/platform/ocp/48304000.epwmss/48304180.
    eqep# ls
driver  driver_override  enabled  modalias  mode  of_node  period
    position  power  subsystem  uevent
root@beaglebone:/sys/devices/platform/ocp/48304000.epwmss/48304180.
    eqep# echo "setting period to 10 ms"
setting period to 10 ms
```

```
14 root@beaglebone:/sys/devices/platform/ocp/48304000.epwmss/48304180.
       eqep# echo 10000000 > period
15 root@beaglebone:/sys/devices/platform/ocp/48304000.epwmss/48304180.
       eqep# cat position
```

Listing 30: Demonstration of setting poll frequency and reading quadrature pulse count

In the end, one would need to apply the position pulse count to equation 5.3 in order to get the absolute angle of the leg.

5.4.2 Encoder Library

In order to simplify the process in section 5.4.1, the peripheral encoder library was developed. It consists of a class called "encoder", which abstracts all necessary file handling required.

```
38 class encoder{
39 public:
40     encoder();
41     encoder(LEG leg);
42     ~encoder();
43     void setPosition_Paths(std::string[2]);
44     void setPeriod(uint32_t);
45     int32_t getPosition();
46     double getAngle();
47 private:
48     std::string eQEPPath;
49     std::string statePaths [2];
50     LEG leg;
51     bool valid;
52 };
```

Listing 31: The "encoder" class, found in the peripheral encoder library

To utilize an encoder with the "encoder" class, one only has to create an instance of it, stating which leg is to be measured. Afterwards, by calling the "getAngle" member function, the program returns the most up-to-date measurement of the angle between actual leg and the body.

```
1 encoder outerEncoder(OUTER);
2 std::cout << "Relative angle between body and outer leg: " << outerEncoder.getAngle
      () << std::endl;
```

Listing 32: Example of using the encoder class

6 Actuation

The following section describes how actuation of the robot is done and the theory behind it. The biped robot have two kinds of actuators; a pair of motors which act as revolute joints and two pairs of servos which function as prismatic joints through a crankshaft mechanism. Actuation is done through a series of controlled *pulse-width modulated* signals (PWM).

6.1 Pulse-Width Modulation

Pulse-With Modulation is a way for digital signals to mimic the energy transfer of analog signals. One does so by chopping a constant electrical signal up into discrete parts, leaving a period consisting of a high cycle and a low cycle. By distributing the duration of the period into the low and the high cycle, one is able to adjust the average voltage fed to the external load. If this is done at a high enough frequency, the device will not be able distinguish the difference between a PWM-signal and the corresponding analog signal.

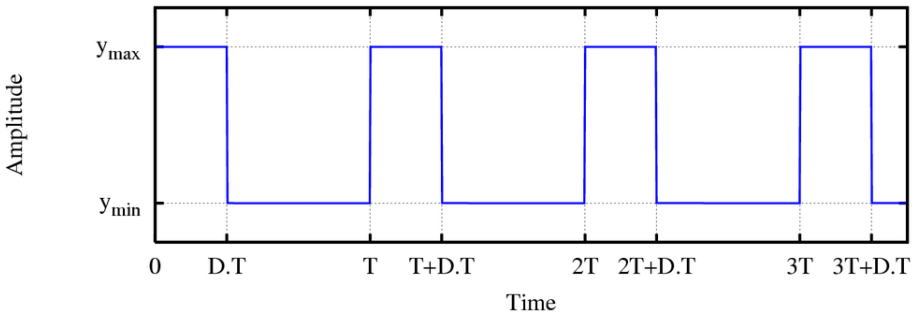


Figure 6.1: Generic form of a PWM-signal

One is able to find the average value of the signal by applying the following equation to the generic PWM-signal.

$$y_{average} = \frac{1}{T}\int_0^T y(t)dt \tag{6.1}$$

$$y_{average} = \frac{1}{T}\Big(\int_0^{DT} y_{max}dt + \int_{DT}^{T} y_{min}dt\Big) = \frac{1}{T}\Big(\Big[y_{max}\Big]_0^{DT} + \Big[y_{min}\Big]_{DT}^{T}\Big) \tag{6.2}$$

$$y_{average} = \frac{1}{T}\Big(y_{max}DT + y_{min}(T - DT)\Big) = y_{max}D + y_{min}(1 - D) \tag{6.3}$$

One is able to see that the term D is the only deciding variable for the average voltage value. D is commonly referred to as duty cycle and is the variable used for control.

6.2 Embedded PWM modules

As mentioned in section 2, the BeagleBone Black features eight PWM outputs. The associated signal output is generated by a PWM module of which the BeagleBone Black has three; EHRPWM0, EHRPWM1 and EHRPWM2. Each of the modules run at its own given frequency and is able to generate two signals with different duty cycles. The last two outputs are generated by the eCAP modules, but these are required for reading the encoder angles.

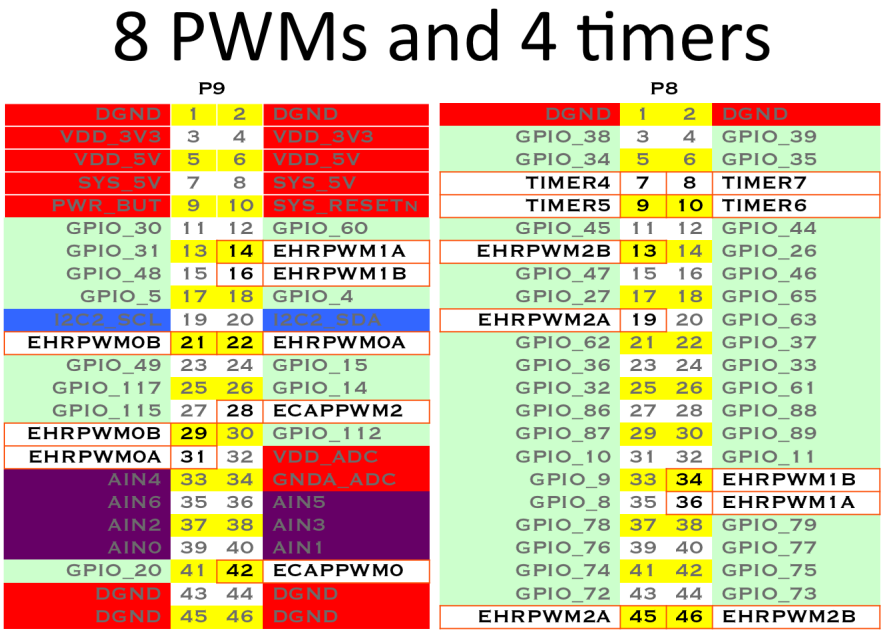


Figure 6.2: PWM layout on the BeagleBone Black [12]

If one wanted to generate a PWM signal out of for example P9.14, the first step is to configure the pin.

```
root@beaglebone:~# cd /sys/devices/platform/ocp/ocp:P9_14_pinmux
root@beaglebone:/sys/devices/platform/ocp/ocp:P9_14_pinmux# echo
    pwm > state
```

Listing 33: Configuring P9.14 to PWM mode

Then one would need to export the right channel and set the desired parameters. In order to do this, one needs to know which pwm chip is associated with the desired module, which in P9.14's case is EHRPWM1.

```
root@beaglebone:~# cd /sys/class/pwm
root@beaglebone:/sys/class/pwm# ls
pwmchip0  pwmchip1  pwmchip3  pwmchip4  pwmchip6  pwmchip7
```

Listing 34: The folder containing the pwmchips.

One can check which memory these are linked to by doing the following:

```
root@beaglebone:/sys/class/pwm# ls -lh /sys/class/pwm
total 0
lrwxrwxrwx 1 root pwm 0 Oct  7 16:40 pwmchip0 -> ../../devices/
    platform/ocp/48300000.epwmss/48300100.ecap/pwm/pwmchip0
lrwxrwxrwx 1 root pwm 0 Oct  7 16:40 pwmchip1 -> ../../devices/
    platform/ocp/48300000.epwmss/48300200.pwm/pwm/pwmchip1
lrwxrwxrwx 1 root pwm 0 Oct  7 16:40 pwmchip3 -> ../../devices/
    platform/ocp/48302000.epwmss/48302100.ecap/pwm/pwmchip3
lrwxrwxrwx 1 root pwm 0 Oct  7 16:40 pwmchip4 -> ../../devices/
    platform/ocp/48302000.epwmss/48302200.pwm/pwm/pwmchip4
lrwxrwxrwx 1 root pwm 0 Oct  7 16:40 pwmchip6 -> ../../devices/
    platform/ocp/48304000.epwmss/48304100.ecap/pwm/pwmchip6
lrwxrwxrwx 1 root pwm 0 Oct  7 16:40 pwmchip7 -> ../../devices/
    platform/ocp/48304000.epwmss/48304200.pwm/pwm/pwmchip7
```

Listing 35: Finding the pwmchips associated memory.

The memory is revealed in the stated directory path as "4830xxxx.[pwm/ecap]". In *AM335x and AMIC110 Sitara™ Processors Technical Reference Manual*[18], page 184, one finds the associated memory to the PWM modules. By cross referencing this with the findings in listing 35, one is able to derive the following:

ePWM0 (EHRPWM0)	pwmchip1	0x4830_0200 to _0260
ePWM1 (EHRPWM1)	pwmchip4	0x4830_2200 to _2260
ePWM2 (EHRPWM2)	pwmchip7	0x4830_4200 to _4260

Table 9: PWM modules and their associated chips and memory

In order to use EHRPWM1, one has to use pwmchip1. In figure 6.2, P9.14's PWM signal is generated by the EHRPWM1A, where the "A" is the channel of the module. To use it, one has to export it by passing a "0" to the export file with the echo command. Likewise, if the desired channel ought to be "B", one would pass a "1".

```
root@beaglebone:/sys/class/pwm# cd pwmchip1
root@beaglebone:/sys/class/pwm/pwmchip1# ls
device  export  npwm  power  subsystem  uevent  unexport
root@beaglebone:/sys/class/pwm/pwmchip1# echo 0 > export
root@beaglebone:/sys/class/pwm/pwmchip1# ls
device  export  npwm  power  pwm-1:0  subsystem  uevent  unexport
root@beaglebone:/sys/class/pwm/pwmchip1#
```

Listing 36: Exporting the pin's associated channel.

The last thing one has to do is to set the parameters.

```
root@beaglebone:/sys/class/pwm/pwmchip1# cd pwm-1\:0
root@beaglebone:/sys/class/pwm/pwmchip1/pwm-1:0# ls
capture  device  duty_cycle  enable  period  polarity  power
    subsystem  uevent
```

```
root@beaglebone:/sys/class/pwm/pwmchip1/pwm-1:0# echo 1000 > period
root@beaglebone:/sys/class/pwm/pwmchip1/pwm-1:0# echo 100 >
    duty_cycle
root@beaglebone:/sys/class/pwm/pwmchip1/pwm-1:0# echo 1 > enable
root@beaglebone:/sys/class/pwm/pwmchip1/pwm-1:0#
```

Listing 37: Demonstration of manually setting PWM parameters, thus actuating the motor

There are two notes to be taken here; the time unit of the period is in nanoseconds and duty_cycle is the time duration of the high cycle, meaning the actual given duty cycle being 0,1 in listing 37.

6.3 Motor control

6.3.1 Torque and Angular Speed relation in a DC-motor

Motors are used for generating mechanical energy in order to move a physical body. If one were to choose an electrical DC motor for this, one need to understand the relation between the angular velocity and torque and how to control these. To create such a relation, one may start by considering the conversion between electrical and mechanical energy by combining their following relations to power:

$$P_{Electrical} = P_{Mechanical} \tag{6.4}$$

Where

$$P_{Electrical} = UI \ , \ P_{Mechanical} = \tau\omega \tag{6.5}$$

This does not take energy loss into account, thus making the conversion unrealistic. To fix this, one has to consider the characteristics of the DC motor. Modelling a DC-motor is commonly done through some variation of the following drawing:

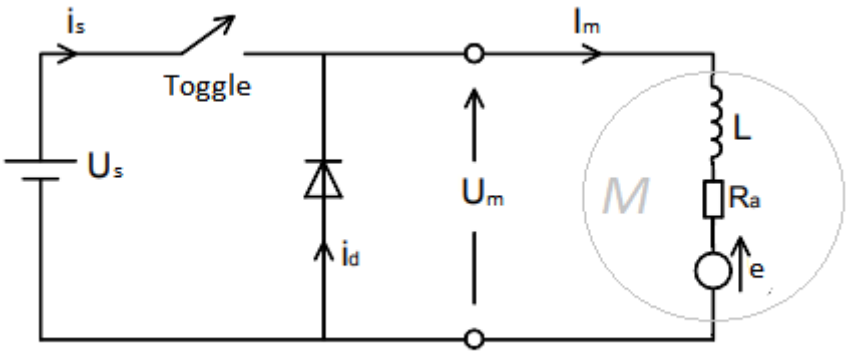


Figure 6.3: Controlling a DC-motor with a step-down chopper scheme

One can see on the right side that a coil, a resistor and a voltage source is connected in series. By

considering the following equations, one is able to express the sum of the power loss:

$$P = UI = I^2R \ , \ U_L = L\dot{I} \tag{6.6}$$

Thus

$$P_{Loss} = P_{Ra} + P_L = U_{Ra}I_m + U_L I_m = I_m^2 R_a + L\dot{I_m} I_m \tag{6.7}$$

One is then able to combine equation 6.4 and 6.7, and thereby make the ensuing relation:

$$P_{Electrical} = P_{Mechanical} + P_{Loss} \tag{6.8}$$

$$U_m I_m = \tau\omega + I_m^2 R_a + L\dot{I_m} I_m \tag{6.9}$$

By dividing equation 6.9 by current, one develops an expression for a balance of voltages.

$$U = \frac{\tau}{I_m}\omega + R_a I_m + L\dot{I_m} \tag{6.10}$$

The term $\frac{\tau}{I}$ is an important ratio as it is known as the motors torque constant, which is found in every motors specifications sheets. Since this is a constant ratio, the following expression applies:

$$\tau = \frac{\tau}{I_m} I_m = K_t I_m \tag{6.11}$$

Now for this project's intents and purposes, equation 6.11 might seem sufficient for controlling the motors, but it does not describe distribution of power and speed of the motor. By substituting the I terms in equation 6.10 with 6.11 and then solving for ω, one develops a differential equation describing the relation between speed and torque.

$$\omega = \frac{1}{K_t}V - \frac{R_a}{K_t^2}\tau - \frac{L}{K_t^2}\dot{\tau} \tag{6.12}$$

This describes the motor in both steady and transient state and might prove to be useful in the development of the robot's gaits. By setting the differential term to zero, one would only consider the speed-torque relationship during the motors steady state.

$$\omega = \frac{1}{K_t}V - \frac{R_a}{K_t^2}\tau \tag{6.13}$$

This implies that the stall torque, which is developed when ω is forced to zero (due to some form of load), is given by

$$\tau = \frac{K_t V}{R_a} = K_t I_m \tag{6.14}$$

However, one should consider equation 6.7, as high torque-zero angular speed operation means nearly all power is being delivered to the inner resistance and coil. This leads to a temperature build that may damage the inner wiring and cause failure. For this reason, every motor has a maximum continuous current that must be respected in further design.

6.3.2 Implementation

6.3.2.1 PWM Current Control

The motor in figure 6.3 is being supplied by a step-down chopper scheme, which is applicable to the project's extent as it involves an ESCON 70/10 Servo Controller. The controller, which functions as a PWM-generator, also features current control which makes I_m directly controllable. This means one only needs to find the required torque to work out which amperage is going to be applied to the motor by using equation 6.11. As the current configuration features a gear ratio by 6, the resulting torque is six times the developed torque from the motor, not considering gear efficiency.

$$\tau = 6\tau_m = 6(K_t I_m) \tag{6.15}$$

This implies that the required amperage is given by

$$I_m = \frac{\tau}{6K_t} \tag{6.16}$$

The maximum required torque was found in the Masters thesis *Stable Gaits for an Underactuated Compass Biped Robot with a Torso*, chapter 4, to be about 2.6Nm. This was rounded up to be 3Nm as having some extra margin if error might be desirable. By applying equation 6.16, the required amperage was found to be 8.3A.

To configure the controller, one has to connect it to a laptop and use the *Setup Wizard* in *ESCON Studio*[58]. Configuration should be set to current control of a DC motor and the following parameters can be found in the Maxon Motor 148877 datasheet[59]. In order for the motor to actuate in both directions, one has to configure the controller to enable either direction by a digital pin.

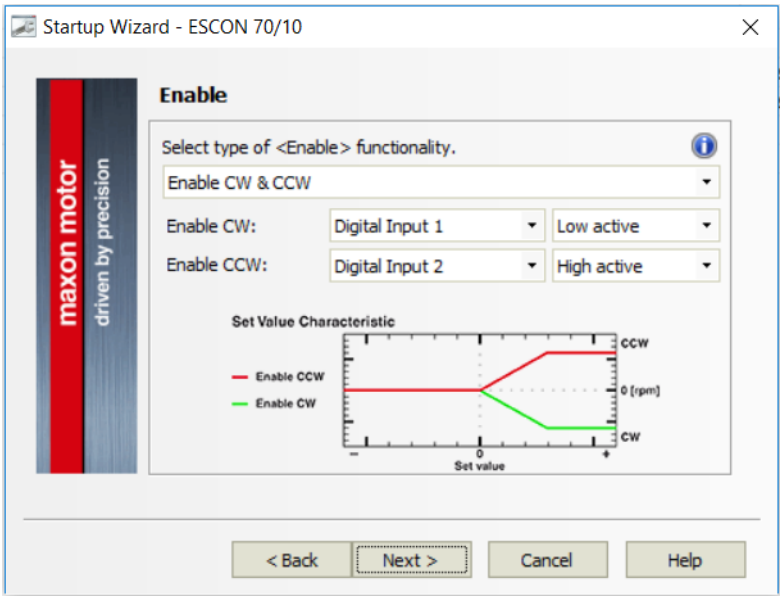


Figure 6.4: Configuring bi-directional motor operation

By leaving the clockwise enabling as low-active and counterclockwise as high-active, one can effectively steer the direction with a single wire and connecting the inputs in parallel.

To control the desired reference of current, one has to feed the controller an analog signal to its differential input. The BeagleBone Black is able to generate a PWM-signal with amplitude 3.8V, although the controller is going to read this as 3.3V as there is a voltage drop of 0.5V over a inner resistance. For this reason, the input is going to be mapped *0 .. 3.3[V]*, while the output is *0 .. 8.3[A]* as shown in figure 6.5

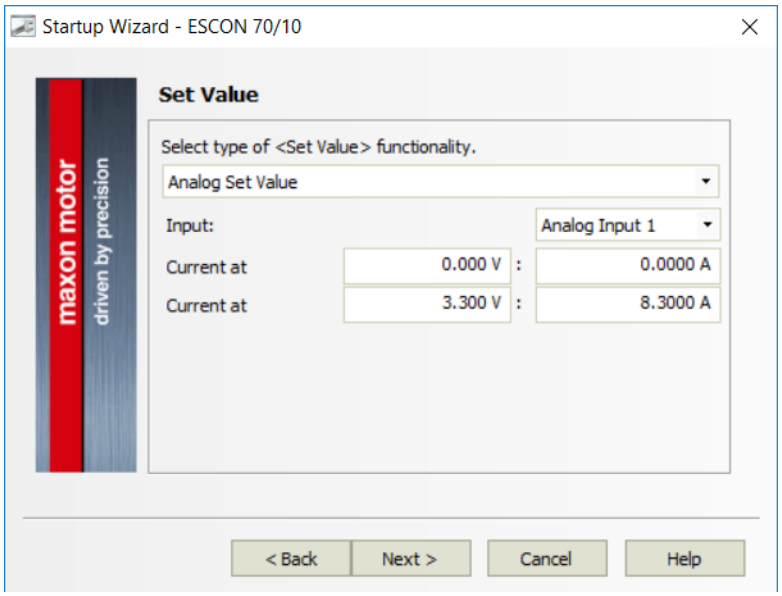


Figure 6.5: I/O mapping of current controller

As demonstrated, it is not a complicated task to reconfigure the mappings if the need should arise. After finishing the configuration, one will be prompted to autotune the controller.

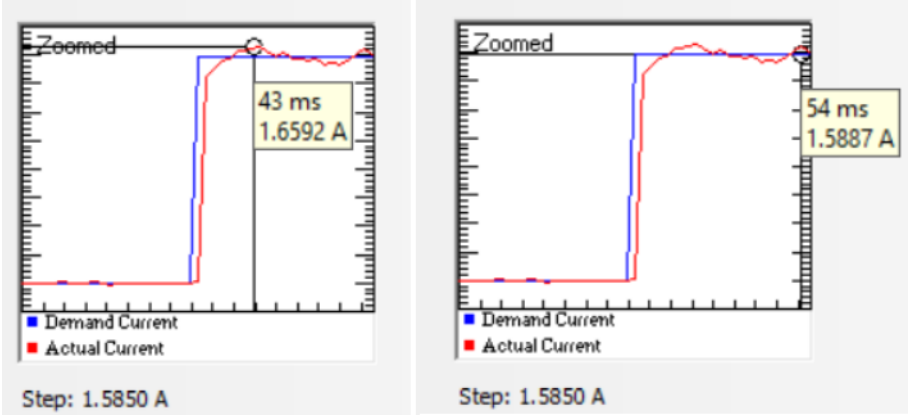


Figure 6.6: Autotuning the controller

This is desirable as the controllers speed and precision become satisfactory. By studying the figures in 6.6, one is able to see a transient time about 1ms and the current does not deviate more that 70mA from its reference.

6.3.2.2 Control with BeagleBone Black

In section 6.3.2.1, the PWM controllers were configured to have an analog and a digital input. As described in section 6.2, to generate a PWM signal with the BeagleBone Black, one has to configure the proposed pin, export the right channel in the associated PWM module and set the desired parameters such as PWM frequency, duty cycle and enable-state. All of this can be done through overwriting textfiles by using the *ofstream* syntax demonstrated in section 2.4.4. This functionality is handled through a class called "motor" in the peripheral "motorLib" library.

```
20 class motor {
21 private:
22   LEG leg;
23   double frequency;
24   int nanoPeriod;
25   double dutyCycle;
26   double sign;
27   std::string anPath;
28   std::string digPath;
29   int setDutyCycle(double dc);  //dc: 0:100 [duty cycle]
30   void exportPWM();
31   void enableDigPinO();
32   void setPolarity(bool pol);
33
34 public:
35
36   motor();              //default INNER
37   motor(LEG leg);
38   ~motor();
39   int setActuation(double act); //act: -100:100 [%]=> -8.3:8.3 [A]~> -3:3 [Nm]
40   double getActuation();
41   void printStatus();       //prints period, on-period and duty cycle to console
42   void setFreq(double inFreq);
```

```
43 };
```

Listing 38: The "motor" class, found in the peripheral "motorLib" library

To utilize a motor with the motor-class, one has to create an instance of it and state which leg is associated with it. Then, one just has to call one of its following member functions listed on line 20 - 23 above.

```
1 motor innerMotor(INNER);     //Create an instance for inner motor control.
2 innerMotor.setActuation(30.0);   //Actuates motors at 30 % duty cycle.
3 innerMotor.printStatus();
```

Listing 39: A demonstration of using the "motor" class.

6.4 Servo control

A digital servo works like a standard feedback control system (figure 6.7) with an actuator and a measuring unit. Usually in hobby servos, a DC motor actuates the servo and a potentiometer gives feedback regarding the servos current position. A control circuit computes the PWM signal(reference) and the voltage from the potentiometer(feedback) into equal units, compares them and gives an output to the DC-motor to remove any difference between the two.

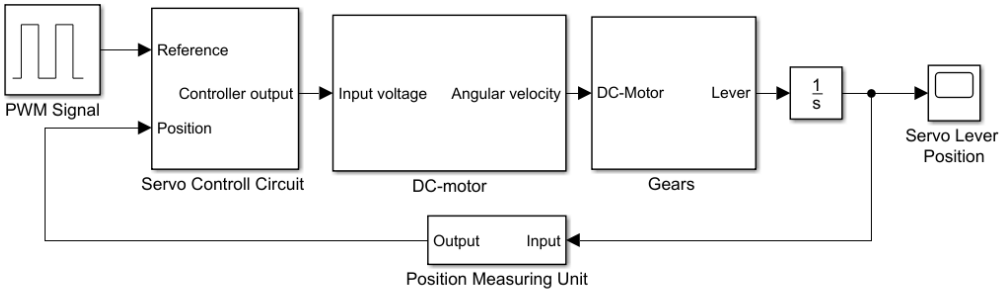


Figure 6.7: Functionality Servo

A PWM-signal within a specific pulsewidth-range is used to control the servo. In hobby servos it is common to utilize a frequency of 50Hz and a pulse width of 1ms as the lower limit and 2ms as the upper limit (some servos span from 0.5 to 2.5 ms). The neutral position is usually somewhere around 1.5ms, but this is not exact because it is not perfectly linear. Also, the limits are not exact and will vary from each servo, even within the same model of servos. The limits are usually not given from the manufactures. However, the neutral position could be, meaning it is needed to find the servo's limits to use its full range. Driving the servo past its limits may cause damage to the servo depending on if it has a fail safe. Many servos also have a physical barrier at their limits, causing damage to the gears one tries to exceed the limit. Especially if the gears are made of plastic which most are. The futuba s2954 has a fail safe within the control circuit cutting the power to the DC-motor if the control signal exceeds the limits. This makes it quite convenient to

find the limits, because whenever the limits are exceeded the servo stops holding its position. Also the sound of the servo adjusting will quiet when the limits are exceeded.

6.4.1 Servo Experiment

In order to find these limits, an experiment was created using an Arduino Mega ADK to generate the needed PWM-signal. To measure the pulsewidth, an oscilloscope was used in addition to the arduino. An additional library (eRCAGuy Timer2 [60]) increased the resolution for time measurements from $4\mu s$ to $0.5\mu s$. See figure 6.8 for the main results of the experiment and appendix D for the complete results.

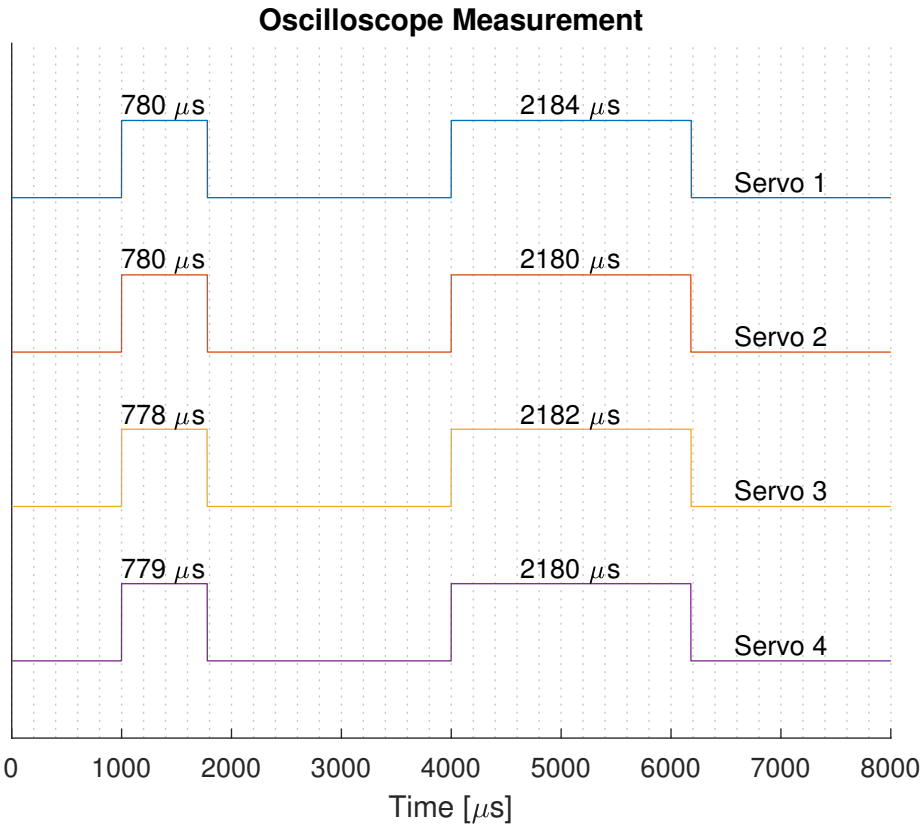


Figure 6.8: Pulsewidth limits

6.4.2 Servo Replacement

Unfortunately, one of the servos got fried in the later stages of the project, servo nr 4. The reason for this is not clear, but it is suspected reversing the connection may have caused this issue. The servo has not been replaced yet due to the timespan of this project as well as a little dilemma regarding availability. See section 10.1.7 for possible solutions.

6.4.3 Control with BeagleBone Black

Controlling the servos were done similarly to the motors, but with some slight modifications. Since the prismatic leg joints were intended to only be fully retracted or extended, the Beaglebone Black was programmed to confine a servo into one of two such boolean states.

In 6.4.1, it was found that the servos are controlled in a specific operating range of PWM signals. As the limits of the operating ranges were not exactly equal, the outermost of the common high cycle durations were chosen; 780..2180μs. These exact signals would also be used for controlling the state of a servo.

In addition, since the robot is strictly intended to move in the saggital plane, the servos on the legs would have to be operated in pairs. This would require extra handling as the servos on a given leg is mounted in opposite directions. This practically means that the servos would have to be operated in different directions in order for the leg's prismatic joints to be in the same state. For extracted state, the right servo in the pair would be given as the 780 μs signal, while the left would be given the 2180 μs one.

Actuation of the servos was implemented in the BeagleBone Black with its own pheriperal library called "servoLib". Within this library, the "servoPair" class exists, which handles all the necessary instructions for controlling a pair of servos in a strict two-state operation.

```
24 class servoPair {
25 private:
26   LEG leg;
27   const int nanoPeriod = 20006000;
28   double frequency = 1000000000/nanoPeriod;
29   std::string pwmPath[2];
30   bool state;
31   std::string ocpPath[2];
32   void exportPWM(std::string path, int i);
33   bool setStatePWM(std::string path);
34   void setFreq(std::string path);
35 public:
36   servoPair();              //default INNER
37   servoPair(LEG leg);          //leg: INNER / OUTER.
38   ~servoPair();
39   void setActuation(bool state);     //state 0 : Retracted,   1 : Extended
40   bool getState();
41 };
```

Listing 40: The "servoPair" class, found in the peripheral "servoLib" library.

The resulting interface for the inner servos would simply be the following:

```
1 servoPair innerServos(INNER);
2 innerServos.setActuation(1);  //Expands
3 std::cout << "Current state of inner servos: " << innerServos.getState() << std::
      endl;
4 innerServos.setActuation(0);  //Retracts
```

Listing 41: Demonstration of using the "servoPair" class.

7 Model identification

The purpose of this task was to make a theoretical model for the swing of each leg of the robot, as well as to identify the localized masses and centers of gravity for each of the robots appendices. As the legs are not symmetrical, each of them requires its own model to describe movement when no controlled forces are acting upon them. Unfortunately, due to an unforeseen loss of equipment, only data for the inner leg was available when making the dynamic models of the legs swinging freely. Because of this, only the model of the inner leg was made. However, the method used in finding this model is applicable to the other leg as well. These models are useful as they describe frictional forces acting upon the robot, both in air drag and sliding friction in the joints and rotary parts. The leg was modeled by swinging it like a pendulum and recording the exact position using the encoders located at the hips.

7.1 Method

Different approaches were tested in order to model the dynamics of the leg. The first attempt was to find a sinusoidal function such that:

$$f(t) = Ae^{-at}cos(\omega t) \tag{7.1}$$

Where A is the initial amplitude, ω is the frequency of oscillation and t is time. This method was based on a 2. degree transfer function with complex poles. Such a function will create an exponentially decaying sinusoidal output where the dampening effect ζ and the oscillating frequency ω are functions of the systems time constant T_k, where T_k is the time it takes the amplitude of the sinusoidal to reach $A_0 * (1 - e^{-1})$. The attempt was not successful, as the system does not fit such a transfer function. Another attempt was made in trying to utilize the Matlab curve fitting toolbox to approximate a sinusoidal function, but this also failed as the data shows that the real system is not a true sinusoid.

The real system is more complex than originally assumed. In reality, a pendulum is a robust object, which means one has to account for rotational forces rather than the common $ma = mgsin(\theta)$. In addition, the oscillatory frequency of the pendulum is not constant, but decreases as the amplitude of the swing increases. Because of this, the next attempt at modeling the leg was based in the dynamics of the swing using differential equations.

7.1.1 Approximation based on harmonic balance

An ideal rigid pendulum with no frictional forces rotating about a fixed horizontal axis gives the following differential equation:

$$I\dot{\theta} = -mgasin(\theta) \tag{7.2}$$

Where I is the moment of inertia about the rotational axis, m is the mass of the pendulum, g is the gravitational constant, a is the distance from the pendulums center of gravity to the rotational axis and θ is the angle of the pendulum. The left-hand side of the equation is the rate of change of the angular momentum and the right-hand side is the system's restoring torque. This equation is commonly rewritten as:

$$\ddot{\theta} + \omega_0^2 sin(\theta) = 0 \tag{7.3}$$

Where ω_0^2 is introduced in place of $\frac{mga}{I}$. The dynamics of such a pendulum is approximately equivalent to a simple pendulum for infinitely small oscillations, with ω_0 representing the angular harmonic frequency of the oscillations. However, due to the presence of friction, the model does not accurately describe the real system. It is assumed that both air drag and sliding friction are acting as non-trivial dampening effects on the pendulum. As for making a more accurate representation of the system, it was decided that Patrick T. Squire's paper on pendulum dampening from 1986 would be used as a basis[61]. This is because his model takes both viscous friction (air resistance) and dry friction (sliding friction) into account, making it more true to a real pendulum than most theoretical models commonly used. Air friction will be approximately proportional to the square of the angular velocity in most circumstances, except for situations with near laminar airflow, in which the drag is proportional to the angular velocity of the 1. power. Both approximations will be used in the full model. The dry friction is assumed to be a constant force, independent of the pendulum's velocity. After adding these terms to the equation, it is now written as:

$$\ddot{\theta} + a|\dot{\theta}|\dot{\theta} + b\dot{\theta} + csgn(\dot{\theta}) + \omega_0^2 sin(\theta) = 0 \tag{7.4}$$

Where a, b and c are coefficients of friction. A modulus sign has been introduced to the quadratic term to ensure that energy always leaves the system. This is also true for the sgn-function in the dry friction term. The sgn-function represents the positive or negative sign of the value of $\dot{\theta}$. The method used to tackle the problem is based on linearization and simplification of the governing equation 7.4 as it cannot be solved exactly. It is assumed that the dampening of the amplitude for each individual oscillation is near linear and a trial function is introduced in place of theta. A trial function is an approximation of the expected solution to equation 7.4 and will substitute the real value θ. Such a function does not actually exist, but the method allows for a linearized approach to the problem. Due to the swing of a pendulum naturally taking the form of a cosine, the trial function takes the form of:

$$u(t)cos(wt) \tag{7.5}$$

where u(t) is a decaying linear function. As shown in figure 7.1, the function u(t) will be approximately equal to:

$$u(t) = A(1 - \frac{kt}{T}) \tag{7.6}$$

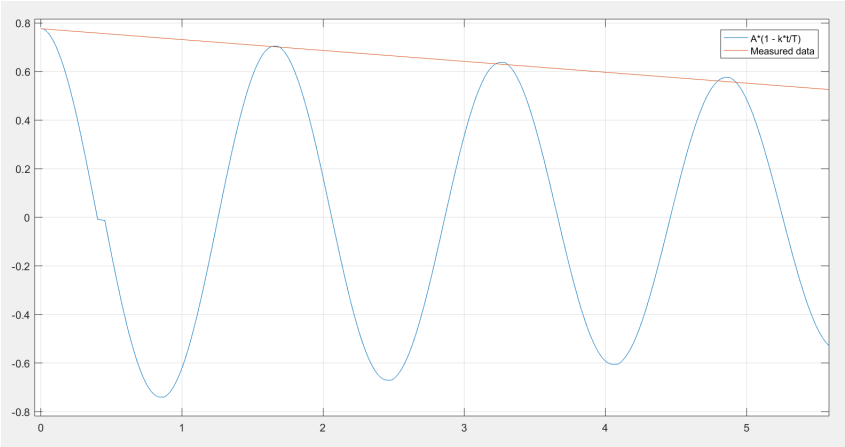


Figure 7.1: Decay in amplitude is approximately $A(1 - \frac{kt}{T})$ for a single oscillation

for a single oscillation. A is the amplitude at the start of the swing, k is the ratio between the change in amplitude between the peaks(ΔA) an the amplitude of the original peak(A), t is time and T is the period of the oscillation. This gives the complete trial function:

$$\theta = A(1 - \frac{kt}{T})cos(wt) \tag{7.7}$$

The method used for solving the equation is based on the theory of harmonic balance. The method of harmonic balance says to equate all harmonic terms of a magnitude equal to the number of adjustable parameters in the original equation to zero. This means that when calculating the result of equation 7.4 with the substituted trial function, all coefficients of harmonics of the 1. order will be set to zero due to there being only one adjustable parameter[62]. It is important to note that assuming the dampening of the pendulum's amplitude over one oscillation is linear becomes less accurate for larger friction coefficients. This means that the method is only applicable to systems with a sufficiently slow dampening. For now it is assumed that this is the case. Equation 7.4 contains three nonlinear terms and is therefore not solvable for an exact solution. To make the problem more manageable, the sgn-function is replaced with the new function

$$sgn(\dot{\theta}) = \frac{2}{\pi} arctan(x\dot{\theta}) \tag{7.8}$$

with the approximation getting better as the adjustable parameter x approaches infinity. Then $arctan(x\dot{\theta})$ and $sin(\theta)$ are both approximated to the first two terms of their respective taylor-series. From this, equation 7.4 now looks like:

$$\ddot{\theta} + a|\dot{\theta}|\dot{\theta} + b\dot{\theta} + c\frac{2}{\pi}(x\dot{\theta} - \frac{(x\dot{\theta})^3}{3}) + \omega_0^2(\theta - \frac{\theta^3}{6}) = 0 \tag{7.9}$$

While the equation is still not linear, it is now possible to substitute inn the trial function and solve it. When the trial function is inserted into the squared term for friction, the two factors $\dot{\theta}$ and $|\dot{\theta}|$ cannot mix. Here the trial function is derivated and becomes the following term:

$$a|\dot{\theta}|\dot{\theta} = \frac{-A^2 k cos(\omega t)|kcos(\omega t) + (T - kt)\omega sin(\omega t)| - A^2\omega(T - kt)sin(\omega t)|kcos(\omega t) + (T - kt)\omega sin(\omega t)|}{T^2} \tag{7.10}$$

By nature of k being "small", the cosine factor may also be considered small. This lets one look at only the sine terms and allows for the approximation:

$$a|\dot{\theta}|\dot{\theta} \approx -A^2\omega sin(\omega t)|\omega sin(\omega t)| \tag{7.11}$$

Figure 7.2 shows a comparison between the exact term and the approximation. The approximation is considered sufficient for a single oscillation and is therefore used.

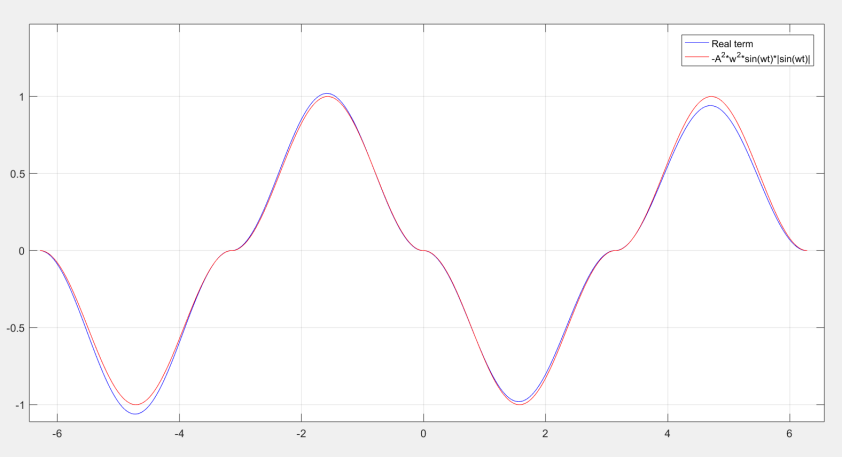


Figure 7.2: Comparison of $A^2\omega|sin(\omega t)|sin(\omega t)$ and $|\dot{\theta}|\dot{\theta}$ over two perods

When solving equation 7.9, all terms containing k^n where $n \in \mathbb{N} \wedge [2, \rightarrow]$ may be neglected due to k being small by nature. The complete answer to equation 1 is gathered in terms of sine and cosine and is as follows:

$$\begin{aligned}
&cos(wt) * (-A\omega^2(1 - \frac{kt}{T}) - b\frac{Ak}{T} - c\frac{2Akx}{\pi T} - \omega_0^2\frac{A^3}{8}(1 - \frac{3kt}{T}) + \omega_0^2 A(1 - \frac{kt}{T})) \\
&+sin(wt) * (\frac{2A\omega k}{T} - a\frac{8A^2\omega^2}{3\pi} - bA\omega(1 - \frac{kt}{T}) + c\frac{A^3\omega^3x^3}{2\pi}(1 - \frac{3kt}{T}) - c\frac{2A\omega x}{\pi}(1 - \frac{kt}{T})) \\
&+cos(3wt) * (-\omega_0^2\frac{A^3}{24}(1 - \frac{3kt}{T})) \\
&+sin(3wt) * (a\frac{8A^2}{15\pi} - c\frac{A^3\omega^3x^3}{6\pi}(1 - \frac{3kt}{T})) = R
\end{aligned} \tag{7.12}$$

Note the use of the identities:

$$cos^3(x) = \frac{1}{4}(3cos(x) + cos(3x))$$
$$sin^3(x) = \frac{1}{4}(3sin(x) - sin(3x)) \tag{7.13}$$

The R on the right-hand side of equation 7.12 is the residual. This value is necessary to balance out the equation since the trial function is non-exact. If all the assumptions and approximations made thus far are accurate, the residual will be negligible. As stated earlier, the next step is to equate the coefficients of the first order sine and cosine terms to zero. This results in two new equations containing the constants ω_0^2, a, b and c.

$$-A\omega^2(1 - \frac{kt}{T}) - b\frac{Ak}{T} - c\frac{2Akx}{\pi T} - \omega_0^2\frac{A^3}{8}(1 - \frac{3kt}{T}) + \omega_0^2 A(1 - \frac{kt}{T}) = 0 \tag{7.14}$$

$$\frac{2A\omega k}{T} - a\frac{8A^2\omega}{3\pi} - bA\omega(1 - \frac{kt}{T}) + c\frac{A^3\omega^3x^3}{2\pi}(1 - \frac{3kt}{T}) - c\frac{2A\omega x}{\pi}(1 - \frac{kt}{T}) = 0 \tag{7.15}$$

One can see that equation 7.15 contains all the friction coefficients, making it possible to solve for their value. Here it is noted that k, a, b and c are all assumed to be quite small, which means that all terms containing ka, kb and kc can be considered negligible. Solving the remaining terms for k gives the following equation:

$$k = a\frac{8A}{\pi} + b\frac{T}{2} - c\frac{Tx}{\pi}(1 - \frac{A^2\omega^2x^2}{4}) \tag{7.16}$$

k is defined as the ratio between ΔA and A, so by multiplying both sides by A the new equation is solvable by putting inn measured data of the pendulum's oscillations. The substitustion $\omega = \frac{2\pi}{T}$ is also introduced.

$$\Delta A = a\frac{8A^2}{\pi} + b\frac{TA}{2} - c\frac{ATx}{\pi}(1 - \frac{A^2x^2\pi^2}{T^2}) \tag{7.17}$$

The resulting equation is only dependent on the coefficients a, b, c and the adjustable parameter x. Next up is finding the value of ω_0^2 using equation 7.14. One can approximate all terms containing ka, kb, kc and k^n in the same way as with equation 7.15. The remaining terms can be solved for $\omega 0^2$ and the resulting equation is given as:

$$\omega_0^2 = \frac{\omega^2(1 - \frac{kt}{T})}{1 - \frac{kt}{T} - \frac{A^2}{8} + \frac{3ktA^2}{8T}} \tag{7.18}$$

An important thing to note is that the adjustable parameter x is seen in the equation for a, b and c coefficients. The variable was originally introduced to more accurately approximate the sgn-function, but as the equations show, increasing the value of x will have a huge impact on the result. Because of this, it was decided that x would remain 1 when calculating the answers. This means that data of small oscillations will give less accurate results as $\frac{2}{\pi}arctan(\dot{\theta})$ is not equal to

± 1. The same can also be said for the factor k which increases in value the smaller the amplitude of the oscillation. It was decided that any data where $k > 0.15$ would not be used.

There were 46 available data sets of a full dampening of the pendulum. The final values for ω_0^2, a, b and c are averages of all the result from each data set. Equation 1 was solved by making three separate equations using measurements from three different oscillations in the same set of data. These three equations were then solved as a set. This exact method is applicable to both legs of the robot.

7.2 Results

7.2.1 Initial result of harmonic balance

As mentioned, the results were derived from averaging values given from 46 different data sets. The final value given for ω_0^2 was $18.9001 s^{-1}$. However, when looking more closely at the answers from each data set, it was discovered that ω_0^2 would vary a great deal between different data (shown in figure 7.3).

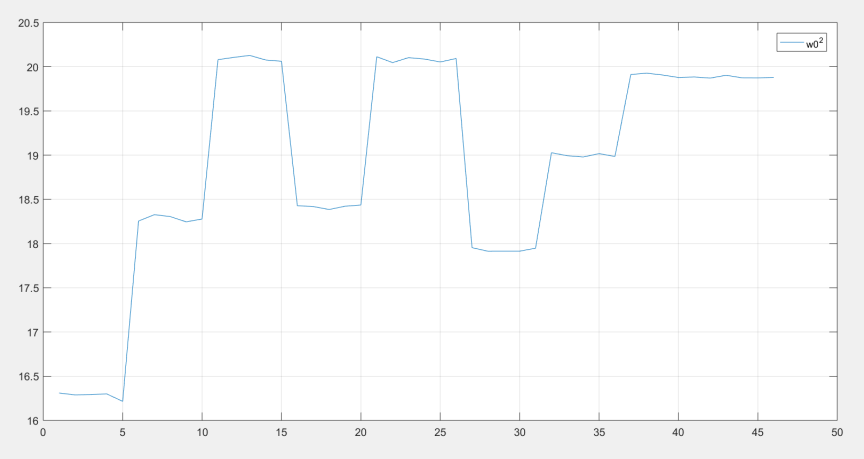


Figure 7.3: ω_0^2 varies substantially between different data sets

Sets off five recorded data sets would share approximately the same result, but the following five sets would give a noticeably different answer. It is currently assumed that this is because the data was originally recorded five at a time and at different times during the day. The data with the slower angular frequency was presumably recorded when the motor was cold, thus increasing the dry friction and increasing the period of oscillation. The different sets of data also vary substantially in length because the cold motor caused the pendulum to dampen much quicker. After this discovery, it was decided that only data from a warmed-up motor would be used to calculate a final answer, as this data most closely resembles the leg when the robot is in use. The angular frequency of the inner leg was given as:

$$\omega_{0,warm}^2 = 19.9930$$
$$\omega_{0,cold}^2 = 16.2814$$

Seeing as there was very little data for a completely cool system, the value of $\omega^2_{0,cold}$ is likely not very exact in comparison to the value given for a warm motor. More data would have been recorded had it not been for an unfortunate loss of equipment in the late stages of the project. The constants a, b and c were calculated to be the following values:

$$a = -0.0572$$
$$b = 0.2130$$
$$c = 0.0939$$

It is important to note that the calculated values for a, b and c varies drastically for each computation (as shown in figure 7.4). A possible way to minimize the influence of variance would be to include more data or to remove any value that deviates more than a set amount from the average. All values used to find a, b and c were within the average value ± 5. A clear trend in the data where b and c follow each other while a mirrors them.

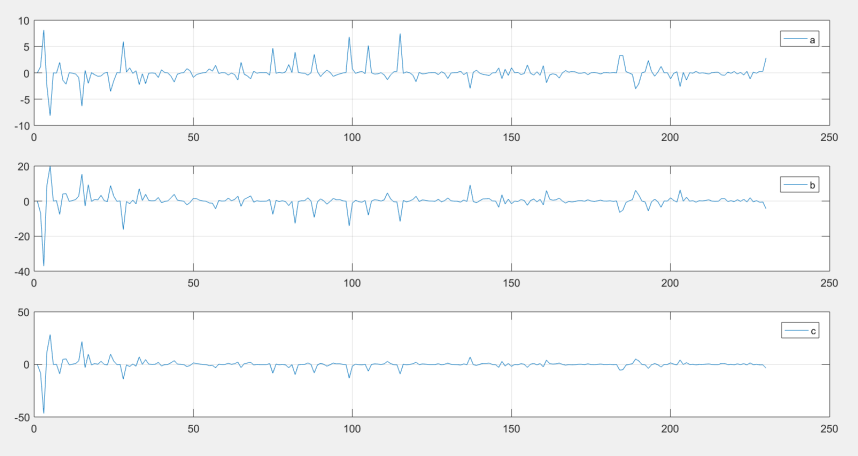


Figure 7.4: High variance in calculated friction (the plot for b and c look similar, but they are not the same plot)

The complete theoretical model of the robots inner leg is

$$\ddot{\theta} - 0.0572|\dot{\theta}|\dot{\theta} + 0.2130\dot{\theta} + 0.0939sgn(\dot{\theta}) + 19.9930sin(\theta) = 0 \tag{7.19}$$

7.2.1.1 Simulation

A program was written in Matlab to simulate the swing of the pendulum using the constants as inputs in the original differential equation. The simulated swing was then compared to multiple real swings to see how accurately the model fits reality. Figure 7.5 shows the comparison to four different data sets.

As one can clearly see, the model does not fit the real swing of the pendulum. The shape of the plot is not correct due to the fact that the constant a is negative. This means the dampening term $a|\dot{\theta}|\dot{\theta}$ is acting in reverse and is adding energy to the system. The initial conditions also seem to

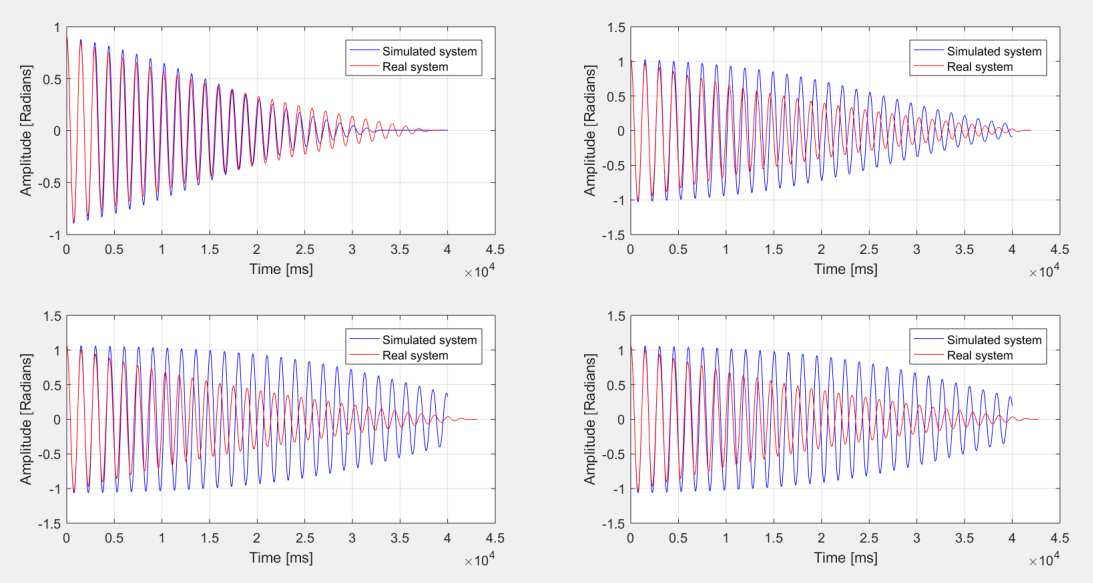


Figure 7.5: Comparison between the simulated system model and the real system

affect the model in a drastic way. The first plot is far better than the following three due to a lower starting point. Both dampening and frequency of oscillation are affected. The simulated system oscillates slower than the real system and falls out of phase. Figure 7.6 shows a simulation where the starting point of the angle is 1.1006 radians. The system becomes unstable and grows indefinitely.

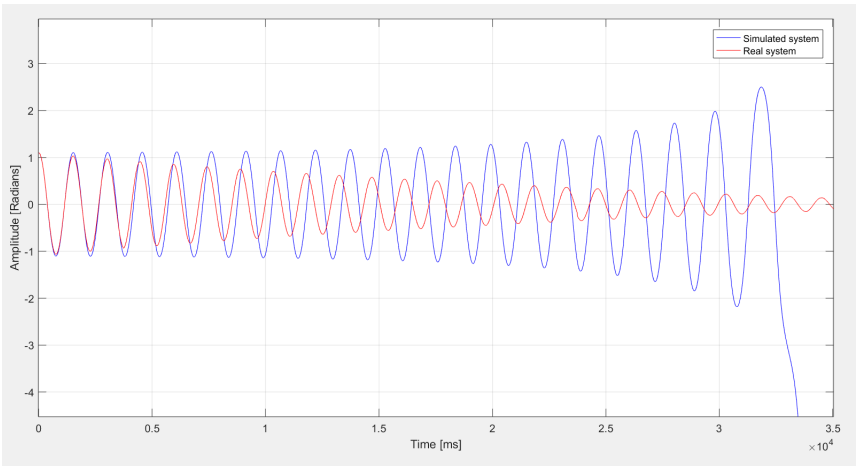


Figure 7.6: The model becomes unstable when the initial conditions of the pendulum are large

There is also variance in how one filters the data for a, b and c. Simply removing all values outside a given threshold or keeping all data will give different results. However, the trend seen in figure 7.4 shows that a will be the opposite sign of b and c in nearly every case, making the argument for filtering data redundant. Needless to say, these results do not make a satisfactory model for describing the swing of the robot's leg.

7.2.2 Result with adjusted method

The method used thus far did not yield satisfactory results, so the focus shifted back to the original differential equation (7.4). Squire's paper shows that he made several different approximations when solving his equation, one of the most important ones being his approximation of the sgn-function [61]. The same method was used again, but this time with the same simplifications used by Squire. The sgn-function is equated as:

$$sgn(\dot{\theta}) = -\frac{2k}{\pi^2}cos(\omega t) - \frac{4}{\pi}sin(\omega t) \tag{7.20}$$

When the trial function is used. The resulting solution to equation 7.4 in terms of sine and cosine is as follows:

$$\begin{aligned} &cos(\omega t) * (-A\omega^2(1 - \frac{kt}{T}) - \omega_0^2\frac{A^3}{8}(1 - \frac{3kt}{T}) + \omega_0^2 A(1 - \frac{kt}{T})) \\ &+sin(wt) * (\frac{2A\omega k}{T} - a\frac{8A^2\omega^2}{3\pi} - bA\omega + c\frac{4}{\pi} \\ &+cos(3wt) * (-\frac{A^3}{24}(1 - \frac{3kt}{T})) = R \end{aligned} \tag{7.21}$$

This leads to new equations for the constants ω_0^2, a, b and c.

$$\omega_0^2 = \frac{\omega^2}{1 - \frac{A^2}{8}\frac{1-3k}{1-k}} \tag{7.22}$$

$$\Delta A = a\frac{8A^2}{3} + b\frac{TA}{2} + c\frac{T^2}{\pi^2} \tag{7.23}$$

And in turn new values for each constant.

$$\begin{aligned} \omega_0^2 &= 19.9269 \\ a &= 0.0086 \\ b &= 0.0496 \\ c &= 0.0984. \end{aligned}$$

This leads to the full model of the pendulum:

$$\ddot{\theta} + 0.0086|\dot{\theta}|\dot{\theta} + 0.0496\dot{\theta} + 0.0984sgn(\dot{\theta}) + 19.9269sin(\theta) = 0 \tag{7.24}$$

7.2.2.1 Simulation

This time there was no filtering of the data for a, b and c. The variance was already limited to the average ± 5. Seeing as this was the threshold chosen originally, the same method was used in this case as well. Figure 7.7 shows the new simulated model in comparison to four different sets of data.

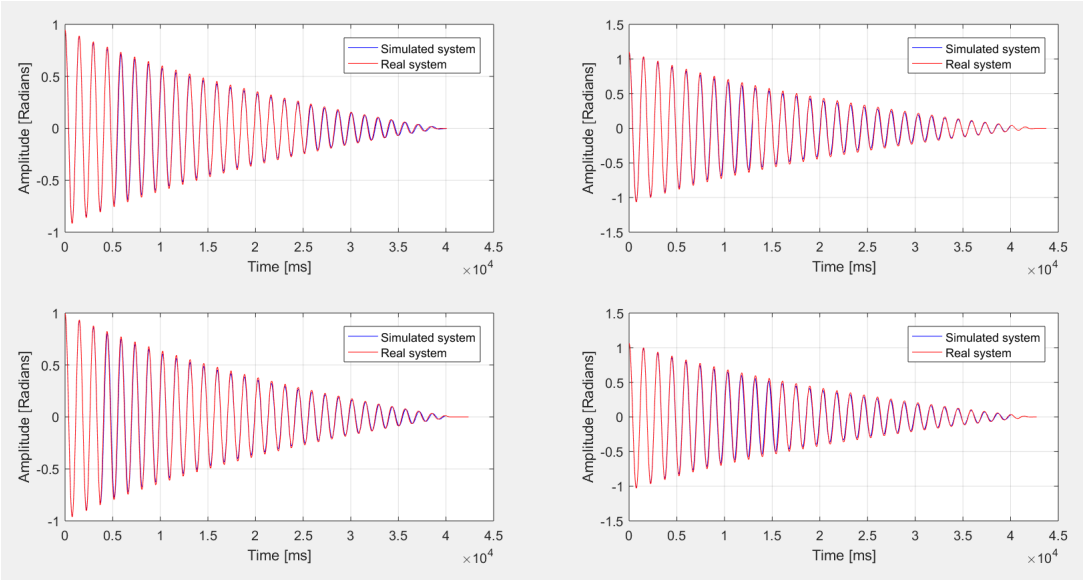


Figure 7.7: Comparison between the new simulated system and the real system

The new values are a vast improvement on the model. The two plots overlap for a large majority of the swing in all four cases. The shape of the dampening curve is very close to exact, with a slightly more exponential form. Figure shows the absolute deviation of the simulation compared to measured data.

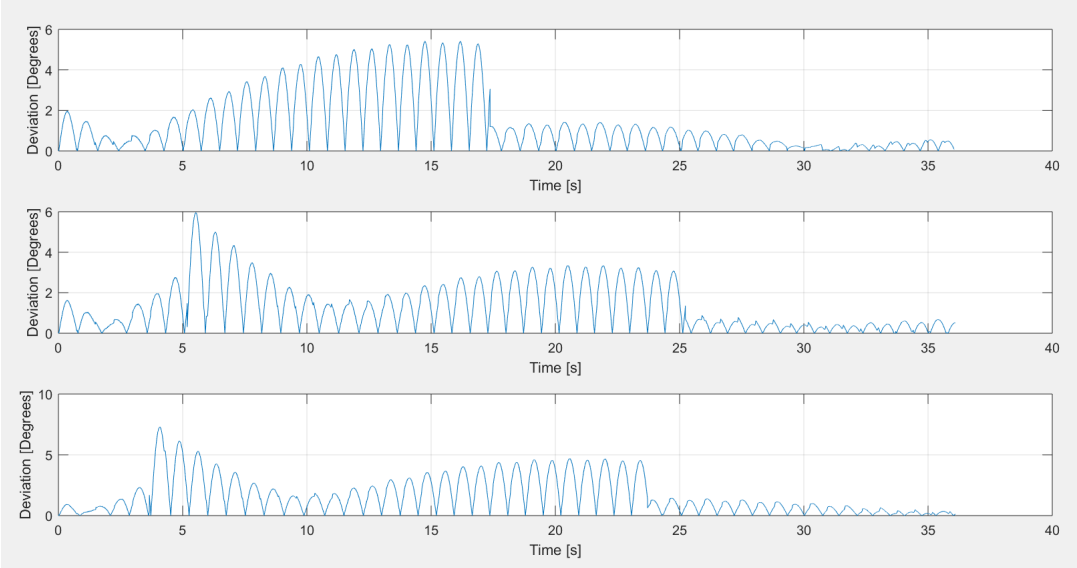


Figure 7.8: Absolute deviation for original model and optimized model

The difference between the simulation and data peaks at more than 7°for the given sample data.

The shape of the curve is noticeably different between each plot.

7.2.2.2 Physical measurements

The respective masses and centers of gravity for each leg was found through blueprints made in the program AutoCAD. A detailed digital model of the entire robot had been made, including the weight of all parts used. The model was deconstructed to get a measurement for each leg. The torso was ignored as it has been modified after the model was made, making the information unreliable. The masses and centers of gravity are as follows:

Outer leg	Inner leg
$m = 2019.01gram$	$m = 1912.67gram$
$a = 182mm$	$a = 228.5mm$

Where a is the distance from the rotational axis to the center of gravity and m is the total mass of the body.

The center of gravity is located an approximately equal distance from each side of the leg, meaning it is centered in the middle of the leg. The digital model shows the robot before any wiring or other components were added during the work done in this project. All mass added to the legs of the robot can be considered negligible, as most equipment with any considerable mass was mounted on the torso. However, because the digital model and real model are no longer the same, these measurements should not be considered highly accurate. It is recommended that they are used as broad approximations, not exact parameters.

8 Power supply

The motors, electronics and servos will have different voltage operating levels with varying currents. This requires reliable solutions in order to avoid voltage drops and damage to equipment. The different voltages are distributed through a crafted distribution circuit(appendix C.6).

8.0.1 Consumption overview

Table 10 lists data found (and calculated) from the components datasheets.

Component	**Power consumption**	**Operating voltage**
Maxon Motor 14887	150W	48V
ESCON 50/5 (409510)	$\leq$ 15W	50V
Futaba S9254	n/a	4.8V
SCANCON 2RMHF	n/a	5.0V
BeagleBone Black	$\leq$ 5W	5V $\pm .25V$

Table 10: Component power requirements

8.0.2 Actuator/Controller

The actuators are the greatest consumers, responsible for balance and counteracting friction loss when the robot is walking. The actuators will be controlled by motor controllers. The actuators and controllers are made by the same manufacturer. Recommended supply values for the specific motor can be found in the controllers datasheet[63]

Characteristics	**Values**	Notation
Nominal voltage	48V	U_N
No load speed	7590rpm	n_O
Nominal speed	7000rpm	n
Nominal torque	187mNm	M
Speed/torque gradient	3.04rpm/mNm	$\frac{\Delta n}{\Delta M}$

Table 11: Maxon DC motor characteristics

Equation 8.1 from controller manual with the given characteristics solves for the required operating voltage for the motor controller.

$$V_{CC} \geq \frac{U_N}{n_O} \cdot (n + \frac{\Delta n}{\Delta M} \cdot M) \cdot \frac{1}{0.98} + 1V \tag{8.1}$$

$$\begin{gathered} V_{CC} \geq \frac{48V}{7590rpm} \cdot (7000rpm + 3.04rpm/nNm \cdot 187nNm) \cdot \frac{1}{0.98} + 1V \\ V_{CC} \gtrsim 50V \end{gathered} \tag{8.2}$$

The result is within the controllers nominal voltage threshold and compensates for its maximum voltage drop. Lower limit combined with the datasheet's maximum rating gives the full operating

range.

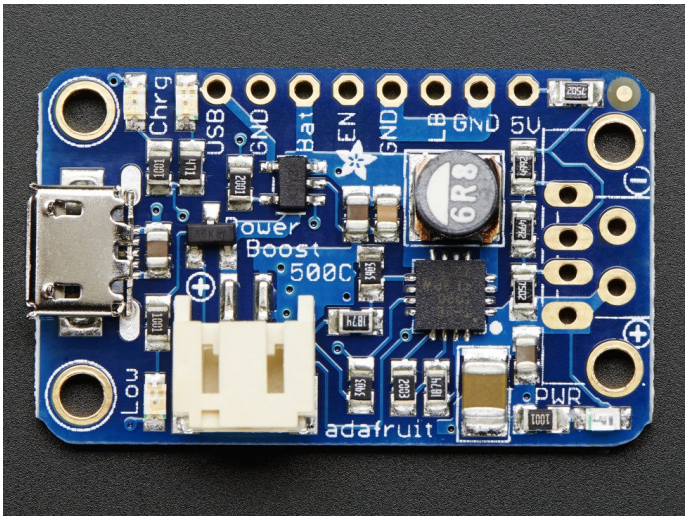
$$56V \geq V_{CC} \gtrsim 50V \tag{8.3}$$

8.0.3 Servos

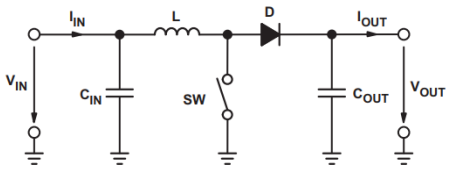
The Futaba S9254 servo motor has a strict operating voltage at 4.8V and pulls a large, varying current (See appendix D). This requires accurate line regulation. Configuring one of the output source terminals to handle the regulation gives the best results because of its response time and accuracy.

8.0.4 BeagleBone Black and sensors

The BeagleBone Black requires a minimum of 1.2A, but 2.0A is recommended when powering external devices. With output terminals configured to 50V and 4.8V, the best option is to convert the output on the 4.8-terminal. The Adafruit PowerBoost 500 solves this issue with a step-up converter (figure 8.1).



(a)



(b)

Figure 8.1: Circuit Board[13]/PowerBoost Power Stage [14]

Figure 8.1b shows the step-up chopper stage of the chip. This stage uses a coil to prevent current drop and a PWM-controlled transistor to maintain the coil current. Capacitors smoothens the noise, resulting in a higher output voltage. The maximum output current is directly proportional to the input voltage (Fig. AdaInOut).

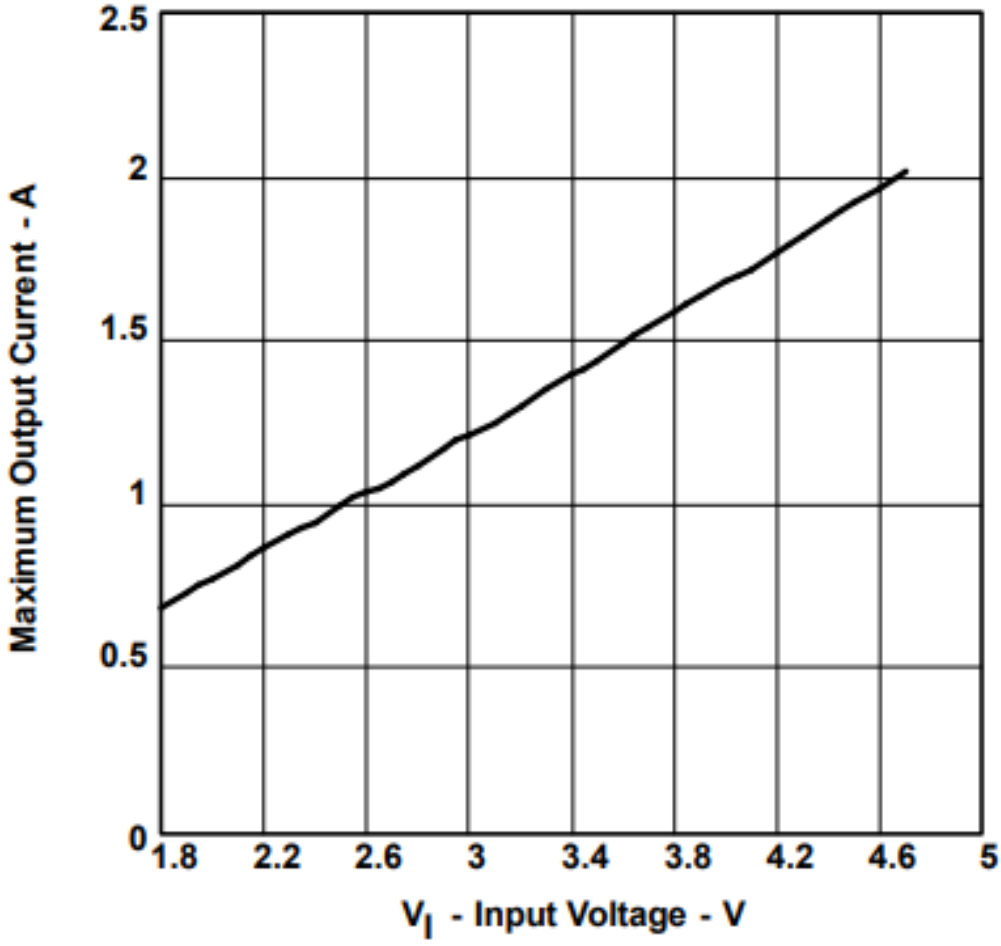


Figure 8.2: Maximum current Adafruit PowerBoost 500 [15]

With 4.8V consistent input, the boost converter supports up to approximately 2A.

8.0.5 Output source

Using the AimTTI QPX600D power supply, it is possible to supply two reliable voltage references. The supply responds quickly to load changes and has an accuracy which satisfies the components needs.

Type	**Minimum**	**Maximum**
V_o	0V	60V/80V
I_o	0.01A	50A
Regulation	**Accuracy**	
Load	$< 0.01\% \pm 5mV$	
Line (10% change)	$< 0.01\% \pm 5mV$	
Transient response (5-95% load change)	$< 2ms$ to $\pm 100mV$	

Table 12: Resistance measurement

8.1 Battery supply

A detachable battery voltage supply should be considered before calculating centers of mass on the robot. Mounting a detachable supply on the robot's torso adds new possibilities to testing environments. This subsection presents a possible solution, but is not implemented on the robot.

8.1.1 Choice of battery

LiFePO4, LiPo and Li-Ion were considered for the battery pack. Construction gives LiPo and LiFePO4 protection at higher discharge rates and offers a greater life cycle. However, Lithium-ion batteries seems to be easier to obtain at a reasonable price in Norway.

With respect to costs, Sony US18650VTC4-batteries have a decent capacity, supporting high discharge rates. Circuit design and calculations are done according to its technical data[16]

Nominal Capacity (0.2C discharge)	**2100mAh 7.77Wh**	**average capacity 3.70V (average discharge voltage)**
Rated Capacity (0.2C discharge)	**2000mAh 7.40Wh**	**minimum capacity**
Capacity at 1C	**2002mAh 7.30Wh**	**average capacity**
Capacity at 10A	**2035mAh 7.01Wh**	**average capacity**
Nominal Voltage	**3.7V**	
Internal Impedance	**12mΩ Typ.**	**measured by AC1kHz**
Cycle Performance	**60% Min. of Initial capacity at 500 cycles**	**10A discharge**

Figure 8.3: Sony US18650VTC4 performance data

8.2 Battery management system

Full protection from dangerous discharge/charge rates can be implemented in a custom battery management system (BMS). This circuit monitors voltage on all battery cells and will cut battery connection if a cell voltage is too high/low.

Maximum/cutoff voltage is given in the battery's discharge characteristics (Fig.8.4)

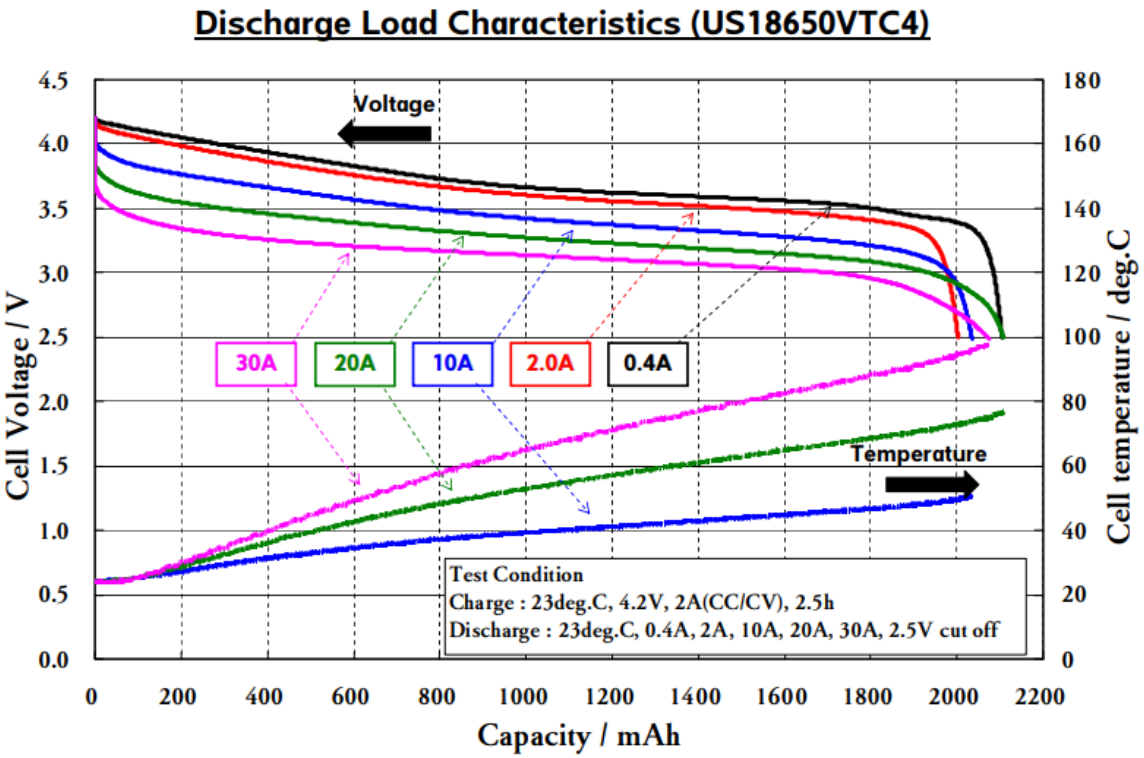


Figure 8.4: Discharge characteristics[16]

The characteristics of each cell will vary and the voltage drop of all cells depend on many variables. To avoid explosion hazards the battery needs to be monitored continuously.

8.2.1 Circuit design

This circuit measures all cell voltages, but scales them to a resolution within the measurement range of the micro controller pins. A MOSFET is used to control output with pulse width modulation.

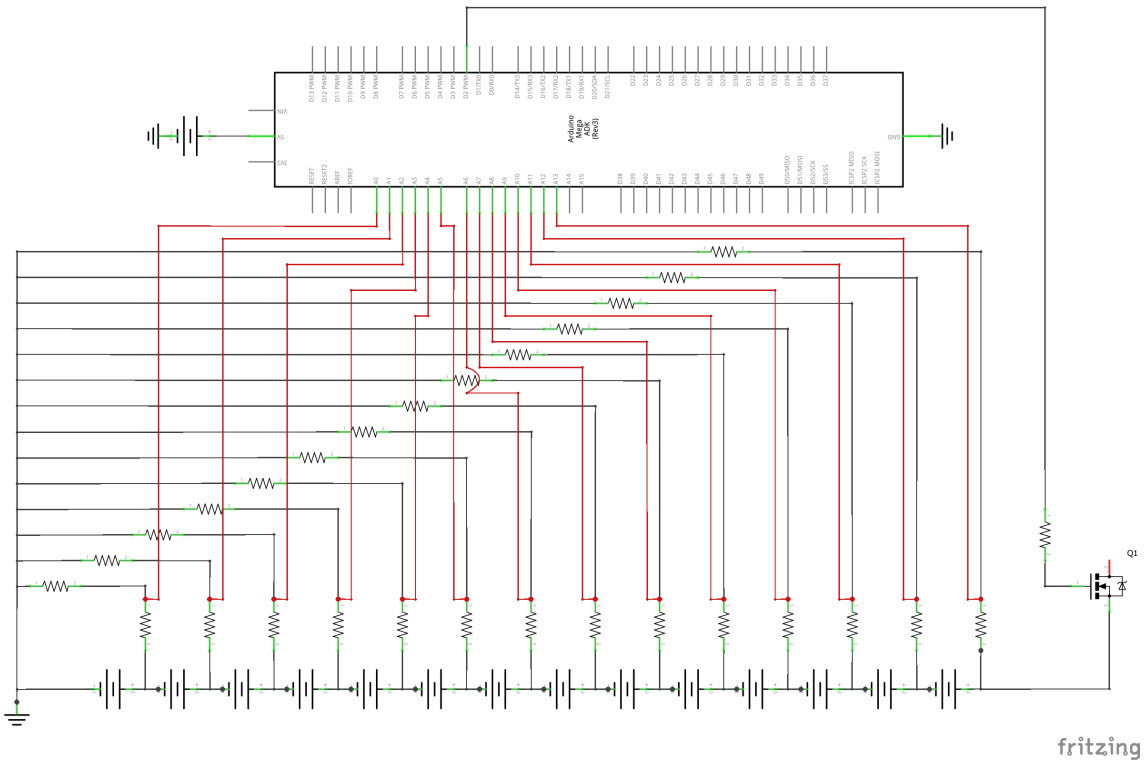


Figure 8.5: Cell monitor circuit

Zener diodes could be mounted between each battery cell to limit max voltage. A prototype has been made for the circuit.

8.3 Voltage dividers

Voltage dividers will be used to keep all cell voltage measurements within the Arduino's input threshold. These measurements have to cover the full operating range of each cell with the best resolution possible. Availability determines the size of used resistors, limited to 3 resistors for each divider.

One resistor value will be preset and maximum cell voltage is given for each cell. This leaves the other two resistor values as answers to the amplification criteria:

$$\frac{V_o}{V_i} = \frac{V_{Arduino,max}}{V_{\text{Cell}}} = \frac{R_2 + R_3}{R_1 + R_2 + R_3} \tag{8.4}$$

With exception of cell 1 & 2, R_1 will have optimal value $820M\Omega$ (Due to availablity), which gives the following ideal amplification and resistor values:

	Ideal values		
Cell	$V_{\text{Cell,max}}$[V]	$R_2 + R_3[k\Omega]$	$\frac{V_o}{V_i}$
1	4.3	820	1.00
2	8.6	590	0.581
3	12.9	519	0.388
4	17.2	336	0.291
5	21.5	248	0.233
6	25.8	197	0.194
7	30.1	163	0.166
8	34.4	139	0.145
9	38.7	122	0.129
10	43.0	108	0.116
11	47.3	96.9	0.106
12	51.6	88.0	0.097
13	55.9	80.6	0.089
14	60.2	74.3	0.083

Table 13: Monitor circuit resistance values

Note: A prototype circuit is made using Figure 13s values. Zener diodes has been mounted between each cell. For future use, the divider's operation ranges could be found by connecting the power supply to the cell inputs. By varying voltage, the input necessary to produce 5V could be found for each divider.

8.3.1 Capacity, weight and cost

While serial connecting the batteries increases voltage, a parallel connection will increase capacity. Cells with different voltage potential causes a current flow, which could damage them. In order to balance the cells, a resistor could be placed in between to lower the current. When the cell potentials are equal the resistor could be removed.

The US18650VTC4 technical data[16] estimates an average weight of 45g for each battery. With 14 cells required to produce >50V (table 14).

Cells	**Average capacity [mA]**	**Average weight[kg]**	**Cost [NOK]**
14	2100	0.63	630,-
28	4200	1.26	805,-
42	6300	1.89	1208,-

Table 14: Battery weight and capacity

9 Discussion

9.1 Programming

Code development was time-consuming due to lack of programming-experience at the start of the project. Rewriting and compiling code gradually became an easier task. Porting library code and routing signal into/out of correct pins gave a better understanding of how the low-level operations work, both in Linux and C/C++. The resulting code has a good structure and can easily be changed because of its modularity.

Even though the main program is fully able to communicate with the sensors and actuator controller, the classes do not feature fully fledged exception handling. This is not essential for a program of this size, but does however make the code easier to read more suited for further scaled development.

The current solution is forced to run on a static sample time of 25ms, but the active runtime during a loop without IMUs was found to be 2.2ms. The IMUs time between reads should be lowered, and threads should be implemented to run while waiting for new readings.

Python could have been a better alternative for a fast approach, since there are lots of Python-resources able to communicate with BeagleBones hardware. Still, this could complicate communication with the PRU-ICSS, which requires compiled code.

9.2 BeagleBone Black

The implementation and testing done using BeagleBone was demanding on time. Still, the usage (and understanding) of Linux, C/C++ and assembly gives the user advantages compared to an average microcontroller in terms of clock speed, connectivity and being able to utilize an OS.

Alternatively, if an Arduino had been used throughout the project, a lot more time could have been used to perform measurements and to identify the robot. Much of the time spent on the BeagleBone Black has been used to learn about tools and techniques which are already well-documented. The educational gain from this is great, though not useful for any future users of the robot if they are already familiar with the operating system (And the programming languages).

On the other hand, deep diving into new disciplines only to solve specific problems could result in bad/unusable solutions. This would leave future students with less general information to work with.

9.3 Programmable Realtime Units

The PRUs did not end up with any practical purposes in this project. The information about its subsystem given in this thesis and its relation to realtime capabilities, is hopefully useful for future work. Due to the main processors speed, they might not even be necessary to use.

Still, the experience of searching through reference manuals to find the boards capabilities was very educational.

9.4 IMU

The problem statement implies the IMU was to be placed on the torso. It was found necessary to obtain data justifying the placement of the IMU, hence the simulation of the IMU, which also lead to the experiments. Additionally, the simulation gave an indication of how to configure the sensor fusion to get the best results. A weakness of the simulation is the complexity of the process. Faults in the simulation setup might not be obvious, giving plausible yet wrong results. Also, a key flaw of the simulation is as mentioned not taking the impact on the ground into account. Another fault is the wrong physical parameters. However, the rest of the provisional simulation is considered adequate, resulting in seemingly correct results. The simulation and the experiments resulted in a change of preferred positioning of the IMU, as well as a sturdy reasoned sensor fusion configuration. Due to the issues regarding cable length to the IMU, it is temporarily placed on the torso, but as discussed in section 10.1.1 this problem should be solvable. The IMU-part of the problem statement may be considered partly completed, except for the slack in the actuator transmission, which has not been prioritized. As seen from findings throughout the project, there are improvements to be made to the momentary implementation, which should be considered for future work.

9.5 Model identification

The final model of the dynamics of the inner leg was surprisingly accurate based on our initial expectations for the method used. When Squire wrote his paper on pendulum dampening[61], the system used was not as heavily impacted by friction as this one. The example figures he provided showed that his pendulum would swing for more than twice the duration of our system. Many of his approximations are based on a, b and c being “small”, which is why we were hesitant. The plot of the result shows a great deal of overlap both in dampening of amplitude and frequency of oscillations. However, the results were far from ideal. As was shown in figure 7.8, the deviation between the theoretical model and the real system peaked at more than 7°. Based on the plots comparing the simulated and real systems, one can assume that the large deviations come from an inaccurate estimation of angular frequency rather than dampening of amplitude. Figure 9.1 shows a plot of the period of each oscillation for a given set of data.

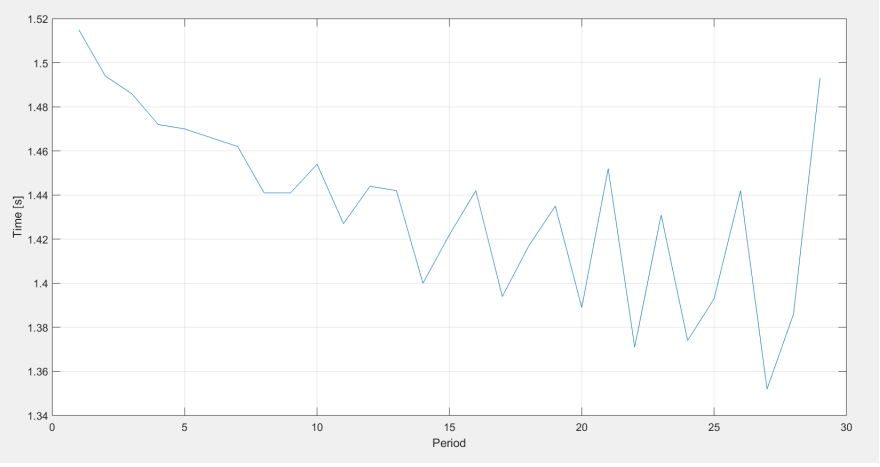


Figure 9.1: Duration of each period of a random set of data

Contrary to what one would assume, the period does not decrease for every oscillation, but spikes up with larger variance as time goes on. The plot in figure 7.7 also shows that the simulation overlaps for different sections of time while missing the target data in between these sections. This is presumed to be caused by various wires, cables and other small obstructions hindering the leg. Such factors are not only likely to change with future work on the model, but they are also extremely hard to model/predict and will therefore not be accounted for in this paper. One can also observe that there are big differences between the three graphs, supporting the idea that these irregularities are not universal for all data gathered. It is also possible that the governing differential equation does not properly describe the dynamics of the system. For example, while experimenting with a rudimentary optimization algorithm, it was noted that the factors a and b seem to not be mutually exclusive. Increasing one and decreasing the other seems to have very little effect on the end result. Based on this, one might not include both terms when making the differential equation. The optimization algorithm mentioned did not improve on the model, but this was to be expected as there was not much time to properly explore this approach. Given more time, such an algorithm would be a priority in improving on the results we have thus far.

A very important thing to note is that all the plots and comparisons shown in this paper are from data used in designing the theoretical model. Due to the loss of equipment, new data could not be made for the sake of testing. This means that the model has not been properly tested against an independent set of data.

10 Conclusion

Instrumentation of the robot was the main objective of this project. This has been completed with some shortcomings in certain areas. A fully functional configuration has been made, actuation and instrumentation controlled with a SBC, BeagleBone Black. Each part of the robot is supplied with correct voltage, using both channels on the power supply and a DC/DC booster to get the 5V to the BeagleBone. See appendix C.1 for a overview diagram of the complete wiring. The BeagleBone is able to calculate the angle of the torso relative to the world frame from the IMU readings, using a scientifically justified sensor fusion configuration with a proper calibration, allowing for the proper resonance frequency needed. The placement of the IMU is constrained to the torso, even tough findings in this project suggest the ideal position is on the legs, mounted 0.5m above the legtip. Also, it seems favorable to operate with two IMUs, one on each leg. The PWM limits of the servo has been found and are controlled with the BeagleBone. Leg actuation, measurement feedback and logging are easy and functional routines on the BeagleBone Black at the unsatisfactory sample time of $20ms$. A working method for establishing a dynamic model of the swing of the robot's legs has been found. The resulting model is less accurate than one would prefer, with a deviance of more than 7%. The results given are also not properly tested against the real system as there was no access to independent data.

10.1 Future work

This section describes further work to be made on the robot, both urgent problems requiring immediate solutions and recommendations for alternative solutions or improvements on the work that was done.

10.1.1 SPI Improvement

As mentioned in section 4.2.1.3, there are some issues regarding cable length with the current SPI configuration and the Beagle Bone Black. There are several ways to fix this problem, two possible solutions are using a micro controller slave or to modify the transmission protocol used.

An arduino slave to calculate the IMU angles and send these to the beagle bone could be a functional solution. This gives 5V to drive the SPI compared to the 3.3V from the beaglebone which did not work. Also, the Arduino could be placed closer to the IMU, reducing the distance of SPI communication. An Arduino Nano or equivalent board should be more than sufficient for running the IMU-calculation and sending this to the Beagle Bone Black. With some modification to the IMU-mounting, this setup is small enough to fit together with the IMU. Then the standard UART Serial communication from the Arduino might be able to transmit the distance required. This needs some thourough testing before any conclusive decisions are made.

Drivers to transform the SPI signals to a differential signal protocol using higher voltages is another alternative. These are more robust to noise and often able to drive the signals over longer distances. The restraints of speed is, however, something that needs to be taken into account when considering this solution. RS-422 is a standard that could be taken into consideration, able to operate with speeds up to 10 Mbit/s, using differential 4-wire communication operating with $\pm+-6$V.

Along with sampletime, many parameters can be tweaked on the IMU circuit. These parameters are described in the SparkFun-parts of the ported IMU-libary, but also available from its datasheet[51]

10.1.2 Model identification

The primary task for future work would be to make a model for the outer leg of the robot. All methods used in this paper are applicable to the other leg as well. It is also highly recommended that the experiment described in this paper is redone with far more data. A larger pool of data could give different results and it would also allow for proper testing of the finished product. One can also put effort into a sophisticated optimization algorithm for finding more accurate results. The methods used so far have mainly been based on approximations and simplifications of the differential equation. With an optimization algorithm, one can tune the coefficients of ω_0^2, a, b and c without making changes to the governing equation.

It is also recommended that new values are found for the physical properties of each of the robots appendices. So far, only mass and distance from rotational axis to center of gravity are given and only for the legs. These values are also not very precise, so any work that requires such parameters to be near exact is not recommended at this time.

10.1.3 BeagleBone

Both boards used during this project (BBB and BBB Wireless) have experienced crashes a short time after boot. One possible explanation for this is a hardware design flaw, which causes issues to the USB host controller during high traffic.[64] However, the source is outdated and the problem may have been fixed in later revisions of the AM335x-hardware.

Regardless of cause, one solution to this issue would be to install a network adapter for wireless communication. (List of adapters guaranteed to work: [65])

10.1.4 Battery Supply

Before attempting to do anything with the Lithium-Ion batteries, it is important to know their limitations and potential hazards. This is described in the UL 1642 safety standard. [66]

Measurement ranges for the voltage dividers should be measured. In order to charge the batteries, the Arduino needs to control current and voltage in the two charging phases (Figure 10.1).

A rigid mounting bracket needs to be designed to hold circuits and batteries. Its design depends on the number of batteries and how it should be mounted on robots torso. We recommend using Fusion360 and Cura to create 3d-models for printing.

10.1.5 Realtime implementation

The next step to reduce interrupt response time is to modify the kernel, preferably by replacing its restricting parts with a Xenomai nucleus. Switching to another Linux distribution (or other OS) should be considered at this point.

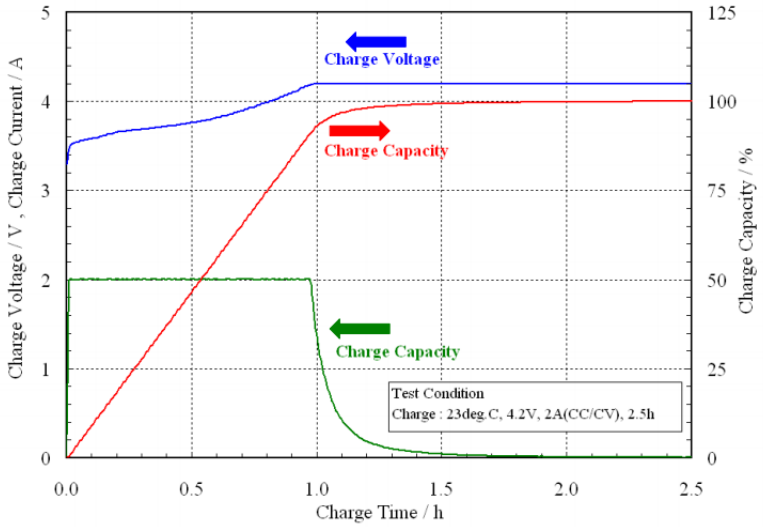


Figure 10.1: US18650VTC4 Charge characteristics

The IMUs can be configured to send interrupt signals on their own wires, allowing them to be received on gpio-pins. Alternatively, the modules used for angle measurements (MCSPI and eQEP) can be configured to send interrupts. [6, p.4914, 2527]

10.1.6 PRU_ICSS

In order to transmit measurements in realtime, one of the PRUs could be configured to manage the transmission. The UART-peripheral within the subsystem[6, p.241] is ideal for passing information serially to a computer or dSPACE. Direct connection to a computer would require a serial to USB converter cable.

10.1.7 Servo

As mentioned in section 6.4.2, a servo replacement is required. The servos currently used are no longer for sale, therefore a used servo must be purchased or a different model must be bought to replace it. If a different model is ordered, the physical size of the servo needs to be the same in order for it to fit in the current mounting system. Also, the actuating speed of the servo should be as close to the original ones as possible. The Futuba S2954 has a rotation speed of 0.06s/60°, which is quite fast. If the new one is any slower, this may become an impediment if the servo is not able to fully extract its joint before the leg impacts the ground. However, this should not be a problem within reasonable limits of the rotation speed or with a slow enough gait pattern.

Mainly based on simplicity, we suggest buying a used servo of the same model, Futuba S2954, to be the preferable option. After a new servo has been installed, the limit of the driving signal needs be found. This should be rather painless, since one only needs to repeat the experiment explained in appendix D. The files, including the Arduino code from this experiment included, can be found at [24, ServoExperiment].

10.1.8 dSPACE

A ransomware attack encrypted the previous dSPACE-installment, forcing a complete wipe of the computer hard drive. A re-installation has been performed by the IT-crew from the Engineering Cybernetics department, but the setup still remains. To complete the setup, some installation files are needed. Retrieval of mentioned files from dSPACE have not been successful yet, since we are awaiting response from dSPACE. Upon the arrival of said files, the setup should most likely be a rather effortless matter. Even though a large part of this project focused on replacing dSPACE with a new platform, getting it back up and running should be useful for several reasons. It could service as a real time platform for user interface during development, e.g communication to the Beagle Bone Black as mentioned in section 4.2.1.1.

Reference

[1] picture of a beaglebone black from sparkfun (figure), . URL https://www.sparkfun.com/products/14162.

[2] Gcc and make compiling, linking and building c/c++ applications (figure), March 2018 (accessed 03-04-2019). URL https://www.ntu.edu.sg/home/ehchua/programming/cpp/gcc_make.html.

[3] Pru software development flow (figure). URL http://downloads.ti.com/docs/esd/SPRUHV7C/software-development-tools-overview-stdz0596323.html.

[4] Demystifying the linux kernel (figure), May 2015 (accessed 11-05-2019). URL https://blog.digilentinc.com/demystifiying-the-linux-kernel/.

[5] Virtual memory (figure), April 2019 (accessed 14-05-2019). URL https://en.wikipedia.org/wiki/Virtual_memory.

[6] Technical reference manual - am335x and amic110 sitara^{TM} processors (figure), March 2017 (accessed 27-03-2019). URL https://www.ti.com/lit/ug/spruh73p/spruh73p.pdf.

[7] Enhancing real-time capabilities with the pru (figure), October 2014 (accessed 18-04-2019). URL https://elinux.org/images/1/1c/Birkett--enhancing_rt_capabilities_with_the_pru.pdf.

[8] Amarpreet Singh Ugal. Hard real time linux using xenomai on multi-core processors (figure). October 2009 (accessed 11-04-2019). URL https://www.intel.cn/content/dam/www/public/us/en/documents/white-papers/multicore-real-time-linux-xenomai-paper.pdf.

[9] LSM9DS1 hook up guide (figure), unknown date of publication (accessed 17-02-2019). URL https://learn.sparkfun.com/tutorials/lsm9ds1-breakout-hookup-guide/all.

[10] Serial peripheral interface (figure), February 2019 (accessed 22-03-2019). URL https://en.wikipedia.org/wiki/Serial_Peripheral_Interface.

[11] Tms320x2833x, 2823x enhanced quadrature encoder pulse (eqep) module (figure). URL http://www.ti.com/lit/ug/sprug05a/sprug05a.pdf.

[12] BeagleBone Black pwm and timers (figure), . URL https://beagleboard.org/Support/bone101.

[13] Adafruit powerboost 500 + charger overview (figure), unknown date of publication (accessed 15-05-2019). URL https://learn.adafruit.com/adafruit-powerboost-500-plus-charger/overview.

[14] Basic calculation of a boost converter's power stage (figure), January 2014 (accessed 08-04-2019). URL http://www.ti.com/lit/an/slva372c/slva372c.pdf.

[15] Adafruit powerboost 500 - page 6 (figure). URL https://cdn-shop.adafruit.com/datasheets/tps61090.pdf.

[16] Sony Energy Devices Corporation - Rev 0.1 (Figure). Ion Rechargeable Battery Technical Information: US18650VTC4 (figure), July 2012 (accessed 07-03-2019). URL https://www.powerstream.com/p/us18650vtc4.pdf.

[17] Quadrature Encoder interface (eQEP) (Table), 2017, (Accessed April 2019). URL https://adafruit-beaglebone-io-python.readthedocs.io/en/latest/Encoder.html.

[18] AM335x and AMIC110 Sitara^{TM} Processors Technical Reference Manual (Table), October 2011 (accessed 14-04-2019). URL http://www.ti.com/lit/ug/spruh73p/spruh73p.pdf. (Figure).

[19] Derek Molloy. Introduction - writing a linux kernel module (listing), April 2015 (accessed 18-05-2019). URL http://derekmolloy.ie/writing-a-linux-kernel-module-part-1-introduction/.

[20] Segmentation fault, April 2019 (accessed 09-05-2019). URL https://en.wikipedia.org/wiki/Segmentation_fault.

[21] C - pointers, Unknown date of publication (accessed 20-05-2019). URL https://www.tutorialspoint.com/cprogramming/c_pointers.htm.

[22] Tad McGeer. *Passive dynamic walking.* April 1990.

[23] Christian F. Sætre. *Stable Gaits for an Underactuated Compass Biped Robot with a Torso.* July 2016.

[24] Bipedal robot prototye github repository, April 2019 (accessed 20-05-2019). URL https://github.com/BiPedRobot/Bipedal-Robot-Prototype.

[25] Embedded System, April 2019, (accessed 05-04-2019). URL https://en.wikipedia.org/wiki/Embedded_system.

[26] Real-time vs. a standard operating system how to choose an rtos, December 2018 (accessed 15-05-2019). URL https://www.microcontrollertips.com/real-time-standard-how-to-choose-rtos/.

[27] Arduino vs rasberry pi, unknown date of publication (accessed 15-05-2019). URL https://www.educba.com/raspberry-pi-3-vs-arduino/.

[28] Microcontroller, April 2019 (accessed 02-05-2019). URL https://en.wikipedia.org/wiki/Microcontroller.

[29] Comparison of Single-Board Computers, 07-05-2019 (accessed 15-05-2019). URL https://en.wikipedia.org/wiki/Comparison_of_single-board_computers.

[30] introduction to beaglebone black, May 2019 (accessed 24-02-2019). URL https://beagleboard.org/getting-started.

[31] introduction to connecting a beaglebone black to internet using usb, date of publicity unknown (accessed 24-02-2019). URL https://www.digikey.com/en/maker/blogs/how-to-connect-a-beaglebone-black-to-the-internet-using-usb.

[32] Sharing internet using network-over-usb in beaglebone black, December 2014 (accessed 09-03-2019). URL https://elementztechblog.wordpress.com/2014/12/22/sharing-internet-using-network-over-usb-in-beaglebone-black.

[33] Download ccs, April 2019 (accessed 16-05-2019). URL http://processors.wiki.ti.com/index.php/Download_CCS.

[34] User's guide - pru optimizing c/c++ compiler, October 2017 (accessed 22-04-2019). URL http://www.ti.com/lit/ug/spruhv7b/spruhv7b.pdf.

[35] User's guide - pru assembly language tools v2.2, October 2017 (accessed 19-04-2019). URL http://www.ti.com/lit/ug/spruhv6b/spruhv6b.pdf.

[36] Derek Molloy. Exploring beaglebone: Tools and techniques for building with embedded linux. URL http://www.minorw.net/data/ExploringBeagleBone.pdf.

[37] Everything is a file, July 2018 (accessed 06-05-2019). URL https://en.wikipedia.org/wiki/Everything_is_a_file.

[38] Lsm9ds1 breakout examples. URL https://github.com/sparkfun/LSM9DS1_Breakout/tree/master/Libraries/Arduino/examples.

[39] Richard M. Stallman, Roland McGrath, Paul D. Smith. Gnu make, May 2016 (accessed 01-04-2019). URL https://www.gnu.org/software/make/manual/make.pdf.

[40] Beaglebone universal-io, November 2017 (accessed 17-05-2019). URL https://github.com/cdsteinkuehler/beaglebone-universal-io.

[41] Alessandro Rubini Jonathan Corbet and Greg Kroah-Hartman. Linux device drivers, third edition, February 2005 (accessed 05-05-2019).

[42] Pru linux application loader api guide, August 2016 (accessed 14-05-2019). URL http://processors.wiki.ti.com/index.php/PRU_Linux_Application_Loader_API_Guide#prussdrv_start_irqthread.

[43] Pru-icss, February 2019 (accessed 17-04-2019). URL http://processors.wiki.ti.com/index.php/PRU-ICSS.

[44] Am335x pru read latencies, February 2019 (accessed 07-05-2019). URL http://processors.wiki.ti.com/index.php/AM335x_PRU_Read_Latencies.

[45] AM335X PRU-ICSS Reference Guide, SPRUHF8A, June 2013 (accessed 22-03-2019). URL https://elinux.org/images/d/da/Am335xPruReferenceGuide.pdf.

[46] Pru assembly instructions, April 2018 (accessed 17-05-2019). URL http://processors.wiki.ti.com/index.php/PRU_Assembly_Instructions.

[47] Pru interrupt controller, January 2017 (accessed 29-03-2019). URL http://processors.wiki.ti.com/index.php/PRU_Interrupt_Controller.

[48] Community made pru package. URL https://github.com/beagleboard/am335x_pru_package.

[49] Processor sdk for am335x sitara processors - linux and ti-rtos support, unknown date of publication (accessed 25-03-2019). URL http://www.ti.com/tool/PROCESSOR-SDK-AM335X.

[50] Pru-icss remoteproc and rpmsg, April 2018 (accessed 10-05-2019). URL http://processors.wiki.ti.com/index.php/PRU-ICSS_Remoteproc_and_RPMsg.

[51] Lsm9ds1, March 2015 (accessed 10-03-2019). URL https://www.st.com/resource/en/datasheet/DM00103319.pdf.

[52] Three-axis Accelerometer, unknown date of publication (accessed 14-03-2019). URL https://www.mathworks.com/help/aeroblks/threeaxisaccelerometer.html.

[53] Three-Axis Gyroscope, unknown date of publication (accessed 08-03-2019). URL https://se.mathworks.com/help/aeroblks/threeaxisgyroscope.html.

[54] Angle estimation using gyros and accelerometer, January 2018 (accessed 27-03-2019). URL https://www.control.isy.liu.se/student/tsrt21/file/pm_sensor.pdf.

[55] IMU errors and Their Effects, February 2014 (accessed 27-04-2019). URL https://www.novatel.com/assets/Documents/Bulletins/APN064.pdf.

[56] Quadrature encoder too fast for arduino with solution, January 2011 (accessed 01-04-2019). URL http://www.hessmer.org/blog/2011/01/30/quadrature-encoder-too-fast-for-arduino-with-solution/.

[57] Torleif Anstensrud. *Industriell automatisering: Robotteknikk Forelesning 9, Instrumentering.* Norges Teknisk-Naturvitenskapelige Universitet, November 2018.

[58] Escon Servo Controllers, unknown date of publication (accessed 02-05-2019). URL https://www.maxonmotor.com/maxon/view/content/ESCON-Detailsite.

[59] Maxon DC motor datasheet, unknown date of publication (accessed 19-04-2019). URL https://www.maxonmotor.com/maxon/view/product/motor/dcmotor/re/re40/148877.

[60] Timer2 counter, March 2015 (accessed 20-02-2019). URL https://github.com/ElectricRCAircraftGuy/eRCaGuy_TimerCounter/blob/master/eRCaGuy_Timer2_Counter.cpp.

[61] Patrick T. Squire. Pendulum damping. https://aapt.scitation.org/doi/pdf/10.1119/1.14838?class=pdf, November 1986. (Accessed 18-05-2019).

[62] J. R. Acton and P. T. Squire. *Solving Equations with Physical Understanding.* Adam Hilger Ltd, August 1985.

[63] ESCON 50/5 Hardware Reference, September 2013 (accessed 21-03-2019). URL https://www.maxonmotorusa.com/medias/sys_master/8810873815070/409510-ESCON-50-5-Hardware-Reference-En.pdf.

[64] Am335x: Usb babble interrupt, December 2013 (accessed 01-05-2019). URL http://e2e.ti.com/support/processors/f/791/t/308549.

[65] Beaglebone black wifi adapters, May 2019 (accessed 01-05-2019). URL https://www.elinux.org/Beagleboard:BeagleBoneBlack#WIFI_Adapters.

[66] Underwriters Laboratories Inc. Ul1642 - lithium battery safety standard, August 2007 (accessed 19-05-2019).

[67] Gerald Coley. BeagleBone Black System Reference Manual - A5.2, April 2013 (accessed 20-02-2019). URL https://cdn-shop.adafruit.com/datasheets/BBB_SRM.pdf.

[68] SCANCON Type 2RMHF - 1.0, March 2011 (accessed 15-03-2019). URL http://static6.arrow.com/aropdfconversion/f26d6ace0eac9fcc92725ffa1e41167344708855/2rmhf-specifications.pdf.

[69] Serial Peripheral Interface, 17-05-2019 (accessed 08-05-2019). URL https://en.wikipedia.org/wiki/Serial_Peripheral_Interface.

[70] Introduction to SPI, September 2018 (accessed 16-02-2019). URL https://www.analog.com/media/en/analog-dialogue/volume-52/number-3/introduction-to-spi-interface.pdf.

[71] Ion battery product porposal, Unknown date of publication (accessed 04-03-2019). URL https://batterionline.no/batterier/ladbart-batteri/18650-li-ion-3-6-3-7v/sony-us18650vtc4-2100mah-30a.

[72] Eugene I. Butikov. *simulations of oscillatory systems: with Award-Winning Software, Physics of Oscillations*. CRC Press, March 2015.

[73] Phiyu Dhaker, September 2018 (accessed 04-04-2019). URL https://www.analog.com/media/en/analog-dialogue/volume-52/number-3/introduction-to-spi-interface.pdf.

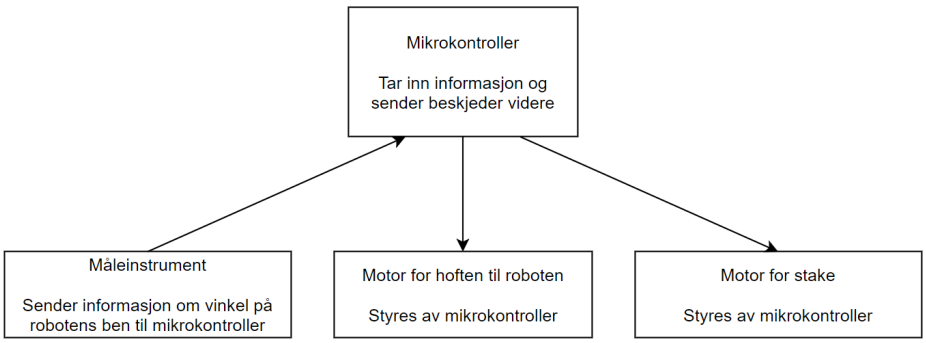
A Appendix - Review Article

Energieffektiv gange for roboter

Mai 2019

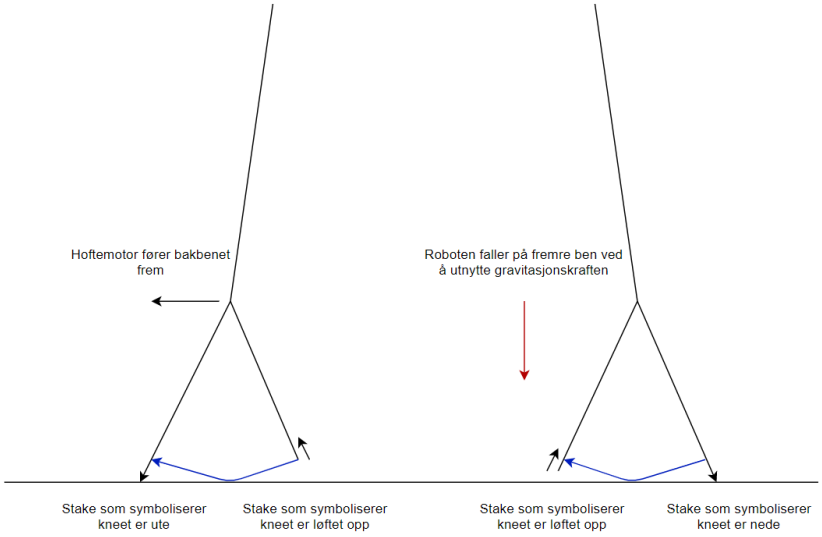
Har du noen gang tenkt over hvordan du går? Eventuelt, tenker du noen gang over hvert steg du tar? Når du går, løfter du opp ett ben med kneet ditt, og bruker hoftemuskulatur til å dytte benet frem. Før dette skjer, vil hjernen sende signaler til muskelfibrene, via nervecellene dine. Signalene muskelfibrene får fra hjernen bestemmer blant annet hvor høyt kneet skal løftes og i hvilket tempo hoftemuskulaturen skal dytte benet frem. Det er mulig å bruke denne analogien til å forklare oppbyggingen til en bipedal robot og hvordan den beveger seg. Vi er en gruppe på fem som studenter ved NTNU, som har hatt i bacheloroppgave å fullføre instrumenteringen på en bipedal robot. I denne artikkelen vil du få en nærmere forklaring på hvordan roboten er bygd opp og hvorfor den er blitt utviklet.

Roboten vi har jobbet med er hovedsakelig oppbygd av en hjerne i form av en mikrokontroller. Derreter har den hoftemuskulatur bygd opp av to motorer og til slutt to knær i form av motorstyrte staker. Stakene er plassert på enden av hvert ben og gjør at bena er like over bakken når de svinger frem.



Figur 1: Diagram som beskriver informasjonsstrøm i roboten

Når du er ute å går, utføres gangebevegelsene i stor grad uten at du bevisst tenker over det. I roboten vi har jobbet med, må hver enkelt del av gangen automatiseres. Strukturen er bygd opp slik at visse betingelser må være på plass, før det første steget kan tas. Når et av robotens ben føres frem av hoftemotoren, har mikrokontrolleren et behov for å vite nøyaktig hvor benet er. Et måleinstrument sender kontinuerlig informasjon til mikrokontrolleren om hvor benet er til en hver tid. Når mikrokontrolleren mottar en bestemt posisjon vil den sende et signal som setter ut staken til benet i bevegelse, og løfte opp staken til benet som ikke er i bevegelse. Da vil roboten lande på benet med staken som er ute, og hoftemotoren vil føre det andre benet frem. Ved å utnytte gravitasjon vil roboten falle på benet som er ført frem. Dette resulterer i en energieffektiv gange som krever lite motorkraft og dermed mindre strøm. Følgende bilde er en illustrasjon av robotens gange.

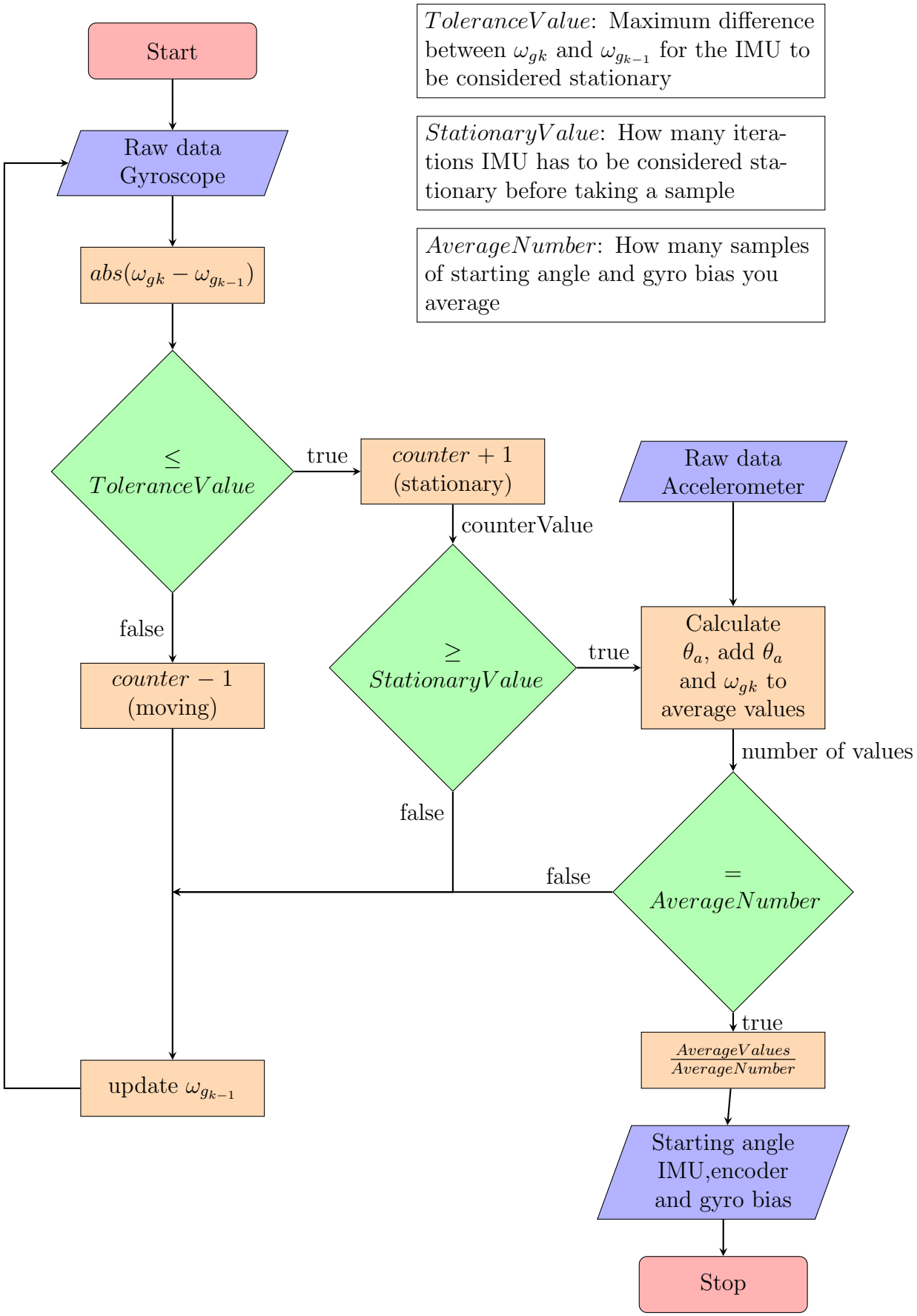


Figur 2: Illustrasjon av robotens gange

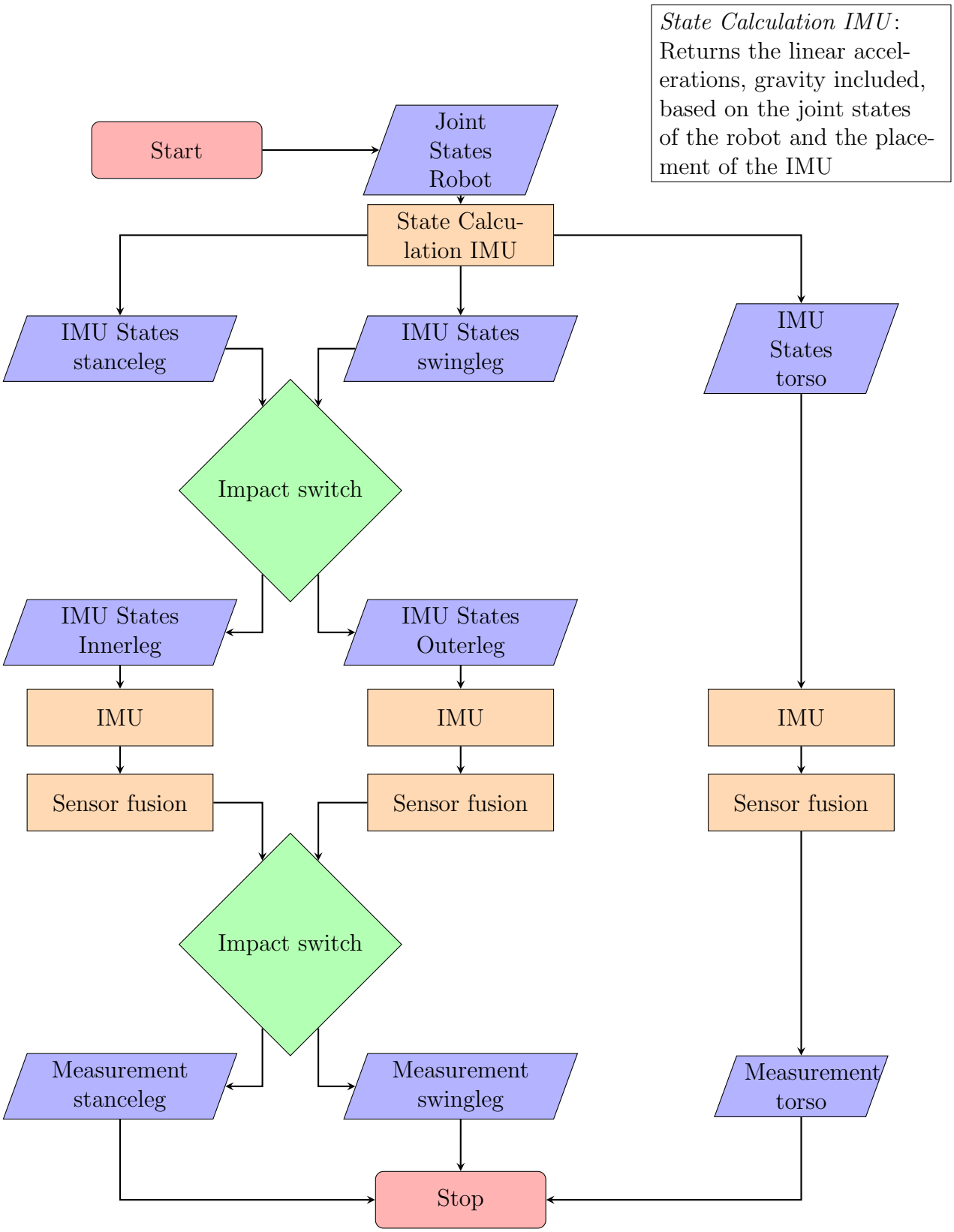
Roboter er i ferd med å bli en større del av samfunnet vårt. Det utvikles blant annet roboter som hjelper barn ned til ett års alderen å lære seg språk og industriroboter som utfører svært nøyaktige sveiseopperasjoner. Foreløpig er selve gangen i roboter noe som krever mye energi. De mest brukte løsningene i dag bruker mye energi på å stabilisere selve roboten med store motorer. Dersom det blir en suksess å konstruere bipedale roboter, vil de kunne bruke mindre energi og bli billigere. Dermed kan roboter få et bredere bruksområde og bli et nytteprodukt i smarthjem og for gamle og syke mennesker som har problemer med å gå.

B Appendix -Flowchart

B.1 Calibration

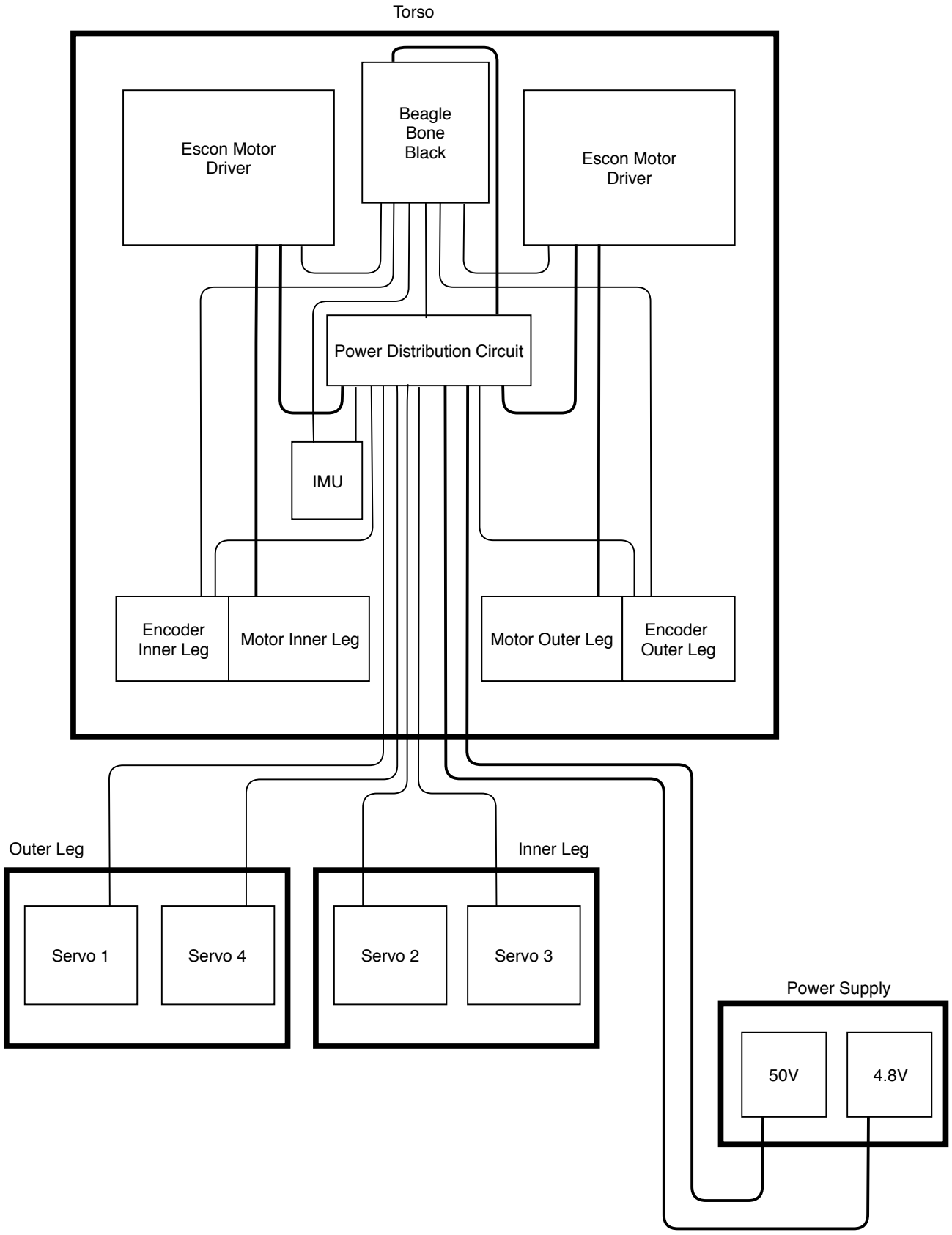


B.2 IMU Simulation



C Appendix - Wiring

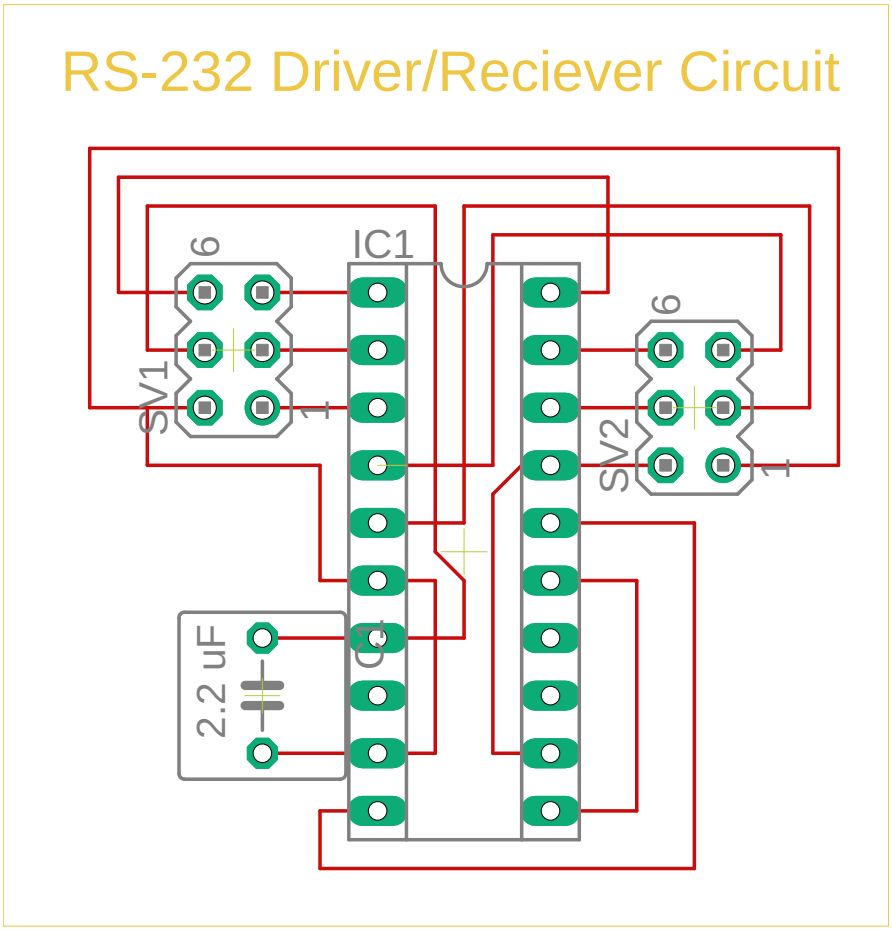
C.1 Overview Diagram

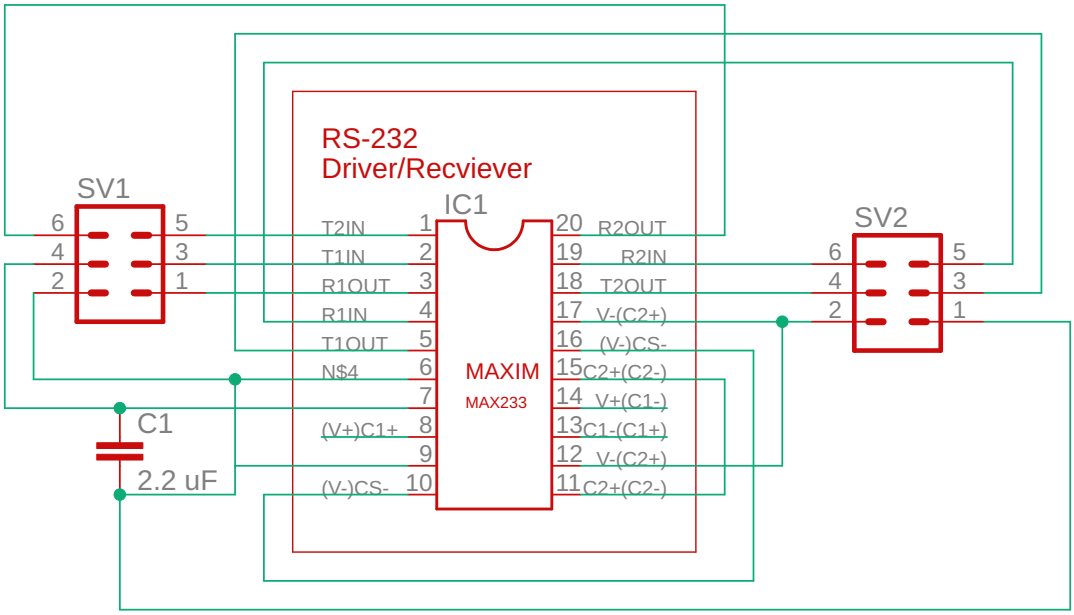


C.2 RS-232 Driver

Connection D-Space RS-232 /Arduino

Arduino		**Cable**		**SV 1**		**MAX 233 CPP**
Pin		Color		Pin		Pin
5V		Orange		5		7
GND		Green		4		6
TX3		Yellow		2		2
RX3		Blue		1		3
		Red		3		1
		Brown		6		20
RS 232 Connector DB-9		**Cable**		**SV 2**		
Pin	Signal	Color		Pin		
1	DCD	Purple		5		18
2	RXD	Blue		2		4
3	TXD	Gray		3		5
4	DTR	White		6		19
5	GND	Green		1		6
6	DSR			4		
7	RTS					
8	CTS					
9	RI					


RS-232 Driver/Reciever Circuit
IC1
SV1
SV2
6
1
6
1
C1
2.2 uF


RS-232
Driver/Recviever
IC1
MAXIM
MAX233
SV1
SV2
C1
2.2 uF
T2IN
T1IN
R1OUT
R1IN
T1OUT
N$4
(V+)C1+
(V-)CS-
R2OUT
R2IN
T2OUT
V-(C2+)
(V-)CS-
C2+(C2-)
V+(C1-)
C1-(C1+)
V-(C2+)
C2+(C2-)

C.3 IMU Levelshifter

IMU-connection w/Level shifter to Arduino

LevelConvertingChip							Cable		Arduino
IMU	Level Shifter		Color		Chip Connector		Color		
VDD	LV+3.3V		Brown		1		Black		3.3V
	HV +5V		Orange/White		2		White		5V
GND	GND	GND	Black		3		Gray		GND
	Low Voltage	High Voltage			4 (not in use)				
CS M	A1	B1	White		5		Purple		49
CS AG	A2	B2	Green		6		Blue		48
SCL	A3	B3	Purple		7		Green		52
SDA	A4	B4	Blue		8		Yellow		51
SDO M/AG			Grey		9		Orange		50

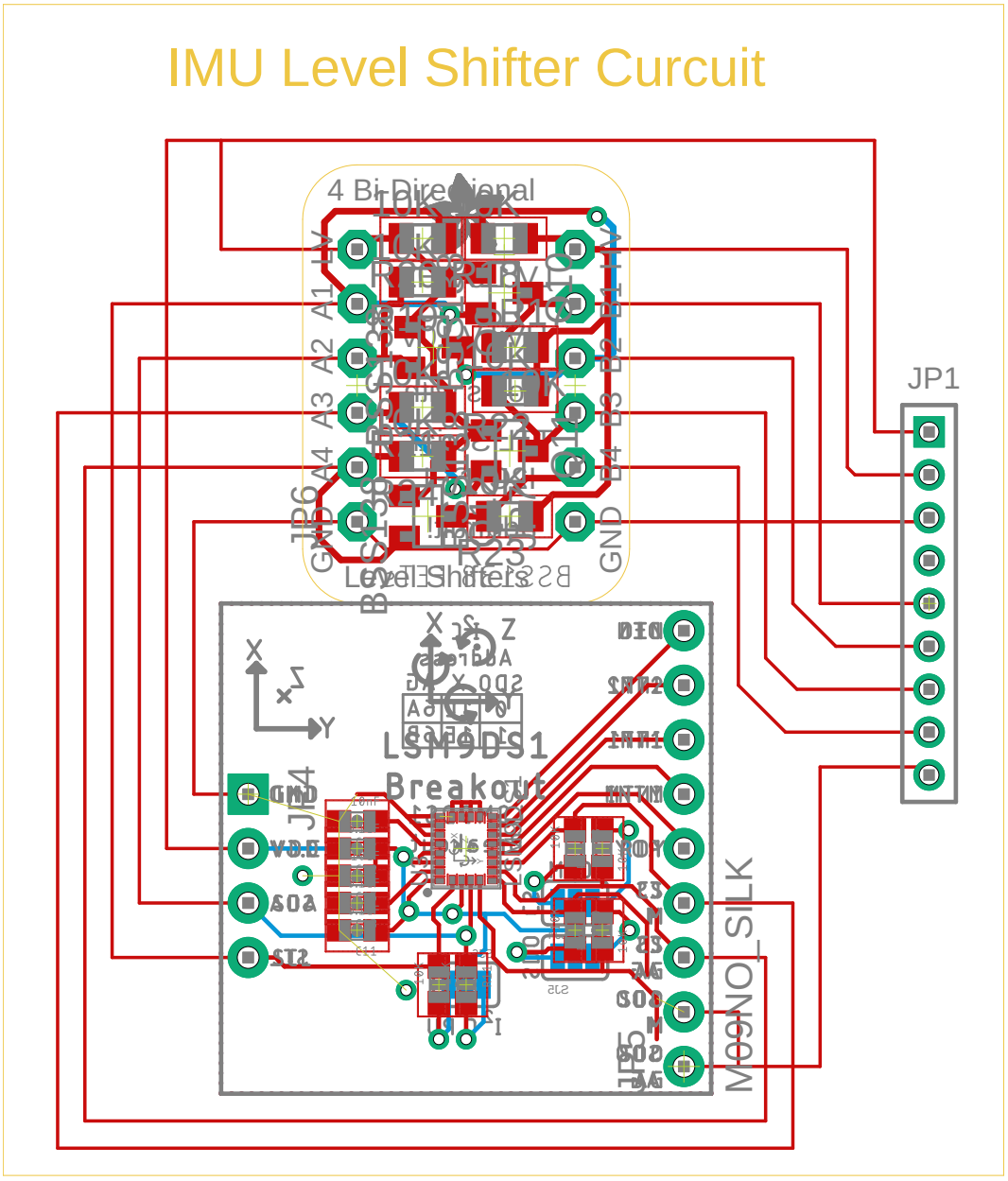
IMU-connection without Level shifter to Beaglebone

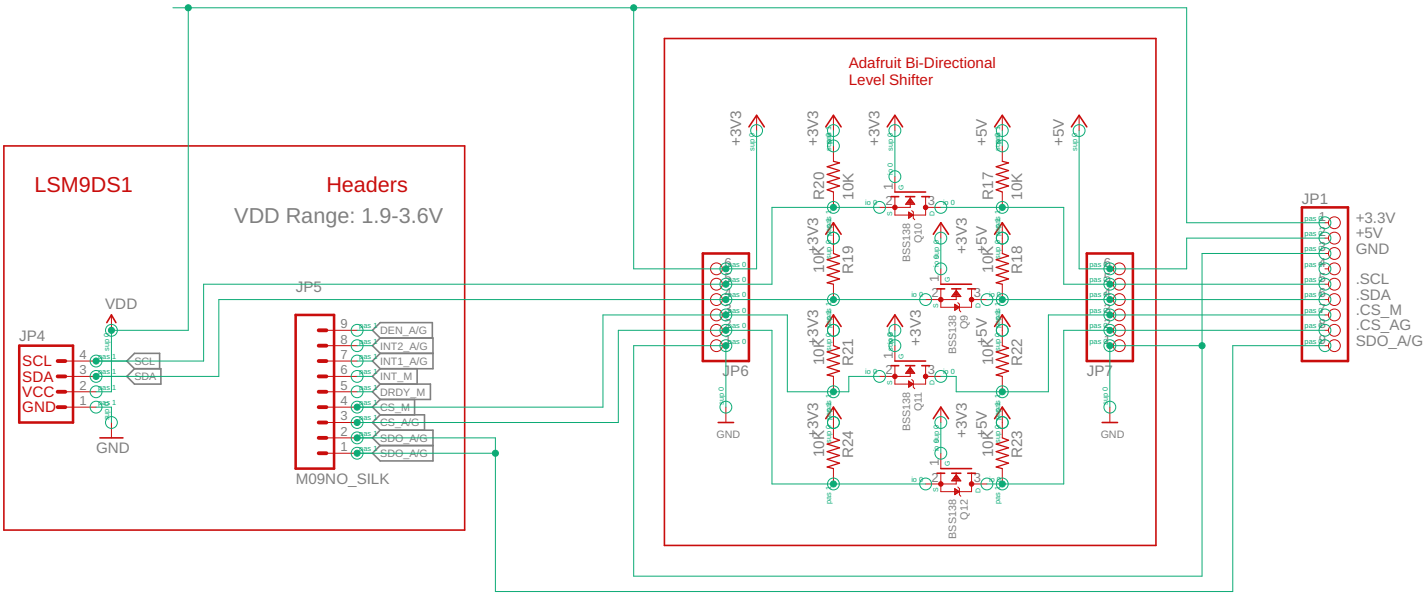
INNER LEG

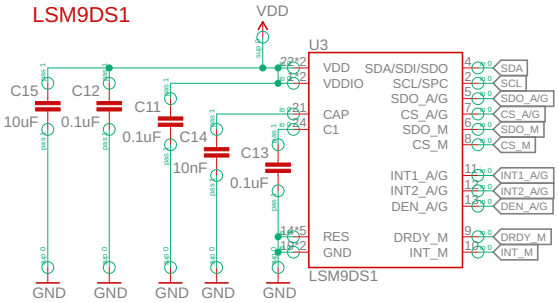
LevelConvertingChip						Cable		Beagle Bone Black
IMU	Short Circuit Channels 1-4		Color		Chip Connector	Color		
VDD	LV+3.3V		Brown		1	Purple		3.3V
	HV +5V		Orange/White		2	Gray		5V
GND	GND	GND	Black		3	White		GND
	Low Voltage	High Voltage			4 (not in use)			
CS M	-	-	White		5	Black		P923
CS AG	-	-	Green		6	Brown		P924
SCL	-	-	Purple		7	Red		P922
SDA	-	-	Blue		8	Orange		P918
SDO M/AG			Grey		9	Yellow		P921

OUTER LEG

LevelConvertingChip						Cable		Beagle Bone Black
IMU	Short Circuit Channels 1-4		Color		Chip Connector	Color		
VDD	LV+3.3V		Brown		1	Purple		3.3V
	HV +5V		Orange/White		2	Gray		5V
GND	GND	GND	Black		3	White		GND
	Low Voltage	High Voltage			4 (not in use)			
CS M	-	-	White		5	Black		P925
CS AG	-	-	Green		6	Brown		P926
SCL	-	-	Purple		7	Red		P922
SDA	-	-	Blue		8	Orange		P918
SDO M/AG			Grey		9	Yellow		P921


IMU Level Shifter Curcuit
4 Bi-Directional
Level Shifters
JP1
JP4
JP5
JP6
LSF9DS1
Breakout
MOONO_SILK



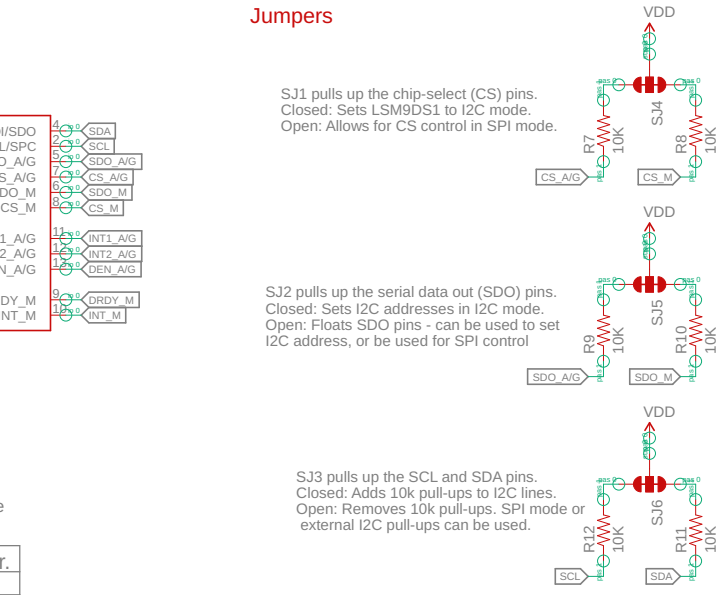


I2C Address Table

Set the SDO_AG and SDO_M to set the accel/gryo and mag I2C addresses:

SDOM/AG	AG Addr.	M Addr.
0	0x6A	0x1C
1	0x6B	0x1E

Jumpers



C.4 Encoder Wiring table

Encoder INNER LEG			
Encoder	**Cable**		**BeagleBone Black**
GND	Brown		GND
VCC	Red		+5V
A+	Orange		P8:31
B+	Yellow		P8:32
A-			
B-			

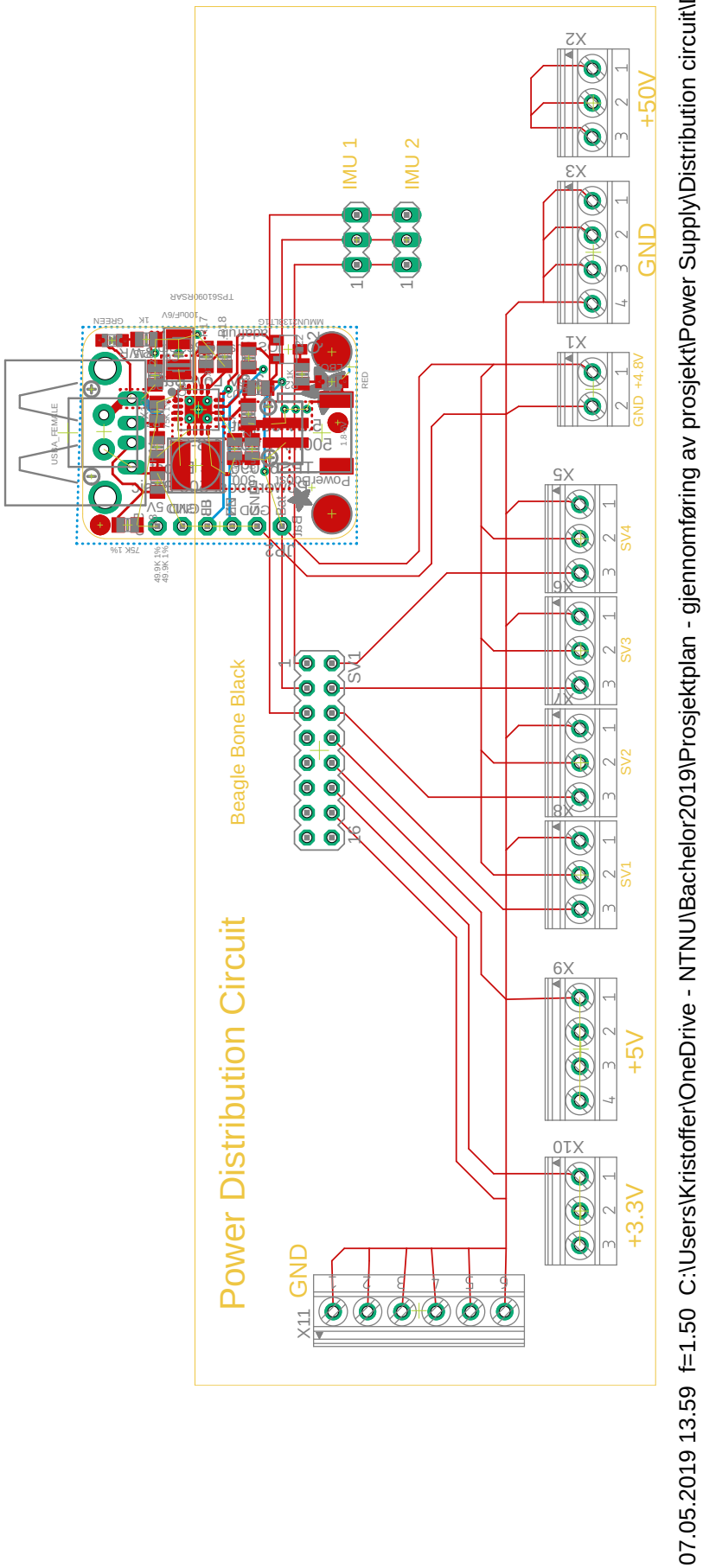
Encoder OUTER LEG			
Encoder	**Cable**		**BeagleBone Black**
GND	Brown		GND
VCC	Red		+5V
A+	Orange		P8:33
B+	Yellow		P8:35
A-			
B-			

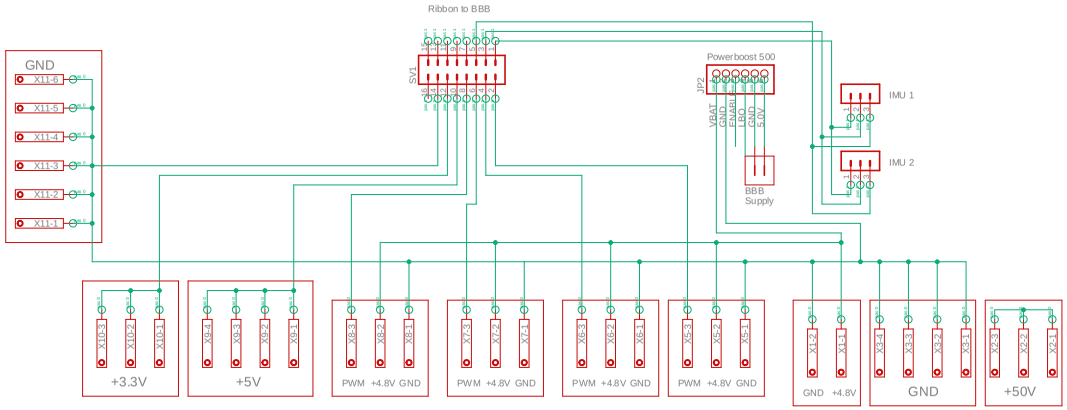
C.5 Motor Wiring table

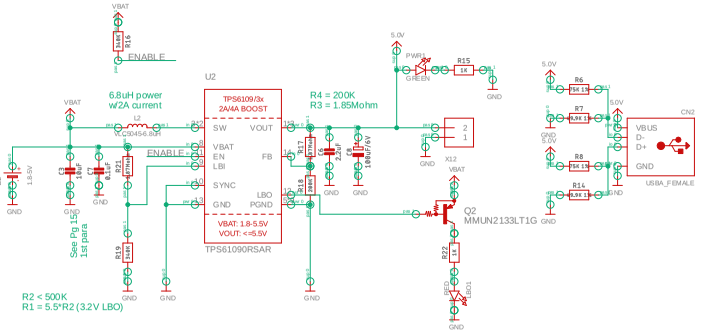
Motor Wiring Outer Leg		
Motor	**Cable**	**Escon Motor Driver**
+VCC	Red	J2:2
GND	Black	J2:1
Power Distribution Circuit		
+50V	Red	J1:2
GND	Gray	J1:1
GND	Black	J6:7
BeagleBone		
P914	Green/White	J5:1
P913	Blue/White	J3:1
Internal Connections Escon Motor Driver		
J5:2	Yellow	J3:1
J6:2	Black	J6:7
J5:5	Black	J6:2

Motor Wiring Inner Leg		
Motor	**Cable**	**Escon Motor Driver**
+VCC	Blue	J2:2
GND	Brown	J2:1
Power Distribution Circuit		
+50V	Red	J1:2
GND	Gray	J1:1
GND	Black	J6:7
BeagleBone		
P916	Green/White	J5:1
P915	Blue/White	J3:1
Internal Connections Escon Motor Driver		
J5:2	Yellow	J3:1
J6:2	Black	J6:7
J5:5	Black	J6:2

C.6 Power Supply Distribution Circuit







ISSUE	>COMPANY	© 2014
DRAWN >DRAWN	TITLE	REV >REV
CHECKED >CHECKED	DATE not saved!	DRG Nº >DRGNO
DATE >DATE	FILE: Distributioncrt	PAGE: 1/1

Connection Power Distribution Circuit to Beagle Bone Black			
Power Distribution Circuit	**Ribbon Cable Connector**		**Beagle Bone Black**
SPI CLK	1	Ribbon Cable	P922
SV4 PWM	2		P931
SPI SDA	3		P918
SV3 PWM	4		P813
SPI SDO	5		P921
SV2 PWM	6		P819
-	7		-
SV1 PWM	8		P929
-	9		-
5V Breakout	10		P904/P905
-	11		-
3.3V Breakout	12		P902/P903
-	13		-
GND Breakout	14		P900/P901

D Appendix - Servo experiment

Servo input signal range identification

This experiment is carried out to determine the PWM-signal required for the servo's endpoints. In addition to identifying the power consume for each servo.

In order to discover the pulse width corresponding to the end points we change the pulse-width of the control signal to the servo. When the servo stops trying to hold its position and turn freely the PWM signal has gone past the servo's limits. Then the pulse width was set to its previous value and this was identified as the limit value. However, there was a zone inbetween where the servo did not function consistent, making the limits the last value before the servo stopped working consistently. Even though both Arduino and oscilloscope were used for measuring, the results turned out to be based only on the oscilloscope due to its superior reliability.

In order to separate each servo, they have been assigned with numbers. While the robot is mounted on the torso bracket. Servo nr 1 is the right servo Outer leg, in the walking direction of the robot. The side were the electronics are mounted are considered the front of the robot. Furthermore, the numbering just goes from right to left in the walking direction. Servo number 2 is the right servo on the Inner leg etc.

Results

Servo Nr	1	2	3	4
Lower Limit [µs]	780	780	778	779
Upper Limit [µs]	2184	2180	2182	2180

Equipment list:

Servo : Futuba S2954

Micro Controller: Arduino Mega ADK

Oscilloscop: Tektronix TDS 2012C

Software Oscilloscop : Tek TDS 2012C OpenChoise Desktop

References:

1. https://www.rchelicopterfun.com/digital-servos.html
2. https://github.com/ElectricRCAircraftGuy/eRCaGuy_TimerCounter/blob/master/eRCaGuy_Timer2_Counter.h

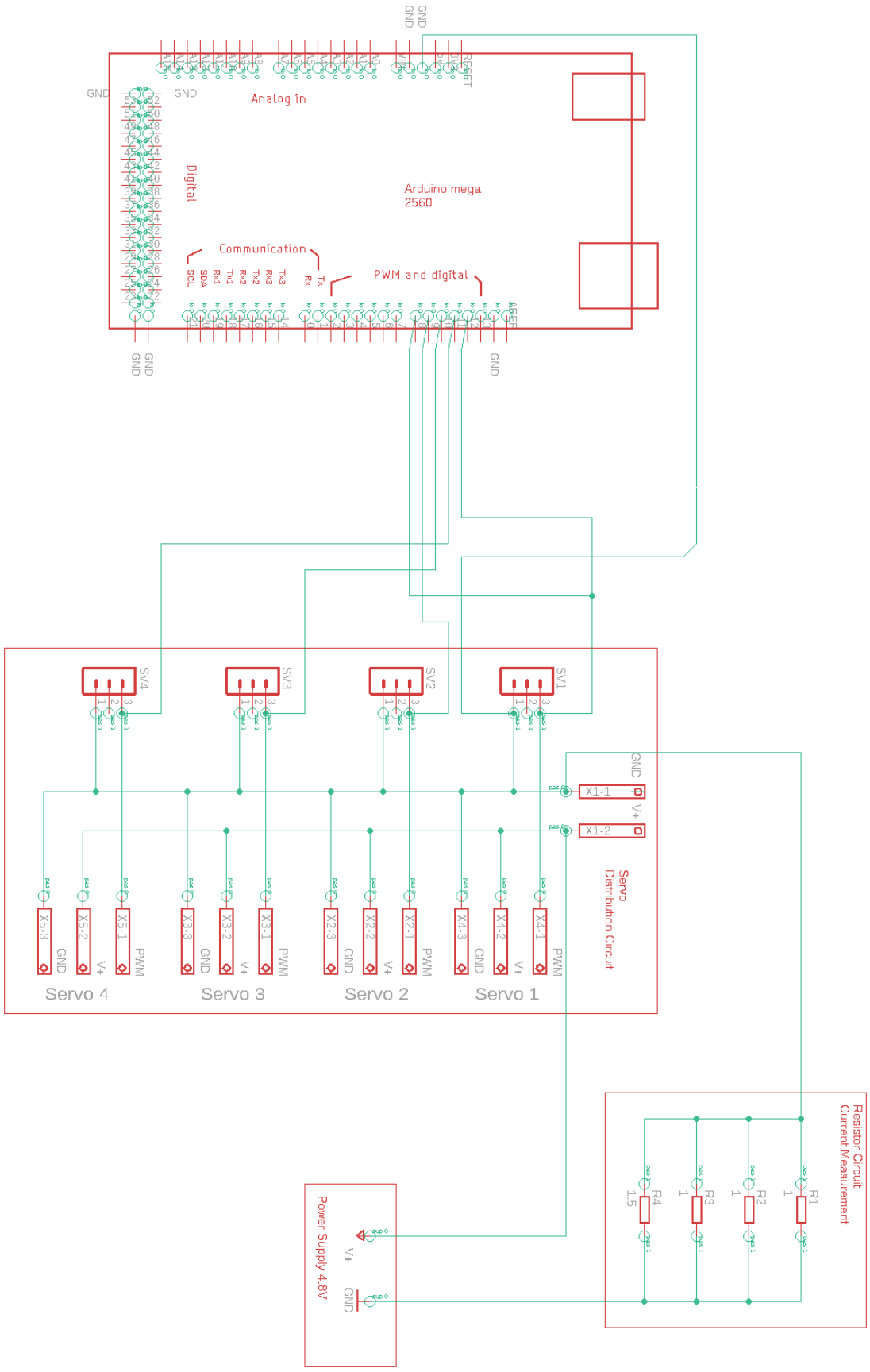
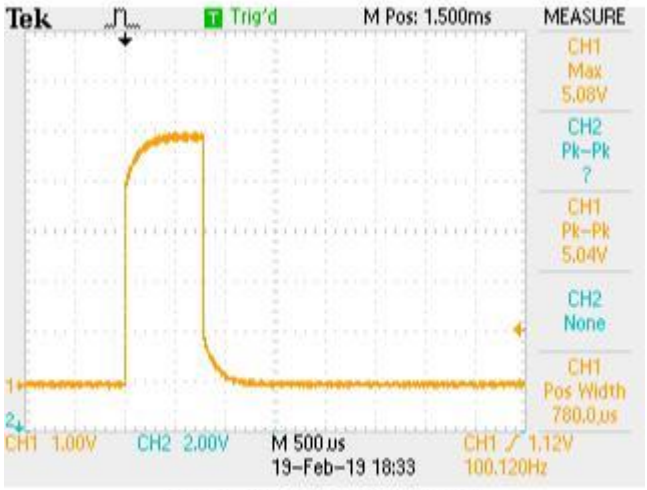
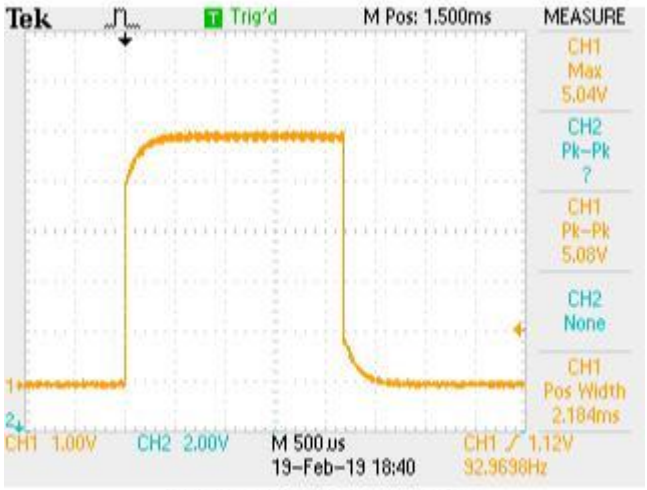


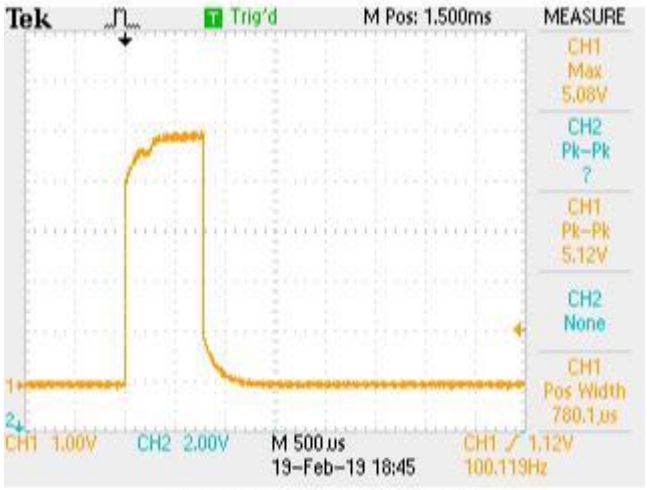
Figure 1: Schematics Servo Experiment

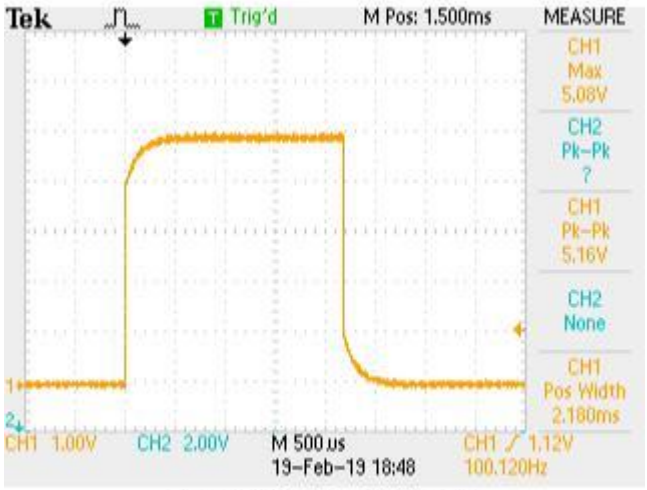
Servo 1:

PWM input Lower Limit	779 µs (Worked infrequent at 778 µs)
Measurement Arduino	Pulsetime(µs) = 785.00 Period_us(µs) = 20006.00 PulseFreq(Hz) = 49.99
Oscilloscope	 19.02.2019 11:54:04 Servo 1

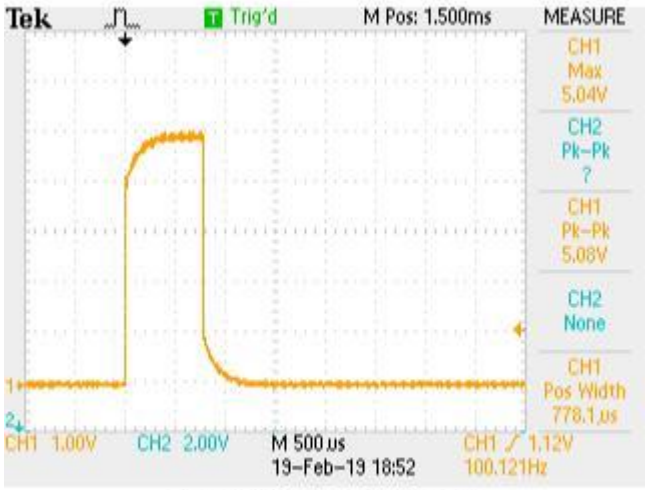
PWM input Upper Limit	2184 µs
Measurement Arduino	Pulsetime(µs) = 2191.00 Period_us(µs) = 20005.00 PulseFreq(Hz) = 49.99
Oscilloscope	 19.02.2019 12:00:35 Servo 1

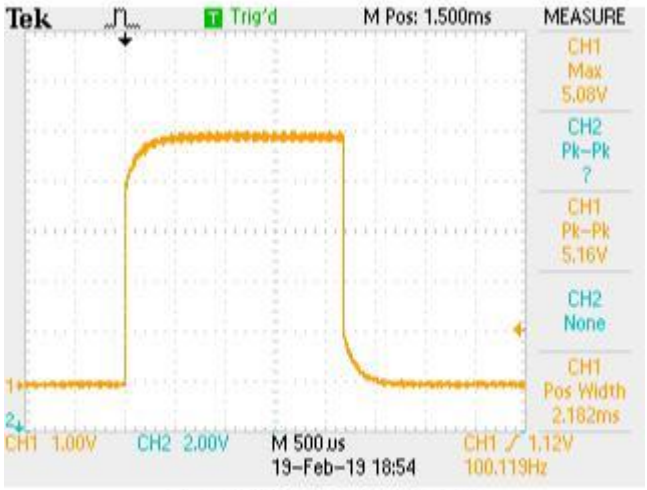
Servo 2:

PWM input Lower Limit	779 µs (Worked infrequent at 778 µs)
Measurement Arduino	Pulsetime(µs) = 785.50 Period_us(µs) = 20005.50 PulseFreq(Hz) = 49.99
Oscilloscope	 19.02.2019 12:05:53 Servo 2

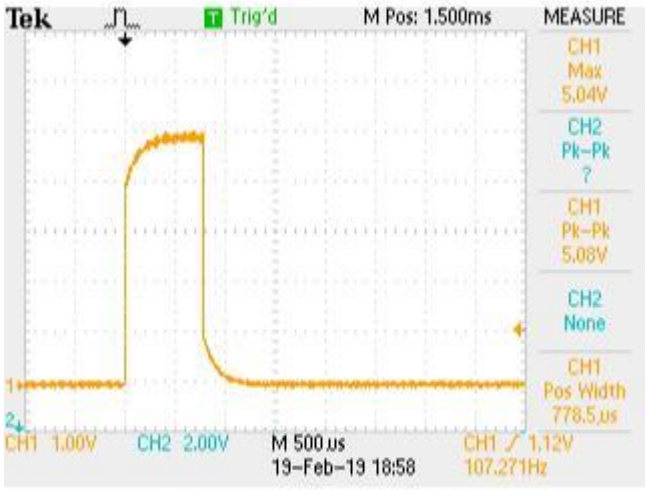
PWM input Upper Limit	2183 µs
Measurement Arduino	Pulsetime(µs) = 2190.00 Period_us(µs) = 20005.50 PulseFreq(Hz) = 49.99
Oscilloscope	 19.02.2019 12:08:47 Servo 2

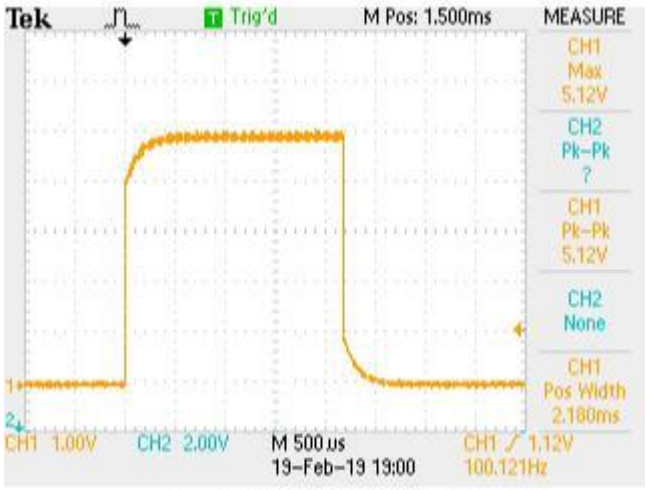
Servo 3:

PWM input Lower Limit	779 µs (Worked infrequent at 778 µs)
Measurement Arduino	Pulsetime(µs) = 785.50 Period_us(µs) = 20005.50 PulseFreq(Hz) = 49.99
Oscilloscope	 19.02.2019 12:12:43 Servo 3

PWM input Upper Limit	2183 µs
Measurement Arduino	Pulsetime(µs) = 2189.50 Period_us(µs) = 20005.50 PulseFreq(Hz) = 49.99
Oscilloscope	 19.02.2019 12:14:51 Servo 3

Servo 4:

PWM input Lower Limit	778 µs
Measurement Arduino	Pulsetime(µs) = 784.50 Period_us(µs) = 20005.50 PulseFreq(Hz) = 49.99
Oscilloscope	 19.02.2019 12:18:45 Servo 4

PWM input Upper Limit	2181 µs
Measurement Arduino	Pulsetime(µs) = 2187.50 Period_us(µs) = 20005.50 PulseFreq(Hz) = 49.99
Oscilloscope	 19.02.2019 12:21:02 Servo 4

Current Measurement with Oscilloscope:

In order to measure the current without a clamp we can take advantage of the linearity of ohm's law. If we put a small enough resistor in series with the load, it won't affect the output effect of the load to much, but still give a measureable voltage over the resistor. Which is proportional with the current.

A resistor circuit was put in series connected to ground.

$$R_{tot} = \left(\frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3} + \frac{1}{R_4}\right)^{-1}$$

$$\left(\frac{1}{1\,\Omega} + \frac{1}{1\,\Omega} + \frac{1}{1\,\Omega} + \frac{1}{1.5\,\Omega}\right)^{-1} = \frac{3}{11}\Omega \approx 0.27\Omega$$

From this we get the current from the formula:

$$I_{servo} = \frac{U}{R} = \frac{U(t)}{\frac{3}{11}\Omega}$$

With a measurement of 175 mV peak voltage we get:

$$I_{servo} = \frac{175 \cdot 10^{-3} V}{\frac{3}{11}\Omega} = \frac{1925}{3} A \approx 642\ \text{mA}$$

Which means the maximum current from all four servos should approximately be

$$I_{tot} = 4 \cdot I_{servo} \approx 2.5\ A$$

However the servos should only be operated in pairs, resulting in a maximum current

$$I_{max} = 2 \cdot I_{servo} \approx 1.25\ A$$

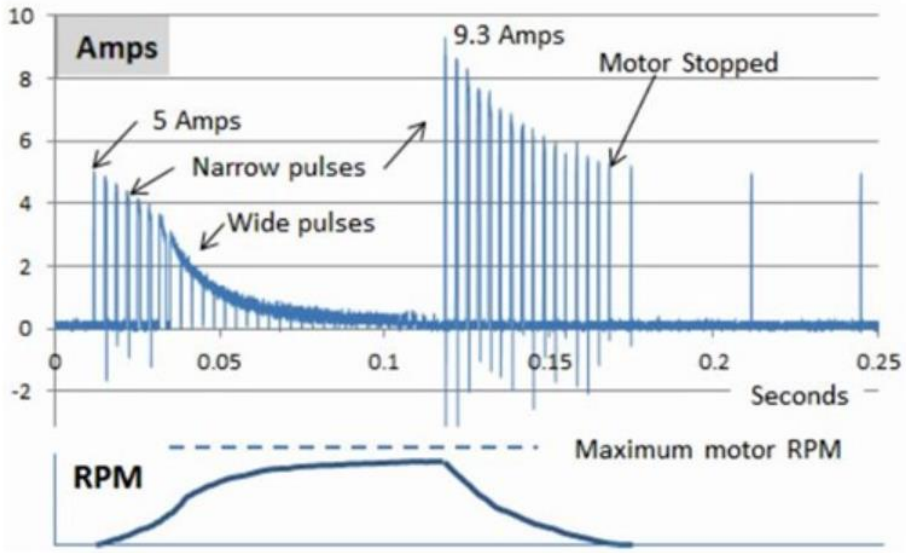


Figure 2: Illustration step test servo[1]

From Figure 2 we can see the current we should expect the servo to draw, as we drive it to a different position. From the RPM-curve we see the first part is to speed up the motor, and the second is to stop it.

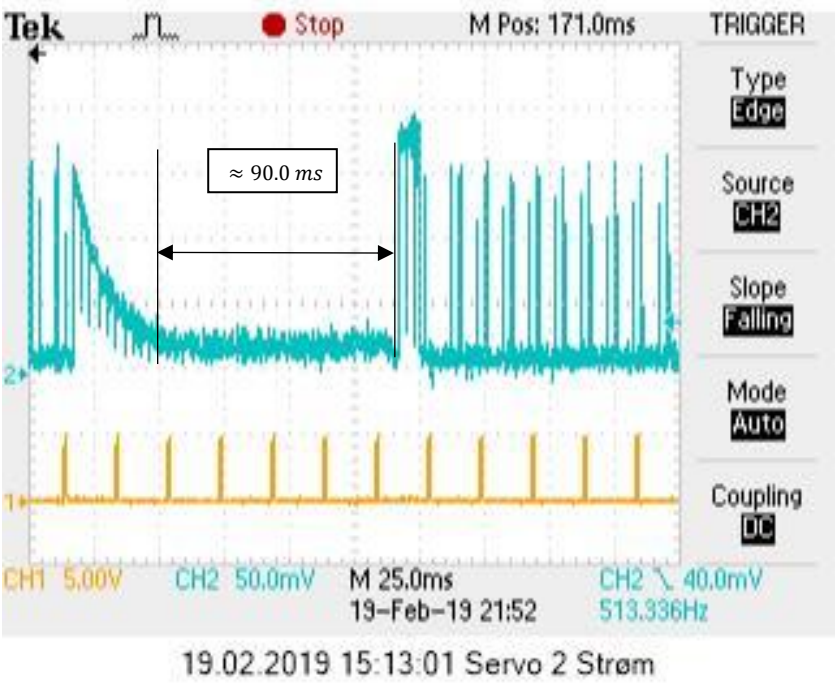
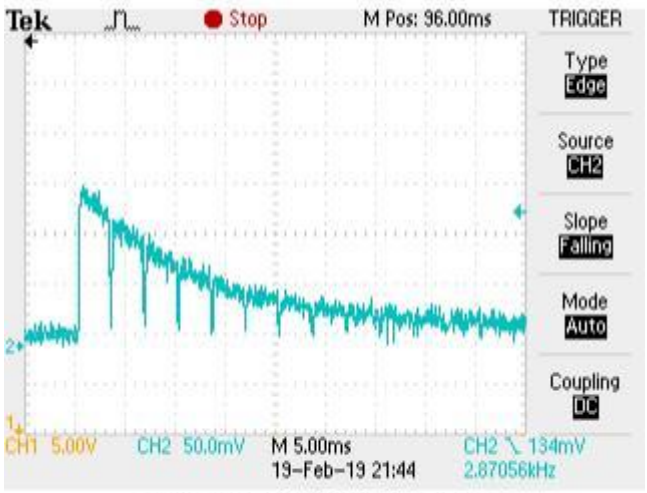
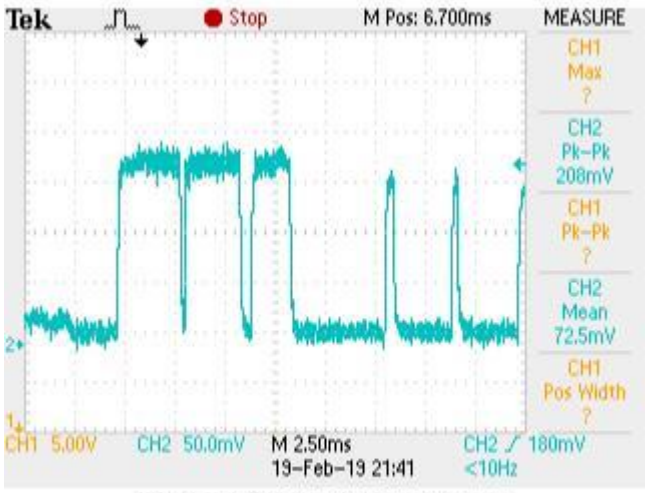


Figure 3: Results current measurement

We get a similar result from our servo with a step from one end stop to the other. The stopcurve is slightly different. The yellow curve shows the PWM signal. We notice the servo draws a mentionable current at startup and to stop. However, in-between it almost draws no current at all. We could take advantage of this and drive the other servo so the peaks will not align. This should reduce the maximum current almost by 50%. This implies the high servo current turns out to be a problem. If not, simultaneous run is preferred due to its simplicity.

Startup Curve	
Peak Value: $\approx 150\ mV$ $\approx 550\ mA$ End Value : $\approx 10mV$ $\approx 37mA$ Time : $\approx 35\ ms$	 19.02.2019 15:05:30 Servo 2 Strøm
Stop Curve	
Peak Value: $\approx 175\ mV$ $\approx 642\ mA$ Time : $\approx 35\ ms$	 19.02.2019 15:02:21 Servo 2 Strøm