University of Essex

CE301: Final report

A wheeled balancing robot

Student: Jonathan Tam - 0945206

Project Supervisor: John Woods

11

Contents

1. Abstract

Self-balancing robots are known to be more energy efficient than robots with more wheels. This concept has been implemented in commercial products such as the Segway Personal Transporters.

The aim of this project is to produce a two wheeled self-balancing robot, including a close loop PID controller to constantly adjust the angle of the robot relative to gravity. In addition the robot is able to self-adjust to inclined planes, via negative feedback from the wheel encoders.

Gravity is measured using an accelerometer, given that the robot is static. When the robot experiences other components of acceleration (i.e. when the robot is moving) the tilt angle provided by the accelerometer is incorrect. This is solved by the addition of a gyroscope, a sensor that measures angular rotation but drift is expected from this type of sensors.

2. Introduction

There are distinct advantages in a robot emulating the human ability to balance. [1] The main advantage is the ability to move and stop without toppling over. This concept can be applied onto a 2 wheeled robot, enabling it to utilise less ground surface area and has a zero turning radius. An example is the Segway currently on the market for personal and commercial use. [2]

Most robots/vehicles are fitted with 3 or more wheels for increased stability both longitudinally and laterally. This in turn increases the friction proportionally to the number of wheels present, thus it is ideal to have less wheels fitted to increase power efficiency. [2]

Ideally, a one wheeled robot should be the most power efficient, but balancing two axes dynamically is relatively complex task. It is better to start off with a two wheeled vehicle, to dynamically balance the longitudinal axis then, progress onto to balancing a robot on a ball which is both longitudinally and laterally unstable. [2]

There are many advantages for having two wheels instead of four. Navigation (Dead reckoning) is much simpler and more accurate, because the there is no turning radius and the wheels are less likely to slip. Secondly, as moving a heavy load requires a much smaller amount of energy to overcome friction and inertia, mainly because gravity is used to the vehicle's advantage. "Driving the wheels in the direction that the upper part of the robot is falling." [3]

Humans evolved to conserve energy and walk efficiently through shifting the centre of mass vertically. This is achieved by transferring the weight between the left and right legs simultaneously while walking. [4] The central of gravity is shifted forward when we walk, thus using gravity to our advantage. [2]

A two wheeled balancing robot is laterally unstable, but it can be dynamically stabilized using gyros and accelerometers, alongside an embedded controller.

Advantages:

- Greater stability on inclines as in [5]
- Less slip and friction
- More agile due to zero turn radius
- Simple navigation with dead reckoning
- Tolerance to impulsive forces
- Uses gravity to its advantage to move itself [6]

As the centre of gravity is shifted forward, gravitational pull draws the object forward and down, whereas accelerating a tilted object induces a drag force and up thrust. When the gravitational pull and up thrust are at equilibrium, the object no longer accelerates in the y direction. [2]

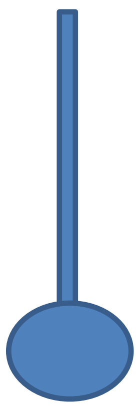


Figure 1.1: Stationary

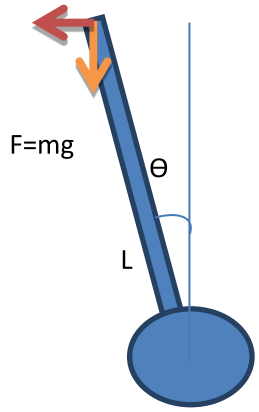


Figure 1.2: Shifted CG

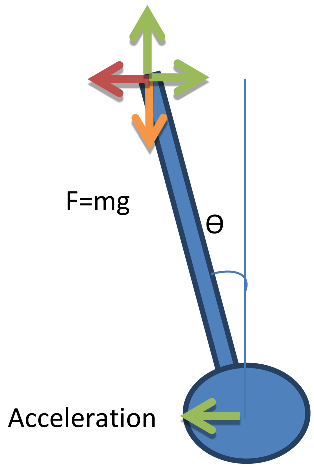


Figure 1.3: Forces Balanced

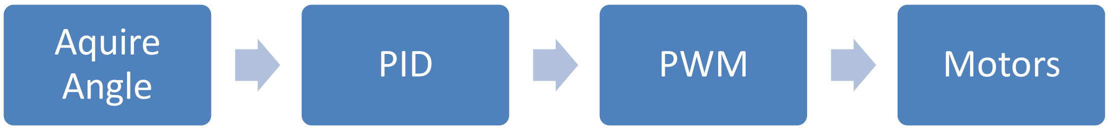
This is essentially an inverted pendulum.

2.1 Overview

This project is aimed to design and implement a two wheel self-balancing Robot. In order to achieve balance the angle with respect to the positive y axis is maintained to as small as possible assuming the centre of gravity is at the centre of the robot. To solve the classic inverted pendulum on a cart problem without solving it as a complex mathematical problem, a PID controller is used. The PWM output then drives the motor to make angle corrections.

Acquire the robot's current orientation with respect to the Y axis (Roll) involves 3 steps:-

1. Acquire accelerometer angle.
2. Acquire gyro angular acceleration (rate of change of angle).
3. Complement Sensor Fusion.



3. Project Goals

The ultimate goal of the project is to produce a robot that balances dynamically on two wheels, by using an embedded processor as a closed loop controller to keep the robot perpendicular to the floor. [2]

<u>Goal outline</u>

1. Balancing robot with a potentiometer as the tilt sensor.
2. Implement PID feedback control loop to **drive** motors.
3. Implement a wired remote for navigation.

Sensor upgrades

4. Upgrade tilt sensor to ultra-sonic sensor.
5. Upgrade tilt sensor to gyroscope.
6. Upgrade tilt sensor to accelerometer.
7. Gyroscope and accelerometer, Sensor fusion with Kalman/complementary filter.

Other upgrades

8. Build a structure/ frame for the robot
9. Remote control for robot using JavaScript and remote procedure call or zigbee transceivers.
10. Use of motor speed governors (quadrature encoders).

Further upgrades

11. Remote control using infrared receiver
12. Write Sequences/ waypoints for the robot to follow
13. Convert robot into a ball balancer

The **First goal** is to write a program in C/C++ to determine the tilt angle of the robot by reading an analogue value from a potentiometer that has an arm extending to the floor. This doesn't work on an inclined plane so gyroscopes and an accelerometer are going to be used to determine the tilt angle.

The **Second goal** is to implement a closed looped PID control loop to drive the motors. Using the tilt angle as the process value, the targeted value would be 0 when stationary and an offset can control the speed at which the robot travels.

The **Third goal** is to upgrade the robot to measure tilt with an ultrasonic sensor. The response would be much quicker with the ultrasonic as the detection method does not involve moving parts.

The **Fourth goal** is to upgrade the tilt measuring mechanism to measure the angular rate as the ultrasonic sensor fails to provide the correct tilt angle on an inclined slope.

The **Fifth and sixth goal** is to add an accelerometer to counter gyro drift.

The **Final goal** is to implement a remote to navigate the robot. If a Wi-Fi router is on board the targeted client could be any JavaScript enabled mobile device such as the iPhone. The mbed cookbook has instructions for running an HTTP server and creating JavaScript enabled webpages for RPC(remote procedure call). If the robot doesn't have any networking capabilities, then the best way is to build another embedded controller and interface it with the robot embedded controller with a pair of zigbee wireless transceivers.

4. Background Reading

4.1 Inertial Measurement Unit

IMU measures inertial movement with two sensors, a gyroscope and an accelerometer.

- Accelerometers are accurate over the long term, but do not provide a high enough resolution to give an accurate tilt angle.
- Gyros provide an accurate tilt angle but the value slowly drifts.

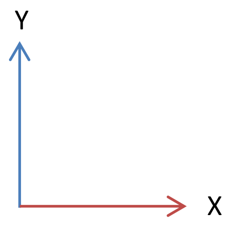
Fusing the sensor information from both gives an accurate tilt angle result over time. The gyro angles are good and stable for a short period of time, but they quickly drift and become inaccurate. The accelerometer gives good angles over a longer period of time, but in the short run they are noisy.

4.2 Gyroscopes and accelerometers

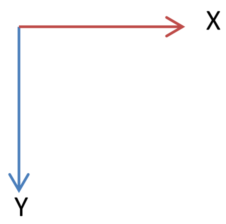
This project will use two electric motors, a gyro, an accelerometer and a microcontroller to balance a robot upright between two wheels.

Gyroscope measures angular rate (speed of rotation), integrating the angular rate sampled over time gives the tilt rotation in angle. Clockwise rotation gives a positive reading, and anti-clockwise rotation gives a negative reading. When it’s stationary it gives a zero reading.

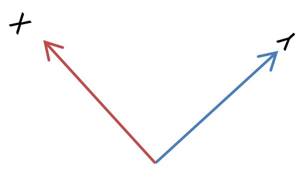
The accelerometer is measured in force per unit mass (a=F/m). It measures the force of gravity (*g*) [7]. Having two accelerometers placed perpendicularly to each other, the orientation can be calculated using trigonometry.



(Y = -1, X = 0)



(Y = 1, X = 0)



(Y = -0.5, X = -0.5)

4.3 Sensor fusion

There are several known methods that are useful for gyroscope and accelerometer sensor fusion.

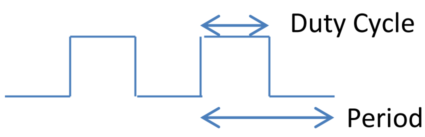
One example being the Kalman filter [7]. The Kalman filter fuses the information from both the gyroscope and the accelerometer to provide an accurate tilt angle without gyro drift. It is mathematically complex and is not so ideal on lower powered embedded controllers. On the other hand a much simpler complementary filter achieves the same result without using as much computing power.

4.4 DC motor drivers

Embedded systems usually do not come with an onboard motor driver to drive DC motors, as DC motors are known to draw a higher current than what the built in outputs can provide. If the targeted embedded controller is an Arduinos, the L298 Motor driver shield can be used, which provides a dual 2A H-bridge.

A PWM (Pulse Width Modulation) output from the embedded controller sends a frequency modulated square wave to the driver, turning the motor on and off at a high speed. This effectively controls the current going into the motor. Varying the duty cycle is more advantageous to varying the voltage, as the torque is not directly proportional to voltage.

Because electric motors generate electromagnetic fields to induce the turning force, it produces Counter-electromotive force. Which are spike currents in the reverse direction, so fast recovering diodes are required to protect the motor driver from the spike current? The inductive property also acts a filter to smooth out the PWM signal into DC voltage.



4.5 Speed sensing

4.5.1 Encoder

Speed sensing is usually done with rotary quadrant encoders or a tachometer, although there are other methods such as back EMF sensing.

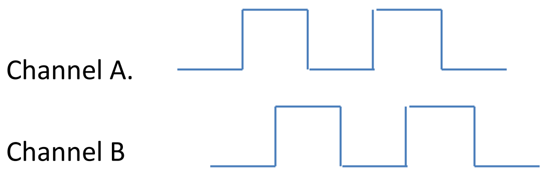
Tachometer increments a counter built in to the embedded controller, by calculating the displacement per count:

$$\textbf{Displacement per count} = \boldsymbol{\pi} \times \left(\frac{\textbf{Wheel diameter}}{\textbf{Counts per revolution}}\right)$$

Differentiating the displacement over time, gives the velocity.

$$\mathbf{V} = \frac{\Delta \mathbf{S}}{\Delta \mathbf{T}}$$

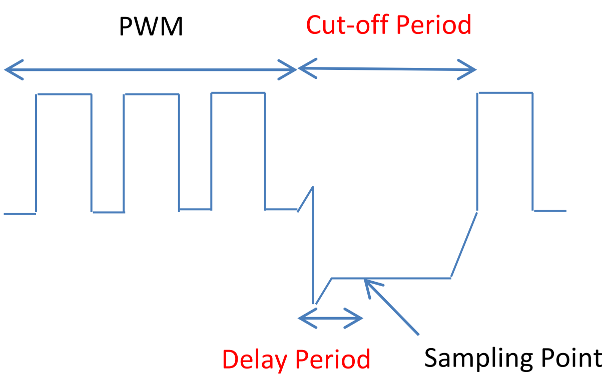
The quadrant encoder provides 2 outputs, channel A and Channel B (90 degrees out of phase to each other). Depending on which pulse comes in first, the direction of rotation can be derived.



4.5.2 Back-EMF [8]

The Back-EMF speed sensing method involves turning off the motor drivers for a short period to measure the potential difference across the motor terminals using an ADC pin on the micro controller and potential dividers.

When the current supply is disconnected, the motor acts as a dynamo producing a voltage. A delay period is given for the back-EMF to settle, once the voltage stabilises the ADC on the microcontroller samples the voltage to give a linear speed reading and distance can be obtained by integrating the reading over time.



4.6 PID closed loop control

There are factors that cannot be mathematically taken in account of balancing the robot. The robot won't be operating under a controlled environment, exposed to unexpected events, such as inclined planes, uneven surfaces, and wind. Due to these factors, controlling the robot with no feedback is very unpredictable, and a lot of the measurements are unpredictable if the robot is not working in a controlled environment. It is best to use a closed loop control system as opposed to an open loop control system.

A Proportional Integral Differential (PID) controller calculates the error between its process values with its desired set point and attempts to correct the calculated error. [9]

4.7 Motors

DC brushed electric motors generates torque by supplying a DC current, internal commutation supplies current to the electromagnetic rotor. Speeds are varied by controlling the current going into the motor, most commonly with PWM.

4.8 Gearboxes

Gearbox is a device that uses gears and gear trains to provide speed and torque conversions. [10] They are used to increase torque for electric motors.

4.9 Bluetooth

Bluetooth is standard for exchanging data over short distances in the ISM band (between 2400-2480); it is originally designed to be a wireless alternative for RS232. [11] Bluetooth is cheap and efficient because it does not require a linear amplifier due to the clever selection of Gaussian frequency-shift keying.

The PlayStation 3 game console interfaces with its controller via Bluetooth. To use the PlayStation 3 controller, the communication between the console and the controller is reverse engineered. The balancing robot then emulates the ps3 console to fetch information from the controller.

4.10 Previous implementations of the robot

4.10.1 nBot Balancing Robot

Figure 1 [3]

This robot is capable of turning by adding a voltage to one wheel and subtracting a voltage from another wheel, without affecting the central of gravity. The robot can operate autonomously following routines based on the input of the wheel encoders.

```
           +---------------------+--------------+
+--------< | Motor shaft encoder |              |
|          +---------------------+   Motor PWM  | <-----+
|    +---< | Angle sensor        |              |       |
|    |     +---------------------+--------------+       |
|    |                                                  |
|    |     +--------+     |\ angle                      |
|    |     |        |     | \ velocity                  |
|    +-----| Deriv  |-----|K1-----+                     |
|    |     |        |     | /      \                    |
|    |     +--------+     |/        \                   |
|    |                               +-----+            |
|    |                    |\ angle   |      \           |
|    |                    | \ pos    |       \          |
|    +--------------------|K2--------+        \         |
|                         | /        |         \        |
|                         |/         |          \       |
|                                    |  SUM      +------+
|          +--------+     |\ wheel   |          /
|          |        |     | \velocity|         /
+----------| Deriv  |-----|K3--------+        /
|          |        |     | /        |       /
|          +--------+     |/         |      /
|                                    +-----+
|                         |\ wheel  /
|                         | \ pos  /
+-------------------------|K4-----+
                          | /
                          |/
```

Figure 2 [3]

The nBot balance algorithm differs from the conventional PID controllers; all the inputs are multiplied by a coefficient and summed to provide an output.

4.10.2 Simple Analogue Balancing Bot

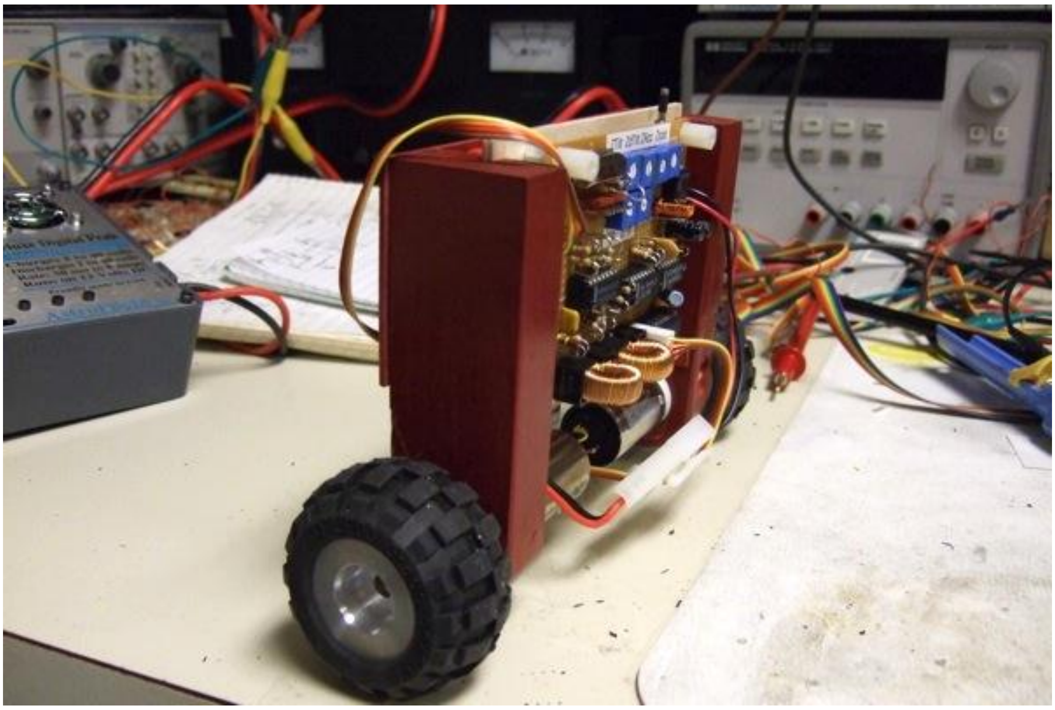


Figure 3: Dale's balancing robot [12]

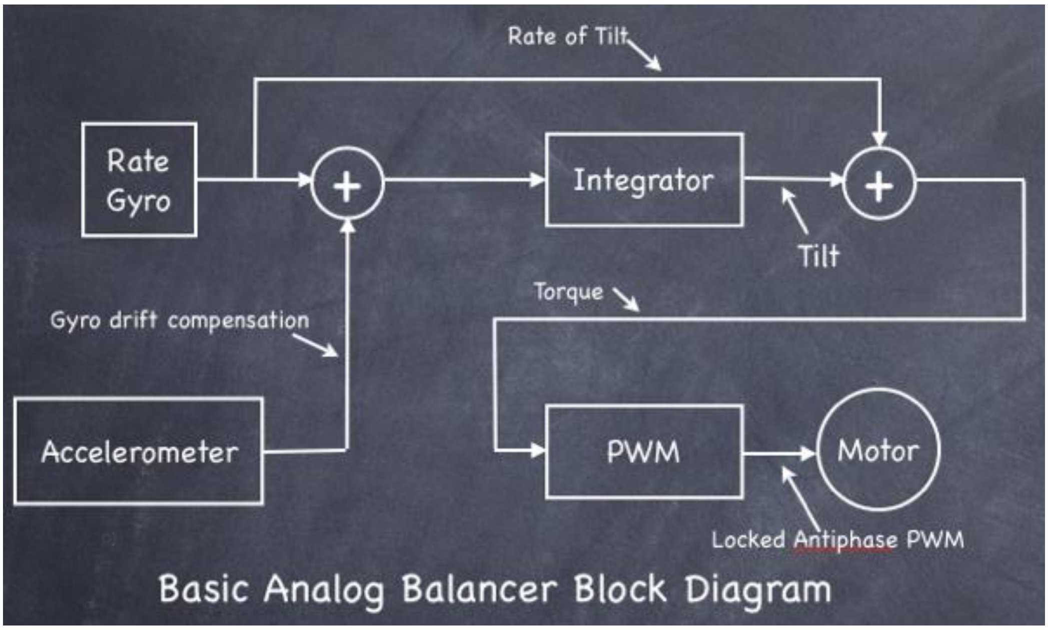


Figure 4: Block Diagram [12]

The Dale Analogue Ball balancer has an analogue PID controller that revolves around capacitors, inductors and OPAMPs. Trim pots are used to tune the PID coefficients and the PWM signal is generated using a comparator and a triangle wave generator.

5. Specification

5.1 Qualitative Requirements

- Adjustable balancing point (possibly a potentiometer)
- Display current angle on pc via usb serial
- Maintain system orientation within ± 5^{o} of the balancing point
- Maneuverable by adjusting offsets to the balancing point
- Remote Controllable
- User accessible data record

5.2 Hardware

5.2.1 Component list (First Robot)

Component	Model	Function	Price (£)
ARM Controller	mbed NXP LPC1768	Processor + PID + PWM	£40
5DOF IMU	IDG500/ ADXL335	Detect robot angle	£27.99
2x Lithium Battery	LC 14500	Robot power supply	£4.9
Geared motor	Pololu Micro Gearmotor 100:1	Manoeuvres robot	£19.86
Potentiometer	Generic	Trim Balancing angle	£0.50
Dual Full Bridge Driver	L298N	PWM + Direction motor Driver	£1.84
8x Schottky Diode	1N5819	Prevent BackEMF	£0.72
5V Voltage Regulator	LM7805	Provide Logic Voltage to mbed and l298n	£0.78
Bluetooth USB Adapter	Generic	Interface with PS3 Controller	£3.49
Project Sum			£100.08

See Appendix 1 for component datasheets

5.2.2 Component list (Second Robot)

Component	Model	Function	Price (£)
ARM Controller	mbed NXP LPC1768	Processor + PID + PWM	£40
6DOF IMU	ITG3200/ADXL345	Detect robot angle	£40.29
2x Lithium Battery	LC 14500	Robot power supply	£4.9
Geared motor	Pololu Micro Gearmotor 30:1	Manoeuvres robot	£19.86
Dual Full Bridge Driver	L293N	PWM + Direction motor Driver	£1.84
Potentiometers	10K	Speed feedback scaling	£0.10
5V Voltage Regulator	LM7805	Provide Logic Voltage to mbed and l298n	£0.78
Bluetooth USB Adapter	Generic	Interface with PS3 Controller	£3.49
Project Sum			£111.16

5.2.3 Additional Parts for experimentation

Component	Model	Function	Price (£)
AVR Controller	Arduino UNO R3	Processor + PID + PWM	£23.95
9DOF IMU	L3G4200D/ LSM303DLM	Detect robot angle and heading	£35.95
Project Sum			£59.9

The additional parts were bought with my own money as the Arduino is needed for another home automation project.

The 9DOF IMU has a 3-axis magnetic compass built in; tilt compensation for the compass is done in software by adding the 3d vector of the accelerometer to the magnetic compass 3d vector before computing the heading. The tilt compensated compass is fused with the Y axis gyro for providing quick responses.

5.2.4 Hardware justification [2]

Mbed NXP LPC1768 – is the core of the system, it constantly acquires information from the IMU producing an output to correct the robot's orientation.

Figure 5 [13]

Advantages of using Mbed over other microcontroller:-

- No additional programmer is required.
- Secondary ARM processor acts as programmer, USB serial and mass storage device.
- Cross platform rapid development (web based compiler)
- Preliminary work done using the mbed
- Built in Ethernet and usb host

5DOF IMU – in this project only 3-axis from the IMU is required to obtain the Roll (angle with respect to the positive Y axis). The Y and Z axis from the accelerometer and the X axis from the Gyroscope, by putting the accelerometer values into atan2(-Y,-Z) the Roll angle is calculated. The gyro produces a rotation rate, so by integrating the rate the angle is calculated. It costs more to buy the sensors separately even if the sensors have less degree of freedom; it makes most sense to opt for the IMU.

Figure 6 [14]

Lithium Battery – lithium batteries have a high energy density and also they have a higher voltage (3.7V a cell). Two lithium batteries provides more than enough voltage to drive the motors at its full speed, if alkali batteries were used 2.5 times the number of cells are needed to give the same voltage output.

Geared motor – a 100:1 gear motor is selected for this project as they rotate at a sufficient speed (120 rpm), and they provide a high enough torque to make the robot oscillate.

Figure 7 [15]

Dual Full Bridge Driver – the L298N are able to drive 2x motors in both directions because of its dual H-bridge. They are relatively cheap to buy for a high output rating of 2A.

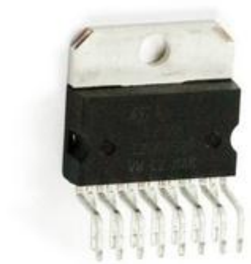


Figure 8 [16]

Voltage Regulator – with the addition of a 5V voltage regulator the system is experience less brownouts and the voltage regulators on the mbed no longer heats up as much.

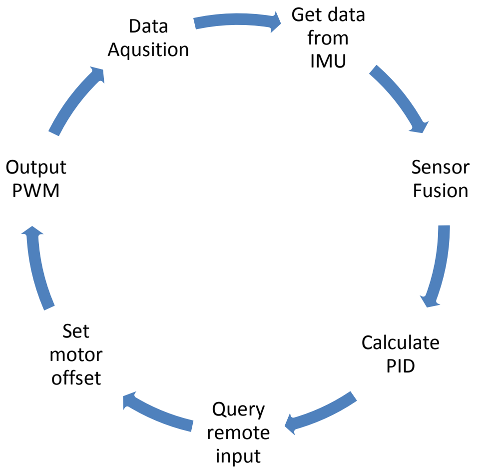
Bluetooth USB Adapter – Most Bluetooth modules are expensive and they are usually designed to perform a single function e.g. Bluetooth serial, Bluetooth Audio.

Whereas the Bluetooth USB adapter can support a various range of functionality as long as the software Bluetooth stack is ported to the microcontroller. As such adapters are abundant in the consumer market; they come in a relatively low price.

Figure 9 [17]

5.3 Software [2]

The software will be written in C++ using the mbed.org online compiler. Most of the codes are available online can be applied on to this project with minor modifications, such as sensor fusion and PID routines.



6. Design

6.1 Mechanical design

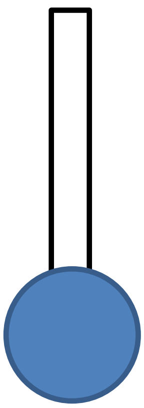


Figure 11

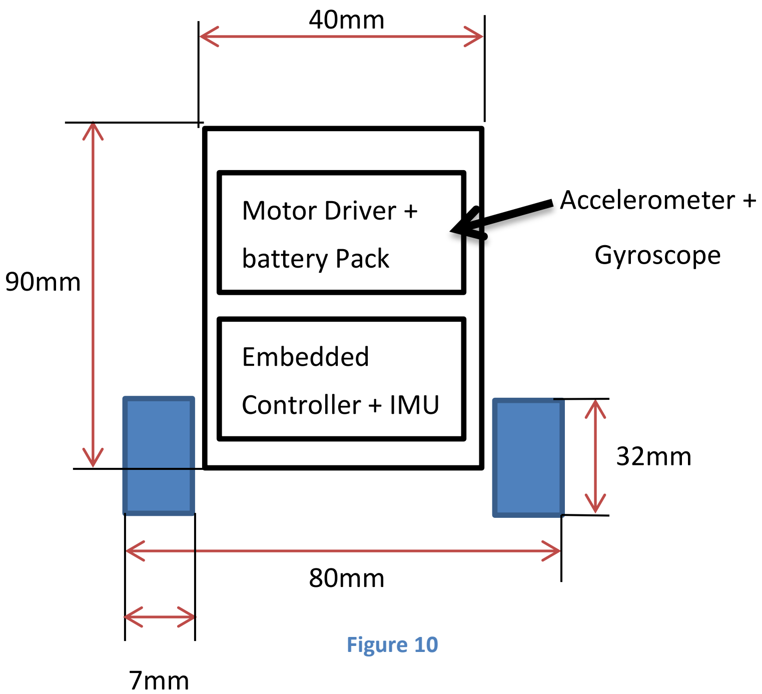


Figure 10

6.2 System Block Diagram

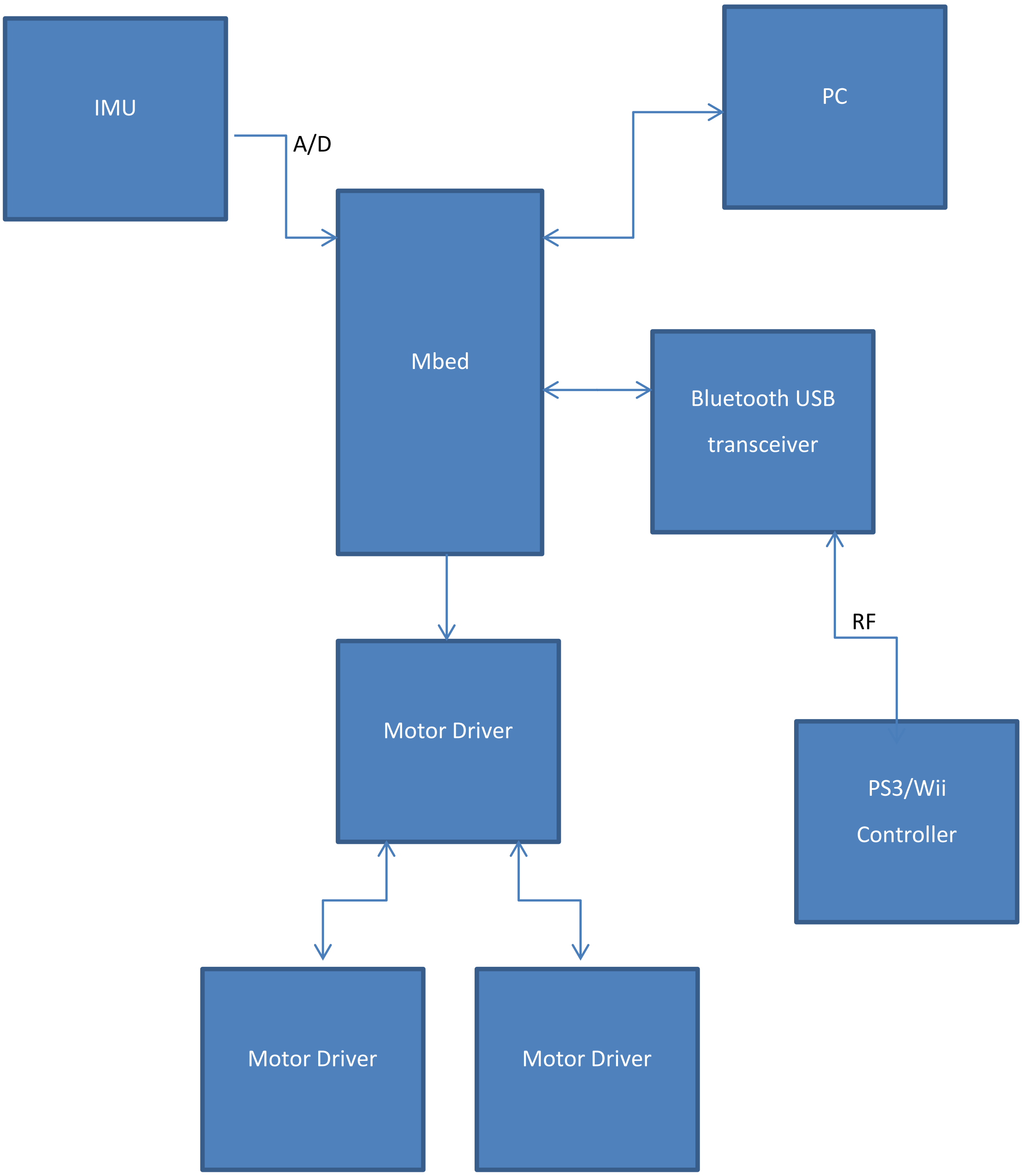
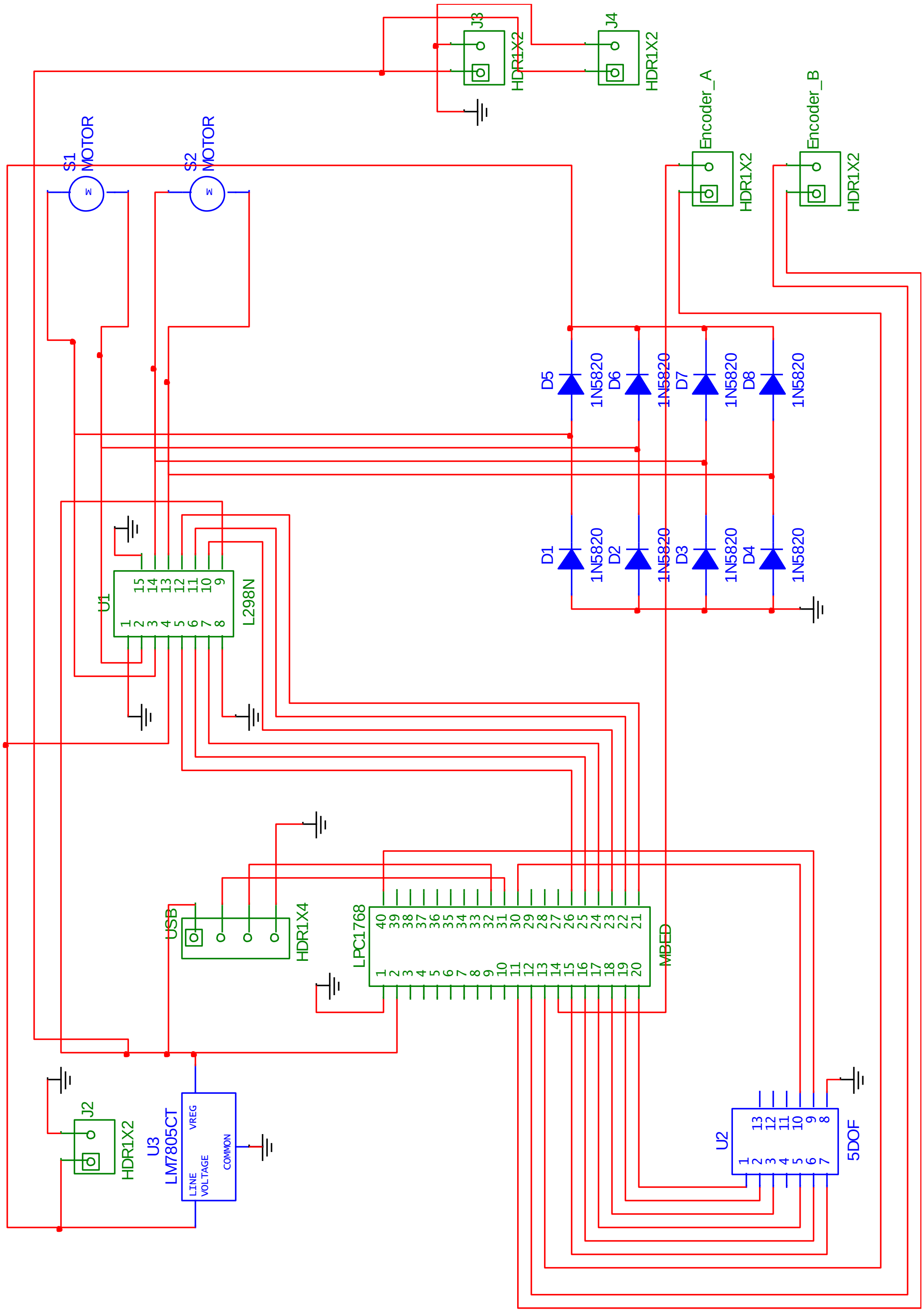


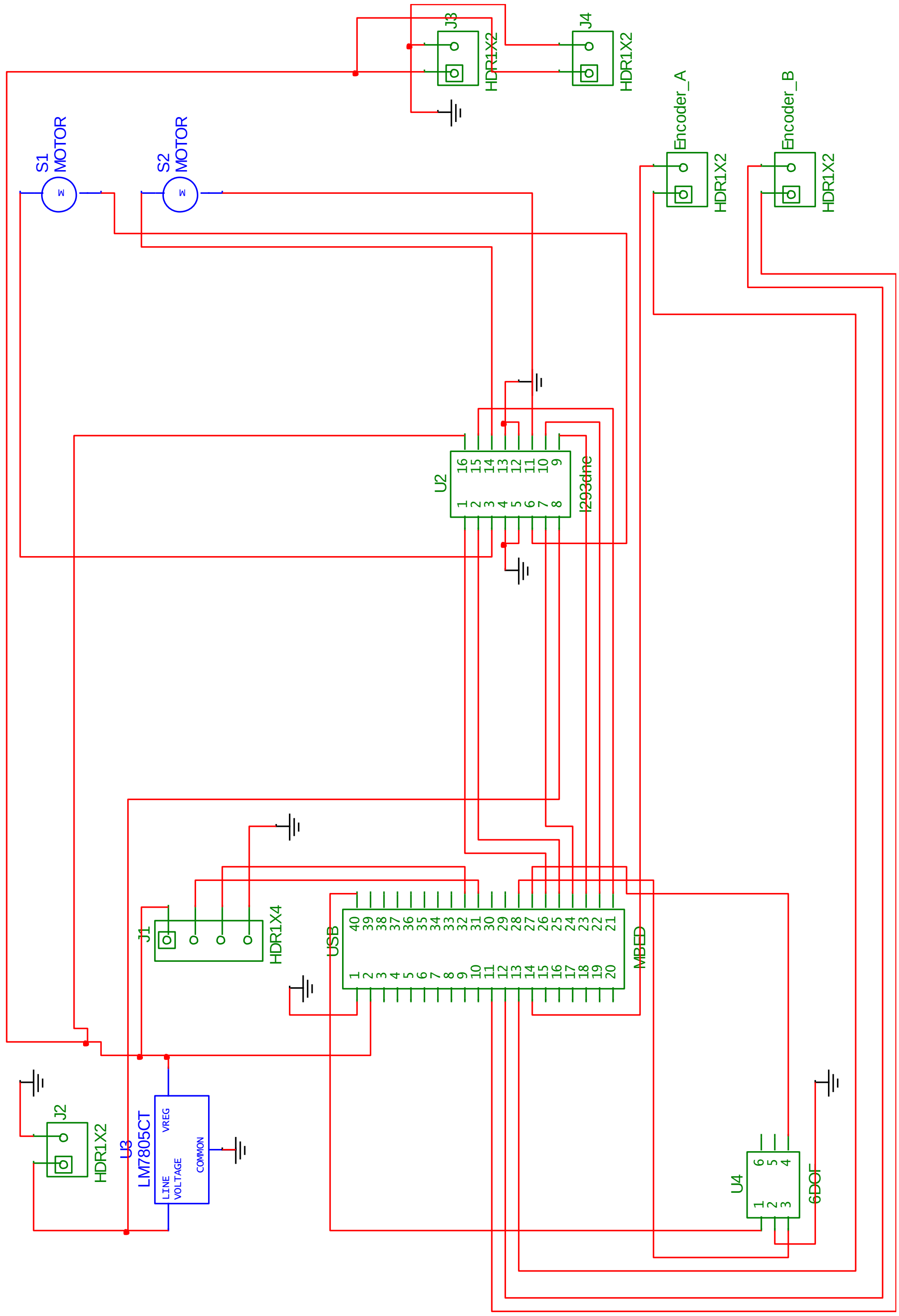
Figure 12

6.3 Circuit Design

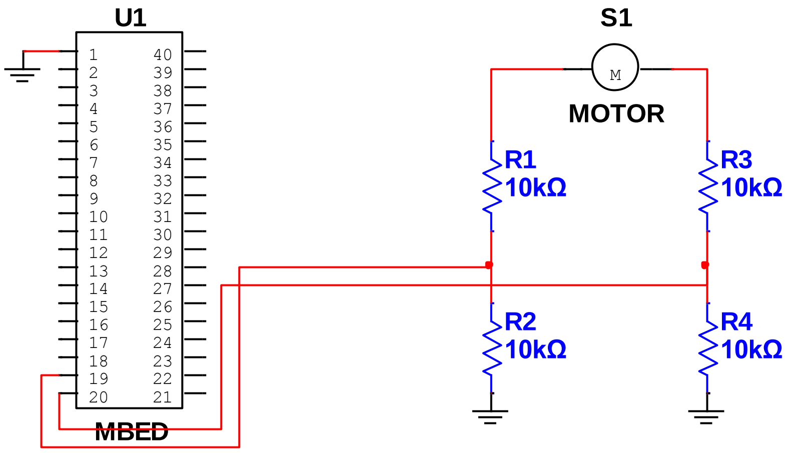
6.3.1 Analogue sensor and wheel encoder



6.3.2 Digital sensor and Back-EMF speed sensing



6.3.1 B-EMF Circuit



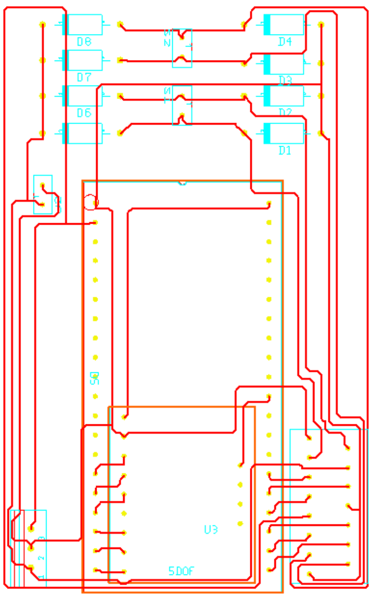
6.4 PCB design

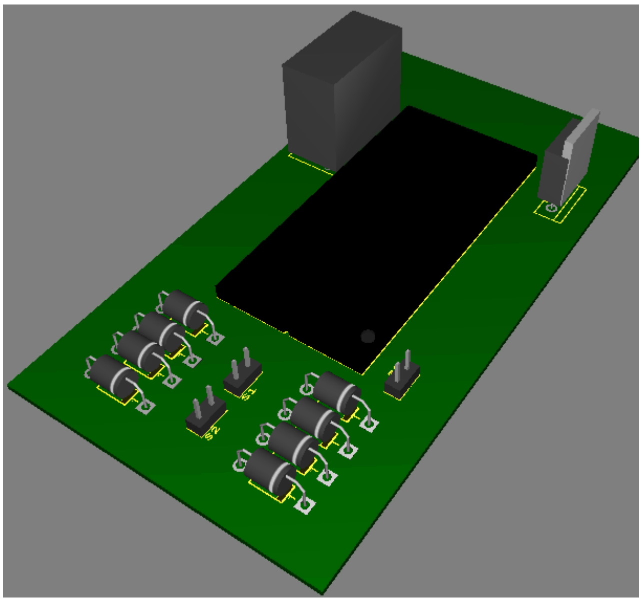
PCB is designed on the 26th of November with NI Ultiboard.

Steps:-

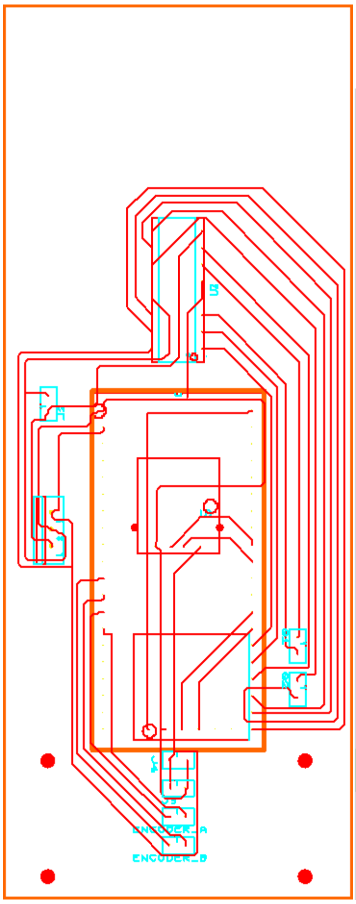
1. Create circuit in multisim
2. Transfer circuit from multisim to Ultiboard
3. Arrange components carefully to allow auto routing on a single layer

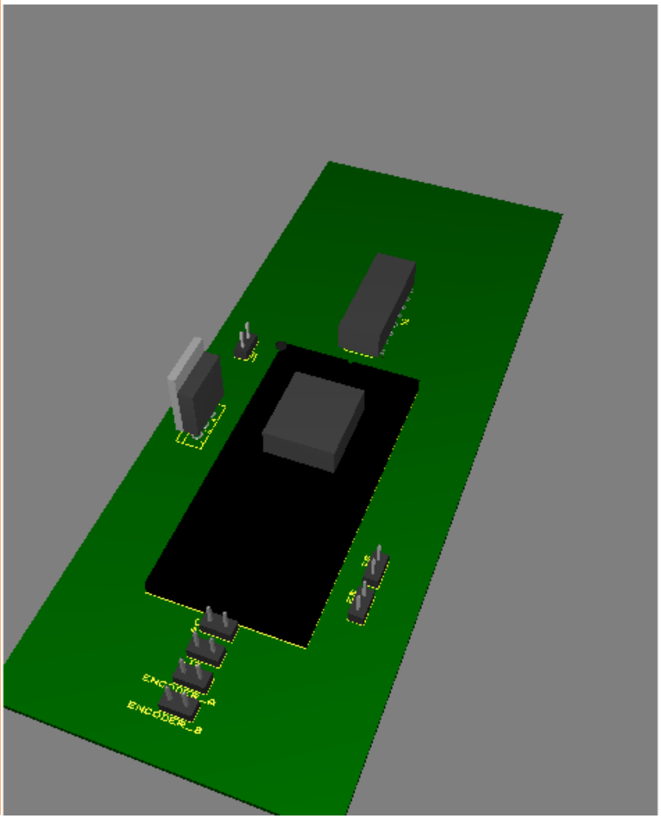
6.4.1 Analogue sensor





6.4.2 Digital sensor and B-EMF speed sensing





Single layer PCBs can help to keep the production cost of the device down.

Figure 13: Toner transfer method prior to Copper Etching.

6.5 Software Design

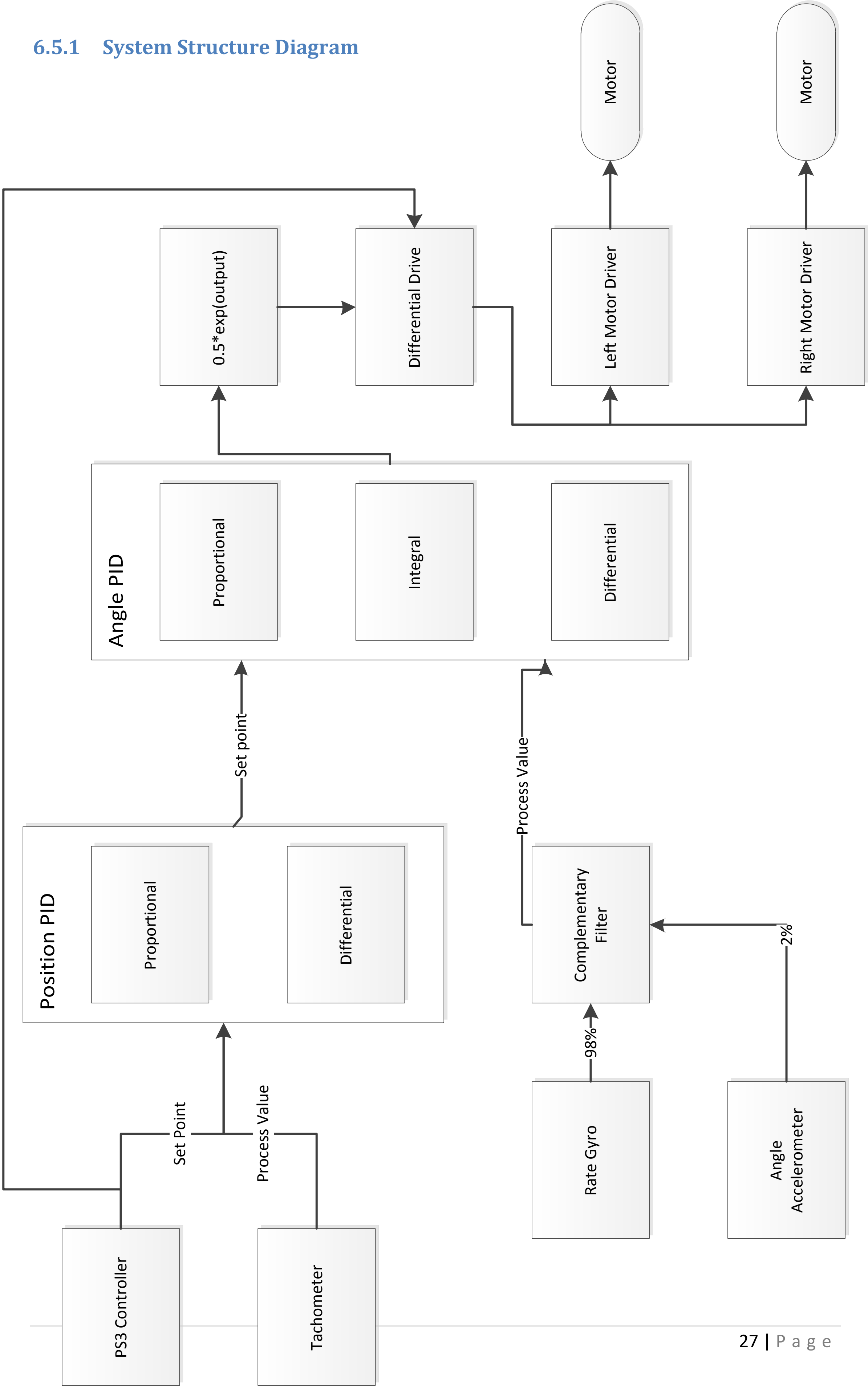
Three programming approaches were tested:

1. Sequential execution in a loop
2. Timer Interrupts
3. Asynchronous threads running under an RTOS

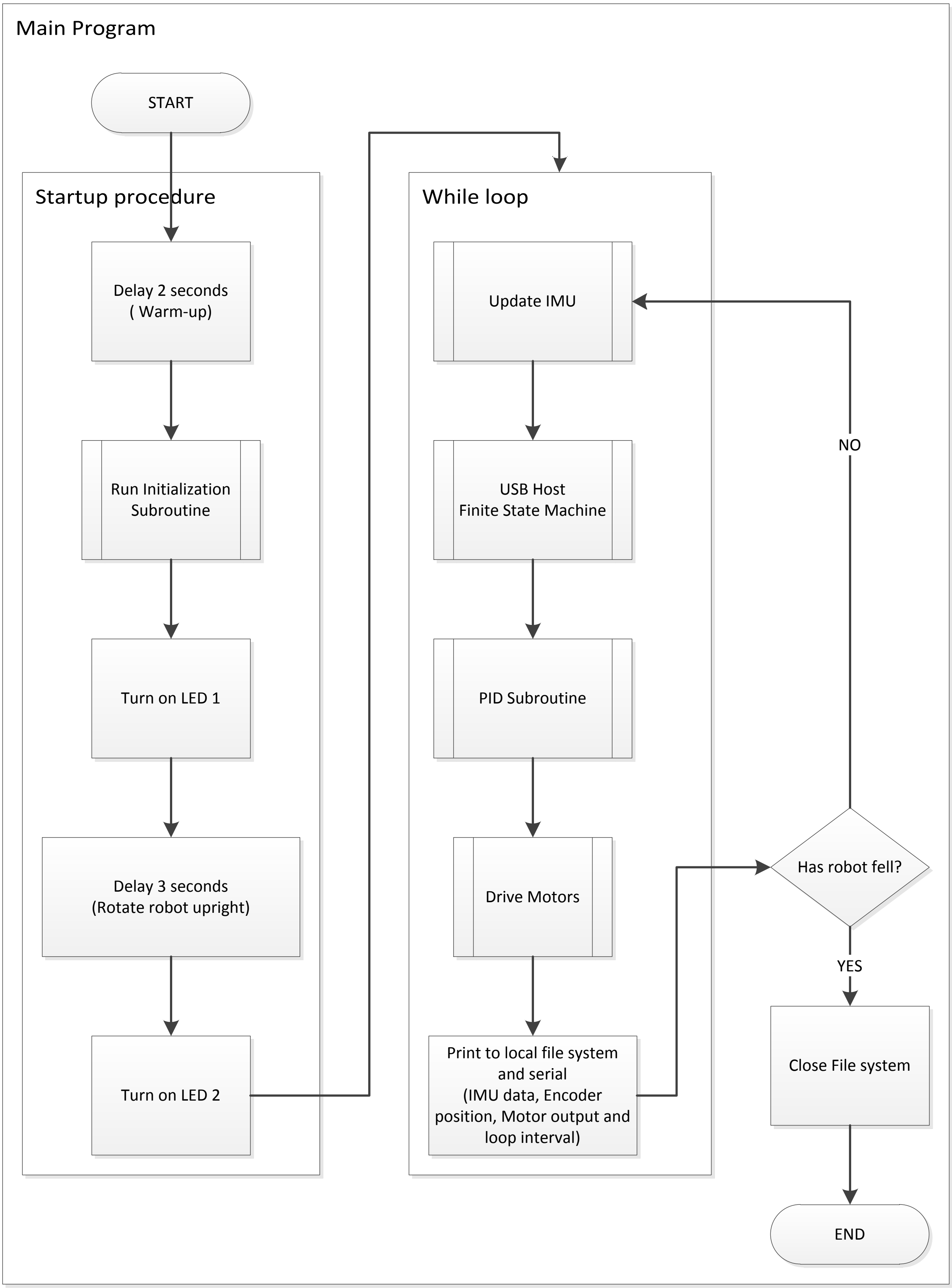
The Sequential task execution method was the initial approach that worked. The Timer interrupt approach is causing a lot of problems as there is always a system interrupt that has a higher priority, causing inconsistent loop times.

The Asynchronous thread execution under an RTOS with pre-emptive priority based scheduling, ensures that lower priority threads doesn't affect the process time of higher priority threads. This ensures modularity of the system as the behaviour of the robot will not change drastically when additional modules are added. This approach was used when testing the BEMF motor speed regulation for two motors running concurrently.

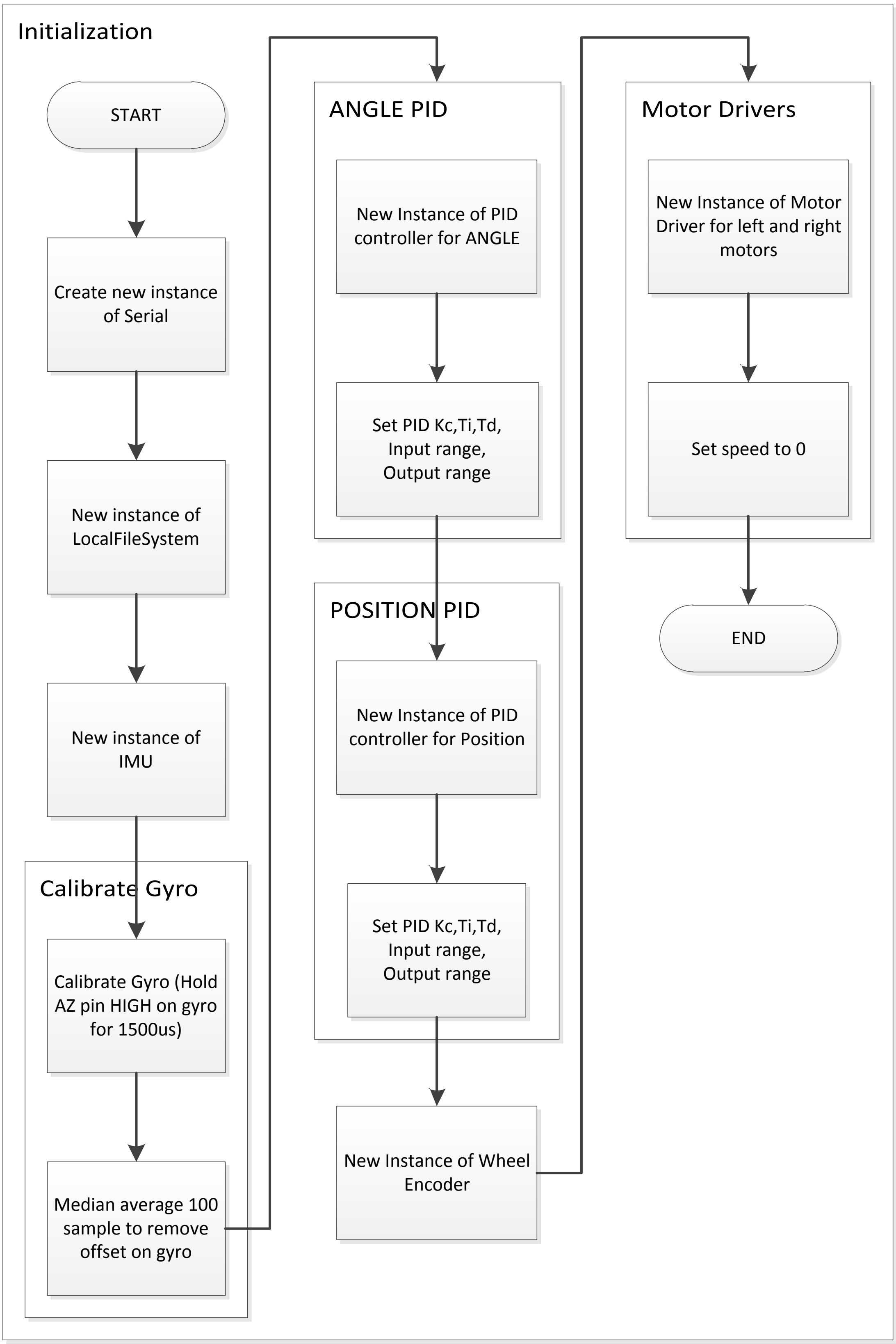
6.5.1 System Structure Diagram



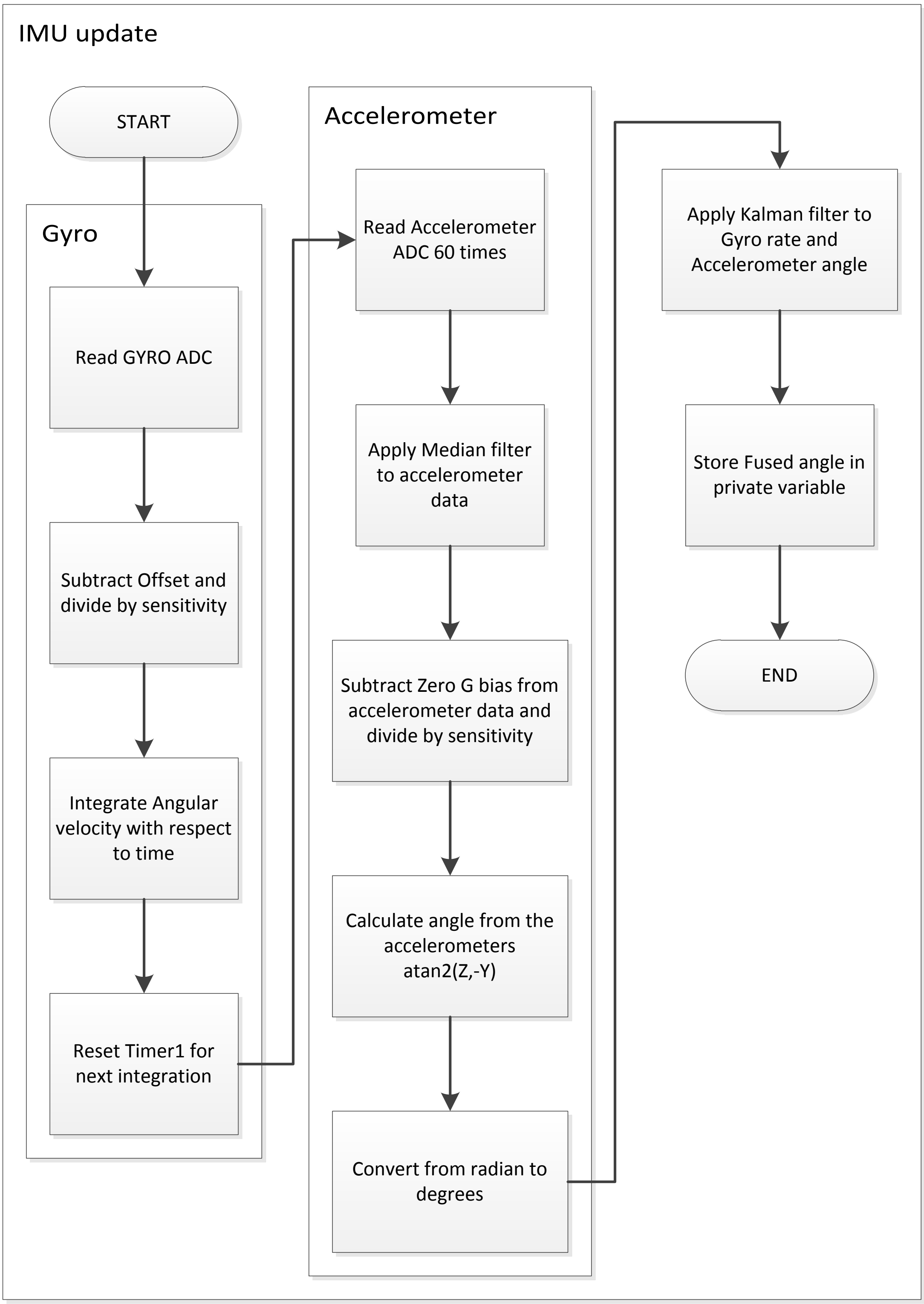
6.5.2 Algorithm Flow Chart



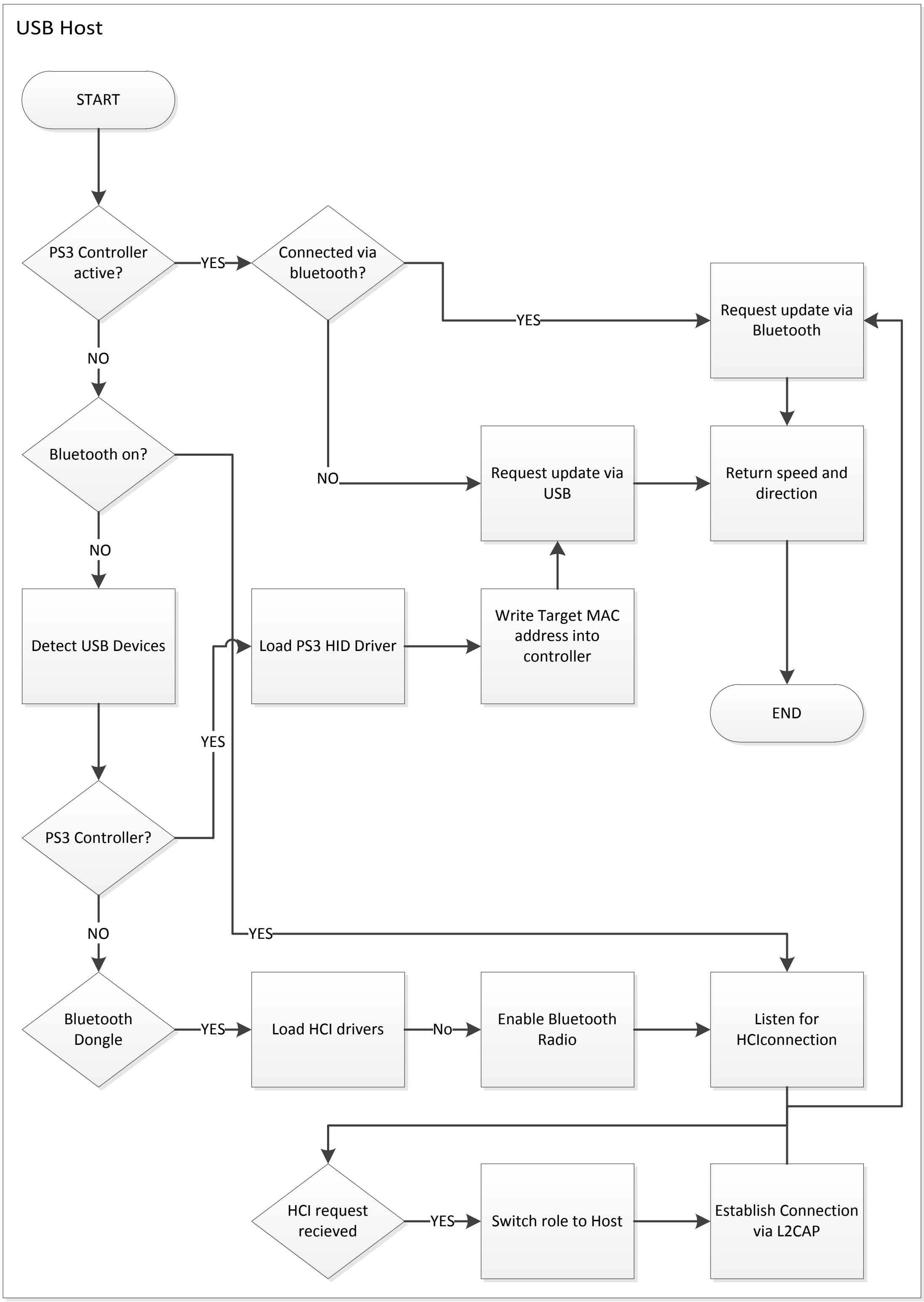
6.5.2.1 Initialisation Subroutine



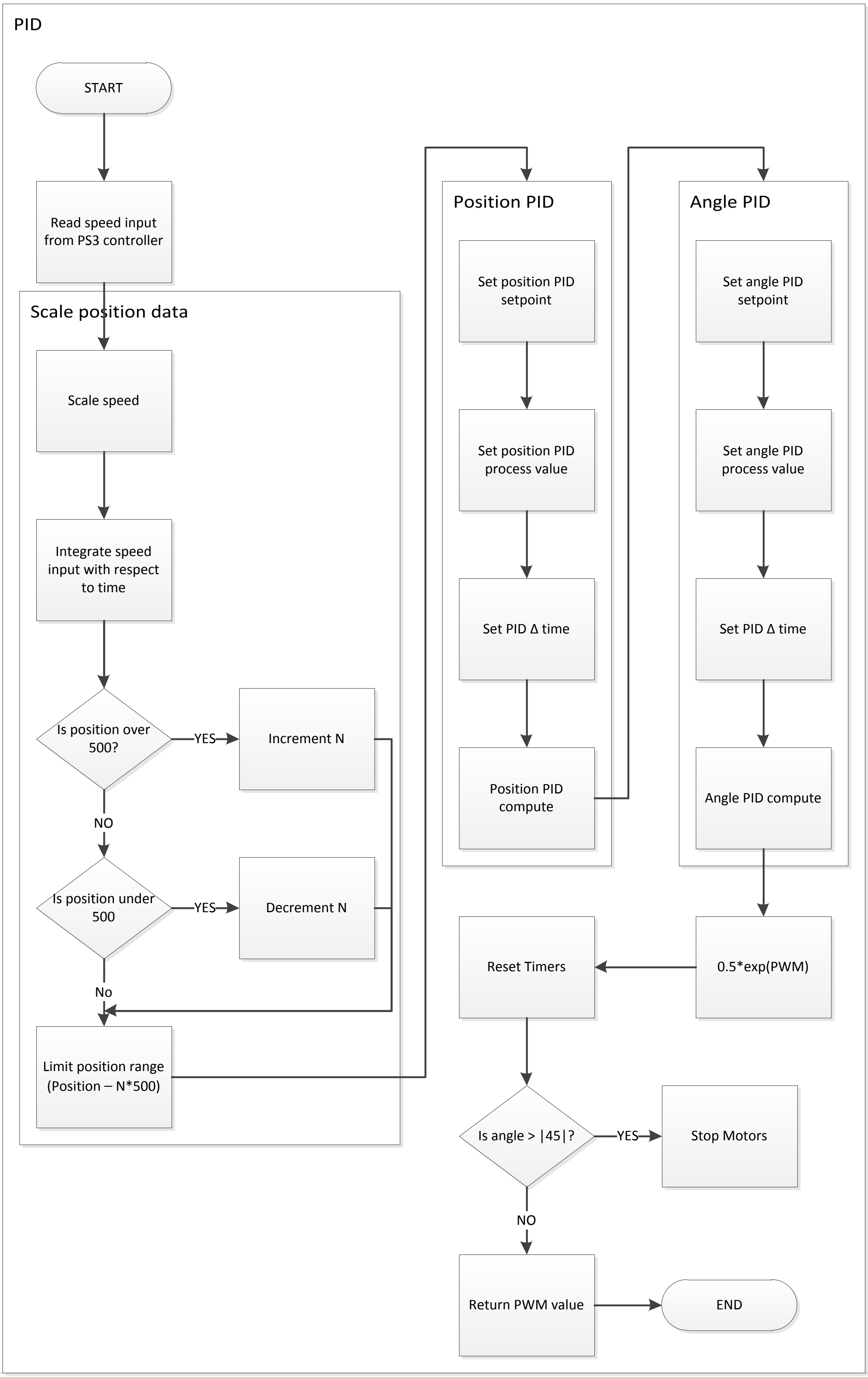
6.5.2.2 IMU-update Subroutine



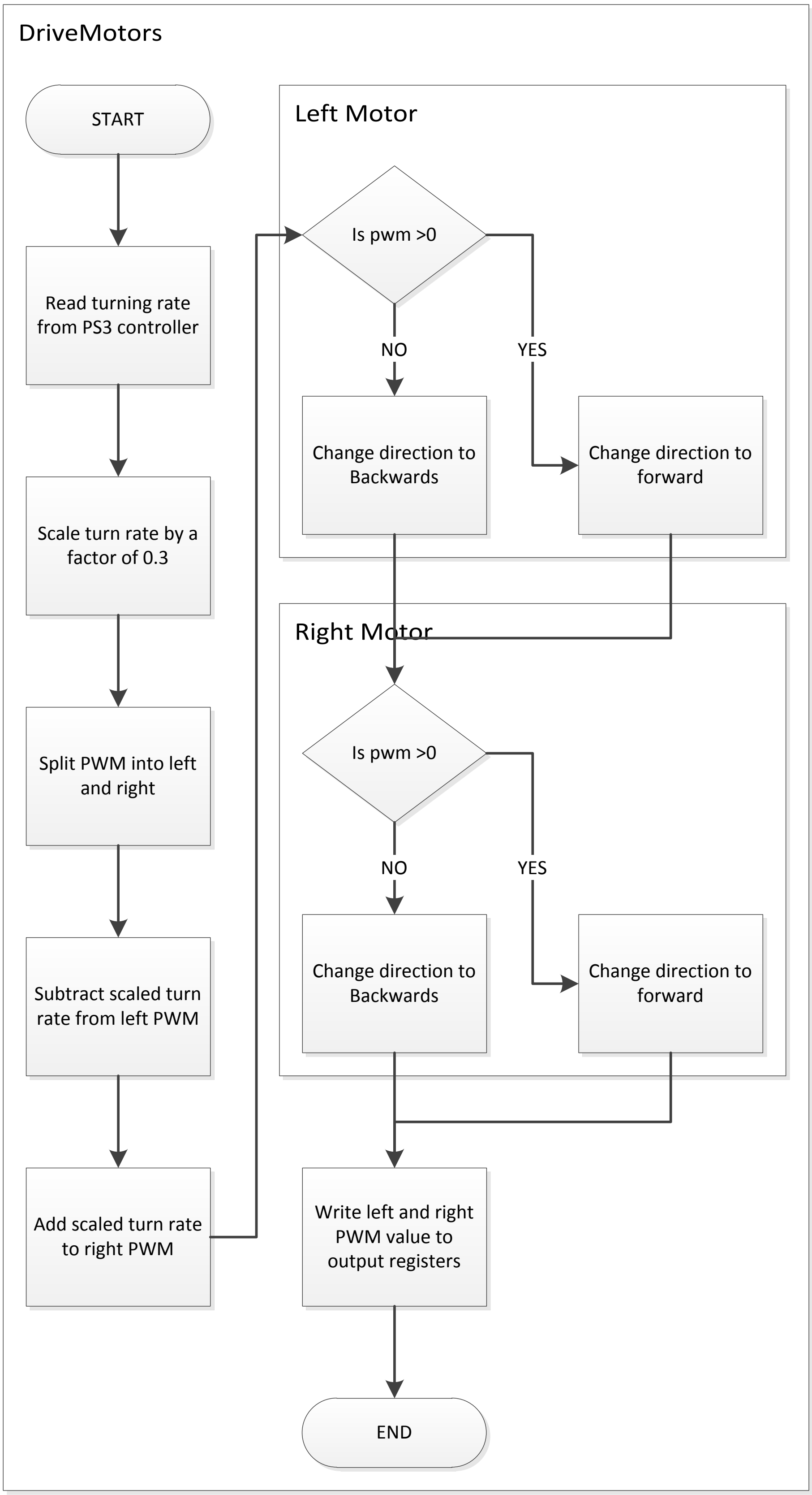
6.5.2.3 USB-Host Subroutine



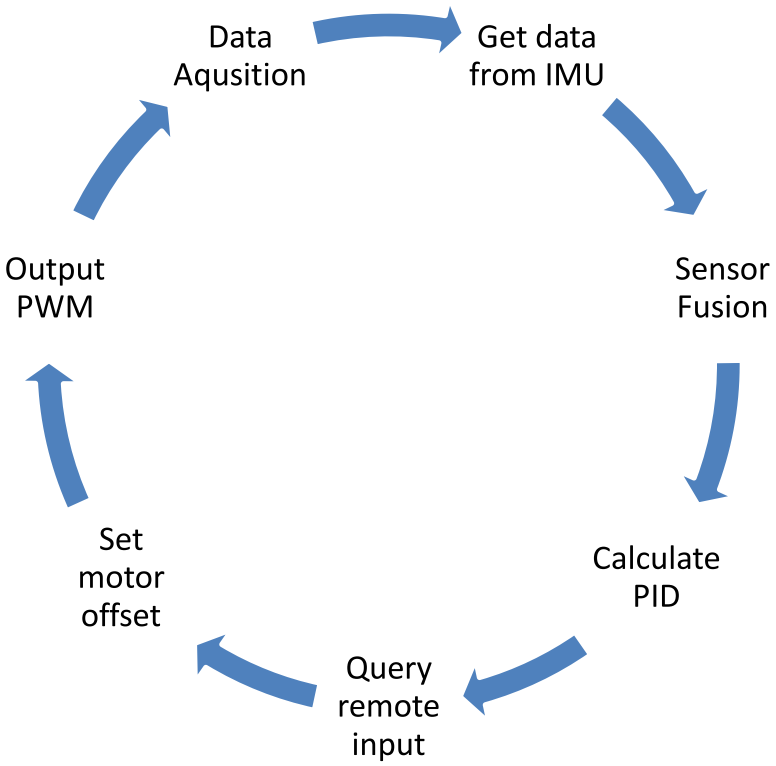
6.5.2.4 PID Subroutine



6.5.2.5 Drive-Motor Subroutine



6.5.3 Main Loop [2]



When the microcontroller boots up, the initialisation phase begins. This involves setting various pins to input and others to output, waiting for a period of time for the components to warm up and calibrating the Sensors.

After initialisation the program goes into a loop that executes itself 50 times a second

1. First the system acquires data by digitizing the Analogue output from the accelerometer and gyroscope, the acquired voltage is then subtracted by the sensor's default Zero G voltage and divided by its sensitivity
2. The Sensor information is then fed into a complement filter [Roll = (0.98)*(Roll + Xrate*0.02) + (0.02)*(AccAngleX);]
3. The calculated Roll is then fed into the PID controller to maintain it at a set point defined by the user.
4. If the mbed receives an instruction from the PS3 Controller it will either change the angle or induce a differential to the wheel speed.
5. PWM drives the motors

6.5.4 Bluetooth Communication

The microcontroller pretends to be the ps3 allowing a ps3 controller to establish a connection and push input data to the microcontroller via Bluetooth upon request, to achieve this a Bluetooth 2.0 EDR module is attached to the microcontroller via an internal USB host controller communicating via an HCI driver.

Initialisation:
1. Setup Bluetooth dongle using the HCI protocol. 2. Wait for the incoming request from the PS3 Controller 3. Accept request when PS3 Controller attempts to connect, and change role to master. 4. Listen on the Bulkin endpoint, for the HCI Control channel (PSM: 0x11). 5. Respond by sending a connection response. First with the result: pending, and then with the result success. 6. Send a configuration request. The controller will then respond with a configuration request as well.
Setup Interrupt:
1. Listen on the Bulkin endpoint, for the HCI Control channel (PSM: 0x11). 2. Respond by sending a connection response. First with the result: pending, and then with the result success. 3. Send a configuration request, sets up the HID Interrupt (PSM: 0x13) channel.
Request input:
1. Send a Set Feature Report (0x53) with a report ID (0xF4) and the following data: 0x42, 0x03, 0x00, 0x00.

Table 1 [18]

The communication was initially reverse engineered using btscanner 2.0 under Backtrack 5 R1 (a Linux penetration testing distribution). With the information in table 1 and sample source codes on the mbed website, the robot can use the ps3 controller as an input device via a CSR Bluetooth USB dongle.

After Initialisation and interrupt setup the main loop is only required to execute the function USBLOOP(), to retrieve the status of the PS3 Controller.

6.6 PID Tuning

Procedures used for PID Tuning:-

1. increase the P gain until oscillation occurs
2. Increase derivative term until 0 overshot
3. Increase integral to increase response speed

Kc 3.0 Ti 0.02 Td 0.03

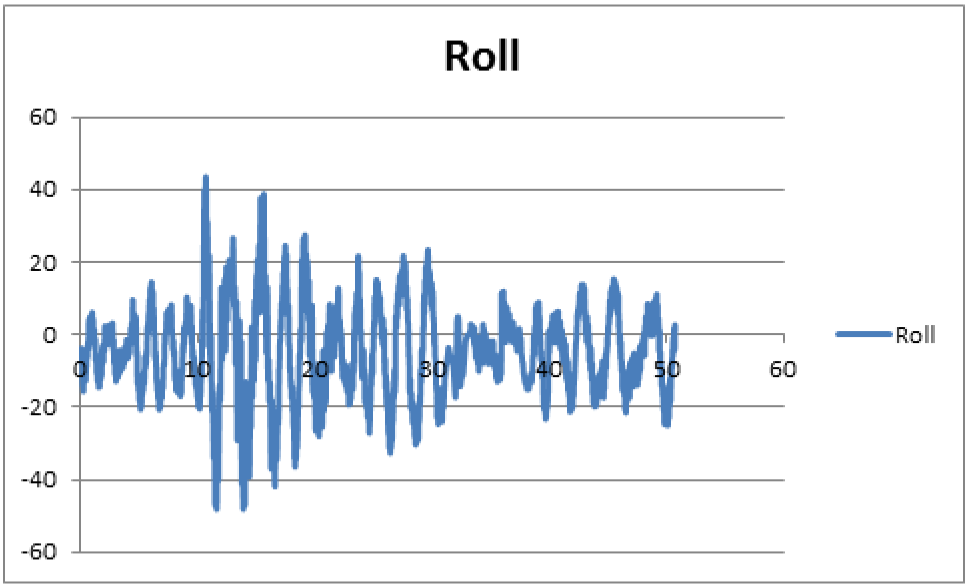


Figure 14

Kc 3.0 Ti 0.02 Td 0.03

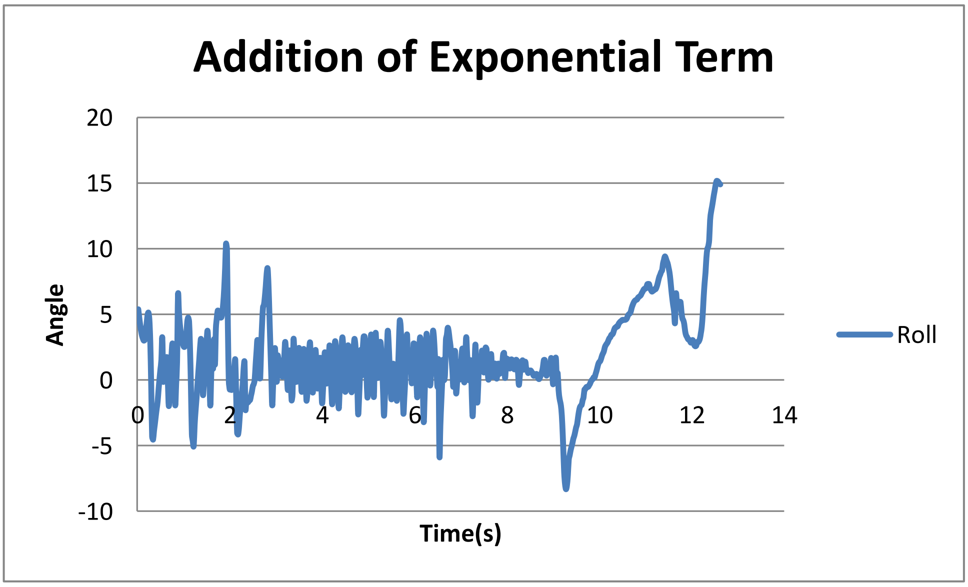


Figure 15

The above charts are plotted from the data stored on the CSV file, the values are stored every 0.02 seconds while the program is executing the main while loop.

From Figure 9, I can tell that with an addition of an exponential term to the output of the PID, the robot tries to overcompensate a lot more, which prevent the robot from constantly catching up with the fall.

6.7 Motor PID tuning (B-EMF)

The motor PID values are calculated, by performing step test from 70% output to 60% output. The Process gain constant, Process Time Constant, Dead Time Constant are derived from the data logging files.

Tuning Constants

$$\mathrm{Kp} = \frac{\Delta \mathrm{Pv}}{\Delta \mathrm{CO}} = \frac{2.08 - 3.13}{-10\%} = \frac{0.105\mathrm{V}}{\%}$$

$$\mathrm{Tp} = -2.96 + 3.08 = 0.12$$

$$[\mathrm{theta}]\mathrm{p} = 2.96 - 2.92 = 0.04$$

A formula for PI tuning is applied to obtain the tuning constants.

- Aggressive tuning
  - Tc = max(0.1 * Tp, 0.8 * [theta]p) = max(0.008, 0.008) = 0.008
  - Kc = (1 / -100) * (0.08 / (0.01 + 0.008)) = -0.04
  - Kc (dimensionless) = -0.04 * (10500/100) = -4.7
- Conservative tuning
  - Tc = max(10 * Tp, 80 * [theta]p) = max(0.8, 0.8) = 0.8
  - Kc = (1 / -100) * (0.08 / (0.01 + 0.8)) = -0.001
  - Kc (dimensionless) = -0.001 * (10500/100) = -0.105

Figure 16 [19]

The conservative tuning takes a longer time to reach the desired speed, but it performs better at low speed regulation than the aggressive tuning.

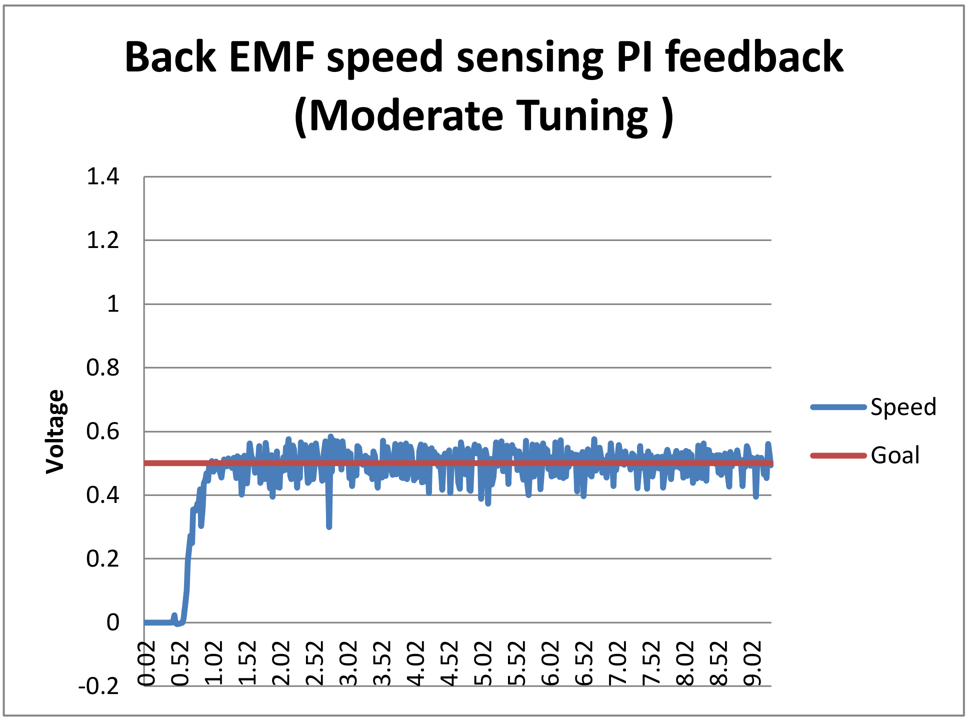


Figure 17

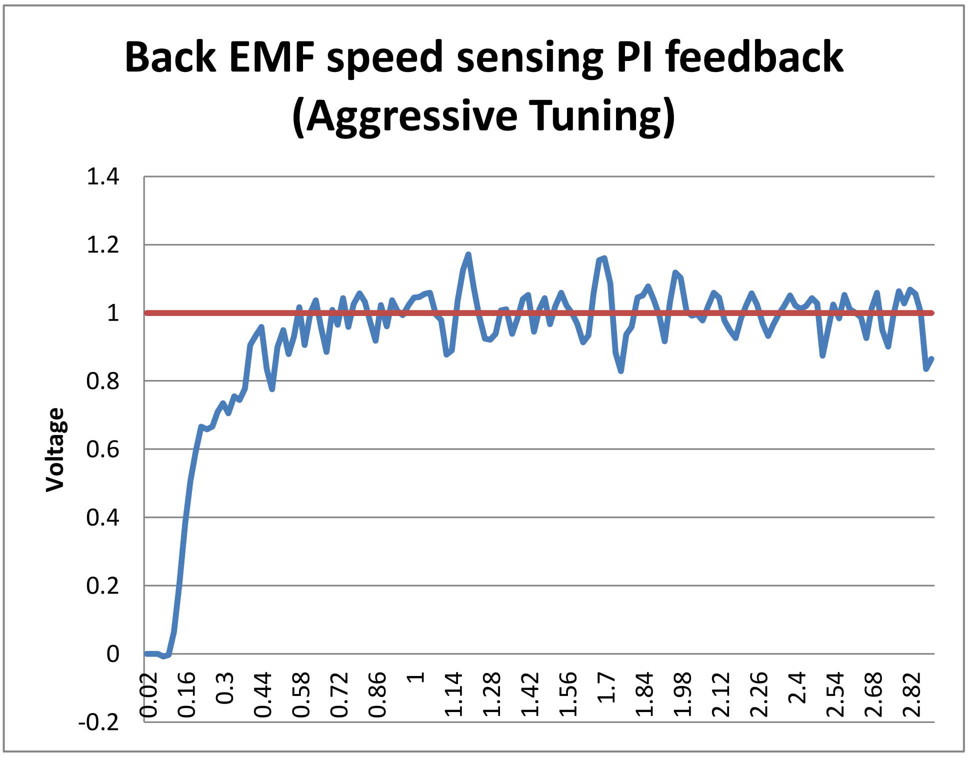


Figure 18

6.8 An attempt to auto tune PID with LABVIEW

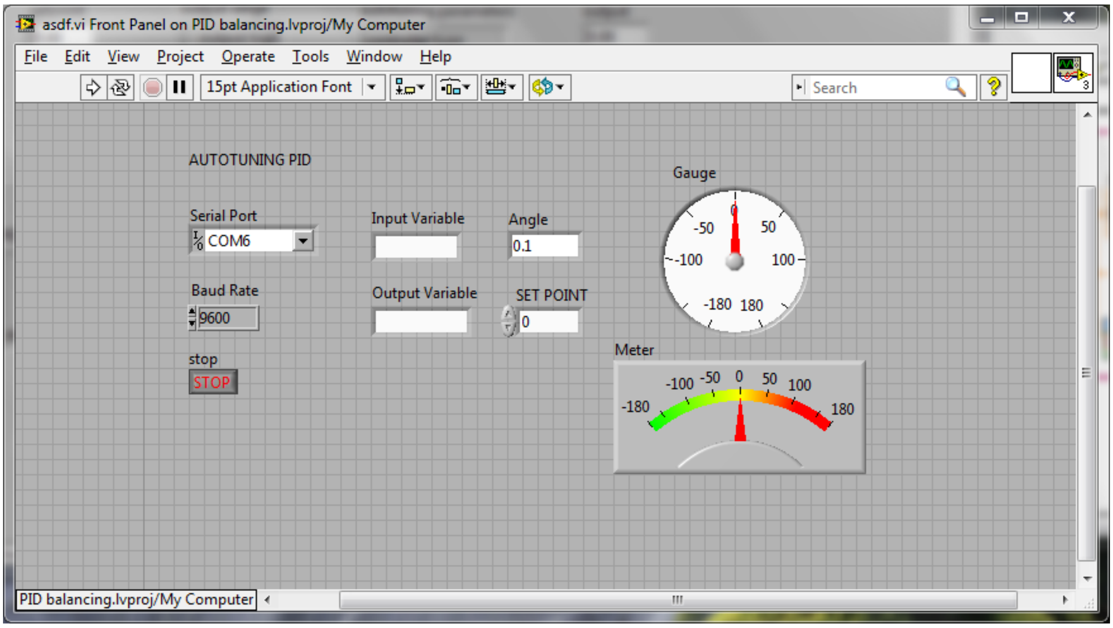


Figure 19

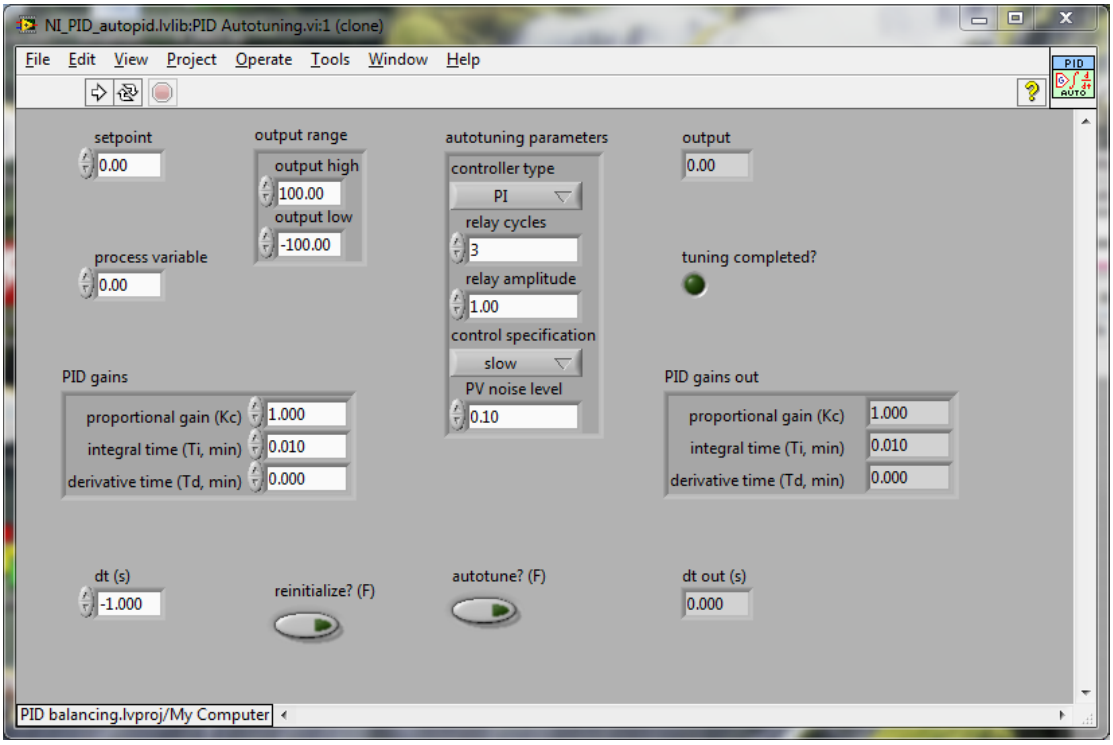


Figure 20

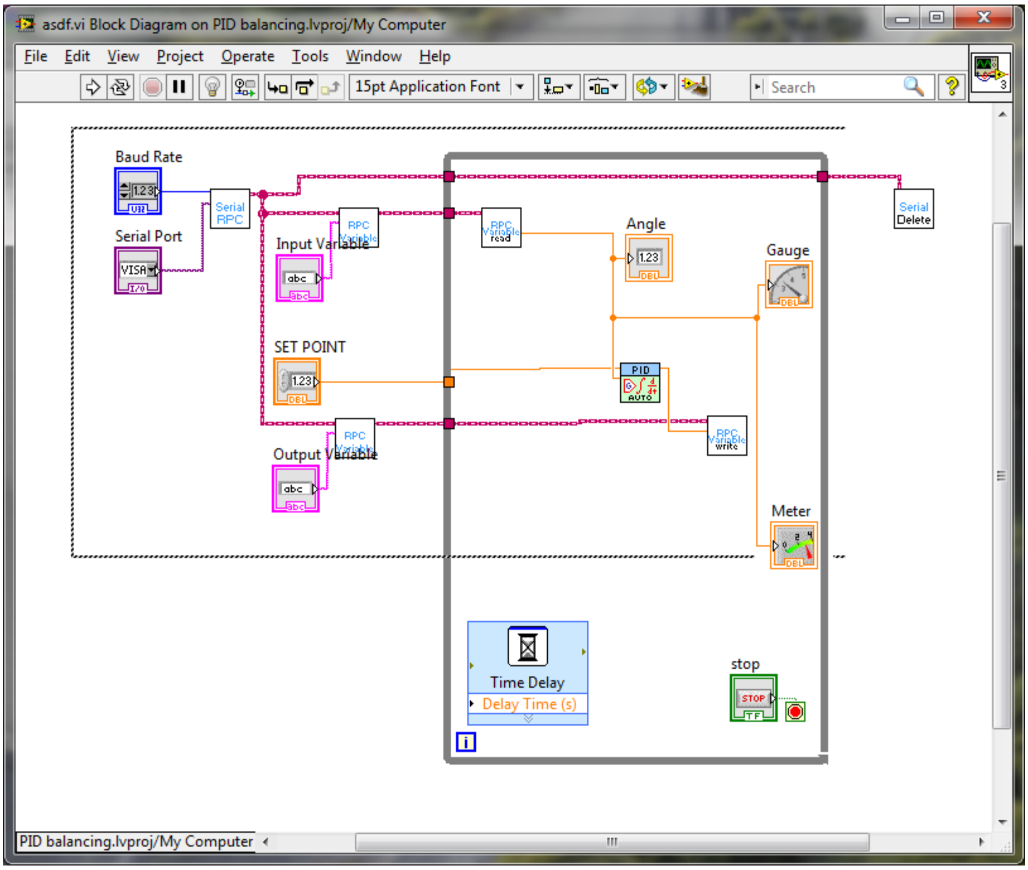


Figure 21

The microcontroller is responsible to acquire data from the sensors, perform sensor fusion and send them to LABVIEW via RS-232. LABVIEW is then configured to run a Close loop PID controller with the ability to automatically derive a close estimate of the PID tuning parameters. The output data is then relayed back to the robot via RS-232 to drive the motors via an H-bridge.

7. Testing

7.1 Sensor Fusion Methods Comparison

7.1.1 Complementary Filter

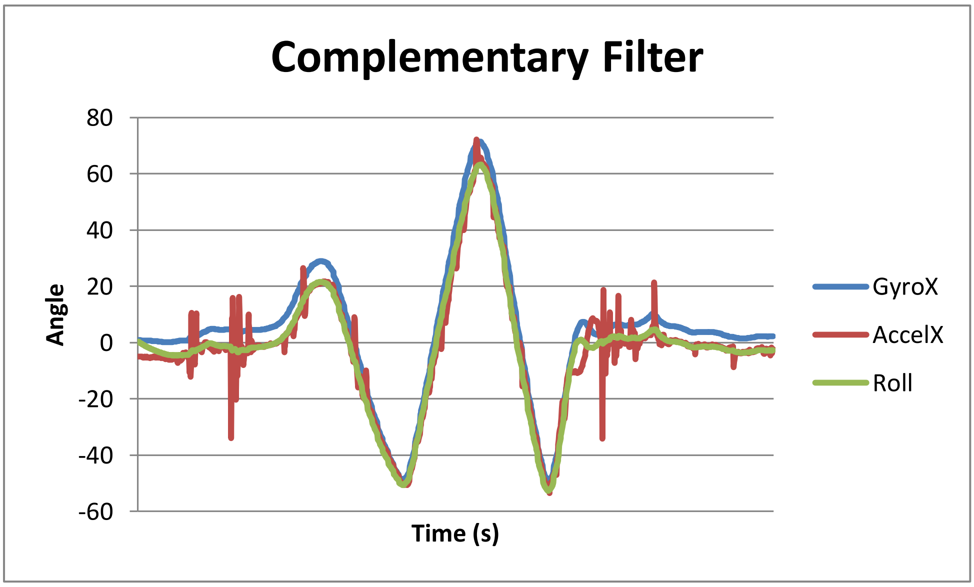


Figure 22

7.1.2 Kalman Filter

Q_angle = 0.001; Q_gyro = 0.003; R_angle = 0.03;

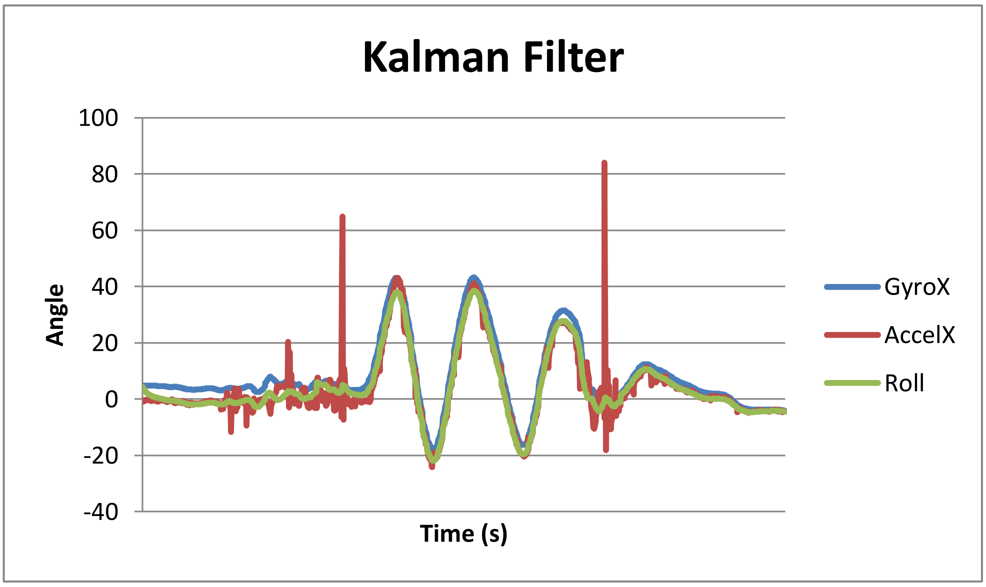


Figure 23

Both the Complementary and the Kalman filter managed to compensate for gyro drift, the Kalman filter has a quicker response compared to the complementary filter (weighted average), the Kalman filter is also less prone to vibrations and sudden accelerations due its prediction mechanism.

The Kalman filter is usually preferred despite the complex calculations that require more processing power, as raw data can be fed directly into the filter without any external additional noise rejection filters.

Three parameters are required to be tuned, GYRO covariance, Angle covariance and Measurement covariance. These parameters determine the amount of averaging required to reject the right amount of noise.

7.2 B-EMF speed sensing

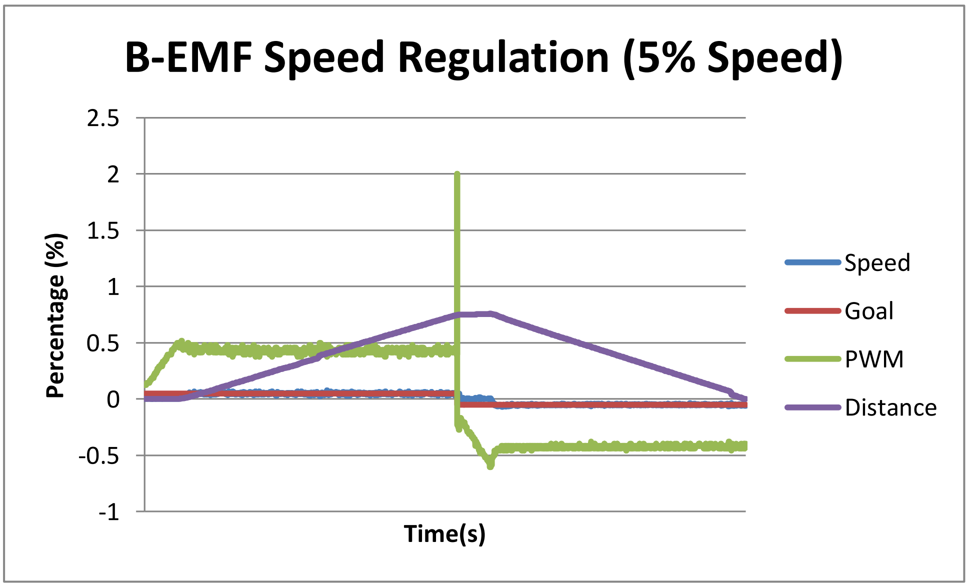


Figure 24

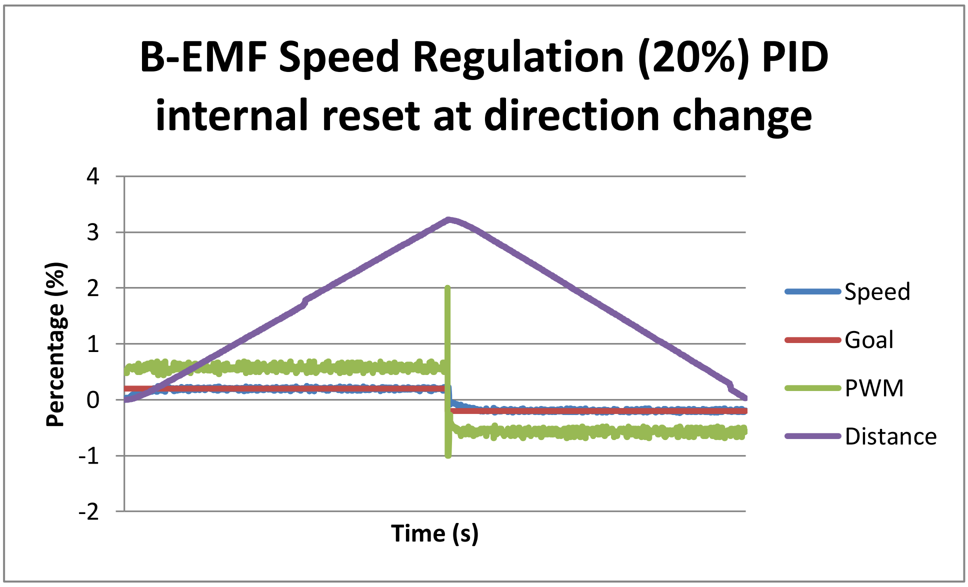


Figure 25

See appendix 4 for the b-emf speed detection library written for the mbed.

By resetting the internal process variables when there's a direction changed, the integral offset correction is cleared so that no additional negative integral is present to slow the direction change down.

7.3 Data logging

The mbed development board has a secondary ARM processor underneath the board, this processor is interfaced to an I2C 2MB non-volatile memory. With the correct library, the mbed can mount the non-volatile memory as a local file system and print runtime data into a text file.

The secondary processor also behaves as a mass storage device when it's plugged into the computer, enabling the user to access the runtime data written onto the I2C memory chip.

A compiler directive "`#ifdef LOG`" `tells the compiler to compile lines responsible for logging data.`

7.4 Balancing data

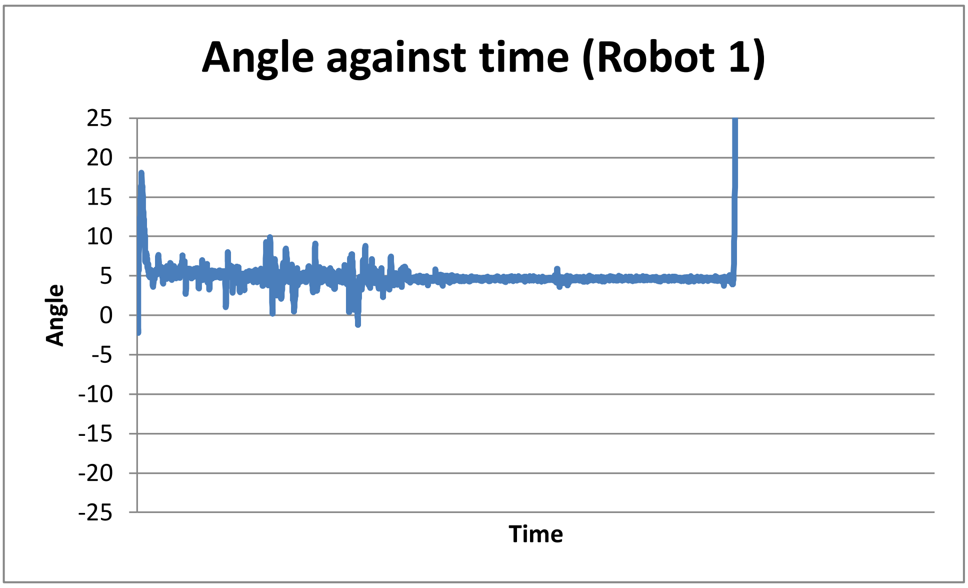


Figure 26

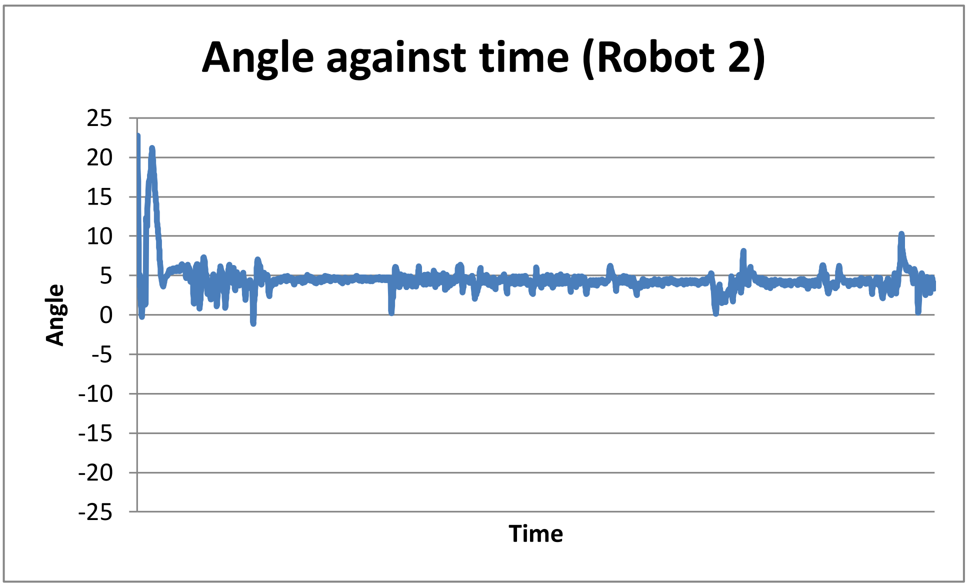


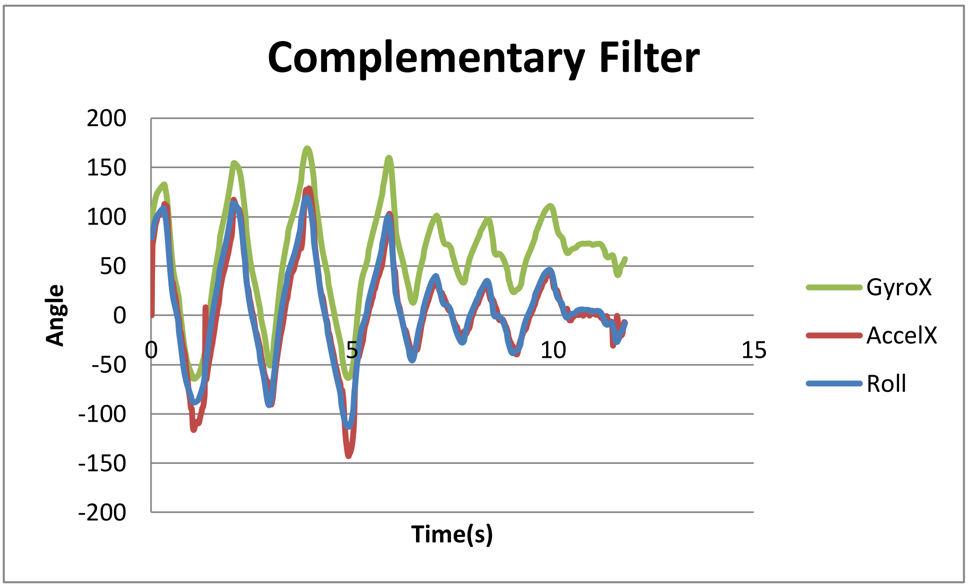
Figure 27

8. Achievements

Work began when the parts (IMU, Motors, Wheels and Diodes) arrived on the 18th November 2011. The very first program written was reading an analogue value from one of the AnalogueIn Pins, a **potentiometer** and a pc was attach to test if the system returned the correct value.

The first revision of the program was completed on the 20th of November 2011, it calculated all three axes from the accelerometers and returned the values to the pc of which virtual horizon software was installed. Unfortunately the baud rate of the program and the microcontroller did not match the program crashed immediately.

On the next day, the **complementary filter** was tested. Because the IMU is always on its side, the filter didn't work until the angles are kept between -180 to +180. The sensor acquisition sequence is then moved to an interrupt routine that executes every 0.02 seconds, this endured that printing real time data over serial doesn't affect data acquisition.



On the 22nd the components were transferred to a fiberglass Vero board, which cut down the size and the weight of the robot significantly. Data logging was also introduced on this day enabling visualisation of the angle over time.

The PID was further tuned on the 25th of November 2011which enabled it to stand upright for around 15 seconds. A single layer PCB was designed, a lot of pins had to be swapped in order to make it a single layer board.

An Exponential term was added to the output of the PID system later on, the system response was greatly enhanced.

An encoder was added to the robot on the 19th of January 2012, the balancing angle is no long impacting the balance. The robot can remain stationary even if extra weight was added to shift the central of gravity, and that the robot is capable of uneven surface compensation.

29th January 2012 – Most sub routines are turned into classes to simplify the program and to prevent complications with variable when multiple instance of the routine is needed.

1st February 2012 - PS3 controller is able to interface with the microcontroller via Bluetooth.

10th February 2012 – Implemented Back-EMF detection in code, written library.

25th February 2012 – Ported the 9dof IMU Arduino library over to mbed.

26th February 2012 – Implemented Tile Compensated magnetic compass.

12th March 2012 – Second robot completed

9. Discussion

Two robots were built to tackle this classic inverted pendulum problem; the first robot was constructed on a Vero board mounted on two 120 rpm motors. It has a 5 degree of freedom IMU of which 2 accelerometer axes and 1 gyro axis were used. A complementary filter was used to fuse the gyro and accelerometer data together and a wheel encoder is used for measuring horizontal displacement. A 1A dual H-bridge motor driver was used to drive the motors. And finally a Bluetooth module was used to interface a ps3 controller to the robot as a remote controller.

There was room for improvement as the motors did not provide enough speed to overcome the acceleration of gravity once the tilt of the robot is greater than 10 degrees. And hence the robot cannot recover from a disturbance great enough to offset the angle to more than 10 degrees. This can be improved by reducing the gear ratio of the gearbox attached to the motor from 100:1 to 30:1. Instead of modifying the working robot, a second prototype was built.

Given the opportunity of building a second robot, different components were selected to ameliorate the ability to recover from a disturbance, an alternative speed sensing technique was employed to lower the overall cost of the robot and an additional magnetic compass is added to detect robot heading.

The back-EMF speed sensing method was used in place of the wheel encoders, reducing the cost from £30 to 8 pence as 4 fixed potential dividers were the only components needed to scale the back-EMF to 0-3.3v for the ADC converters to detect the speed. This is a viable method as the back-EMF is directly proportional to the speed of the motor.

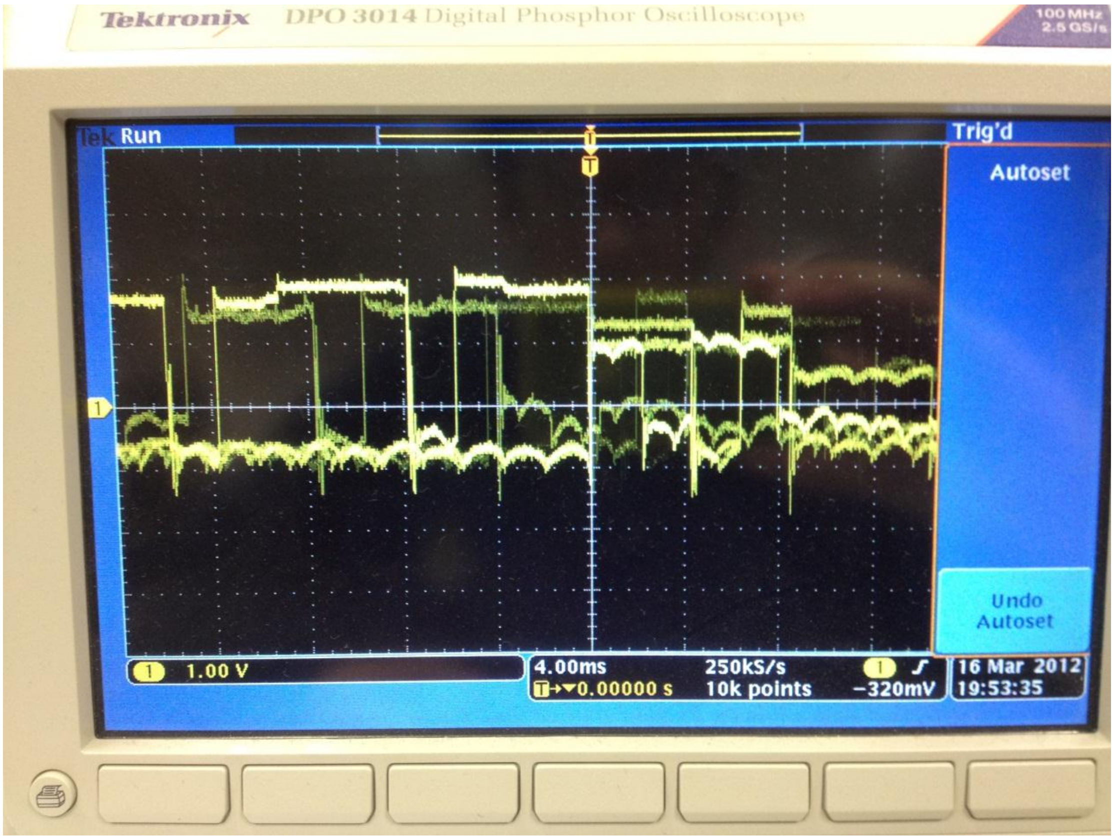


Figure 28: I2C noise when motor is rotating at a low speed

The second robot interfaces with the IMU using I2C; from I2C communication is prone to Electrostatic noise generated by the motors as shown in Figure 28. The problem with digital communication in a noisy environment is that you cannot recover any useable information as the connection simply fails and the only method to reduce motor interference is to add an additional ground plane around the traces and to add several decoupling capacitors (tantalum + electrolytic).

Digital sensors can be programmed with different measurement range and sensitivity. The more expensive IMUS have built in signal conditioning and sensor function. One of the main advantages of I2C is that multiple devices can operate on the same signal line, reducing circuit complexity.

The analogue sensors can operate in a noisy environment as long as filtering is applied; the Gaussian noise can be reduced significantly. Despite the analogue sensor requires an ADC pin for each of the axis, they come with a smaller price tag. And using the ADC built into the microprocessor reduces communication overheads, meaning that more samples can be taken for filtering.

As a result of the noise, the second robot has to move the motors further away from the IMU by adding an extra piece of plastic to the bottom of the robot.

Gyro drift is the same between the two sensors as long as it is calibrated at start.

Infrared communication between the robot and the remote was originally used, but because the infrared decoding program, interrupts from the wheel encoder and serial communication will disrupt the decoding process. To use the infrared as a communication channel, an additional microprocessor is required example of such processor can be an AVR tiny 8. Adding additional processors to the circuit increases complexity and hence should be avoided. Instead, the PS3 Controller is used in place as the communication is managed by the USB dongle.

A balancing robot can stand up using all sorts of balancing algorithm, but to judge the performance, reliability and efficiency of the robot quantifiable measurements has to be made to compare the robots. Efficiency can be measured by the amount of power required to stay balanced which is correlated to the amount of motor movements. An efficient robot should always be making minor adjustments instead of large corrections.

Reliability of the robot can be measured by the magnitude of disturbance the robot can recover from and the duration of the balance.

Performance can be measured by the maximum velocity of the robot whilst remaining upright.

Robot	**Performance**	**Reliability**	**Efficiency**
Prototype 1	Slow movement	Stands up indefinitely (battery not flat)	Less than 2 degree fluctuation
Prototype 2	Can travel 3 times faster than prototype 1	Stands up indefinitely (battery not flat)	Around 7 degree fluctuation

Figure 29

It turns out that the mbed pins have no protection mechanisms against voltages higher than 5v, when the positive lead on the lithium ion polymer battery pack came loose, the mbed chip was fried. Seeing that the magic interface chip is still functional, by replacing theLPC1768 chip should restore the mbed to a working order.

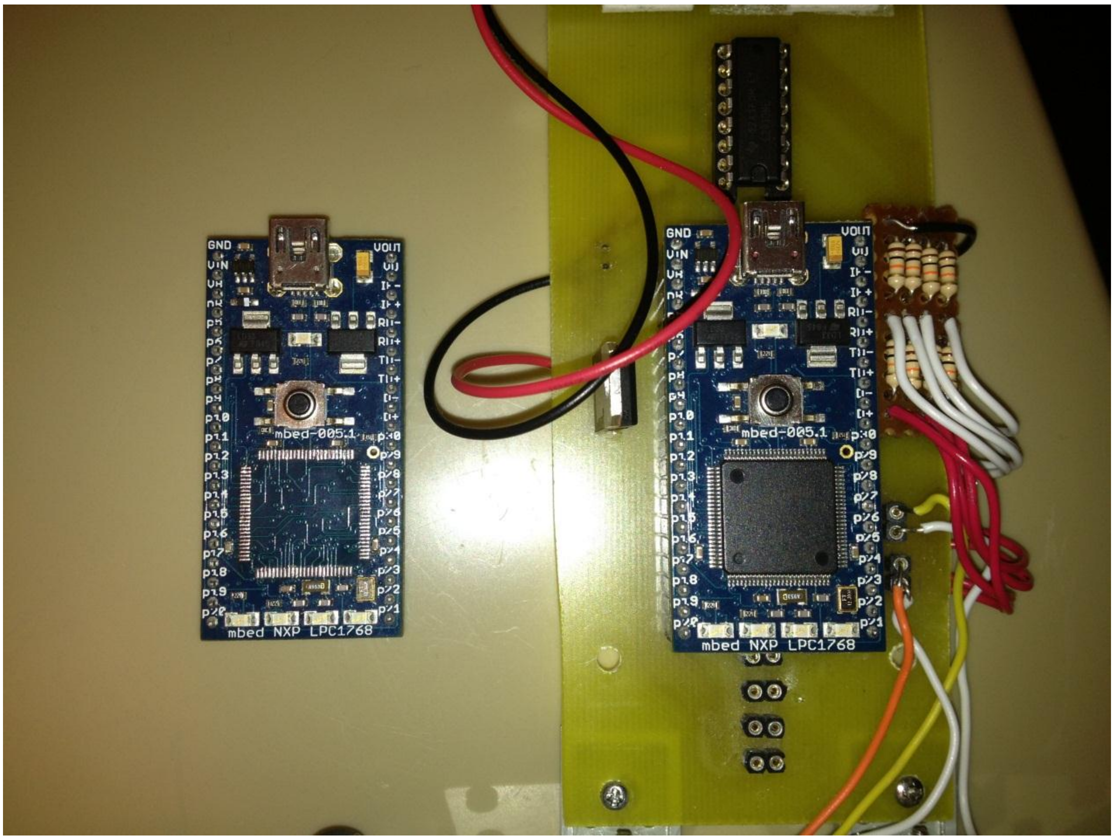


Figure 30

The replacement chip costs £7 if ordered in bulk, but if a single chip is ordered farnell charges a hefty 12 pounds shipping totaling at £19. Because the cost is considerably higher than £7 and that replacing the chip does not guarantee the restoration of the mbed, the repair has been abandoned.

SHOPPING CART

Qty	Item	Price
1 x	IMU Analog Combo Board - 5 DOF IDG500/ADXL335	£26.48
2 x	50:1 Micro Metal Gearmotor HP	£13.00 x2
1 x	Pololu Micro Metal Gearmotor Bracket Extended Pair	£3.79
1 x	Pololu Wheel 32x7mm Pair - Black	£5.50
1 x	Resistors 10k 1/8W (pack 20)	£0.20
1 x	Stripboard 95 x 127mm	£1.75
1 x	Pololu 5V Boost Regulator NCP1402	£4.96
1 x	Baby Orangutan B-328 Robot Controller	£16.99
1 x	Enclosed Battery Box 2x AAA	£0.40

Sub-Total (ex vat): £71.72
VAT 20%: £14.34
Total: £86.06

View Cart | Checkout

Figure 31 [20]

(Alternative Low cost version that runs on normal AA or AAA batteries, Baby Orangutan Controller and low cost analogue IMU)

10. Further Improvements

At present, the balance controller runs on an expensive development board. By designing a PCB for the actual SMD LPC1768 chip the cost of the microcontroller can be reduced from £49 down to £7 pounds. Or an alternative AVR controller can be used such as the ATMEGA328p.

A larger balancing robot can be built with solar panels, as long as the balancing algorithm is efficient enough, the motors shouldn't draw too much energy while remaining stationary.

Localization and mapping can be implemented with several ultrasonic sensors for mapping and an additional behavior for wandering around, given a high resolution encoder.

11. Conclusion

The PID implantation of a balance controller, did not produce too much overshoot to compromise the efficiency of the robot, this is shown in figure 26 and 27. Both of the robot is balancing with minor correction movements except robot is slight more energy efficient than the other. They both succeed at balancing with different components Therefore, the design was successful.

In both robots, wheel velocity detection exists to correct offsets in the balance angle which prevents the robot from drifting forward or backwards depending on the direction offset.

From figure 24, conclusion can be drawn that B-EMF speed feedback is reliable and works in both directions.

Several of the sensor upgrade goals was skipped as the more advanced sensors worked, mainly because the simple weighted average sensor fusion mechanism was working flawlessly.

Infrared remote controls can cause complications due to amount of processor time required to decode the signal correctly and Bluetooth communication are just as cost effective as PS3 controllers does not have to be in the cost.

The project marginally meets the £100 requirement that was set out at the start of the project, as long as the parts are ordered in bulk the price will fall below £100, another cheaper alternative is listed in figure 30.

The usage of the Gantt chart has greatly aided the planning of the project as it helps visualising the tasks and deadlines for you to work towards. Most of the task has been fulfilled, except for several optional upgrades that might break the robot. As soon as the robot was working, no further hardware modification should be performed unless easily reversible. To get around that a second robot was built although extra time was consumed during the PCB etching process. The main discrepancies occurred when the parts take longer than expected to ship, and that PCB etching takes longer than expected

12. Context

This project correlates to the following fields:

- Control systems (PID)
- Embedded systems (AVR and ARM)
- Navigation systems and avionics (telemetry using the Attitude and heading reference system)
- Telecommunications (Bluetooth)
- Data Acquisition (IMU)
- Segway PT

There are legal and safety issues with the Segway PT which is the industry that corresponds to this project the most. (An RTA spokesman confirmed they were illegal on roads "or road related areas" because they don't comply with vehicle safety standards: "In simple terms, riders are way too exposed to mix with general traffic on a road and too fast, heavy and consequently dangerous to other users on footpaths or cycle paths.") [21]

The source codes either fall under the gnu license or the creative commons category. The source code is either written by me or it is a modification of another program that falls under the GNU license. Programs that fall under the GNU license are open-source and can be freely modified even for commercial use.

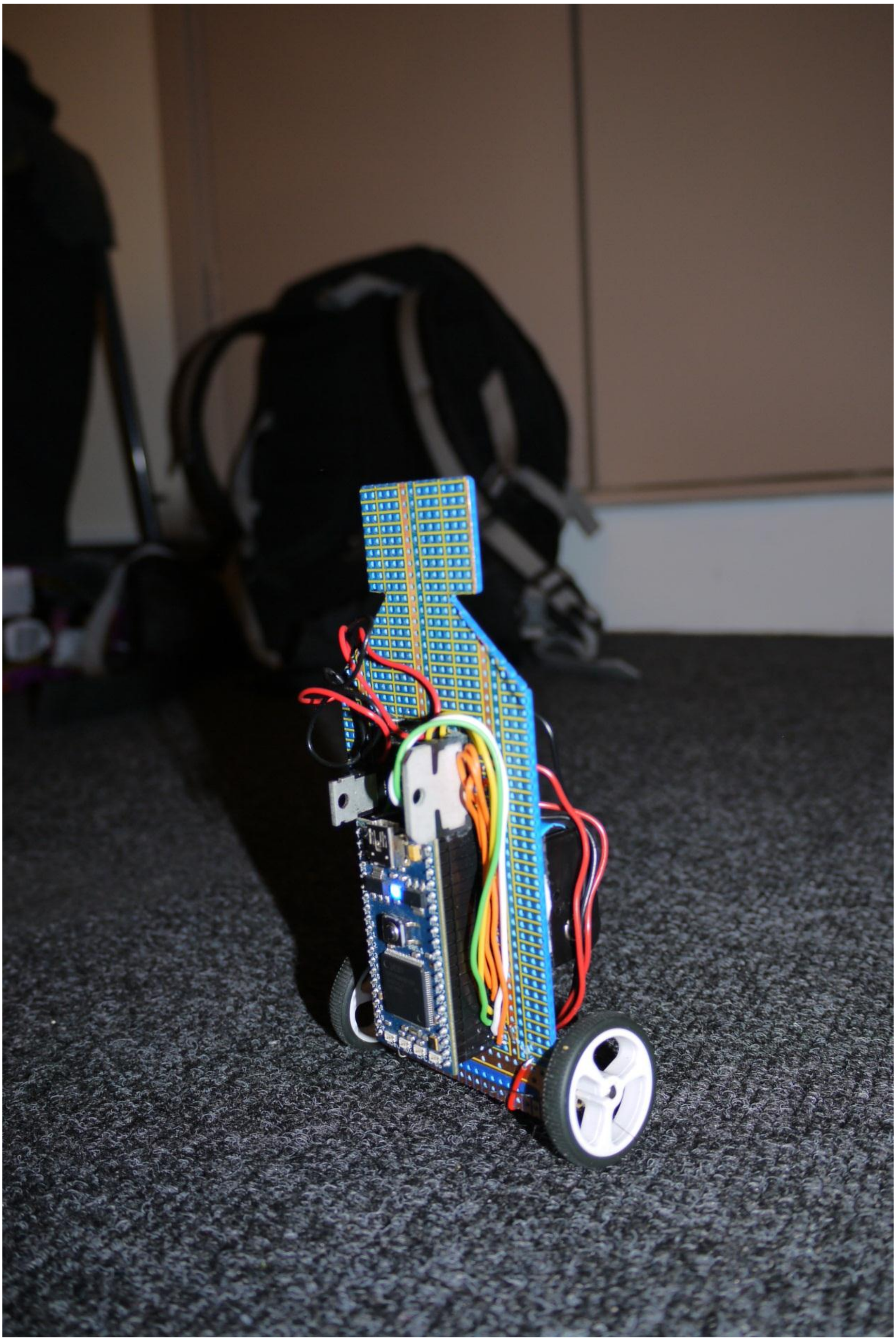


Figure 32: Picture was taken on the 25th November 2011

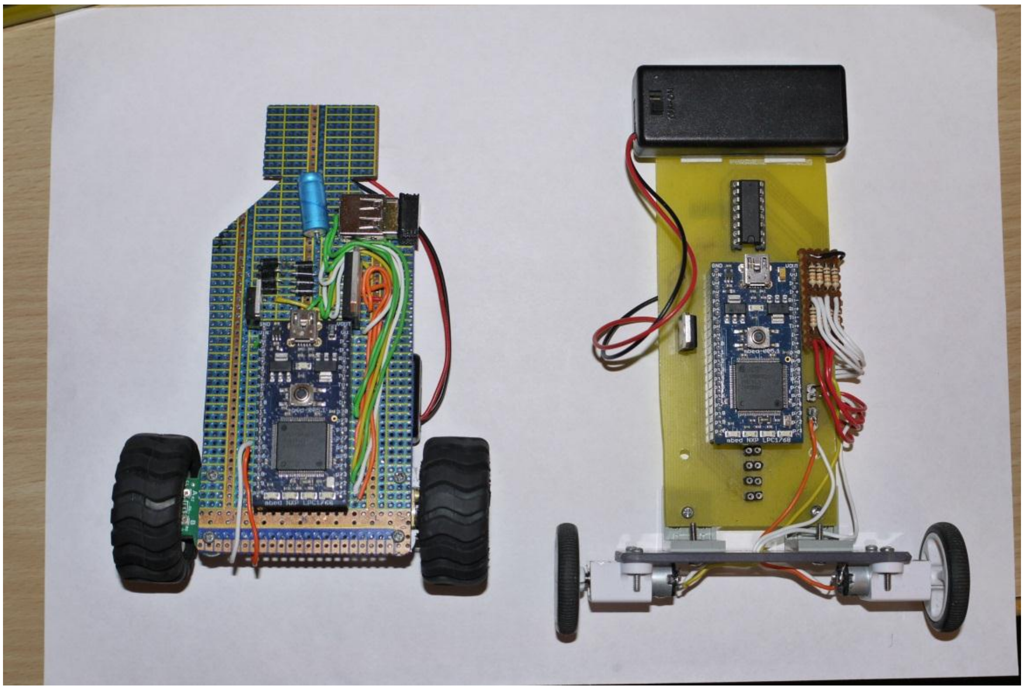


Figure 33: Picture was taken on the 12th of April 2012

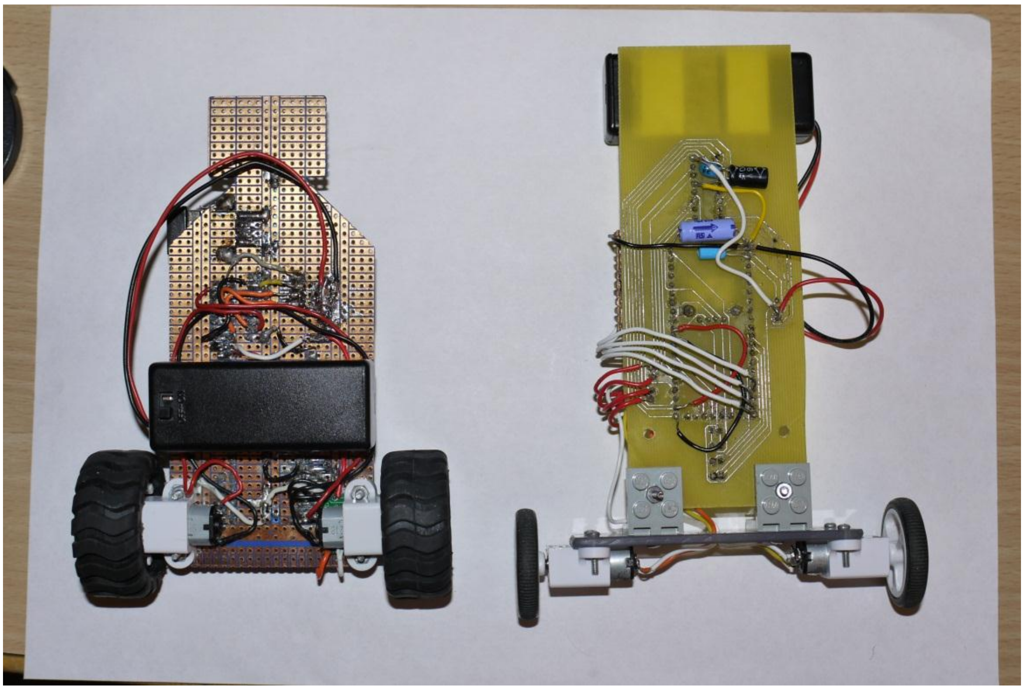


Figure 34: Picture was taken on the 12th of April 2012

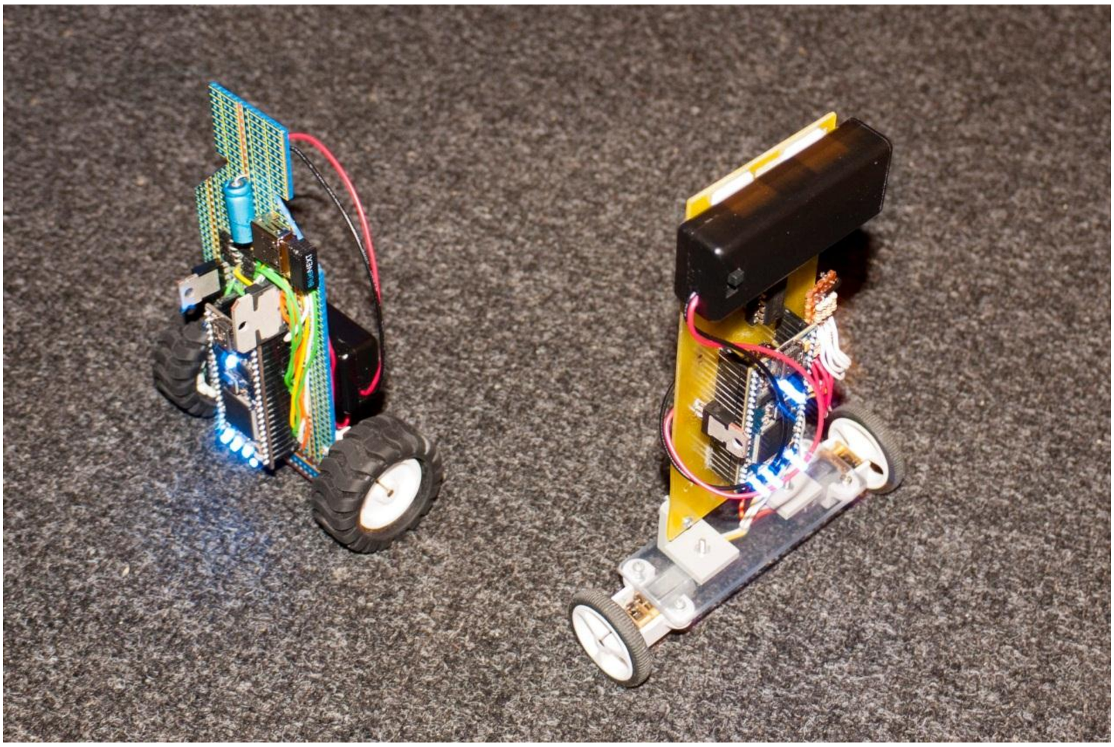


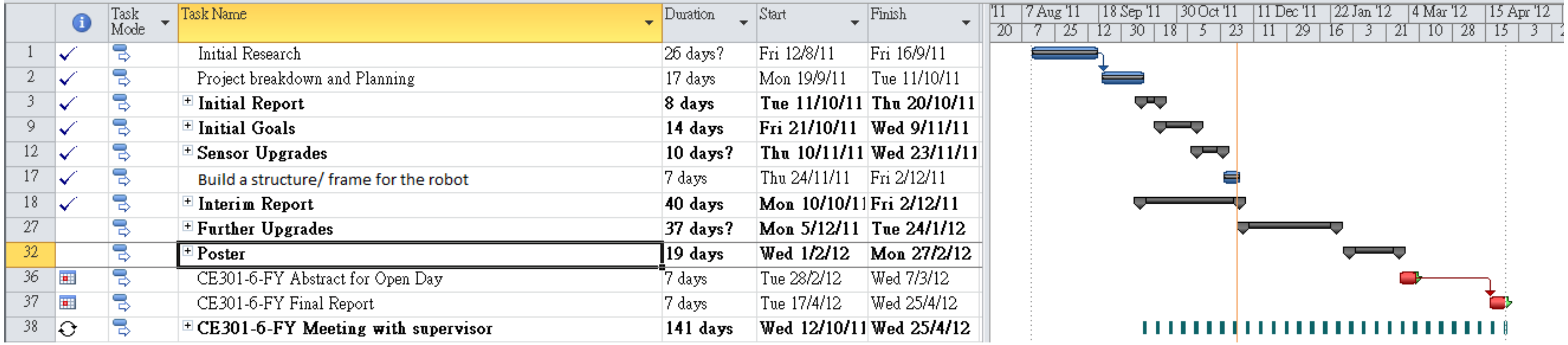
Figure 35: Picture was taken on the 31st of March 2012

13. Project Planning

Project planning mainly done on a Gant Chart, it is a very useful project management tool for visualising task, progress and milestones on the calendar. This will enable specific steps of the project to be completed on time, or before the planned completion date. If so, then more time can be used on the next task, without disrupting the set completion date of the project or bringing it forward. However, should the task take longer than usual, the chart will help with the planning of reducing the waiting time for another task.

(Figure 5.1-5.4) Below are Gant charts generated using Microsoft Project for the duration of this project.

Project start date	12^{th} August 2011
Project end date	25^{th} April 2012
Project duration	257 days / 36 weeks



	Task Mode	Task Name	Duration	Start	Finish
1		Initial Research	26 days?	Fri 12/8/11	Fri 16/9/11
2		Project breakdown and Planning	17 days	Mon 19/9/11	Tue 11/10/11
3		+ **Initial Report**	**8 days**	**Tue 11/10/11**	**Thu 20/10/11**
9		+ **Initial Goals**	**14 days**	**Fri 21/10/11**	**Wed 9/11/11**
12		+ **Sensor Upgrades**	**10 days?**	**Thu 10/11/11**	**Wed 23/11/11**
17		Build a structure/ frame for the robot	7 days	Thu 24/11/11	Fri 2/12/11
18		+ **Interim Report**	**40 days**	**Mon 10/10/11**	**Fri 2/12/11**
27		+ **Further Upgrades**	**37 days?**	**Mon 5/12/11**	**Tue 24/1/12**
32		+ **Poster**	**19 days**	**Wed 1/2/12**	**Mon 27/2/12**
36		CE301-6-FY Abstract for Open Day	7 days	Tue 28/2/12	Wed 7/3/12
37		CE301-6-FY Final Report	7 days	Tue 17/4/12	Wed 25/4/12
38		+ **CE301-6-FY Meeting with supervisor**	**141 days**	**Wed 12/10/11**	**Wed 25/4/12**

Figure 36: Summary view of all the tasks

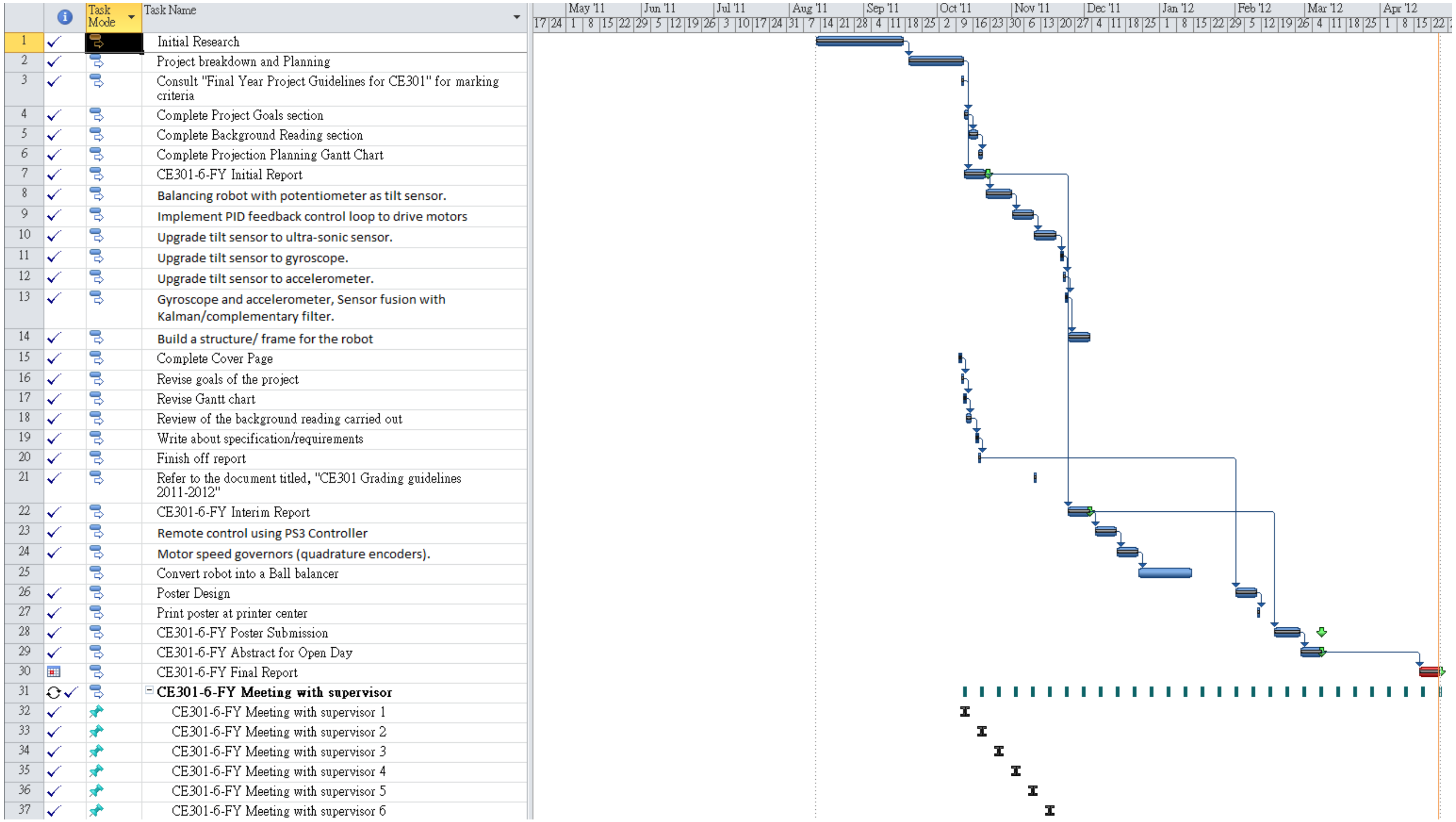
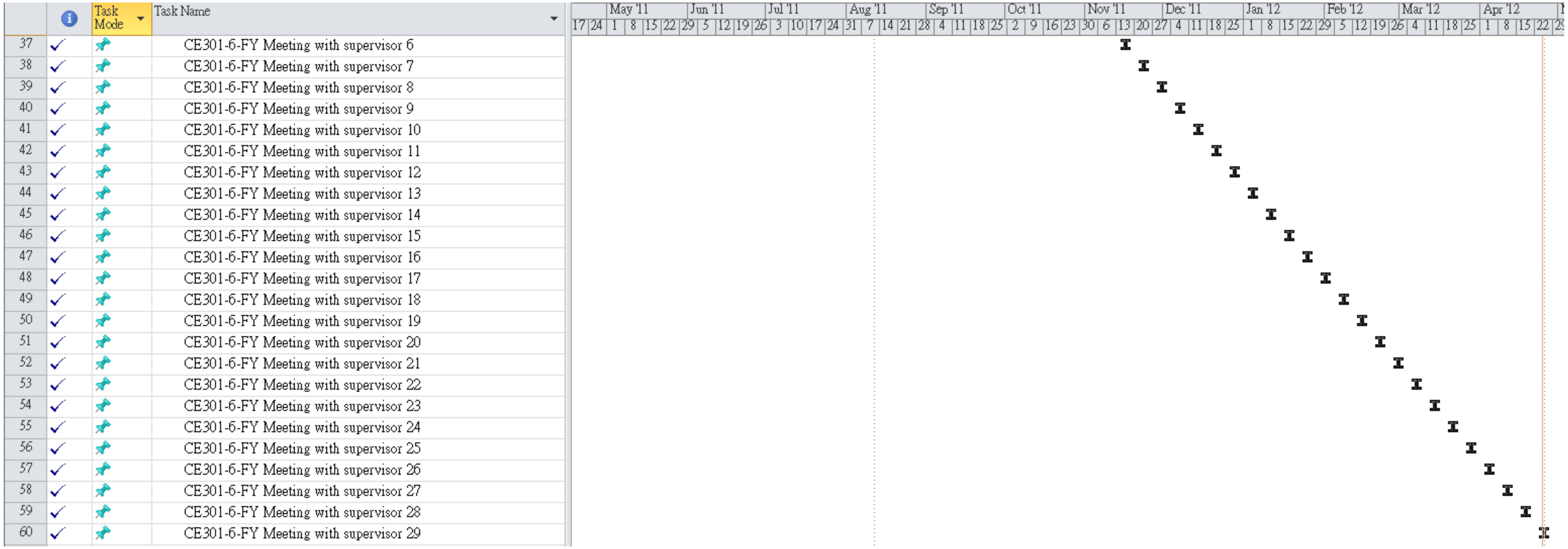


Figure 37: Full project duration overview (Task 1-37)



	Task Name
37	CE301-6-FY Meeting with supervisor 6
38	CE301-6-FY Meeting with supervisor 7
39	CE301-6-FY Meeting with supervisor 8
40	CE301-6-FY Meeting with supervisor 9
41	CE301-6-FY Meeting with supervisor 10
42	CE301-6-FY Meeting with supervisor 11
43	CE301-6-FY Meeting with supervisor 12
44	CE301-6-FY Meeting with supervisor 13
45	CE301-6-FY Meeting with supervisor 14
46	CE301-6-FY Meeting with supervisor 15
47	CE301-6-FY Meeting with supervisor 16
48	CE301-6-FY Meeting with supervisor 17
49	CE301-6-FY Meeting with supervisor 18
50	CE301-6-FY Meeting with supervisor 19
51	CE301-6-FY Meeting with supervisor 20
52	CE301-6-FY Meeting with supervisor 21
53	CE301-6-FY Meeting with supervisor 22
54	CE301-6-FY Meeting with supervisor 23
55	CE301-6-FY Meeting with supervisor 24
56	CE301-6-FY Meeting with supervisor 25
57	CE301-6-FY Meeting with supervisor 26
58	CE301-6-FY Meeting with supervisor 27
59	CE301-6-FY Meeting with supervisor 28
60	CE301-6-FY Meeting with supervisor 29

Figure 38: Full project duration overview (Task 37-60)

14. Works Cited

[1] J. Woods, University of Essex. [Online].

[2] J. Tam, "CE301: Interim Report," 2010.

[3] D. P. Anderson, "nBot Balancing Robot," SMU (c) 1993-2010 David P. Anderson , 01 3 2010. [Online]. Available: http://www.geology.smu.edu/~dpa-www/robo/nbot/. [Accessed 20 10 2011].

[4] S. Kondō, Primate morphophysiology, locomotor analyses, and human bipedalism., Tokyo: University of Tokyo Press, 1985.

[5] D. M. Ling and W. W. Kong, "Two Wheeled Balancing Robot," *Robot head to toe,* vol. 4, pp. 1-3, 2010.

[6] J.-S. Hu, J.-J. Wang and G.-C. Sun, "Self-balancing control and manipulation of a glove puppet robot on a twowheel mobile platform," *in Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems,* pp. 424-425, 2009.

[7] S. Colton, "The Balance Filter," 25 6 2007. [Online]. Available: http://web.mit.edu/scolton/www/filter.pdf. [Accessed 20 10 2011].

[8] Acroname Robotics, "Back-EMF Motion Feedback," 07 08 2006. [Online]. Available: http://www.acroname.com/robotics/info/articles/back-emf/back-emf.html.

[9] C. Yong and F. C. Kwong, "Wíreless Controlled Two Wheel Balancing Robot," *International Journal of Network and Mobile Technologies,* vol. 2, no. 2, pp. 88-109, 2011.

[10] G. R. P. a. J. E. S. J. J. Uicker, Theory of Machines and Mechanisms, New York: Oxford University Press, 2003.

[11] Wikipedia, “Bluetooth,” [Online]. Available: http://en.wikipedia.org/wiki/Bluetooth.

[12] Dale, “Simple Analog Balancing Bot,” 2008. [Online]. Available: http://www.wa4dsy.com/robot/balancing-robot/analog-balancing-bot.

[13] mbed, “mbed NXP LPC1768 HandBook,” [Online]. Available: http://mbed.org/handbook/mbed-NXP-LPC1768. [Accessed 30 12 2011].

[14] Sparkfun Electronics, “IMU Analog Combo Board - 5 Degrees of Freedom IDG500/ADXL335,” [Online]. Available: http://www.sparkfun.com/products/9268. [Accessed 30 11 2011].

[15] Sparkfun Electronics, “Micro Metal Gearmotor 100:1,” [Online]. Available: http://www.sparkfun.com/products/8910. [Accessed 30 11 2011].

[16] Sparkfun Electronics, “Full-Bridge Motor Driver Dual - L298N,” [Online]. Available: http://www.sparkfun.com/products/9479. [Accessed 30 11 2011].

[17] Hobby Electronics, “Mini Bluetooth 2.0 USB Adapter,” [Online]. Available: http://www.hobbytronics.co.uk/bluetooth-usb-dongle.

[18] TINYCLR WIKI, “PS3 Controller,” [Online]. Available: http://wiki.tinyclr.com/index.php?title=PS3_Controller#Bluetooth.

[19] mbed, “PID - Cookbook,” [Online]. Available: http://mbed.org/cookbook/PID.

[20] HobbyTronics, [Online]. Available: http://www.hobbytronics.co.uk/.

[21] Sydney Morning Herald, “ "Segway test: ride a mock horse",” 3 January 2008. [Online]. Available: http://www.smh.com.au/news/technology/segway-test-ride-a-mock-horse/2008/01/02/1198949941691.html?page=3. [Accessed 25 April 2012].

[22] WhaFat Technology, [Online]. Available: http://www.ultrafire.net/products.asp?enBigClassName=Battery. [Accessed 30 11

2011].

[23] Analog devices, “Small, Low Power, 3-Axis ±3 g,” 2009. [Online]. Available: http://www.sparkfun.com/datasheets/Components/SMD/adxl335.pdf. [Accessed 30 11 2011].

[24] InvenSense, “ Integrated Dual-Axis Gyr,” 2008. [Online]. Available: http://www.sparkfun.com/datasheets/Components/SMD/Datasheet_IDG500.pdf. [Accessed 30 11 2011].

[25] Sparkfun Electronics, “Wheel 32x7mm,” [Online]. Available: http://www.sparkfun.com/products/8901. [Accessed 30 11 2011].

[26] Sparkfun Electronics, “Voltage Regulator - 5V,” [Online]. Available: http://www.sparkfun.com/products/107. [Accessed 30 11 2011].

[27] D. J. R. White, “System Dynamics Introduction to the Design and Simulation of Controlled Systems,” University of Massachusetts Lowell, 1997. [Online]. Available: http://www.profjrwhite.com/system_dynamics/sdyn/s7/s7invp1/s7invp1.html. [Accessed 19 10 2011].

[28] D. Harris, “Drift Calculations,” in *Flight instruments & automatic flight control systems*, Wiley-Blackwell, 2003, p. 44.

[29] D. P. Anderson, “nBot Balancing Robot,” [Online]. Available: http://www.geology.smu.edu/~dpa-www/robo/nbot/.

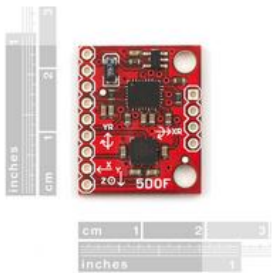
Appendix 1 - Component Specification

Power source –2x LC 14500(£4.9)

[22]

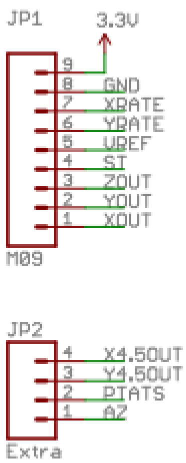
3.6V 900mAh 3.24Wh

2x 3.6V 900mAh 3.24Wh IMU - 5DOF IMU by Sparkfun Electronics (£27.99)


[14]

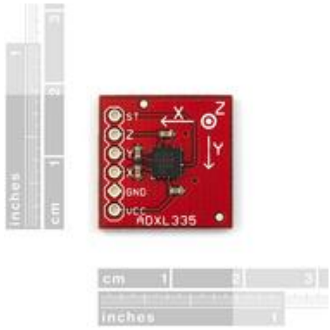
Contains two sensors:	IDG500 and ADXL335
Small foot print :	less than 1 square inch
Weight :	2g

Cheaper than buying the sensors separately, ADXL335 costs £15.53 and an IDG500 costs £24.87, at a total of £40.4



5DOF	mbed
JP1-1	20
JP1-2	19
JP1-3	18
JP1-5	17
JP1-6	16
JP1-7	15
JP2-1	30

Accelerometer - ADXL335

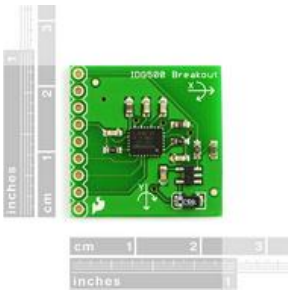


[14]

Parameter	Conditions	Min	Typ	Max	Unit
SENSOR INPUT	Each axis				
Measurement Rang		±3	±3.6		g
SENSITIVITY (RATIOMETRIC)²	Each axis				
Sensitivity at XOUT, YOUT, ZOUT		270	300	330	mV/g
ZERO g BIAS LEVEL (RATIOMETRIC)					
0 g Voltage at XOUT, YOUT	VS = 3 V	1.35	1.5	1.65	V
0 g Voltage at ZOUT	VS = 3 V	1.2	1.5	1.8	V

[23]

Gyroscope – IDG500



Parameter	Conditions	Min	Typ	Max	Unit
SENSITIVITY					
Full-Scale Range	At X-OUT and Y-OUT		±500		°/s
Sensitivity	At X-OUT and Y-OUT		2.0		mV/°/s
REFERENCE					
Voltage (VREF)			1.35		V
ZERO-RATE OUTPUT					
Static Output (Bias)	Factory Set		1.35		V

[24]

2x Motor + Gearbox – Pololu (£9.93)

[15]

100:1 Gear ratio

120rpm @ 6V

30mA @ 6V

420mA stall current @ 6V

13 oz inches torque @ 6V

Wheels - Pololu 32x7mm (£4.33)

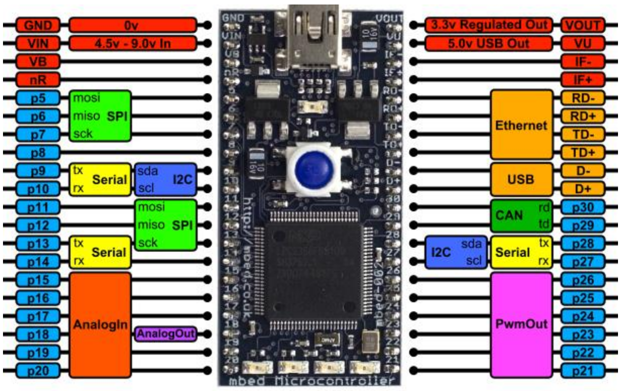
[25]

Silicone tire

Wheel Diameter: 32mm

Tire trackwidth: 7mm

Microcontroller - mbed NXP LPC1768 (£40) [13]



This mbed Microcontroller is based on a Cortex-M3 Core running at 96MHz, with 512KB FLASH, 64KB RAM and a load of interfaces including Ethernet, USB Device, CAN, SPI, I2C and other I/O.

Package

- 40-pin DIP package
- 0.1. pitch, 0.9. pin spacing
- 54mm x 26mm

Power

- Powered by USB or 4.5v - 9.0v appiled to VIN
- <200mA (100mA with Ethernet disabled)
- Real-time clock battery backup input VB
- 1.8v - 3.3v Keeps Real-time clock running
- Requires 27uA, can be supplied by a coin cell
- 3.3v regulated output on VOUT to power peripherals
- 5.0v from USB available on VU (only available when USB is connected!)
- Current limited to 500mA
- Digital IO pins are 3.3v, 40mA each, 400mA max total

Pins

- Vin - External Power supply to the board
  - 4.5v-9v, 100mA + external circuits powered through the Microcontroller
- Vb - Battery backup input for Real Time Clock
  - 1.8v-3.3v, 30uA
- nR - Active-low reset pin with identical functionality to the reset button.
- Pull up resistor is on the board, so it can be driven with an open collector
- IF+/- - Reserved for Future use

Voltage Regulator – LM7805 (£0.78)

[26]

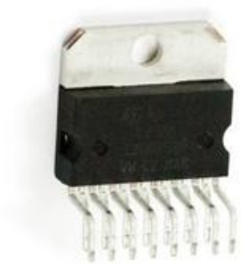
Input Voltage V = 7 V to 25 V

Output Voltage = 5V

Output Current up to 1.5 A

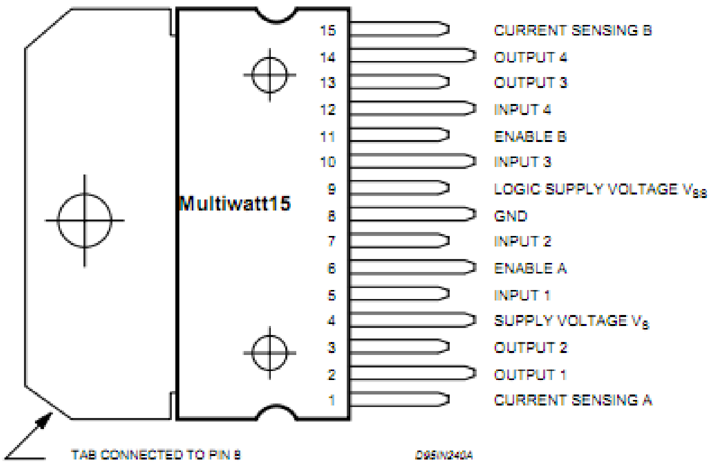
Full-Bridge Motor Driver Dual - L298N (£1.84)

[16]



Parameter	Min	Typ	Max	Unit
Supply Voltage (pin 4)	+2.5		46	V
Logic Supply Voltage (pin 9	4.5	5	7	V
Input Low Voltage	-0.3		1.5	V
Input High Voltage	2.3		1.5	V

PIN CONNECTIONS (top view)



Inputs		Function
V_{en} = H	C = H ; D = L	Forward
	C = L ; D = H	Reverse
	C = D	Fast Motor Stop
V_{en} = L	C = X ; D = X	Free Running Motor Stop

L = Low H = High X = Don't care

Appendix 2 –First Robot Program

```
#define LOG //Compiler directive to enable datalogging
//****************************************************************************/
// Includes
//****************************************************************************/
#include "mbed.h"
#include "IMU.h"
#include "PID.h"
#include "QEI.h"
#include "iostream"
#include "USBHost.h" // Peter Barratt's USB Bluetooth Dongle code
#include "Utils.h"
#include "ps3.h"
//****************************************************************************/
// Defines
//****************************************************************************/
#define RATE      0.02
#define Kc        14
#define Ti        0.15
#define Td        0.001

#define ExpGain   0.5
#define GOAL      5.5//7

#define OVERSHOOT 0.5

#define Kc1       4
#define Ti1       0.0
#define Td1       30

//PC Communication
Serial pc(USBTX, USBRX);

//File
```

```
#ifdef LOG
LocalFileSystem local("local");
FILE* fp;
#endif

IMU IMU(p15, p19, p18, p30);
PID Controller(Kc, Ti, Td, RATE);
PID ControllerQEI(Kc1, Ti1, Td1, RATE);
QEI wheel (p13, p14, NC, 48, QEI::X4_ENCODING);

Timer t;

//****************************************************************************/
// Outputs
//****************************************************************************/
DigitalOut leftDirection[2] = {p22, p23};
DigitalOut rightDirection[2] = {p25, p26};
DigitalOut LED[] = {LED1,LED2,LED3,LED4};

PwmOut leftMotor(p21);
PwmOut rightMotor(p24);

//****************************************************************************/
// Declare global variables
//****************************************************************************/
float PWM, leftpwm, rightpwm;
float TurnDifferential = 0;
float Position = 0;
float timer;
float encoder;
int scale; //map position between -1000 and 1000 for position pid

void initializeMotors(void) {

    leftMotor.period_us(30);
    leftDirection[0] = 1;
    leftDirection[1] = 0;
```

```
    leftMotor = 0;

    rightMotor.period_us(30);
    rightDirection[0] = 1;
    rightDirection[1] = 0;
    rightMotor = 0;

}

void DriveMotors() {
    TurnDifferential = -getDifferential() * 0.3; // Scaledown PS3 input by 70% to avoid
falling

    leftpwm  = PWM - TurnDifferential;
    rightpwm = PWM + TurnDifferential;

    //set motor direction and PWM duty cycle
    if (leftpwm  < 0) {
        leftDirection[0] = 1;
        leftDirection[1] = 0;
        leftMotor  = ExpGain*exp(-leftpwm);
    }
    if (leftpwm  > 0) {
        leftDirection[0] = 0;
        leftDirection[1] = 1;
        leftMotor  = ExpGain*exp( leftpwm);
    }
    if (rightpwm < 0) {
        rightDirection[0] = 1;
        rightDirection[1] = 0;
        rightMotor = ExpGain*exp(-rightpwm);
    }
    if (rightpwm > 0) {
        rightDirection[0] = 0;
        rightDirection[1] = 1;
        rightMotor = ExpGain*exp( rightpwm);
    }
```

```
}

void initializePidControllers(void) {

    Controller.setInputLimits (-180, 180);
    Controller.setOutputLimits(-1.0, 1.0);
    Controller.setBias(0.0);
    Controller.setMode(AUTO_MODE);

    ControllerQEI.setInputLimits (-1000, 1000);
    ControllerQEI.setOutputLimits(-10, 10);
    ControllerQEI.setBias(0.0);
    ControllerQEI.setMode(AUTO_MODE);
}

void PID() {
    //Position Pid (pv: position, sp: ps3 analogue stick integration)
    Position = Position + (getSpeed()*t.read()); //integrate speed
    //PID input scaling -scale*500
    if ( (Position*40 - scale*500) >  500) scale++;
    if ( (Position*40 - scale*500) < -500) scale--;
    //pc.printf("%f", Position-n*50);
    ControllerQEI.setSetPoint(Position*40 - scale*500);
    ControllerQEI.setProcessValue(-wheel.getPulses() - scale*500);
    ControllerQEI.setInterval(t.read());
    encoder = ControllerQEI.compute();

    //Angle PID (pv: angle from IMU, sp: preset angle + offset from encoder pid)
    Controller.setProcessValue(IMU.getRoll());
    if (IMU.getRoll() > (GOAL+encoder))
Controller.setSetPoint(GOAL-OVERSHOOT+encoder);
    if (IMU.getRoll() < (GOAL+encoder))
Controller.setSetPoint(GOAL+OVERSHOOT+encoder);

    Controller.setInterval(t.read());
    PWM = Controller.compute();
```

```
    timer = t.read();
    t.reset();

    if (abs(IMU.getRoll()) > 45) {  //Detect fall - stop motors
        PWM = 0;
        leftDirection[0]  = 0;
        leftDirection[1]  = 0;
        rightDirection[0] = 0;
        rightDirection[1] = 0;
    }

    if (PWM>0)  LED[2] = 1;
    else        LED[2] = 0;
    if (PWM<0)  LED[3] = 1;
    else        LED[3] = 0;
}

//****************************************************************************//
// Main Function
//****************************************************************************//
int main() {
    pc.baud(460800);
    wait(2); // warm up time
#ifdef LOG
    fp = fopen("/local/pidtest.csv", "w");
    fprintf(fp, "GyroX,AccelX,Roll,Goal,PWM,Timer,Position\n");
#endif
    //initialise system
    USBInit();
    initializeMotors();
    IMU.initialise();
    initializePidControllers();
    LED[0] = 1; //Indicate finish initialisation

    wait(3); // Give time for user to rotate robot upright

    LED[1] = 1; //System in operation
```

```
    t.start();
    while (1) {
        IMU.update();
        USBLoop();
        PID();
        DriveMotors();

        //Raw data
        //pc.printf("X Accel: %fG, Y Accel: %fG, Z Accel: %fG, X Gyro %f, Y Gyro %f,
Gyro_offset: %fV\r\n", X, Y, Z, Xrate, Yrate, Gyro_offset);

        //Processed Data
        //pc.printf("angle_x: %.2f, angle_y: %.2f, angle_z: %.2f, gyro_x: %.2f, gyro_y:
%.2f\r\n", AccAngleX, AccAngleY, AccAngleZ, GyroAngleX, GyroAngleY);

        //Fused Data
        //pc.printf("Roll: %.2f, Goal: %.2f, Timer1: %f, Timer2: %f\r\n", Roll, GOAL,
timer1, timer);
        //pc.printf("%.2f\r\n", Roll);

        //pc.printf("%i\n", wheel.getPulses());

#ifdef LOG
        fprintf(fp, "%.2f,%.2f,%.2f,%.2f,%.2f,%f,%i\n", IMU.getGyrox(), IMU.getAccelx(),
IMU.getRoll(), GOAL+encoder, PWM*100, timer,-wheel.getPulses());
#endif

        pc.printf("!ANG:%.2f,%.2f,%.2f\r\n", 0, IMU.getRoll(), 0);
        wait(0.0);
    }
}
```

Appendix 3 –Second Robot Program

```
//Angle PID defines

#define BemfSampleRate 0.02
```

```
#define GOAL            4.5
#define EXP             0.44
#define OVERSHOOT       0.0

#define AngleP          20
#define AngleI          0.15//0.15
#define AngleD          0.1//0.1

#define Kc1             3.2//3.2
#define Ti1             0
#define Td1             8.5//7.3

#include "mbed.h"
#include <imu9dof.h>
#include <motordriver.h>
#include <PID.h>
#include <bemf.h>

float RATE = 0.2;
float position;
float output;

Serial pc(USBTX, USBRX); // tx, rx

LocalFileSystem local("local");
FILE *fp = fopen("/local/out.csv", "w");

Timer t;

minimu9 IMU9( p28, p27 );

//Motor  leftmotor(p23,p22,p21); //pwm, fwd, bwd
Motor  leftmotor(p23,p30,p29); //FOR BROKEN MBED
Motor rightmotor(p26,p25,p24);
PID PIDangle(AngleP,AngleI,AngleD,RATE);
PID PIDposition(Kc1,Ti1,Td1,RATE);
```

```
BEMF sensor(p23,p19,p20); //PWM, FW, BW
BEMF sensor1(p26,p18,p17); //PWM, FW, BW

DigitalOut LED[] = {LED1,LED2,LED3,LED4};

void initialise() {
    PIDangle.setInputLimits  (-180,180);
    PIDangle.setOutputLimits (-1.0,1.0);
    PIDangle.setBias(0.0);
    PIDangle.setMode(AUTO_MODE);

    PIDposition.setInputLimits(-100, 100);
    PIDposition.setOutputLimits(-50, 50);
    PIDposition.setBias(0.0);
    PIDposition.setMode(AUTO_MODE);

    sensor.enable(BemfSampleRate);
    sensor1.enable(BemfSampleRate);
}

void computePID(void) {
    // position
    PIDposition.setSetPoint(0);
    PIDposition.setProcessValue(-(sensor.getPosition()+sensor1.getPosition())/2);
    PIDposition.setInterval(RATE);
    position = PIDposition.compute();

    // Angle
    if (IMU9.getPitch()>(GOAL+position))
        PIDangle.setSetPoint(GOAL-OVERSHOOT+position);

    if (IMU9.getPitch()<(GOAL+position))
        PIDangle.setSetPoint(GOAL+OVERSHOOT+position);

    //PIDangle.setSetPoint(GOAL);
    PIDangle.setProcessValue(IMU9.getPitch());
```

```
    PIDangle.setInterval(RATE);
    output = PIDangle.compute();


    if (output < 0)
        output = -EXP*exp(-output);
    else
        output = EXP*exp(output);

    if (abs(IMU9.getPitch())>70) {
        output = 0;
        fclose(fp);
    }
}



int main() {
    pc.baud(460800);
    wait(1);
    initialise();
    IMU9.calibrateGyro();
    LED[0] = 1; //Indicate finish initialisation
    wait(2);    //Give time for user to rotate robot upright
    LED[1] = 1; //System in operation
    t.start();

    while (1) {
        //50 Hz
        if (t.read() >= 0.009997) {
            RATE = t.read();
            t.reset();

            IMU9.update();
            computePID();
            if (output>0) LED[2] = 1;
            else LED[2] = 0;
            if (output<0) LED[3] = 1;
```

```
            else LED[3] = 0;

            leftmotor.speed(output);
            rightmotor.speed(output);
            //fprintf(fp, "%.2f,%.2f,%.2f\r\n",IMU9.getRoll(),IMU9.getPitch(),
IMU9.getYaw());
            fprintf(fp, "%.2f,%.2f,%.2f\r\n",IMU9.getPitch(),
IMU9.getAccelerometer(),IMU9.getGyro());
            pc.printf("!ANG:%.2f,%.2f,%.2f\r\n",IMU9.getRoll(),IMU9.getPitch(),
IMU9.getYaw());
        }
    }

}
```

Appendix 3 –Tilt Compensated Compass

```
//-----------------------------------------------------------------------------
// IMU Calculation
// Complementary filter (Pitch Roll Yaw)
//
// Tilt Compensated Compass (vector calculation):
// http://iopscience.iop.org/1742-6596/48/1/020/pdf/jpconf6_48_020.pdf
//-----------------------------------------------------------------------------

#include "mbed.h"

#include <L3G4200D.h>
#include <LSM303DLM.h>
#include <vector.h>

//-----------------------------------------------------------------------------
// IMU CALCULATION
//-----------------------------------------------------------------------------
#define ToRad 0.01745329252  // *pi/180
#define ToDeg 57.2957795131  // *180/pi

// LSM303 accelerometer: 8 g sensitivity
// 3.8 mg/digit; 1 g = 256
#define GRAVITY 1024  //this equivalent to 1G in the raw data coming from the accelerometer

// L3G4200D gyro: 2000 dps full scale
// 70 mdps/digit; 1 dps = 0.07
#define Gyro_Gain 0.01750 // Gyro gain

// Compass Calibration
int MAG_MIN[3] = {-685, -638, -507};
int MAG_MAX[3] = { 384,  373,  505};

// Working Variables
int a[3], g[3], m[3];
float A[3], G[3], M[3];
float RA[3], RG[3], RM[3];
```

```
float Aangle[3];
float pitch, yaw, roll;

Serial pc(USBTX, USBRX); // tx, rx
LSM303DLM compass( p28, p27 );
L3G4200D gyro( p28, p27 );
Timer t;

void Compass_Heading();

int main() {
    pc.baud(460800);
    t.start();
    while (1) {
        gyro.read(g);
        compass.readAcc(a);
        compass.readMag(m);

        // Map and Scale Values
        for (int i = 0; i < 3; i++ ) {
            A[i] = (float)a[i] / GRAVITY;
            G[i] = (float)g[i] * Gyro_Gain;
            M[i] = (float)(m[i]-MAG_MIN[i]) / (MAG_MAX[i] - MAG_MIN[i]) * 2 - 1.0;
        }

        // Switch axis (Rotate)

        RA[0] = -A[1];
        RA[1] = -A[2];
        RA[2] = -A[0];

        RG[0] =  G[1];
        RG[1] =  G[2];
        RG[2] =  G[0];

        RM[0] = -M[1];
        RM[1] = -M[2];
```

```
        RM[2] = -M[0];

        // Calculate Pitch and Roll Angle
        Aangle[0] = ToDeg* atan2(-RA[1],-RA[2]);  //arctan = O/A, radian to degree
        Aangle[1] = ToDeg* atan2( RA[0],-RA[2]); //arctan = O/A, radian to degree

        // Calculate Heading
        Compass_Heading();

        // Detect >360 to 0 and >0 to 360 done by compass
        if ((Aangle[2] - yaw) >270) yaw +=360;
        if ((yaw - Aangle[2]) >270) yaw -=360;

        // Complementary Filters
        pitch = (0.98)*(pitch + RG[0]*t.read()) + (0.02)*(Aangle[0]);
        roll  = (0.98)*(roll  + RG[1]*t.read()) + (0.02)*(Aangle[1]);
        yaw   = (0.99)*(yaw   + RG[2]*t.read()) + (0.01)*(Aangle[2]);

        // Limit range to 0-360
        if (yaw <   0) yaw += 360;
        if (yaw >= 360) yaw -= 360;

        // Prevent Singularity
        if (abs(pitch)>90) roll = 0;
        if (abs(roll)>90) pitch = 0;



        t.reset();
        pc.printf("Complementary pitch: %d,  roll: %d,  yaw: %d\r\n", (short)pitch,
(short)roll, (short)yaw);

        wait( 0.02 );
    }

}

void Compass_Heading() {
```

```
    // load accelerometer and magnetometer into vector
    vector temp_a;
    temp_a.x= RA[0];
    temp_a.y= RA[1];
    temp_a.z= RA[2];

    vector temp_m;
    temp_m.x= RM[0];
    temp_m.y= RM[1];
    temp_m.z= RM[2];

    // normalize acceleromter reading
    vector_normalize(&temp_a);

    // compute E and N plane
    vector E;               // east  plane
    vector N;               // north plane
    vector from = {0,-1,0}; // Z axis facing up

    // vector cross product of accelerometer and magnetometer (find east plane)
    vector_cross(&temp_m, &temp_a, &E);
    // normalize east plane
    vector_normalize(&E);
    // vector cross product of east plane and accelerometer (find north plane)
    vector_cross(&temp_a, &E, &N);

    // compute heading
    Aangle[2] = atan2(vector_dot(&E,&from), vector_dot(&N,&from)) * ToDeg;
    if (Aangle[2] < 0) Aangle[2] += 360;
}
```

Appendix 4 –B-EMF speed sensing

```
#include "bemf.h"

BEMF::BEMF(PinName P, PinName F, PinName B):
        pwm(P),Forward(F), Backward(B) {}

void BEMF::enable(float Period) {

    period = Period;
    pwm.period_us(2);
    tick.attach(this, &BEMF::sample, period);
    t.start();
}

void BEMF::sample(void) {
    temp = pwm.read();
    pwm.write(0);

    wait_ms(2); //wait 2 ms for voltage to settle
    //forward = (forward + Forward.read()*3.3)/2;
    //backward = (backward +Backward.read()*3.3)/2;
    forward = Forward.read()*3.3;
    backward = Backward.read()*3.3;
    if (forward > backward)
        speed = forward;
    else
        speed = -backward;

    pwm.write(temp);

    position += speed * t.read(); //speed intergration
    t.reset();
}

float BEMF::getSpeed(void) {
    return speed;
}
```

```
float BEMF::getPosition(void) {
    return position;
}
```