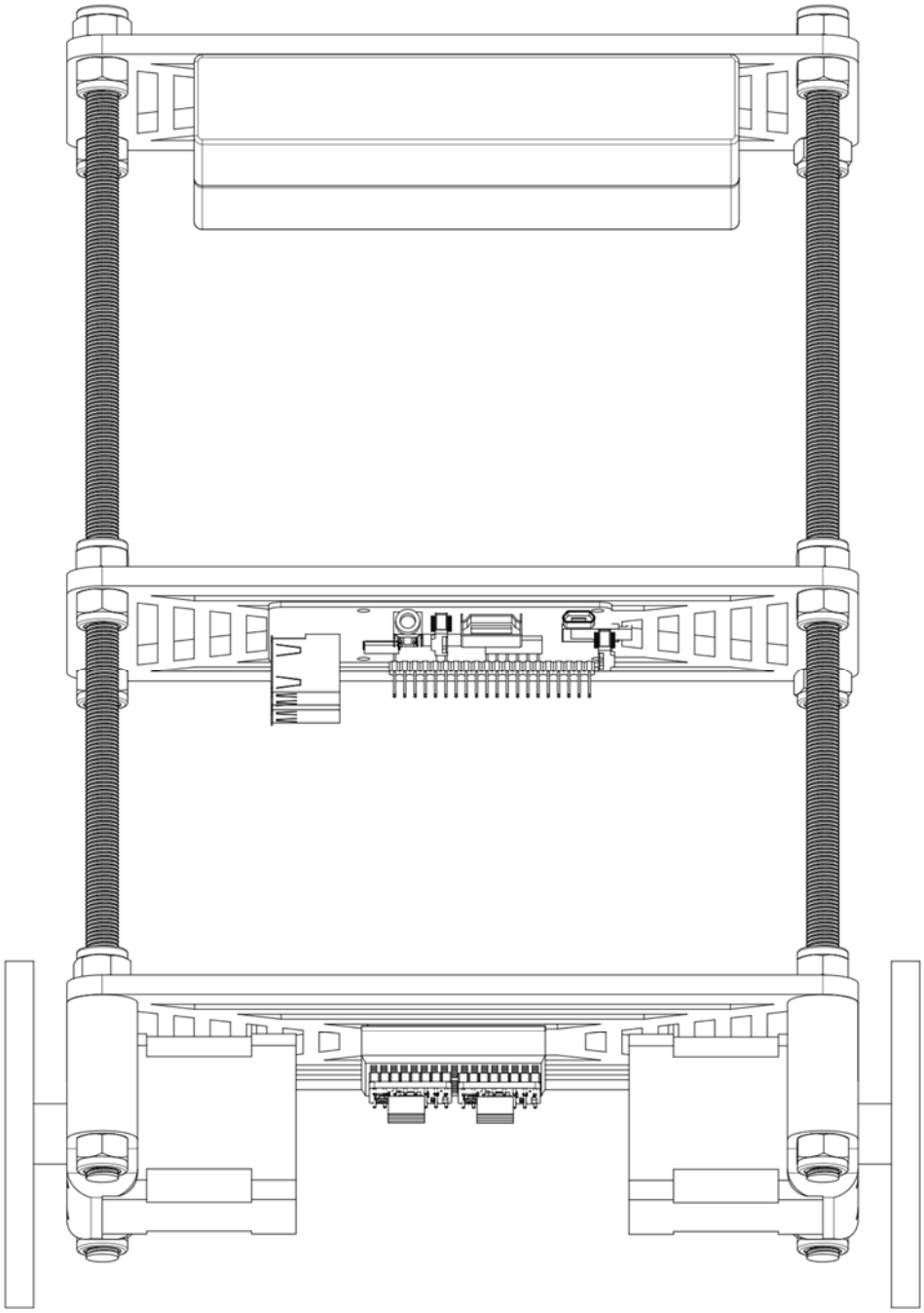

Rollie

Michael Jones - A00916729
Jonas Menge - A00830579

School of Energy

Mechatronics and Robotics Technology

Self-Balancing Robot Final Report

VERSION 0.0.1

ROBT 4491

Attention: Pavlos Paleologou, Chris Baitz, Matthew Rockall
Date: 2016-05-06

Acknowledgments

We would like to thank the Open-source community for providing solid groundwork for us to build upon.

We would also like to thank Chris Baitz and Pavlos Paleologou for giving us creative freedom, and providing helpful advice throughout the project.

Thank-you Matthew Rockall for providing us with the tools to create the necessary supporting documentation.

Copyright

Michael Jones - A00916729
Jonas Menge - A00830579

Abstract

Balancing robotic platforms capable of carrying small payloads have many useful applications. However, in the past most designs needed expensive hardware to eliminate positional error. Our approach eliminates the need for costly positional encoders and is capable of maintaining its position by counting steps of the motors. We use this data to correct any angular position steady state error introduced by mechanical or electrical noise on the Inertial Measurement Unit (IMU). A cascaded Proportional Integral Derivative (PID) controller, that combines positional and angular information, is used to stabilize our platform. We call our approach Rollie, and release it as open source under the GNU free software license.

The open-source approach means that we use software and hardware in our project that are free for use, and modify for the Rollie project. The Libraries used for I^2C connection to the IMU are from the wiringPi Library, and have a GNU LGPLv3 free software license. Most of the hardware used in the Rollie project is open-source hardware including the Raspberry Pi microcontroller, the Sparkfun IMU, and the Stepper motor controllers. The 3D printer used to manufacture the parts for Rollie is a Prusa i3 Mendel RepRap 3D printer that was developed by the open-source community.

The parts for Rollie are designed to be 3D printed. The bed size of the 3D printer determines how wide we are able to make the parts. The parts were all designed to have a large flat surface to make contact with the bed for ease of printing. The parts for Rollie took up to 4 hours each to print. The printing was done gradually over the course of the project, starting with the mounts for the motors. Each part was designed and perfected over a couple of versions before they were added to the final product. We find that the parts are both strong and lightweight which make this method of manufacturing a very good option for our application.

The Raspberry Pi microcontroller is Rollie's control system. The Raspberry Pi runs a full Linux operating system and is connected to the internet. The internet connection allows us to connect to the robot over SSH (Secure SHell), which gives us a wireless terminal into the Raspberry Pi. To be able to connect to Rollie from anywhere with Wi-Fi, we have a script that automatically connects to a cloud hosted server to form a reverse SSH tunnel through the BCIT secure network.

The project has met the requirements as outlined in the request for proposal. To increase functionality and usability further development must be made. We recommend increasing the usability by implementing an ncurses command line menu system that will allow the user to more easily control and improve the performance of the control system.

Table of Contents

Table of Figures

Michael Jones - A00916729
Jonas Menge - A00830579

1 Introduction

Rollie is a self-balancing robot platform designed to carry small payloads, such as cameras or other sensors. It is also a great learning tool for those interested in robotics, computer science or electronics.

Two-wheeled robots are very useful for quick nimble movement which makes them ideal for moving around in tight spots or around complex obstacles. With additional sensors and some clever algorithms Rollie can be used in offices for small deliveries of food or mail.

The scope of Rollie for the Mechatronics and Robotics Capstone project was to design and build a lightweight, balancing platform. Some of the stretch goals of the project were to take this platform and interface additional sensors for autonomous control. We also wanted to build a candy firing mechanism for the kids.

Open source was the theme of the Rollie project, and this methodology was brought into the all aspects of the hardware and software. Open source hardware used in this project includes Pololu motor drivers, Raspberry Pi computer, and Sparkfun IMU. All of the manufacturing for Rollie was done on an open source Prusa i3 Mendel 3D printer. The open source software includes the Rasbian Linux operating system, the Wiring Pi library (for I^2C functions), and GitHub (for version control).

The hardware and software was designed for modularity. The hardware for the power systems, controller, and motor drivers are located in their own modules and are located on appropriate parts of the robot. The software is neatly packaged in different .cpp files each with their own header file according to their functions. Those files are contained within a larger repository stored both locally on the Raspberry Pi, and on Github.com where our files can be collaborated on by the Rollie team.

2 Project Scope / Objectives

The proposed scope of the project as outlined in the project proposal was to design and build a modular self-balancing robotic platform. Each part of the robot was meant to have its own module, and each module was to be connected by a neat cable with a plug. The design proposed was to be completely 3D printed and connected together by threaded rod. The proposed microcontroller was the Raspberry Pi, and the proposed motors were Stepper motors. Also outlined in the proposal was a robust wireless control system utilizing the local Wi-Fi network at BCIT rather than RF or Bluetooth. The proposal also included some stretch goals to complete if the base deliverables were met. These included a candy firing mechanism, a camera or other vision system, and complete control of all movement by an operator.

The project at this time has met the base deliverables of balancing and wireless connectivity. The design is just as outlined in the project proposal. All the parts were 3D modeled and printed on a 3D printer, the parts are connected together with threaded rod. The Raspberry Pi is being used as a controller to read the angular position and control the stepper motors to balance the system.

Some things we were not able to achieve before the project deadline include most of the stretch goals outlined in the proposal, including the candy firing mechanism. There were a few unexpected setbacks in Rollie's development. Rollie was a completely self-funded project with a small budget divided between the members of the project. This budget meant we had to be smart about how we spent the money on the different components of the robot. We were able to save money by taking advantage of components commonly used for other tasks. The motors and motor drivers used in our project are common on 3D printers, so they are relatively inexpensive and widely available from China. There was a mix-up with the motors where they were accidently sent here and back to china, delaying the start of putting the robot together by a couple of weeks.

We also ran into difficulties with configuring the wireless connectivity due to issues with BCIT network security. This is why we decided to use the relay server to avoid having to connect between computers directly on the BCIT network. Another issue we encountered was filtering the noise from the IMU. The noise presented an error in our angle calculation that caused the motors the shift back and forth at steady state, impeding balance.

3 Project Description

This project requires the cooperation of both hardware and software to maintain its position and keep the platform from falling over. Balance is achieved by measuring its angle relative to the ground using an IMU (inertial measurement unit) and positioning its wheels under the mass of the robot. The robot can be modeled as an inverted pendulum. Most of the mass is located at the top of the platform which makes it easier for the controller to correct the angle by increasing the period of the pendulum. The larger the period, the more time the controller has to react. (Lundberg, 1994–2002)

3.1 Design

Our design philosophy was keep it simple and flexible. This is reflected in the modular, expandable sections and the highly reusable code.

3.1.1 Hardware Development

The entire project was 3D modeled in a cloud based modeling software called Onshape. This allowed us to verify design ideas on the computer in before making the parts and assembling the robot. This in conjunction with 3D printing saved time in the shop, and allowed a large amount of flexibility in the design.

3.1.2 Software development

All software was written on the Atom text editor and in the terminal directly. Our build system consists of a makefile that compiles all the C++ source files in the /src/ directory. This makefile lets us determine which additional libraries to include. The makefile is configured to compile using g++ to allow for compiling C++ source files needed to easily create threads. Simply executing the command “make” compiles the project.

Git was incorporated into the project because we had a need for a flexible version control system that allowed us to edit code on one computer, push to the online repository and pull the code onto the Raspberry Pi for testing. Git has proven to be a vital tool for the development of this project.

3.2 Manufacturing

All the parts used in this project were either 3D modeled and printed, or basic hardware from Home depot, such as threaded rod. The 3D printer allowed for lots of time to be saved hand making the parts for Rollie using conventional manufacturing methods, and was also an extremely cost effective way to quickly produce strong, accurate parts for our robot. Rollie can be completely assembled using two tools, an Allen key and a wrench.

3.3 Hardware

The hardware of this project includes batteries, microcontroller, inertial measurement unit for sensing angular position, motors and motor controllers for moving and balancing the robot.

3.3.1 Power Supply

Two 12V 2200mAh Lithium Polymer batteries are used in series to power the robot. This brings the system voltage to 24V which is used directly in the motion controllers. This high voltage is stepped down using a switching, DC-DC converter to 5V. This is used to power the microprocessor and inertial measurement unit.

Testing has shown that the batteries are capable of sustaining about 6 hours of continues operation. However this figure can depend on how well the system is tuned and the amount of noise of the sensor data.

3.3.2 Inertial Measurement Unit

A 6 degree of freedom, 3 axis gyroscope and accelerometer combination board is used to capture the robot's angle. This allows the robot to position the wheels underneath the chassis to prevent it from falling over. The IMU is interfaced to the Raspberry Pi via I^2C. The accelerometer and gyroscope are addressed individually on the I^2C bus. The wiringPiI2CReadReg8() function from the wiringPi library is used to read the individual registers that contain the data from the accelerometer and gyroscope respectively. (SparkFun, 2014)

3.3.2.1 Gyroscope

The gyroscope is initialized using the wiringPiI2CSetup() function which inputs the address of the gyro on the I^2C bus (0x68) and returns a the device id which is used in the I^2C read and write functions when referring to the gyro.

The gyro X, Y, Z axis registers return a value proportional to the angular velocity around each respective axis. The scaling factor is defined by the data sheet, and is 14.375 deg/s/LSB of the 16 bit value. This value is read and integrated at 100 Hz which gives the angular displacement per 0.01 sec. We insure that the data is only read when it is ready by polling the done bit of status register on the gyro.

3.3.2.2 Accelerometer

The accelerometer is initialized using the wiringPiI2CSetup() function at address 0x53 on the I^2C bus. The accelerometer is accurate at steady state, but is very susceptible to mechanical accelerations during movement.

The accelerometer measures the gravity about axis X, Y, Z and store the values can be read from the 16 bit registers, 8 bits at a time using the wiring Pi library. The MSB is read first, shifted over 8 bit, then OR'd with the LSB. The units of this value are in m/s^2 multiplied by a scaling factor that is dependent on the range the accelerometer is set at.

3.3.3 Micro Processor

A Raspberry Pi 2 single board computer is used to perform all calculations necessary to balance the platform and communicate with the server. The raspberry Pi runs at 1 GHz and has 4 cores. This allows us to run multiple processes concurrently while maintaining the timing necessary to read sensor data and send position commands to the motion controllers.

Since it is a fully fledged computer on a PCB (Printed Circuit Board) it allows for many different connections to be made to it. This lets us connect up to 4 USB devices (of which one is used for wireless control). Additionally it features an Ethernet port for wired connections to networking devices.

3.3.4 Motion Controllers

Two stepper motor drivers are used in conjunction with stepper motors to control the position of the robot. Both controllers accept step and direction signals and incorporate selectable micro stepping. The drivers are set to half-step the motors, which changes the normal 400 steps per rotation to 800.

3.3.5 Motors

Two Nema 17 Stepper motors, with an angular resolution of 0.9 degrees per step are used to control the robot. There are 400 full steps per rotation of the motor, using micro stepping changes this to 800 half steps per rotation.

3.3.6 Chassis

The chassis is mainly 3D printed out of PLA (polylactic acid). This offers the benefit of flexibility in design and in rapid prototyping.

Threaded rod is used to complete the structure with nuts and washers holding together the individual components.

3.4 Software

A multithreaded design is employed to run the different process and to get the most out of the multi core architecture of the ARM processor. This has proven to be very effective at maintaining the timing requirements of the different peripherals.

Software flowcharts and block diagrams can be found in Appendix D.
The complete program source code can be found in Appendix E.

3.4.1 Sensor Filtering

The data that comes from the sensor has error due to mechanical and electrical noise. This affects the accuracy of the angle read by the IMU. Filtering is required for the gyro and the accelerometer to be able to produce reliable angle data.

The accelerometer has a tendency to pick up large amounts of mechanical noise from accelerations on the system that mimic gravity. The data from the Gyroscope makes up for 99% of the angle calculation, and the accelerometer for 1%. This is

done because the gyro is much more accurate while moving, but has a tendency to drift in steady state. Since we care much more about having an accurate angular reading during movement, we favor the gyroscope for determining the angle of the system.

The accelerometer is accurate in steady state, but is susceptible to large error while the system is moving. This method of filtering is called a complementary filter. The angle is mostly determined by the gyroscope and the accelerometer is used to correct the angle when it is in steady state. (Jan, 2013)

3.4.2 Cascaded PID Algorithm

A cascaded PID controller is used to control the position of the robot. It utilizes two PID loops, working in unison to maintain the position setpoint of the robot, and the angle of the tilt of the robot.

The algorithm records the position by recording the pulses that are sent to the motor controller. Each pulse corresponds to a fixed angular displacement. By recording the pulses, the position of the robot can be determined. This actual position is compared to the setpoint position and the error is passed into the first stage of the cascaded PID algorithm. The output from this PID is used as the angle setpoint for the angle PID controller.

This angle setpoint is then compared to the angle measured by the IMU and the error is passed into the second stage PID. The second stage tries to keep the error as small as possible by varying the position of the motors. This in turn changes the angle and position of the robot. This system allows the position of the robot to be controlled with the first PID loop and therefore make the robot change its angle to to get to the new position setpoint.

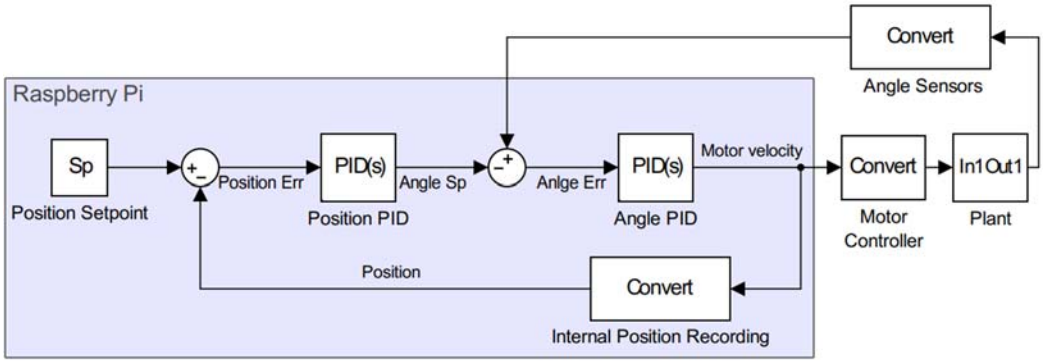

Figure 1 - Cascaded PID Block Diagram

The algorithm and is frequency compensated and can easily run at the maximum sensor rate of 100Hz. (Cvra, 2015)

3.4.3 Motion Control Algorithm

The algorithm designed for moving the robot is linked to the position PID loop. The robot is told to move forward or backward by giving the PID loop successively larger positional setpoints. The amount the robot is to move at one time determines the velocity of the robot. A control menu that reads the inputs from the keyboard arrow keys to tell the robot to move. For turning, another PID loop determines the amount of steps to send each motor individually, which is subtracted or added to the number being determined by the angle PID.

3.4.4 Remote Server

A dedicated server has been built for this project. It is hosted by Digital Ocean and is used as a network relay to communicate with the Raspberry Pi computer. This allows us to control the robot from anywhere and work around the tight network security policies at BCIT.

When the robot is powered up, it automatically connects to the server using a reverse ssh tunnel. This is then used to establish a connection to the user.

3.4.5 Boot System

We have implemented a complex boot procedure to ensure the Raspberry Pi establishes a good connection with the remote server. On boot, a CRON job executes a script that sends an email to a predefined address. This script is located in the ~/rollie/tools/sendip.py .

Additionally, four times a minute, a script (connect_to_server.sh) attempts to create a reverse ssh tunnel to the server. If it detects that the tunnel already is established it outputs the process id. This is stored in a log file for debugging purposes. This log file is located in the home directory (~/boot.log)

3.5 Testing

Much of the testing for each section was performed as each part was finished. For example, the IMU code for determining the angular position was simply tested by running the code, and verifying the correct angle was detected.

Many tests have been conducted by the Rollie team in the pursuit of balance. Once all the sections of code had been built, verified, and added to the complete system there was still the complicated problem of tuning the PID control gains. The procedure for this largely consisted of the guess-and-test method, each time watching the response and making a new more educated guess at the proper gains.

3.6 Future work

Since our project is fully open source, we hope there is a prosperous future and continuous development of the code base. We plan on perfecting our algorithms to be more robust as time progresses.

Features we have planned for the future:

- Ncurses menu - more user friendly and greater control over robot
- Self-tuning algorithm - automatically find the best balancing parameters
- Vision system - item tracking, QR code scanning
- 3D mapping and autonomous operation

3.7 End of Project Plan

Since this project is self-funded, we will be keeping the components for future development or incorporation in future projects.

We plan to maintain the GitHub repository so future students and other interested parties can build their projects upon our work.

4 System Specifications

4.1 Safety

An effort was made to make Rollie follow safety standards outlined by CSA z432-94. Danger related to loss of control is mitigated by designing the wheels to a diameter that makes the robot not run away when it falls over. This is achieved by making the wheels lift off the ground when the robot tilts past a recoverable angle.

The wheels of the robot can only exert a small amount of torque before the motor controllers limit the current and cause the wheels to stop. This is not enough to cause damage to fingers or other appendages.

5 Conclusions and Recommendations

The project as outlined by the request for proposal was to build a lightweight, two-wheeled, self-balancing robot. Our approach was to use 3D printing in the construction of the parts to save money and time. Stepper motors were used for the actuators to provide simple control of the robot. An inertial measurement unit for angular positional feedback, and a Raspberry Pi for running the control system and other software. Over the course of the project we had many successes, including designing a wireless robotic control system using the Raspberry Pi and maintaining positional accuracy without the use of feedback from encoders. This resulted in unprecedented cost savings which previous projects could not achieve. We have been very happy with both the hardware and software design of this project, and the overall package looks and works great.

Future recommendations for this project are to increase usability by incorporating an interactive menu system and to increase stability of the balancing algorithms. Other improvement include object avoidance and mapping the surroundings of the robot.

Michael Jones - A00916729
Jonas Menge - A00830579

References

Cvra. (2015). *PID controller*. Retrieved from https://github.com/cvra/pid

Jan, P. (2013). *pieter-jan*. Retrieved from http://www.pieter-jan.com/node/11

Lundberg, K. (1994–2002). *The Inverted Pendulum System*. Retrieved from MIT: http://web.mit.edu/klund/www/papers/UNP_pendulum.pdf

SparkFun. (2014). *IMU Digital Combo Board*. Retrieved from https://github.com/sparkfun/IMU_Digital_Combo_Board

Appendices

Appendix A - User Guide

Unboxing

Upon arrival of your robot ensure that all parts are intact. It is recommended that the batteries be charged as they may have self-discharged in shipping.

Charging the batteries

> **Warning:** Lithium-polymer batteries may explode if punctured, overcharged or short-circuited. Always use the provided charger to charge the batteries and read the manufacturer's recommendations

To charge the two lithium-polymer batteries;

1. Carefully unplug the connectors from the robot
2. Slide the batteries out of the tie downs
3. Connect the supplied charger to a 12V 6A power supply
4. Connect the battery terminals to the charger, ensuring that the polarity is correct
5. Connect the balancing port of the battery to the charger
6. Select Lipo-Balance from the menu and press start
7. Once the charger says that the battery is full, replace it with the other battery (repeat from step 4)
8. Reinstall the batteries
9. Connect the batteries to the wiring harness

Michael Jones - A00916729
Jonas Menge - A00830579

Connecting to Rollie for the first time

The first time you run Rollie you must connect it to the internet through a router or switch using a standard Ethernet cable.

This is done by locating the Ethernet port on the Raspberry Pi computer (on the second level of the robot). Insert the provided Ethernet cable into it and insert the other end into the router or switch. Connect the battery power to the robot and wait for it to start.

Rollie will then send you an email to the address specified on your order. The email will contain the IP address that it is connected to. This address will allow you to connect to, and control every aspect of the robot.

> **Note:** Windows users may need to install a ssh client to connect to the robot. We recommend PuTTY. It can be downloaded from http://www.putty.org/ for free. Follow the install and usage instructions.

Connect a computer to the same network as the robot. To communicate with the robot a secure shell session must be established. This can be done by running the following command from a terminal (replace the IP address with the one provided in the email).

```
ssh pi@192.168.0.1
```

Enter “raspberry” as the default password to log into the system. This will open a new terminal session.

Changing the default password

The first thing you should do is change the password of the Raspberry Pi. This can be done by entering the following command:

```
passwd
```

You will be prompted to enter the current password (raspberry) and a new password of your choice. You will need to use this password to log into the robot from this point on.

Michael Jones - A00916729
Jonas Menge - A00830579

BCIT Network Setup

To be able to use the robot wirelessly, the network credentials must be entered into the Wi-Fi configuration file.

Update the network configuration file by running the following:

```
sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
```

Then add the following to the configuration file:

```
network={
    ssid="BCIT_Secure"
        key_mgmt=WPA-EAP
        eap=PEAP
        identity="a00######"      <--- Here
        password="super_secret"   <--- And here
        phase2="auth=MSCHAPV2"
}
```

This will allow the robot to connect to our server so that it can be controlled remotely. From this point on, the robot will automatically connect to the server when it is turned on.

Logging in through Wi-Fi

Once the network is configured properly, the Raspberry Pi will automatically connect to the server. The user can then connect to it through the server.

This is done by establishing a SSH connection with the server and then connecting to the Raspberry Pi.

To log into the server run the following command from your computer's terminal emulator:

```
ssh root@159.203.6.212
```

Then connect from the server to the raspberry pi by running:

```
pi2
```

This will open a new terminal session and allow you to control the robot.

Michael Jones - A00916729
Jonas Menge - A00830579

Updating the software

The Rollie project is under constant development. It is highly recommended to update the preinstalled software regularly.

To update the Linux operating system that runs on the Raspberry Pi run:

```
sudo apt-get install update
```

To update the code used to run Rollie, first change into the Rollie directory:

```
cd ~/rollie/
```

Then download the latest code from the Rollie repository:

```
git pull
```

The code needs to be compiled into machine code which the computer can run efficiently. To do this run:

```
make
```

Then to install the program so that it can be executed from any directory run:

```
make install
```

Start-up

To run the robot simply execute the Rollie program.

```
sudo rollie
```

Shut-down

To stop the rollie program when it it is being executed, simply press CTRL+C. This will immediately terminate the program.

To safely shutdown the Raspberry Pi computer run:

```
sudo shutdown now
```

Safety concerns

The most dangerous aspect of this project is the batteries. A fire could be started it they are overcharged, undercharged, short circuited or punctured.

Extreme care should be taken when charging and the instructions for changing should always be followed.

Michael Jones - A00916729
Jonas Menge - A00830579

Troubleshooting

Table 1 - Troubleshooting

Symptom	Cause	Solution
The server cannot connect to the robot.	Stale SSH sessions are blocking the port and inhibiting new connections	On the server run: pstree -p Look for the process ID of the stale SSH sessions and record the numbers kill process number
Rollie will not start	- Incorrect working directory - Not properly installed	Change directory: cd ~/rollie/src/ Compile and install: make install
A stepper motor is not spinning when the signals are sent	Incorrect current setting	Adjust the current setting potentiometer to read a voltage of 1.6V at the test point. (follow manufactures guidelines for more information)

Michael Jones - A00916729
Jonas Menge - A00830579

Appendix B – Calculations

Battery life calculation - used to size system batteries

$2 * 2A = 4A \max draw$

$$\frac{2.2\ mAh}{4A} = 0.55\ hours$$

Actual battery life has proven to be 6 hours. Therefore:

$$\frac{2.2\ mAh}{6.0\ hours} = 0.366\ A\ draw$$

Stepper motor velocity

Velocity of the steppers is controlled by changing the frequency of the pulses sent to the stepper motors.

$$400\frac{Steps}{rotation} * \frac{1}{0.5} microstepping = 800\frac{steps}{rotation}$$

$$1\frac{rev}{\pi * wheel\ diameter} * 800\frac{steps}{rev} = \frac{800}{\pi * diameter} * \frac{Steps}{meter}$$

$\frac{1}{\frac{800}{\pi * diameter} * \frac{Steps}{meter}}$ Period of steps sent to motor

Michael Jones - A00916729
Jonas Menge - A00830579

Appendix C - CAD Drawings

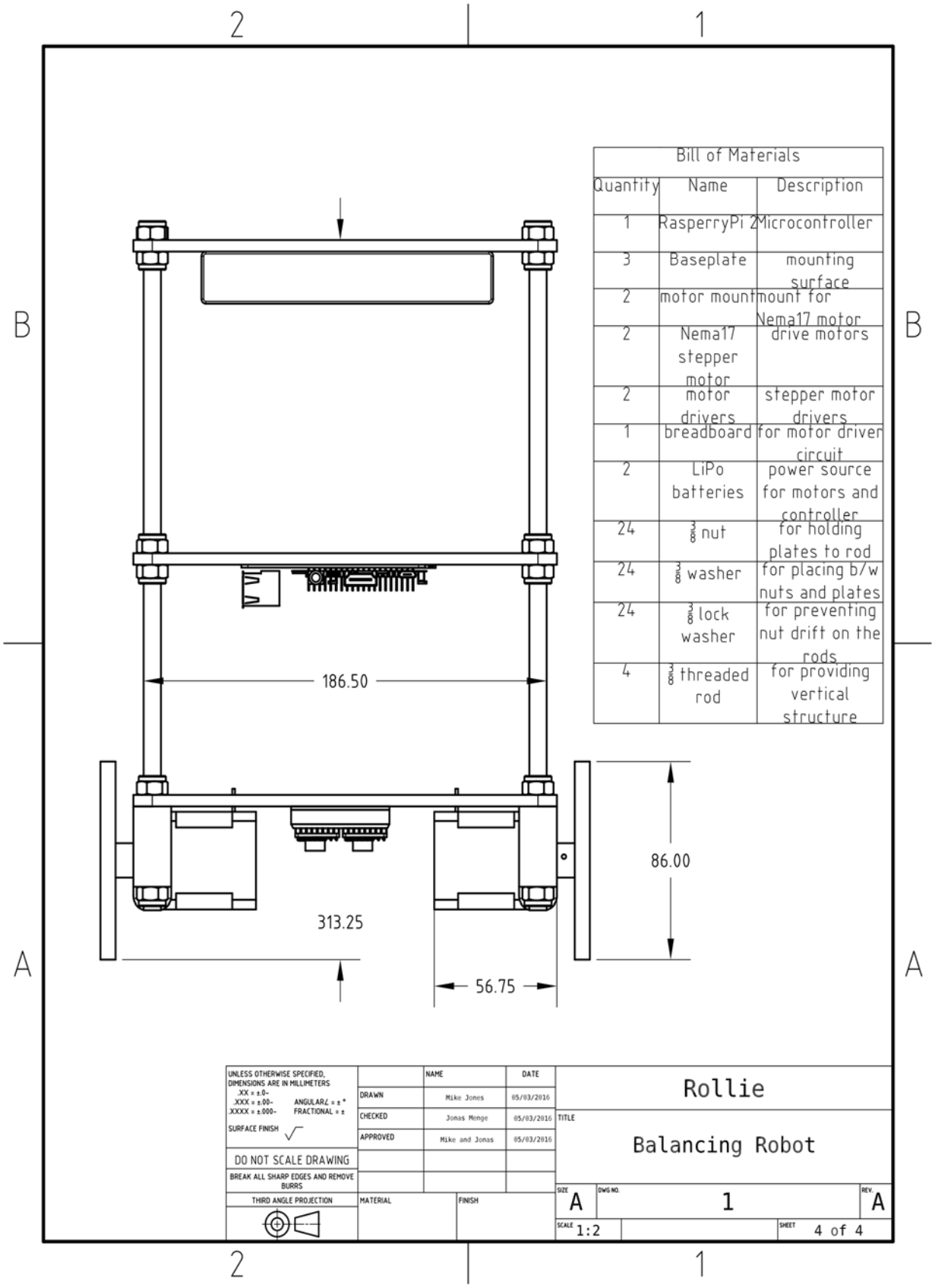

Figure 2 - Rollie Assembly with BOM

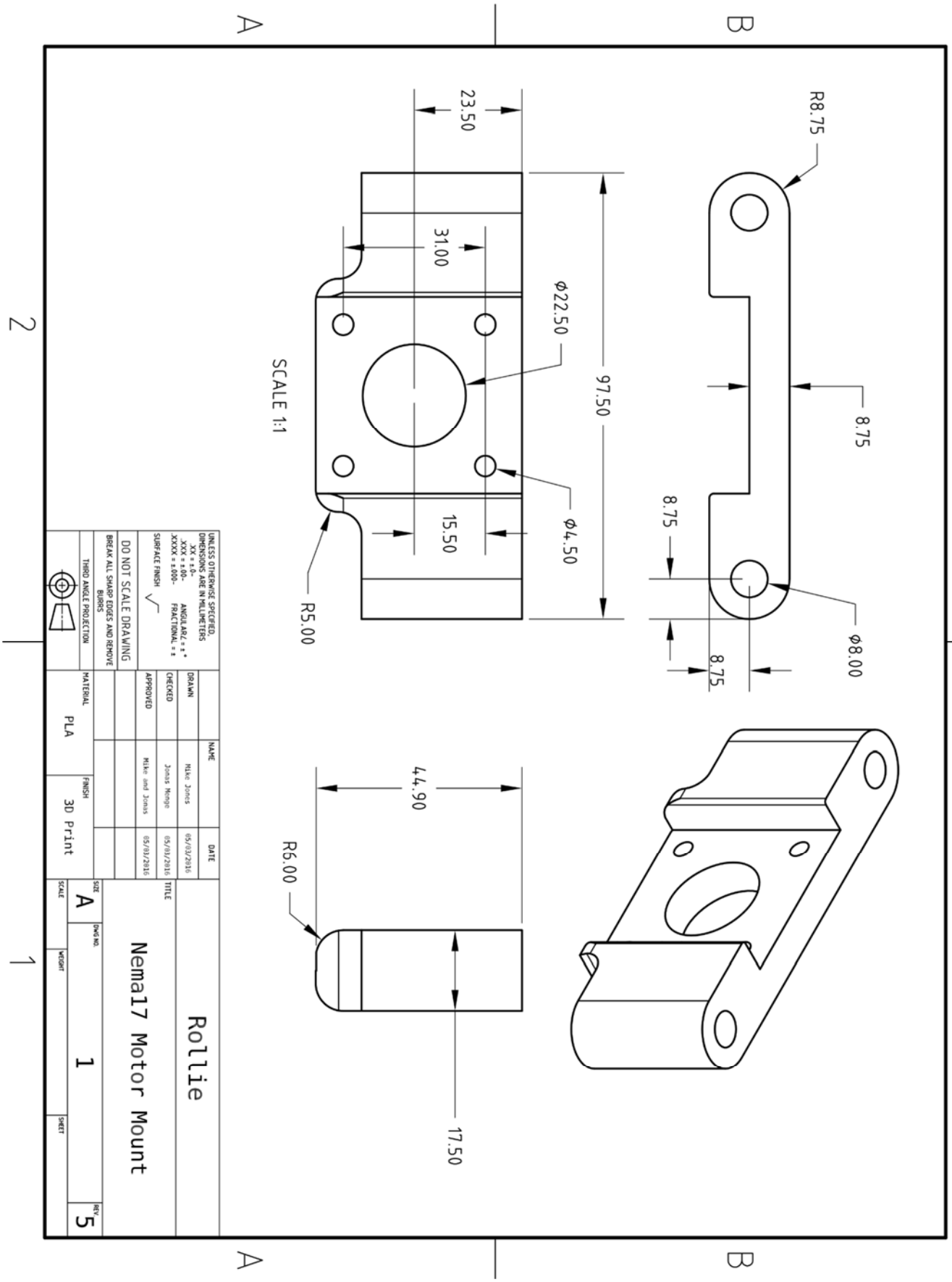

Figure 3 - Motor mount drawing

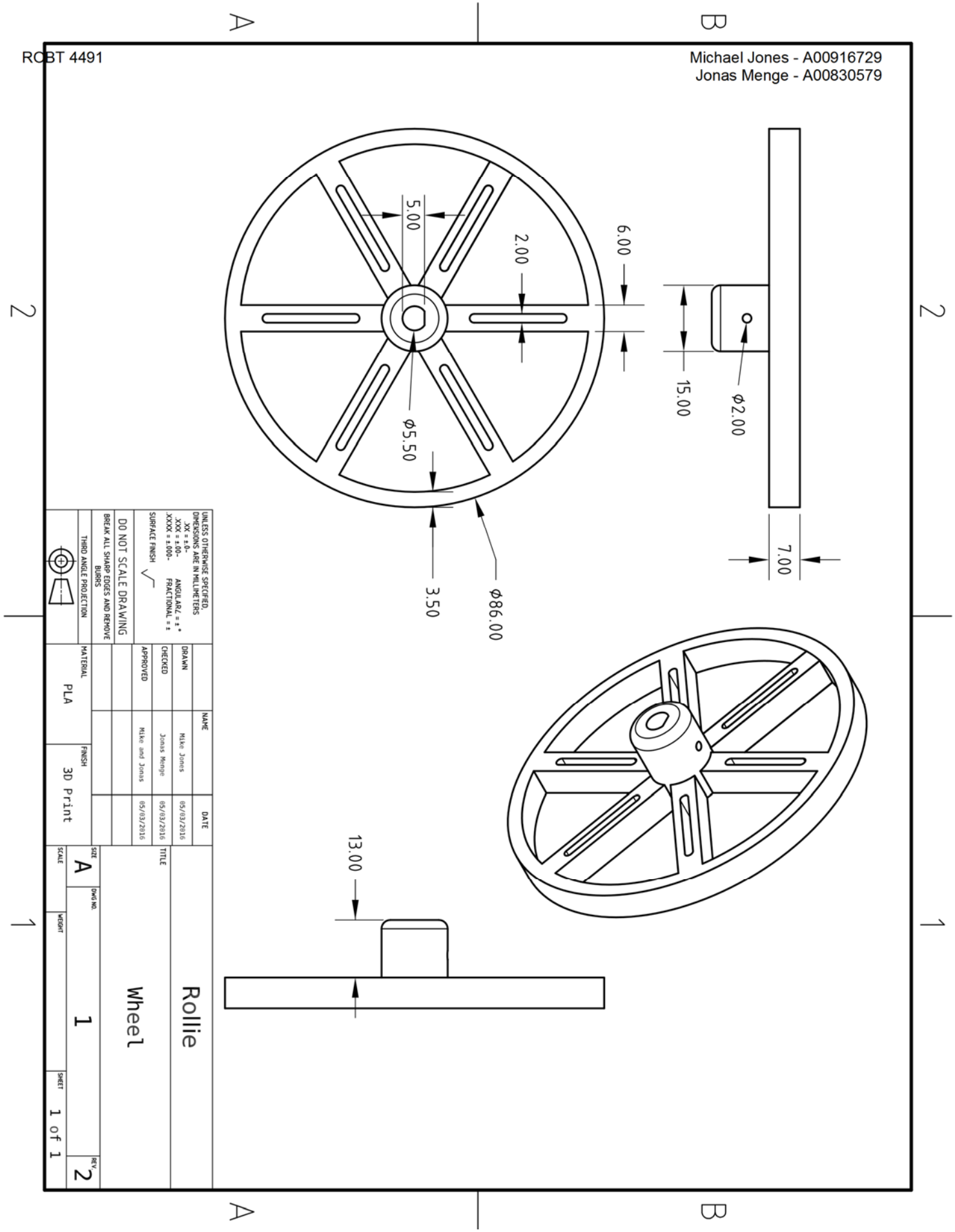

Figure 4 - Wheel drawing

Michael Jones - A00916729
Jonas Menge - A00830579

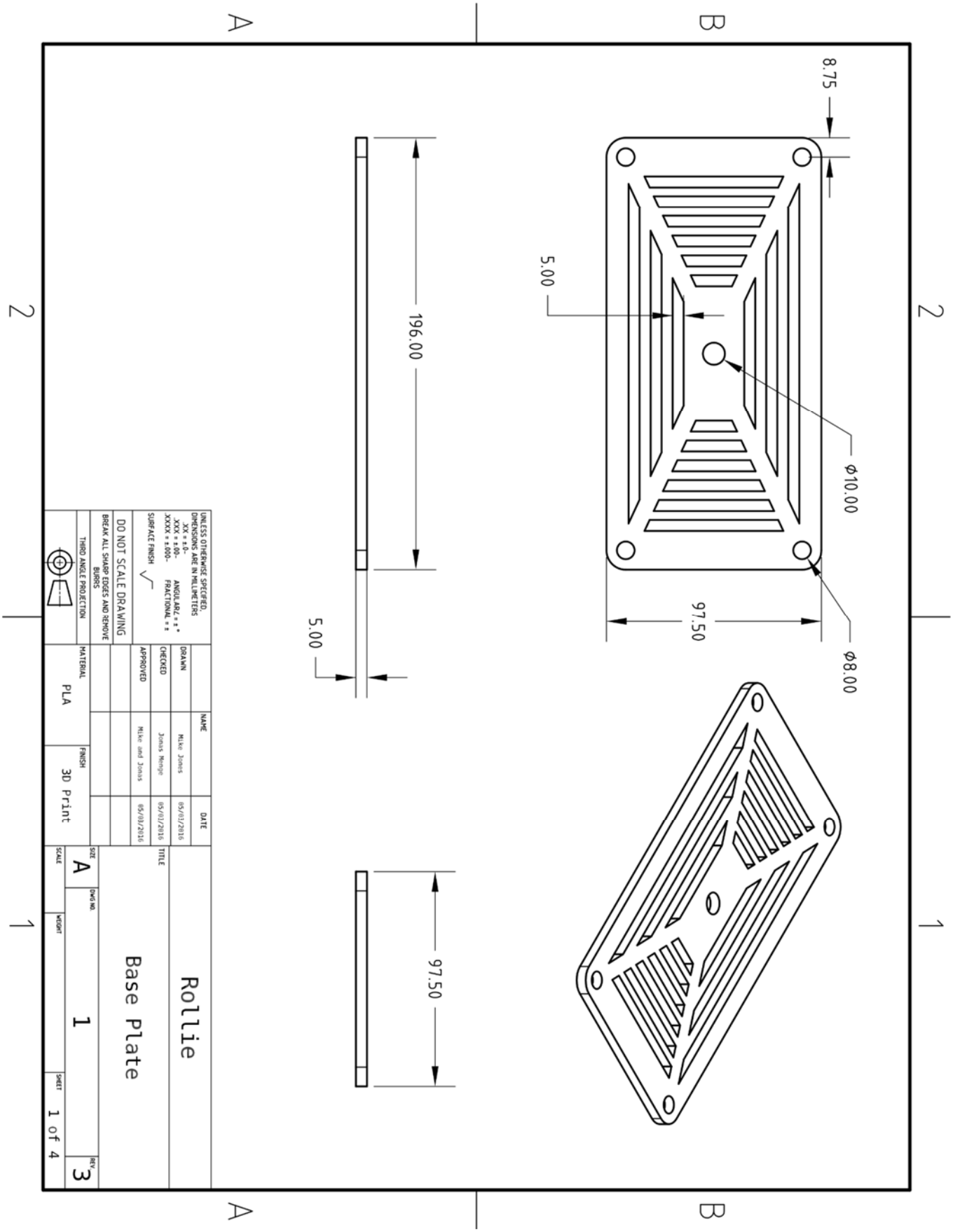

Figure 5 - Base plate drawing

Michael Jones - A00916729
Jonas Menge - A00830579

Appendix D – Schematics

Schematic diagram

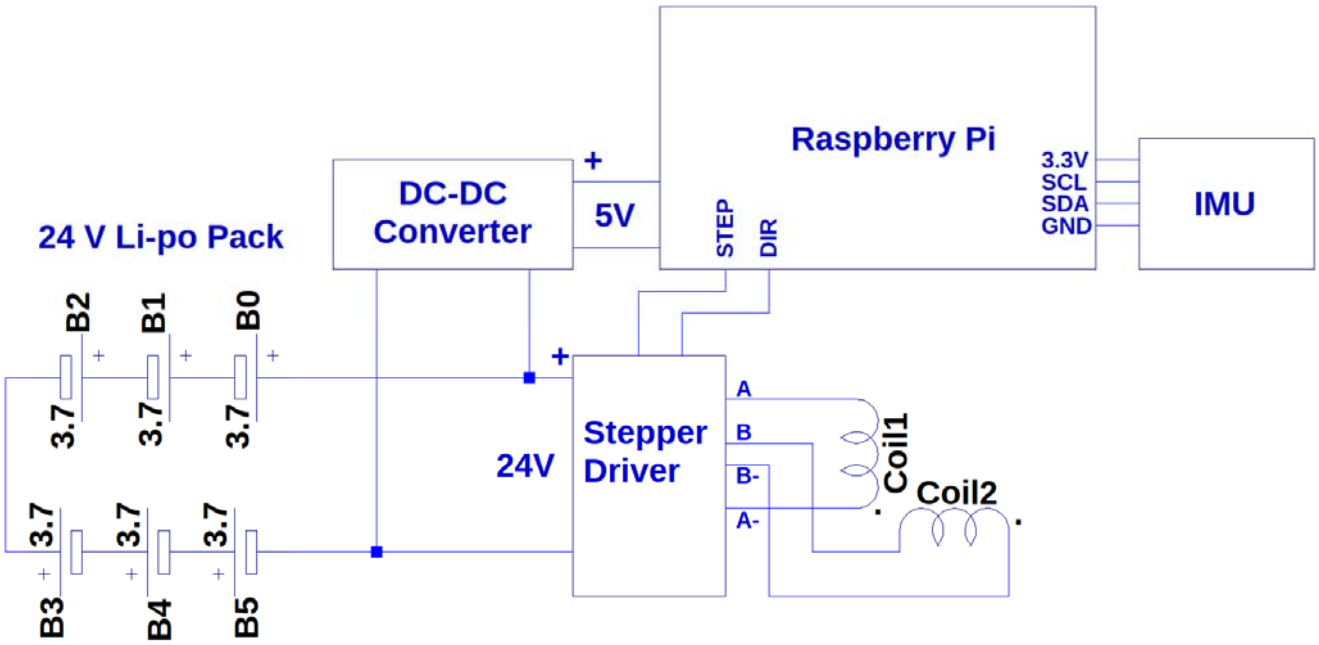

Figure 6 - System Schematic diagram

Michael Jones - A00916729
Jonas Menge - A00830579

Appendix E - Program Flowcharts & Block Diagrams

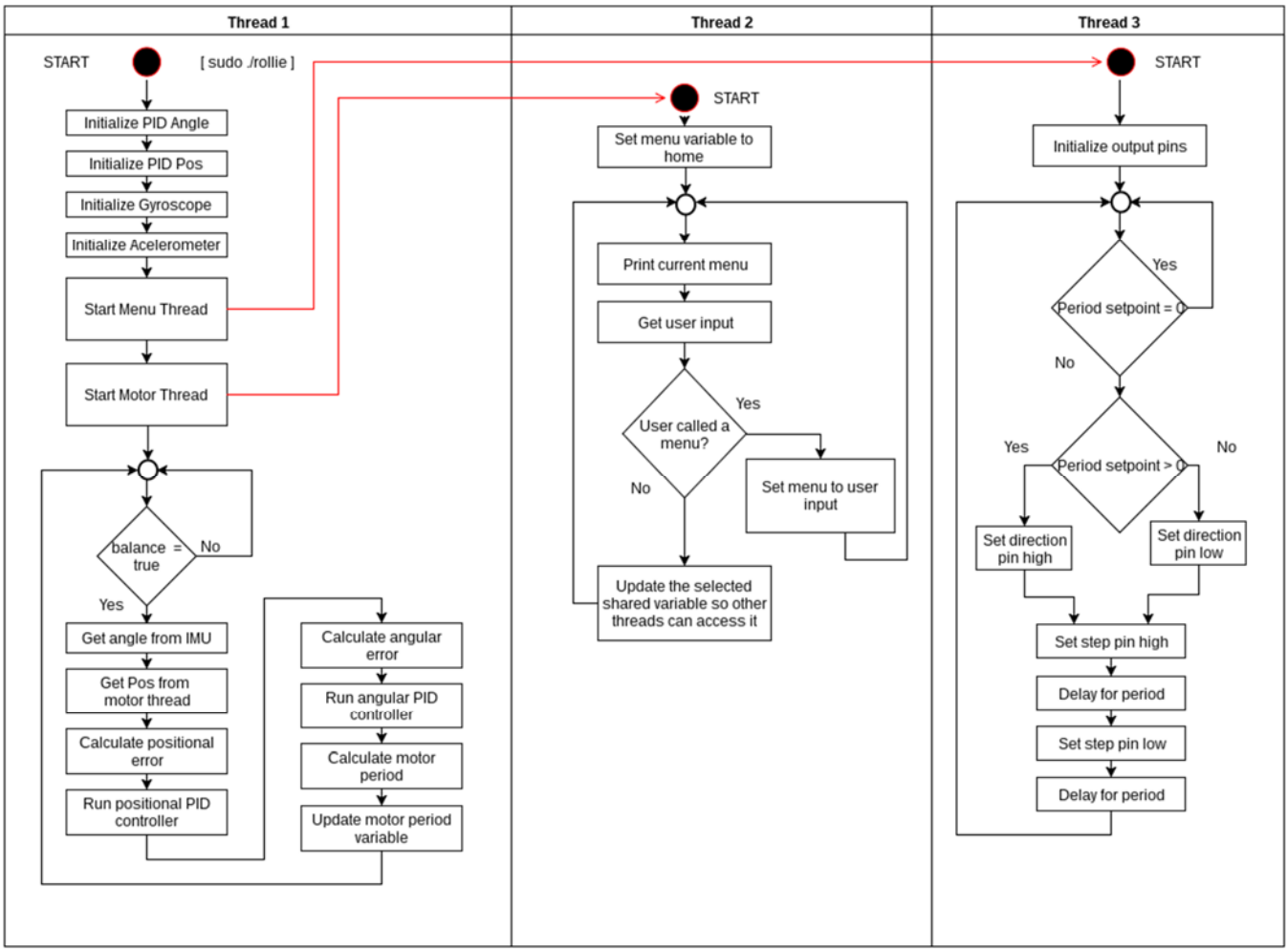

Figure 7 - System Flowchart

Michael Jones - A00916729
Jonas Menge - A00830579

Appendix F - Program Source Code

rollie.cpp

```
/************************************************************
*
*   rollie.cpp - Self balencing robot project
*
*   Authours: Jonas Menge and Michael Jones
*
*   Usage: sudo ./rollie
*
*
************************************************************/
#include "imu.h"        // Inertial measurment unit for getting angles
#include "pid.h"        // PID Controller
#include "stepper.h"    // Stepper motor controller

#include <stdio.h>
#include <wiringPi.h>   // For interfacing whith the Raspberry Pi hardware
#include <thread>       // To run concurrent pocesses
#include <math.h>

#define DEADBAND 0.0    // Absolute amount of angle error that is exeptable

// function prototypes
void loop(pid_filter_t *pidAngle, pid_filter_t *pidPos, int devAccel, int devGyro, stepper *stepper);


int main()
{
   // Angle PID controller setup
   pid_filter_t pidAngle;
   pid_init(&pidAngle);

   // Position PID controller setup
   pid_filter_t pidPos;
   pid_init(&pidPos);

   // Set the pid frequency
   pid_set_frequency(&pidPos, 100);
   pid_set_frequency(&pidAngle, 100);

   // Set the maximum positinal PID limit
   pid_set_integral_limit(&pidPos,1428571.0); // 10 degrees max

   // Preset the integral
   pid_set_integral(&pidPos, 857142.8); // set to 6 degrees

   // Tune the PID controller
   pid_set_gains(&pidAngle, 0.299, 0.0000, 0.000030);
   pid_set_gains(&pidPos, 0.0, 0.00082, 0.00005);

   // Start Stepper Motor Thread
   stepper stepper;                 // Create stepper struture
   setSpeed(0.0, &stepper.period);     // Preset the period to 0.0

   std::thread t_stepper;              // Create the stepper thead
   t_stepper = std::thread(stepperControl, &stepper);


   // Balencing loop
   loop(&pidAngle, &pidPos, devAccel, devGyro, &stepper);


   return 0;
}


void loop(pid_filter_t *pidAngle, pid_filter_t *pidPos, int devAccel, int devGyro, struct stepper *stepper)
{
   float errorPos = 0.0;
```

Michael Jones - A00916729
Jonas Menge - A00830579

```
    float setpointPos = 0.0;
    float pitch = 0.0;
    float errorAngle = 0.0;
    float setpointAngle = 0.0;
    float pidOutput = 0.0;
    float p0,p1 = 0.0;

    while (1){
      // Simple avreraging
      p0 = pitch;
      getAngle(&pitch,devAccel,devGyro);
      p1 = pitch;
      getAngle(&pitch,devAccel,devGyro);
      pitch = (pitch + p0 + p1) / 3.0;

      // Limit positinal range to prevent interator from overflowing too quikly
      if (500 < stepper->count)
      {
        stepper->count = 500;
      } else if (-500 > stepper->count) {
        stepper->count = -500;
      }


      /////////// PID Controller //////////

      // Error = Positional setpoint - actual position
      errorPos = setpointPos - stepper->count;
      setpointAngle = pid_process(pidPos, errorPos);
      // Outputs the setpint angle for the next PID

      // Errorr = Anguar setpoint - acutal angle
      errorAngle = setpointAngle - pitch;
      pidOutput = pid_process(pidAngle, errorAngle);
      // Outputs the motor velocity in m/s

      // Calculate and set the motor speed
      setSpeed(pidOutput, &stepper->period);
      //printf("\r Comp Angle: %0.2f", pitch);
      printf("\rerPos = %f, spAngle = %f, ActualA = %f, erAngle = %f, PID = %f",errorPos, setpointAngle, pitch, errorAngle, pidOutput);

    }
}
```

Michael Jones - A00916729
Jonas Menge - A00830579

stepper.cpp

```
/*  stepper.cpp
*  Written by Jonas Menge and Michael Jones
*
*  Accepts the period between steps
*
*  Returns curren step count
*/

#include "stepper.h"
#define RAMP 0.02


const int motor1Step = 17;
const int motor2Step = 27;
const int motor1Dir = 18;
const int motor2Dir = 22;


void stepperControl(stepper *stepper){
   motorSetup();

   while(1){
      if (stepper->period != 0.0){
         // Do math with the rate to scail it and set dir
         //printf("period = %f",*period);
         if (stepper->period > 0){
            digitalWrite(motor1Dir, HIGH);
            digitalWrite(motor2Dir, HIGH);
            stepper->count--;
         } else {
            digitalWrite(motor1Dir, LOW);
            digitalWrite(motor2Dir, LOW);
            stepper->count++;
         }

         // Do the step
         digitalWrite(motor1Step, HIGH);
         digitalWrite(motor2Step, HIGH);
         wait(&stepper->period);
         digitalWrite(motor1Step, LOW);
         digitalWrite(motor2Step, LOW);
         wait(&stepper->period);

      } else {
         delay(5);
      }
   }
}


// Initialize output pins
void motorSetup(){
   printf("starting motor thread");
   wiringPiSetupGpio(); // Initialize wiringPi -- using Broadcom pin numbers

   pinMode(motor1Step, OUTPUT);
   pinMode(motor2Step, OUTPUT);

   pinMode(motor1Dir, OUTPUT);
   pinMode(motor2Dir, OUTPUT);


}


// Alow updating the time while waiting
void wait(float *time){
   //printf("wait: %f",*time);
   for (float i = -0.01; i <= abs(*time/2.0); i = i + 10.0) // check how long we need to wait
   {
      delayMicroseconds(10);
```

Michael Jones - A00916729
Jonas Menge - A00830579

```
    }
}

// Calculate the new motor speed and store it
void setSpeed(float velocity, float *pulseTimePtr) {
    *pulseTimePtr = 1000000*(1.0 / (MOTOR_STEPS / (MICRO_STEPS * DIA * PI))) / velocity;
    //printf("Period: %f, velocity: %f\n", *pulseTimePtr,velocity);
}
```

Michael Jones - A00916729
Jonas Menge - A00830579

stepper.h

```
#ifndef STEPPER_H_
#define STEPPER_H_

#include <stdio.h>
#include <wiringPi.h>
#include <math.h>
#include <stdlib.h>

//constants
#define MOTOR_STEPS 400.0   // Full steps per rotation of the motor
#define MICRO_STEPS 0.5     // Micro steps (half steps = 0.5)
#define DIA 0.085           // Daiameter in m

#define PI  3.14159265359   // Its pi day... need at least 11 decimals


// Structs
struct stepper {
  float period;
  long int count;
};

// function prototypes
void motorSetup();      // Initialize motor pins
void wait(float *time);  // Wait loop
void stepperControl(struct stepper *stepper);
void setSpeed(float velocity, float *pulseTimePtr);

#endif
```

Michael Jones - A00916729
Jonas Menge - A00830579

pid.h

```
#ifndef PID_H_
#define PID_H_

/** Instance of a PID controller.
 *
 * @note This structure is only public to be able to do static allocation of it.
 * Do not access its fields directly.
 */
typedef struct {
    float kp;
    float ki;
    float kd;
    float integrator;
    float previous_value;
    float integrator_limit;
    float frequency;
} pid_filter_t;

/** Initializes a PID controller. */
void pid_init(pid_filter_t *pid);

/** Sets the gains of the given PID. */
void pid_set_gains(pid_filter_t *pid, float kp, float ki, float kd);

/** Returns the proportional gains of the controller. */
void pid_get_gains(const pid_filter_t *pid, float *kp, float *ki, float *kd);

/** Returns the limit of the PID integrator. */
float pid_get_integral_limit(const pid_filter_t *pid);

/** Returns the value of the PID integrator. */
float pid_get_integral(const pid_filter_t *pid);

/** Process one step if the PID algorithm. */
float pid_process(pid_filter_t *pid, float value);

/** Sets a maximum value for the PID integrator. */
void pid_set_integral_limit(pid_filter_t *pid, float max);

/** Resets the PID integrator to zero. */
void pid_reset_integral(pid_filter_t *pid);

/** Resets the PID integrator to zero. */
void pid_set_integral(pid_filter_t *pid, float value);

/** Sets the PID frequency for gain compensation. */
void pid_set_frequency(pid_filter_t *pid, float frequency);

/** Gets the PID frequency for gain compensation. */
float pid_get_frequency(const pid_filter_t *pid);

#endif
```

Michael Jones - A00916729
Jonas Menge - A00830579

pid.cpp

```
#include <stdlib.h>
#include <math.h>
#include "pid.h"


void pid_init(pid_filter_t *pid)
{
   pid_set_gains(pid, 1., 0., 0.);
   pid->integrator = 0.;
   pid->previous_value = 0.;
   pid->integrator_limit = INFINITY;
   pid->frequency = 1.;
}

void pid_set_gains(pid_filter_t *pid, float kp, float ki, float kd)
{
   pid->kp = kp;
   pid->ki = ki;
   pid->kd = kd;
}

void pid_get_gains(const pid_filter_t *pid, float *kp, float *ki, float *kd)
{
   *kp = pid->kp;
   *ki = pid->ki;
   *kd = pid->kd;
}

float pid_get_integral_limit(const pid_filter_t *pid)
{
   return pid->integrator_limit;
}

float pid_get_integral(const pid_filter_t *pid)
{
   return pid->integrator;
}

float pid_process(pid_filter_t *pid, float value)
{
   float output;
   pid->integrator += value;

   if (pid->integrator > pid->integrator_limit) {
      pid->integrator = pid->integrator_limit;
   } else if (pid->integrator < -pid->integrator_limit) {
      pid->integrator = -pid->integrator_limit;
   }

   output  = pid->kp * value;
   output += pid->ki * pid->integrator / pid->frequency;
   output += pid->kd * (value - pid->previous_value) * pid->frequency;

   pid->previous_value = value;
   return output;
}

void pid_set_integral_limit(pid_filter_t *pid, float max)
{
   pid->integrator_limit = max;
}

void pid_reset_integral(pid_filter_t *pid)
{
   pid->integrator = 0.;
}

// Added By Jonas
void pid_set_integral(pid_filter_t *pid, float value)
{
   pid->integrator = value;
```

Michael Jones - A00916729
Jonas Menge - A00830579

```
}

void pid_set_frequency(pid_filter_t *pid, float frequency)
{
   pid->frequency = frequency;
}

float pid_get_frequency(const pid_filter_t *pid)
{
   return pid->frequency;
}
```

Michael Jones - A00916729
Jonas Menge - A00830579

imu.h

```
#ifndef IMU_H_
#define IMU_H_


// libraries and header files
#include <iostream>
#include <errno.h>
#include <wiringPi.h>
#include "stdio.h"
#include <math.h>
#include <wiringPiI2C.h>
#include <stdint.h>
#include <stdlib.h>



//prototypes
void accPitch(float *,int);
void gyroPitch(float *,int);
int accConfig(void);
void getAngle(float *,int, int);
int gyroConfig(void);



// adxl345 registers

#define ADXL345_REG_DEVID            (0x00)    // Device ID
#define ADXL345_REG_THRESH_TAP       (0x1D)    // Tap threshold
#define ADXL345_REG_OFSX             (0x1E)    // X-axis offset
#define ADXL345_REG_OFSY             (0x1F)    // Y-axis offset
#define ADXL345_REG_OFSZ             (0x20)    // Z-axis offset
#define ADXL345_REG_DUR              (0x21)    // Tap duration
#define ADXL345_REG_LATENT           (0x22)    // Tap latency
#define ADXL345_REG_WINDOW           (0x23)    // Tap window
#define ADXL345_REG_THRESH_ACT       (0x24)    // Activity threshold
#define ADXL345_REG_THRESH_INACT     (0x25)    // Inactivity threshold
#define ADXL345_REG_TIME_INACT       (0x26)    // Inactivity time
#define ADXL345_REG_ACT_INACT_CTL    (0x27)    // Axis enable control for activity and inactivity detection
#define ADXL345_REG_THRESH_FF        (0x28)    // Free-fall threshold
#define ADXL345_REG_TIME_FF          (0x29)    // Free-fall time
#define ADXL345_REG_TAP_AXES         (0x2A)    // Axis control for single/double tap
#define ADXL345_REG_ACT_TAP_STATUS   (0x2B)    // Source for single/double tap
#define ADXL345_REG_BW_RATE          (0x2C)    // Data rate and power mode control
#define ADXL345_REG_POWER_CTL        (0x2D)    // Power-saving features control
#define ADXL345_REG_INT_ENABLE       (0x2E)    // Interrupt enable control
#define ADXL345_REG_INT_MAP          (0x2F)    // Interrupt mapping control
#define ADXL345_REG_INT_SOURCE       (0x30)    // Source of interrupts
#define ADXL345_REG_DATA_FORMAT      (0x31)    // Data format control
#define ADXL345_REG_DATAX0           (0x32)    // X-axis data 0
#define ADXL345_REG_DATAX1           (0x33)    // X-axis data 1
#define ADXL345_REG_DATAY0           (0x34)    // Y-axis data 0
#define ADXL345_REG_DATAY1           (0x35)    // Y-axis data 1
#define ADXL345_REG_DATAZ0           (0x36)    // Z-axis data 0
#define ADXL345_REG_DATAZ1           (0x37)    // Z-axis data 1
#define ADXL345_REG_FIFO_CTL         (0x38)    // FIFO control
#define ADXL345_REG_FIFO_STATUS      (0x39)    // FIFO status

#define WHO_AM_I_REG    0x00
#define SMPLRT_DIV_REG  0x15
#define DLPF_FS_REG     0x16
#define INT_CFG_REG     0x17
#define INT_STATUS      0x1A
#define TEMP_OUT_H_REG  0x1B
#define TEMP_OUT_L_REG  0x1C
#define GYRO_XOUT_H_REG 0x1D
#define GYRO_XOUT_L_REG 0x1E
#define GYRO_YOUT_H_REG 0x1F
#define GYRO_YOUT_L_REG 0x20
#define GYRO_ZOUT_H_REG 0x21
#define GYRO_ZOUT_L_REG 0x22
```

Michael Jones - A00916729
Jonas Menge - A00830579

```
#define PWR_MGM_REG     0x3E

//----------------------------
// Low Pass Filter Bandwidths
//----------------------------
#define LPFBW_256HZ 0x00
#define LPFBW_188HZ 0x01
#define LPFBW_98HZ  0x02
#define LPFBW_42HZ  0x03
#define LPFBW_20HZ  0x04
#define LPFBW_10HZ  0x05
#define LPFBW_5HZ   0x06

//-----------
// Offsets
//-----------
//short int TEMP_OUT_OFFSET = 0;
//short int GYRO_XOUT_OFFSET = 0;
//short int GYRO_YOUT_OFFSET = 0;
//short int GYRO_ZOUT_OFFSET = 0;
#endif
```

Michael Jones - A00916729
Jonas Menge - A00830579

imu.cpp

```
//  Authors Michael and Jonas
//  Description, Program to return an accurate angle from the IMU.
//
//  Outputs angles in degrees


#include "imu.h"

//Global constants
#define PI                          3.14159265359   // Its pi day... need at least 11 decimals
//#define ACCELEROMETER_SENSITIVITY   2047.97       //+-16 g full scale range of accelerometer
#define ACCELEROMETER_SENSITIVITY   1000            //+-16 g full scale range of accelerometer
#define GYROSCOPE_SENSITIVITY       1/14.375        //LSB/(deg/s)
#define dt                          0.01            //sampling rate 0.01 = 10ms, need to add
function input for this
#define UPPER_ACC_FORCE             4095.94         // max force 2g*2047.97
#define PIDIV                    0.31831                 //1/PI because multiplication is
faster

// function to configure the gyroscope
int gyroConfig(){

  int devGyro = wiringPiI2CSetup(0x68);
  wiringPiI2CWriteReg8(devGyro, 0x15, 0x09);
  wiringPiI2CWriteReg8(devGyro, 0x16, 0x1a);
  wiringPiI2CWriteReg8(devGyro, 0x17, 0x01);

  return(devGyro);
}

// function to configure the accelerometer
int accConfig(){

  // setup i2c
  int devAccel = wiringPiI2CSetup(0x53);

  // configure ADXL345 registers
  wiringPiI2CWriteReg8(devAccel, ADXL345_REG_POWER_CTL, 0x08);
  wiringPiI2CWriteReg8(devAccel, ADXL345_REG_DATA_FORMAT, 0x0B);
  wiringPiI2CWriteReg8(devAccel, ADXL345_REG_INT_ENABLE, 0x80);  //
  wiringPiI2CWriteReg8(devAccel, ADXL345_REG_FIFO_CTL, 0x00);    //bypass fifo

  return(devAccel);
}

// accelerometer pitch
void accPitch(float *aPitch, int devAccel){

  int forceMagnitudeApprox;
  float Xa,Ya,Za;
  short X,Y,Z;
  int flag;

  do{

    do{
      flag = wiringPiI2CReadReg8(devAccel,(ADXL345_REG_INT_SOURCE));
      flag = flag & 0x80;
      //   printf("flag = %x\n",flag);
    }while(flag != 0x80);

    // grab raw data from accelerometer
```

Michael Jones - A00916729
Jonas Menge - A00830579

```
    X = wiringPiI2CReadReg8(devAccel,(ADXL345_REG_DATAX1));
    X = (X) << 8;
    X = X | wiringPiI2CReadReg8(devAccel,(ADXL345_REG_DATAX0));

    Z = wiringPiI2CReadReg8(devAccel,(ADXL345_REG_DATAZ1));
    Z = (Z) << 8;
    Z = Z | wiringPiI2CReadReg8(devAccel,(ADXL345_REG_DATAZ0));

    Y = wiringPiI2CReadReg8(devAccel,(ADXL345_REG_DATAY1));
    Y = (Y) << 8;
    Y = Y | wiringPiI2CReadReg8(devAccel,(ADXL345_REG_DATAY0));


    forceMagnitudeApprox = abs(X)+abs(Y)+abs(Z);

    // scale the output for calculating the angle
    Xa = (float)X * 0.0039;
    Ya = (float)Y * 0.0039;
    Za = (float)Z * 0.0039;

    //calc angle
    *aPitch = (atan2(Xa,sqrt(Ya*Ya+Za*Za)) * 180.0)*PIDIV; /// PI;

    //printf("pitch = %lf",*aPitch);

  } while (forceMagnitudeApprox > ACCELEROMETER_SENSITIVITY && forceMagnitudeApprox < 32768);

}

void gyroPitch(float *gyrPitch, int devGyro)
{
  float Yg;
  short Y;
  int flag;

  do{
    flag =wiringPiI2CReadReg8(devGyro,(INT_STATUS));
    flag= flag & 0x01;
    }while(flag != 0x01);


  Y = wiringPiI2CReadReg8(devGyro,(GYRO_YOUT_H_REG));
  Y = (Y) << 8;
  Y = Y | wiringPiI2CReadReg8(devGyro, GYRO_YOUT_L_REG);

  Yg = (float)Y;

  // Integrate the gyroscope data -> int(angularSpeed) = angle
  *gyrPitch = (Yg/* * GYROSCOPE_SENSITIVITY*/); // Angle around the X-axis
}

// Get a new angle in degrees
// Accepts the old pitch (for intigration) and device id of the accel and gyro
void getAngle(float *pitch, int devAccel, int devGyro)
{
       float aPitch,gyrPitch;

       //timing start timer
       unsigned int time = micros();

       accPitch(&aPitch, devAccel);      // get pitch from the accelerometer

       if(abs(aPitch) > 25.0)
```

Michael Jones - A00916729
Jonas Menge - A00830579

```
        {
                aPitch = *pitch;
        }

        gyroPitch(&gyrPitch, devGyro);   //get pitch from Gyro

        //end timer
        time = micros() - time;

        //convert to seconds from micro
        //time = time * 0.000001;
        //printf("time: %f",time);
        *pitch = (*pitch + (gyrPitch * GYROSCOPE_SENSITIVITY *dt /*((float)time * 0.000001)*/))
* 0.99 + aPitch * 0.01;

}
```

Michael Jones - A00916729
Jonas Menge - A00830579

Makefile

```
#### PROJECT SETTINGS ####
# The name of the executable to be created
BIN_NAME := rollie
# Compiler used
CXX ?= g++
# Extension of source files used in the project
SRC_EXT = cpp
# Path to the source directory, relative to the makefile
SRC_PATH = .
# Space-separated pkg-config libraries used by this project
LIBS =
# General compiler flags
COMPILE_FLAGS = -std=c++11 -pthread -Wall -Wextra -g -fmax-errors=5
# Additional release-specific flags
RCOMPILE_FLAGS = -D NDEBUG
# Additional debug-specific flags
DCOMPILE_FLAGS = -D DEBUG
# Add additional include paths
INCLUDES = -I $(SRC_PATH)/ -l wiringPi
# General linker settings
LINK_FLAGS = -l wiringPi -l pthread
# Additional release-specific linker settings
RLINK_FLAGS =
# Additional debug-specific linker settings
DLINK_FLAGS =
# Destination directory, like a jail or mounted system
DESTDIR = ~
# Install path (bin/ is appended automatically)
#INSTALL_PREFIX = usr/local
INSTALL_PREFIX =
#### END PROJECT SETTINGS ####

# Generally should not need to edit below this line

# Obtains the OS type, either 'Darwin' (OS X) or 'Linux'
UNAME_S:=$(shell uname -s)

# Function used to check variables. Use on the command line:
# make print-VARNAME
# Useful for debugging and adding features
print-%: ; @echo $*=$($*)

# Shell used in this makefile
# bash is used for 'echo -en'
SHELL = /bin/bash
# Clear built-in rules
.SUFFIXES:
# Programs for installation
INSTALL = install
INSTALL_PROGRAM = $(INSTALL)
INSTALL_DATA = $(INSTALL) -m 644

# Append pkg-config specific libraries if need be
ifneq ($(LIBS),)
	COMPILE_FLAGS += $(shell pkg-config --cflags $(LIBS))
	LINK_FLAGS += $(shell pkg-config --libs $(LIBS))
endif

# Verbose option, to output compile and link commands
export V := false
export CMD_PREFIX := @
ifeq ($(V),true)
	CMD_PREFIX :=
endif

# Combine compiler and linker flags
release: export CXXFLAGS := $(CXXFLAGS) $(COMPILE_FLAGS) $(RCOMPILE_FLAGS)
release: export LDFLAGS := $(LDFLAGS) $(LINK_FLAGS) $(RLINK_FLAGS)
debug: export CXXFLAGS := $(CXXFLAGS) $(COMPILE_FLAGS) $(DCOMPILE_FLAGS)
debug: export LDFLAGS := $(LDFLAGS) $(LINK_FLAGS) $(DLINK_FLAGS)
```

Michael Jones - A00916729
Jonas Menge - A00830579

```
# Build and output paths
release: export BUILD_PATH := ../build/release
release: export BIN_PATH := ../bin/release
debug: export BUILD_PATH := ../build/debug
debug: export BIN_PATH := ../bin/debug
install: export BIN_PATH := ../bin/release

# Find all source files in the source directory, sorted by most
# recently modified
ifeq ($(UNAME_S),Darwin)
	SOURCES = $(shell find $(SRC_PATH)/ -name '*.$(SRC_EXT)' | sort -k 1nr | cut -f2-)
else
	SOURCES = $(shell find $(SRC_PATH)/ -name '*.$(SRC_EXT)' -printf '%T@\t%p\n' \
	                                                         | sort -k 1nr | cut -f2-)
endif

# fallback in case the above fails
rwildcard = $(foreach d, $(wildcard $1*), $(call rwildcard,$d/,$2) \
	                                                  $(filter $(subst *,%,$2), $d))
ifeq ($(SOURCES),)
	SOURCES := $(call rwildcard, $(SRC_PATH)/, *.$(SRC_EXT))
endif

# Set the object file names, with the source directory stripped
# from the path, and the build path prepended in its place
OBJECTS = $(SOURCES:$(SRC_PATH)/%.$(SRC_EXT)=$(BUILD_PATH)/%.o)
# Set the dependency files that will be used to add header dependencies
DEPS = $(OBJECTS:.o=.d)

# Macros for timing compilation
ifeq ($(UNAME_S),Darwin)
	CUR_TIME = awk 'BEGIN{srand(); print srand()}'
	TIME_FILE = $(dir $@).$(notdir $@)_time
	START_TIME = $(CUR_TIME) > $(TIME_FILE)
	END_TIME = read st < $(TIME_FILE) ; \
		$(RM) $(TIME_FILE) ; \
		st=$$((`$(CUR_TIME)` - $$st)) ; \
		echo $$st
else
	TIME_FILE = $(dir $@).$(notdir $@)_time
	START_TIME = date '+%s' > $(TIME_FILE)
	END_TIME = read st < $(TIME_FILE) ; \
		$(RM) $(TIME_FILE) ; \
		st=$$((`date '+%s'` - $$st - 86400)) ; \
		echo `date -u -d @$$st '+%H:%M:%S'`
endif

# Version macros
# Comment/remove this section to remove versioning
USE_VERSION := false
# If this isn't a git repo or the repo has no tags, git describe will return non-zero
ifeq ($(shell git describe > /dev/null 2>&1 ; echo $$?), 0)
	USE_VERSION := true
	VERSION := $(shell git describe --tags --long --dirty --always | \
		sed 's/v\([0-9]*\)\.\([0-9]*\)\.\([0-9]*\)-\?.*-\([0-9]*\)-\(.*\)/\1 \2 \3 \4 \5/g')
	VERSION_MAJOR := $(word 1, $(VERSION))
	VERSION_MINOR := $(word 2, $(VERSION))
	VERSION_PATCH := $(word 3, $(VERSION))
	VERSION_REVISION := $(word 4, $(VERSION))
	VERSION_HASH := $(word 5, $(VERSION))
	VERSION_STRING := \
		"$(VERSION_MAJOR).$(VERSION_MINOR).$(VERSION_PATCH).$(VERSION_REVISION)-$(VERSION_HASH)"
	override CXXFLAGS := $(CXXFLAGS) \
		-D VERSION_MAJOR=$(VERSION_MAJOR) \
		-D VERSION_MINOR=$(VERSION_MINOR) \
		-D VERSION_PATCH=$(VERSION_PATCH) \
		-D VERSION_REVISION=$(VERSION_REVISION) \
		-D VERSION_HASH=\"$(VERSION_HASH)\"
endif

# Standard, non-optimized release build
.PHONY: release
release: dirs
```

```
ifeq ($(USE_VERSION), true)
	@echo "Beginning release build v$(VERSION_STRING)"
else
	@echo "Beginning release build"
endif
	@$(START_TIME)
	@$(MAKE) all --no-print-directory
	@echo -n "Total build time: "
	@$(END_TIME)

# Debug build for gdb debugging
.PHONY: debug
debug: dirs
ifeq ($(USE_VERSION), true)
	@echo "Beginning debug build v$(VERSION_STRING)"
else
	@echo "Beginning debug build"
endif
	@$(START_TIME)
	@$(MAKE) all --no-print-directory
	@echo -n "Total build time: "
	@$(END_TIME)

# Create the directories used in the build
.PHONY: dirs
dirs:
	@echo "Creating directories"
	@mkdir -p $(dir $(OBJECTS))
	@mkdir -p $(BIN_PATH)

# Installs to the set path
.PHONY: install
install:
	@echo "Installing to $(DESTDIR)$(INSTALL_PREFIX)/bin"
	@$(INSTALL_PROGRAM) $(BIN_PATH)/$(BIN_NAME) $(DESTDIR)$(INSTALL_PREFIX)/bin

# Uninstalls the program
.PHONY: uninstall
uninstall:
	@echo "Removing $(DESTDIR)$(INSTALL_PREFIX)/bin/$(BIN_NAME)"
	@$(RM) $(DESTDIR)$(INSTALL_PREFIX)/bin/$(BIN_NAME)

# Removes all build files
.PHONY: clean
clean:
	@echo "Deleting $(BIN_NAME) symlink"
	@$(RM) $(BIN_NAME)
	@echo "Deleting directories"
	@$(RM) -r build
	@$(RM) -r bin

# Main rule, checks the executable and symlinks to the output
all: $(BIN_PATH)/$(BIN_NAME)
	@echo "Making symlink: $(BIN_NAME) -> $<"
	@$(RM) $(BIN_NAME)
	@ln -s $(BIN_PATH)/$(BIN_NAME) $(BIN_NAME)

# Link the executable
$(BIN_PATH)/$(BIN_NAME): $(OBJECTS)
	@echo "Linking: $@"
	@$(START_TIME)
	$(CMD_PREFIX)$(CXX) $(OBJECTS) $(LDFLAGS) -o $@
	@echo -en "\t Link time: "
	@$(END_TIME)

# Add dependency files, if they exist
-include $(DEPS)

# Source file rules
# After the first compilation they will be joined with the rules from the
# dependency files to provide header dependencies
$(BUILD_PATH)/%.o: $(SRC_PATH)/%.$(SRC_EXT)
	@echo "Compiling: $< -> $@"
```

Michael Jones - A00916729
Jonas Menge - A00830579

```
@$(START_TIME)
$(CMD_PREFIX)$(CXX) $(CXXFLAGS) $(INCLUDES) -MP -MMD -c $< -o $@
@echo -en "\t Compile time: "
@$(END_TIME)
```