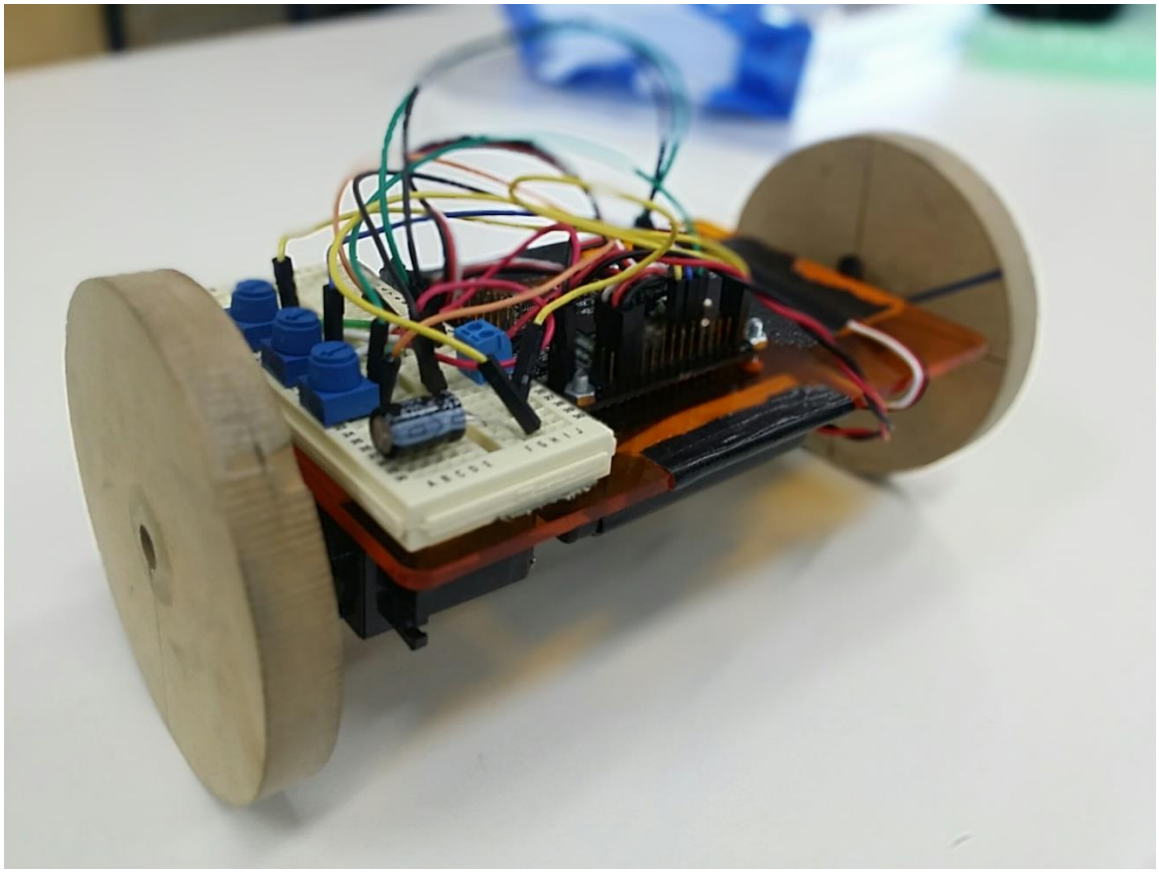
ALEX TRAN | 2014

SELF BALANCING ROBOT

MECHATRONICS ASSIGNMENT
SEMESTER 2



CONAN O'BRIEN
GUNGAHLIN COLLEGE

DESIGN OF ROBOT

The initial design of the robot was it to have two levels, so that it would be 'initially' thought to be easier to balance, however early prototypes showed that if built higher the weight distribution will be harder to control, since the actuators used in this robot were self-modified continuous servos. Where the servos have sufficient amount of torque to handle the weight however was not fast enough for the amount of weight would have been pressured onto it if a two level design was implemented.

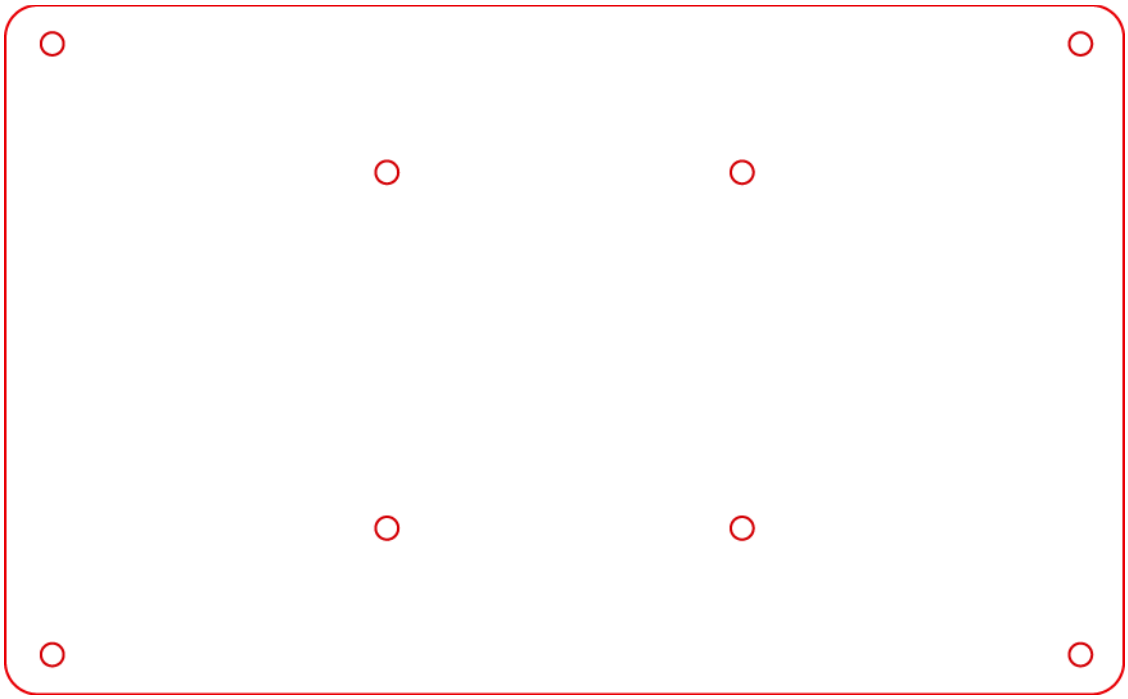


Figure 1

The base that was designed in Adobe Illustrator is shown in Figure 1, the four holes in the middle of the design are holes to bolt down the MultiWii board, and the four other holes was originally so that the an upper layer could be added, however as problems occurred it was scrapped. The design was then cut by a laser cutter where the dimensions were 150mm by 90mm.

The wheels were measured at 90mm radii, as these wheels did not have any rubber grip. Bigger wheels were sufficient to make up for the loss of grip on the wheels. There were other ways to implement more grip so it would have balanced better, however it would have come at a high price. The reason why bigger wheels can make up for the loss of grip is because the movement in the servos even slight would have had a greater effect on the movement in the wheels in bigger wheels than smaller ones. The wheels were cut from a water jet cutter.

Code

Final_Segway_Robot.ino

```
#include <Wire.h>
#include <Servo.h>

Servo myservo, myservo2;
byte servoPin = 9;
byte servoPin2 = 10;
int pot1 = A0;
int pot2 = A1;
int pot3 = A2;

int lastTime; //keep track of time intervals

float x = 0;

float setValue = 90; // 90 = level on robot

float input,output;
float errSum, lastErr;
float kp, ki, kd, KP, KI, KD;

void setup() {
  Serial.begin(19200);
  Wire.begin();
  Wire.beginTransmission(0x68);//mpu 6050 i2c address
  Wire.write(0x6b);        // select register 0x6b for power management
  Wire.write(0x00);  // value to write to register 0x6b to wake the MPU6050
  Wire.endTransmission(true); // release the I2C-bus
  myservo.attach(servoPin);
  myservo2.attach(servoPin2);
}


void loop() {

  updateAccel(); //getting new X value

  ErrorFind(); //calculating error and output to servos

  servoOut(); // function to output values to servos

  serialWrite();
}
```

Accelerometer.ino

```
void updateAccel() {
  Wire.beginTransmission(0x68);//mpu 6050 i2c address
  Wire.write(0x3B);      //register address for the start of accelerometer data
  Wire.endTransmission(false);    // hold the I2C-bus
  Wire.requestFrom(0x68, 6, true);  //the 6 here is the number of bytes to read
  byte i = 0;
  byte buffer[6];
  float accRaw[3];

  while(Wire.available() && i<6)
  {
    buffer[i]=Wire.read();
    i++;
  }
  if ( i != 6)     {
    Serial.println('I2C Error!');
  }

  accRaw[0]=(float)(buffer[0]<<8   |   buffer[1]);   //   starting   components   in
ACCELEROMETER
  accRaw[1]=(float)(buffer[2]<<8 | buffer[3]);
  accRaw[2]=(float)(buffer[4]<<8 | buffer[5]);

  float acc[3];

  for(int j = 0;j<3;j++) {
    acc[j] = accRaw[j]*(3.3/(1024.0*0.800)); //changing to radians
  }

  x = atan(acc[0]/sqrt(pow(acc[1],2)+pow(acc[2],2)))*(360/(2*PI)); // converting to
degrees
}
```

AlignmentCircle.ino

```
float circleCheck(float val,float addition)
{ // function to make sure readings is to level with 90 degrees to balance robot


  float answer = val + addition;
  if(answer > 360) {
    answer -= 360;
  }
  else if(answer < 0) {
    answer += 360;
  }
  else if( answer == 360) {
    answer = 0;
  }
  return answer;
}
```

PotRead.ino

```
void tunings()
{

  kp = map(analogRead(pot1), 0.0, 1023.0, 0.0, 6.0); //reading from potentiometer
  ki = map(analogRead(pot2), 0.0, 1023.0, 0.0, 10.0); //reading from potentiometer
  kd = map(analogRead(pot3), 0.0, 1023.0, 0.0, 6.0); //reading from potentiometer
  ki = ki/10000; //decreasing value for I value for PID control
  kd = kd/10; //gradient for D in PID


}
```

PIDcal.ino

```
void ErrorFind()
{

   int now = millis();
   float timeChange = now - lastTime; //time to caluculate intergral and derivative
controls


   input = circleCheck(x, 90); // 90 to balance / setpoint

   input = constrain(input, -200, 200);


   float error = setValue - input; // calculating error value
   errSum += error * timeChange ; // intergral control error sum
   float gradient = (error - lastErr) / timeChange; // derivative control error
gradient

   tunings(); //  read values from potentiometers

   KP = (kp * error); // final error value for Porpotional Control
   KI = (ki * errSum); // final error value for Intergral Control
   KD = (kd * gradient);  // final error value for Derivative Control

   output = KP + KI + KD; // output to servos


   lastErr = error; //for next derivative control
   lastTime = now; // for next time change
}
```

ServoOut.ino

```
void servoOut()
{
  output = int(output); // so servo only read and write integers
  int servoOutput1 = map(output, -90,90,180,0); //mapping data from error to rotate
servos in direction and speed
  int servoOutput2 = map(output, -90,90,0,180);
  myservo.write(servoOutput1); //writing to servos
  myservo2.write(servoOutput2);

}
```

Serial.ino

```
void serialWrite()
{
  Serial.print("Ang: ");
  Serial.print(input);
  Serial.print(" TimeChange: ");
  Serial.print(lastTime);
  Serial.print(" Output: ");
  Serial.print(output);
  Serial.print(" Kp:");
  Serial.print(kp);
  Serial.print(" Ki:");
  Serial.print(ki);
  Serial.print(" Kd:");
  Serial.print(kd);
  Serial.println();
  delay(10);

}
```

End

Explanation of Function of Robot

Comments were included in the code, where it explains most of the functions and commands that are given to the processor.

In Accelerometer.ino, the code where it communicates with the accelerometer was used in help of Conan's code (O' Brien, 2014). The calculation of the x value is calculated by taking the data read and converting it to radians then to degrees.

The AlignmentCircle.ino is where the angle that is read from the x value, is taken to calculate the value so that 90 is where horizontal level is and 180 or -180 is where it is on its side.

For the PID control, the Proportional control is calculated by having a constant value to increase or decrease which is multiplied with the error value calculated by the input value so it can calculate part of the output. This is the base output value which is the majority of the output, because it intends to reach the set value as it is the biggest value.
The Integral control is calculated by taking the total sum of the error and multiplying it by the change of time during the process, and finally multiplied with the constant value which is in this case, is an extremely small value, due to the time change is recorded in milliseconds which gives a bigger value if multiplied by a simple integer. Progressively the integral control will smoothen out the total output for the system as it calibrates to the surface it is on, it will be steadier at set value, unless it is pushed so it would need to recalibrate.
The Derivative control is calculated by taking the previous error subtracting the current error and dividing them by the change in time. It controls the gradient of the previous error and the current error, so it can figure out the steepness of the gradient it needs to change so it can be a horizontal linear line at the set value.

The three controls are then added up and in ServoOut.ino is just the function where it takes the output value from PID calculation to output to the servos where it has to be 'map' from 0 to 180 to control speed and direction of each servo where one is the opposite to the other as the servos are placed in different states on the robot.

References

Brettbeauregard.com, (2014). *Improving the Beginner's PID – Introduction « Project Blog*. [online] Available at: http://brettbeauregard.com/blog/2011/04/improving-the-beginners-pid-introduction/ [Accessed 8 Sep. 2014].

O'Brien, C. (2014). *Year 11 Avionics - Microsoft OneNote Online*. [online] OneNote - Avionics. Available at: https://onedrive.live.com/view.aspx?resid=57B063F591A310AF!10499&ithint=onenote%2c&app=OneNote&authkey=!AMHRIEHFZ8T_KCI [Accessed 14 Sep. 2014].