

## Lab 3B – Rover PID Control

---

### Table of Contents

1	Overview .....	0
1.1	How it Works.....	0
1.2	Credits .....	1
1.3	Reference Material .....	1
1.4	Mission Checklist.....	1
2	Make the Rover Go Straight.....	1
2.1	Software Checklist.....	1
2.2	Software Instructions.....	3
2.2.1	Addendum to instructions .....	6
2.3	Correct for Drift.....	8
2.3.1	Eliminate Drift (Design Challenge) .....	10
3	PID Control Code.....	10
3.1	GyroCaliRev3.....	11
3.2	ForwardPID .....	16

## 1 Overview

In this lab you will use your gyro to detect when your rover drifts from a straight trajectory caused by any internal or external variables. Closed loop control may be implemented using a P, PD, PID, or state-space control algorithm.

Our sensor, a [L3G420D](#) or [L3G4200D](#) 3-Axis MEMS Gyro with voltage regulator carrier/breakout board (from Pololu) includes a high-precision ST L3G4200D 3-axis gyroscope. The L3G4200D sensor measures the angular rates of rotation (velocity) about the pitch (x), roll (y), and yaw (z) axes with a configurable range of  $\pm 250^\circ/\text{s}$ ,  $\pm 500^\circ/\text{s}$ , or  $\pm 2000^\circ/\text{s}$ . We will only be using the z-axis sensor. The L3G4200 communicates with our microcontroller over an I2C serial interface.

### 1.1 How it Works

To understand the theory and code behind this lab, read [PID Control Lecture](#) plus “[Improving the Beginner’s PID: Direction](#)” by Brett Beauregard, the author of the [Arduino PID library](#)

## 1.2 Credits

The following lab reflects the work of many EE444 students. Including Kevin Nguyen, Nicholas Flenghi, Walter Heth, and Paul Zelaya

## 1.3 Reference Material

1. [L3G420D MEMS gyro](#): Ultra-stable three-axis digital output gyroscope
2. [Arduino library](#) for the L3G420D
3. [Arduino PID](#)
4. Read about Gyroscopes: <http://tom.pycke.be/mav/70/gyroscope-to-roll-pitch-and-yaw> and Kalman Filters (just for fun): <http://tom.pycke.be/mav/71/kalman-filtering-of-imu-data>

## 1.4 Mission Checklist

- ☐ Rover travels in a straight line using the Arduino PID Library.
- ☐ Rover uses shaft encoders to go a specified distance.
- ☐ Rover turns around 180° in order to head back to the starting point.
- ☐ Rover travels in a straight line to the starting point using the Arduino PID Library.
- ☐ Rover turns around 180° to starting position.

# 2 Make the Rover Go Straight

In this section you will download/upload the necessary software to make your rover travel in a straight line. Prior to starting this section, ensure your rover is properly wired with the gyro mounted and connected as described in the gyro lab.

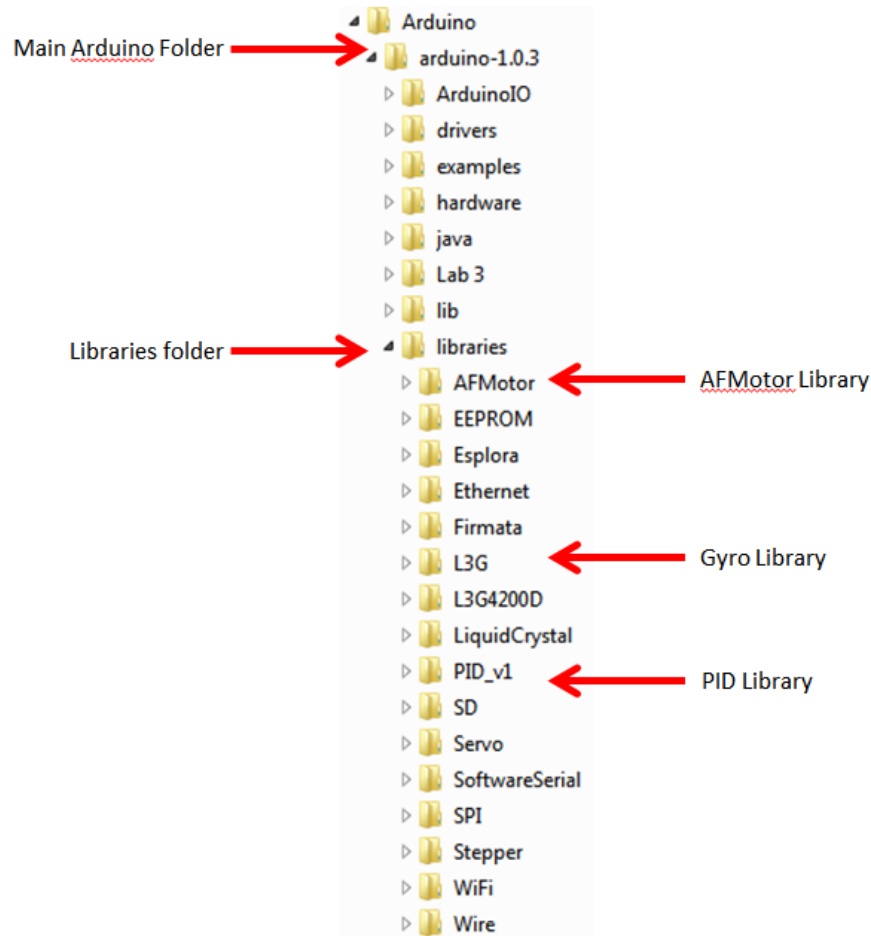
## 2.1 Software Checklist

Download and install the following software and libraries

1. [Adafruit Motor Shield Library](#)
2. [Gyro Arduino Library](#)
3. [Arduino PID Library](#)
4. [Processing](#) - Download the 32 bit version
5. [ControlP5 Processing Library](#)
6. [PID FrontEnd PDE for Processing](#).

### Arduino Libraries

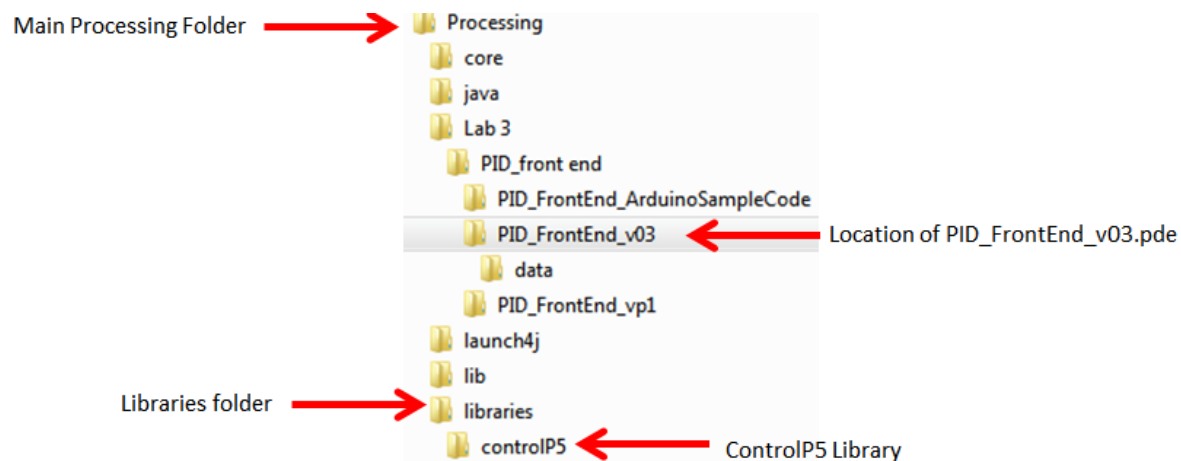
Any Arduino libraries have to be installed in the Libraries folder located inside the Arduino main folder. Place the Motor shield in a folder titled AFMotor, the gyro library in a folder titled L3G, and the Arduino PID library in a folder titled PID\_v1. Figure 1 below shows how the files should be arranged in the main Arduino folder.



**Figure 1** Arduino Library file structure

## Processing Libraries

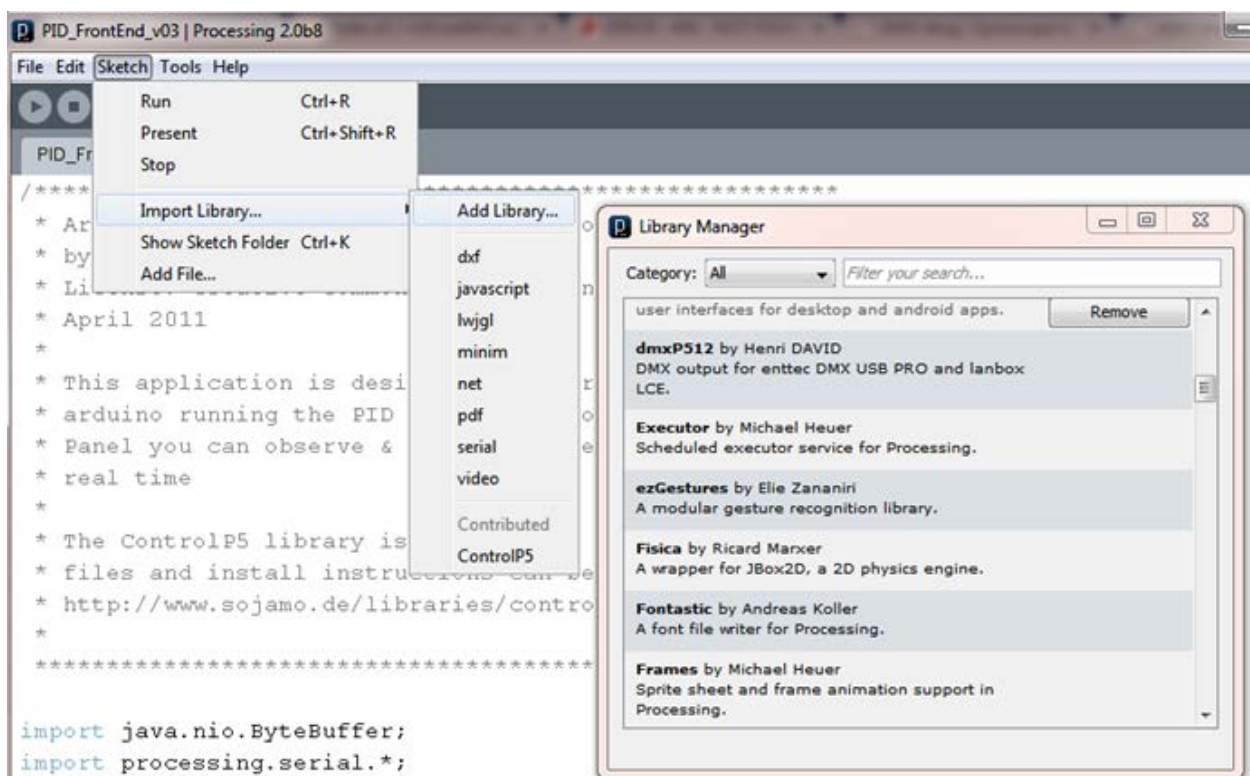
The 32 bit version of the processing program should be used. Computers that have 64-bit operating systems should be able to run the 32-bit version of processing. The arduino PID library page includes a processing file called PID front end. This .PDE file uses a processing library called ControlP5. ControlP5 is a processing GUI library. The .PDE file uses the ControlP5 library to generate a GUI that can send and receive data to the Arduino UNO/Arduino Motor Shield. The ControlP5 library must be installed in the Processing Library folder. The PID front end can be located anywhere inside the main processing folder.



**Figure 2** Processing Library file structure.

## 2.2 Software Instructions

Open the file **PID\_FrontEnd\_v03.pde** with Processing (see Figure 2). First the ControlP5 must be referenced in this file. To ensure that the file is recognized go to sketch, Import Library, Add Library. A library manager box should appear. Find ControlP5, and double click it to add to import the library. It should then appear in the sketch drop down as a contributed library (Figure 3).



**Figure 3** PID\_FrontEnd\_v03.pde

This program will produce a graph that will show the desired rover heading (referred to as set point on the GUI), and the actual rover heading as detected by the onboard gyro (referred to as input on the GUI). The PID control effort will also be displayed in a separate graph (referred to as Output on the GUI).

The graph window areas need to be configured. The screenshots below show what values to change. Set them as they appear in Figure 4 and 5

```

/*****
 * Arduino PID Tuning Front-End, Version 0.3
 * by Brett Beauregard
 * License: Creative-Commons Attribution Share-Alike
 * April 2011
 *
 * This application is designed to interface with an
 * arduino running the PID Library. From this Control
 * Panel you can observe & adjust PID performance in
 * real time
 *
 * The ControlP5 library is required to run this sketch.
 * files and install instructions can be found at
 * http://www.sojamo.de/libraries/controlP5/
 *****/

import java.nio.ByteBuffer;
import processing.serial.*;
import controlP5.*;

/*****
 * User specification section
 *****/
int windowWidth = 900; // set the size of the
int windowHeight = 600; // form

float InScaleMin = -30; // set the Y-Axis Min
float InScaleMax = 30; // and Max for both
float OutScaleMin = -100; // the top and
float OutScaleMax = 100; // bottom trends

int windowSpan = 30000; // number of ms into the past you
int refreshRate = 100; // how often you want the graph

```

**Figure 4** Arduino PID Tuning Front-End

```

void setup()
{
    frameRate(30);
    size(windowWidth, windowHeight);

    println(Serial.list());
    myPort = new Serial(this, Serial.list()[0], 115200);
    myPort.bufferUntil(10);
}

```

**Figure 5** Print Out COM Ports

In Figure 5, the command `println(Serial.list());` prints out all the active COM ports the computer has open. If you do not have an Arduino attached to the computer at the time you are about to run `PID_FrontEnd_v03.pde` you have to set the following array index to zero as shown in Figure 6.

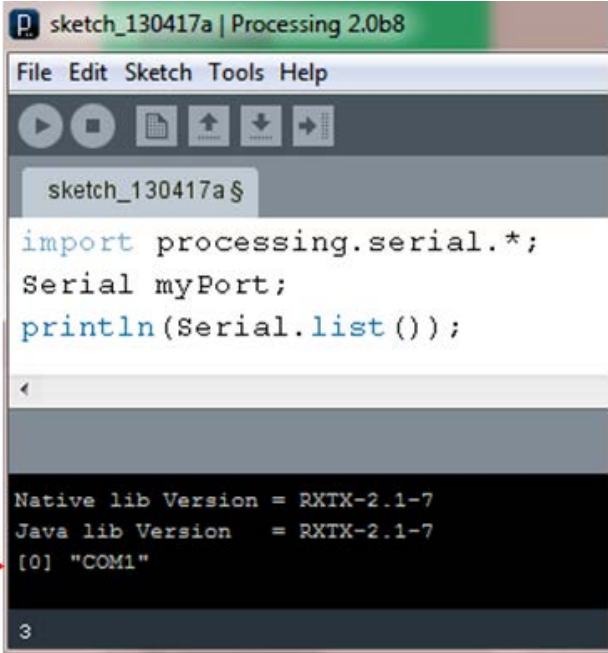
```

println(Serial.list());
myPort = new Serial(this, Serial.list()[0], 115200);
myPort.bufferUntil(10);

```

**Figure 6** Setting Array Index to Zero

If there is an Arduino attached to the computer when **PID\_FrontEnd\_v03.pde** front end is run, the highlighted number in Figure 6 should be set to a 1. If 1 does not work, this can mean that there is another device attached to the computer that has opened up a COM port. To view all available COM ports, open a new processing window by pressing CNTRL+N and enter the following:



```
sketch_130417a $
import processing.serial.*;
Serial myPort;
println(Serial.list());

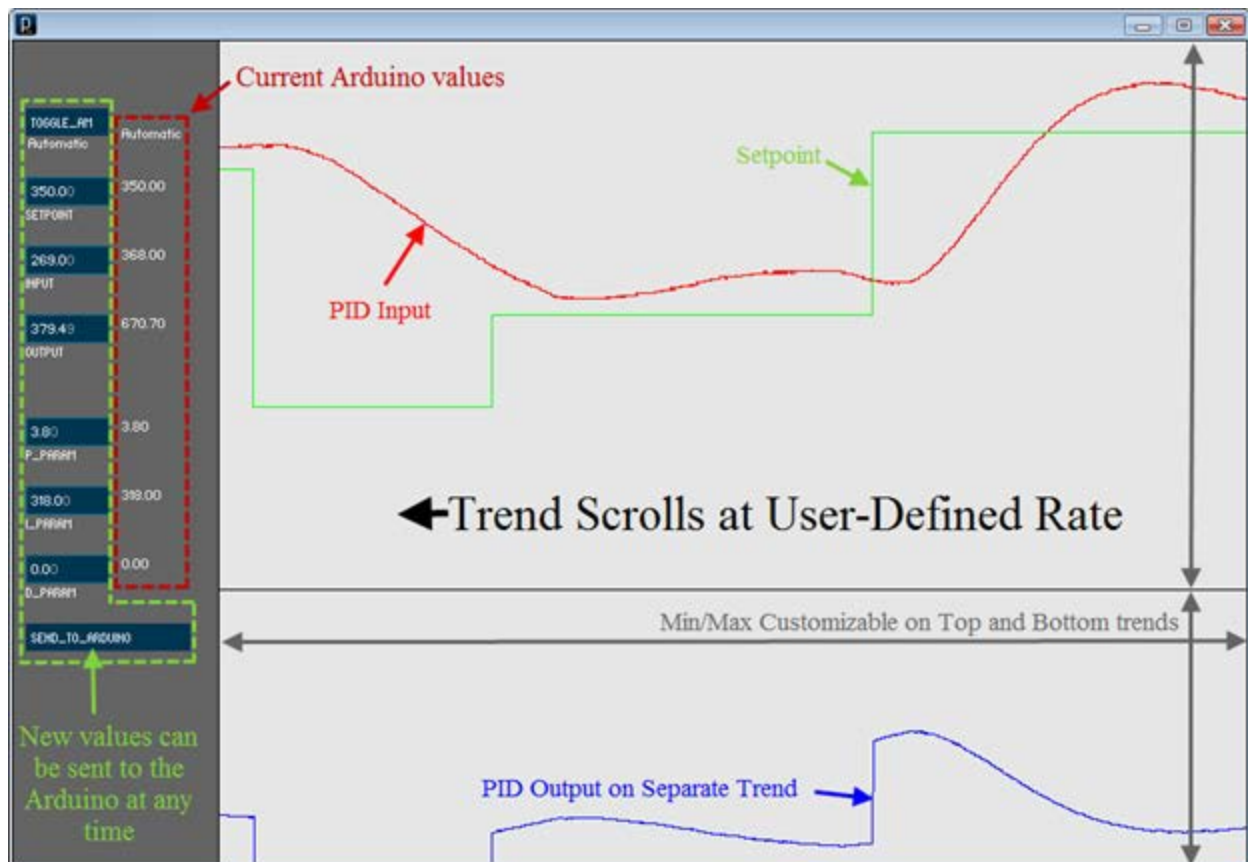
Native lib Version = RXTX-2.1-7
Java lib Version   = RXTX-2.1-7
[0] "COM1"

3
```

**Figure 7** Display all COM Ports

Note that there is one open COM port, which has a [0] entry associated with it. When an Arduino is connected, a COMx will appear with an array entry. Use this entry as shown in Figure 6.

Next the PID program must be uploaded to the Arduino. Upload **GyroCaliRev1.ino** to the Arduino. See Section 6 for this code. Once **GyroCaliRev1.ino** is uploaded, run the **PID\_FrontEnd\_v03.pde** processing program. Once the program is running, two windows should be visible. The top window displays the rover set point, and the rover input. The bottom graph shows the control effort by the PID program on the motors (PID Output). The set point, as well as many other parameters (refer to Figure 8), can be changed on the graph by entering a new set point and hitting the **SEND\_TO\_ARDUINO** button.



**Figure 8** Changing the Set Point

## 2.2.1 Addendum to instructions

Following are notes from students who had problems implementing the instructions as written. In general the problems are result of the original instructions being written for the Mac and mostly implemented within Windows.

### 2.2.1.1 PID\_FrontEnd\_v03 - Windows RXTX version mismatch

*by James Nelson*

The following might be a solution if you are running 'PID\_FrontEnd\_v03' and see the message: (especially if your OS is Windows Vista)

WARNING: RXTX Version mismatch  
 Jar version = RXTX-2.2pre1  
 native lib Version = RXTX-2.2pre2  
 Here is how you fix this problem.

- 1) Go to <http://rxtx.qbang.org/wiki/index.php/Download> and download rxtx-2.1-7-bins-r2.zip from "Binary" column.
- 2) Go to the location where you downloaded the file and copy the following files:
  - a) ... \rxtx-2.1-7-bins-r2.zip\rxtx-2.1-7-bins-r2\RXTXcomm.jar

and copy RXTXcomm.jar to "....\processing-1.5.1\java\lib\ext".  
For me, it was C:\Program Files\processing-1.5.1\java\lib\ext  
b) ... \rxtx-2.1-7-bins-r2.zip\rxtx-2.1-7-bins-r2\Windows\i368-mingw32  
and copy both rxtxParallel.dll and rxtxSerial.dll to  
"....\processing-1.5.1\ java\bin".  
For me, it was C:\Program Files\processing-1.5.1\java\bin

### The result:

Stable Library

=====

Native lib Version = RXTX-2.1-7

Java lib Version = RXTX-2.1-7

You should now be able to run 'PID\_FrontEnd\_v03' every time with no problem, whereas before I tried everything to get some sort of response (there was nothing on the display). This is a known problem with Windows Vista.

## 2.2.1.2 Rover PID code – Problems Locating and Reading rangedata.h file

*by Elizabeth Sandoval*

### Rover PID code

The Rover PID Code can be found at the end of this lab. Copy the code into Arduino IDE and save the sketch. Run the sketch.

### Locating rangedata.h file

If the sketch gives an error about finding rangedata there are two ways to fix it.

1. The simplest solution if to place rangedata.h in your project folder and include parenthesis in the #INCLUDE statement. If the Arduino IDE is open, remember to **close and reopen the Arduino IDE** after you have added this file.

```
#INCLUDE "rangedata.h" // this will look in the project or sketch folder
```

2. Although not recommended, you can also place the rangedata.h file in the Arduino's program folder and use carrots <> in the #INCLUDE statement.

```
# INCLUDE <rangedata.h> // this will look in the Arduino Program libraries folder
```

### Error in reading rangedata.h file

If your program generates an ERROR trying to read the datarange.h file, open it and look for the following line with space.



```
rangedata - Notepad
File Edit Format View Help

,242,243,244,245,246,247,248,249,250,251,252,253,254,255,256,257,258,259,260,261,262,263,264,265,266,267,2
,498,499,500,501,502,503,504,505,506,507,508,509,510,511,512,513,514,515,516,517,518,519,520,521,522,523,5
,754,755,756,757,758,759,760,761,762,763,764,765,766,767,768,769,770,771,772,773,774,775,776,777,778,779,7
,1008,1009,1010,1011,1012,1013,1014,1015,1016,1017,1018,1019,1020,1021,1022,1023
99263,118.862043,117.8129472,116.7821472,115.7691687,114.7735535,113.7948588,112.8326571,111.8865348,110.9
7.58101619,67.23885418,66.90011926,66.5647602,66.23272679,65.90396979,65.57844095,65.25609295,64.93687941,
7205113,46.80564702,46.64040788,46.47632152,46.31337591,46.15155919,45.99085969,45.83126586,45.67276633,45
237,35.85367485,35.75613504,35.65911867,35.56262156,35.46663953,35.37116848,35.27620433,35.18174307,35.087
12865,29.02655154,28.96225661,28.89824203,28.83450597,28.77104662,28.70786217,28.64495084,28.58231087,28.5
96688,24.3632513,24.31770389,24.27232373,24.22710989,24.18206146,24.13717754,24.09245723,24.04789962,24.00
1.97577763,20.94183062,20.90799127,20.87425905,20.84063347,20.80711402,20.77370019,20.74039149,20.70718742,
40355669,18.37728303,18.35108269,18.32495536,18.29890073,18.27291852,18.24700841,18.2211701,18.19540331,18
```

Place a Comma and hit delete button on computer to get the following. Recompile (Verify) and it should be compiled fine.

```
,250,251,252,253,254,255,256,257,258,259,260,261,262,263,264,2
,506,507,508,509,510,511,512,513,514,515,516,517,518,519,520,5
,762,763,764,765,766,767,768,769,770,771,772,773,774,775,776,7
,1014,1015,1016,1017,1018,1019,1020,1021,1022,1023,165,165,165,1
409429,109.1407128,108.2550399,107.3835741,106.5259763,105.681
54.30767455,63.99759485,63.69047279,63.38626625,63.08493391,62
35900548,45.20372218,45.04948922,44.89629599,44.744132,44.5929
1329,34.90133693,34.80884777,34.71684198,34.62531578,34.534265
783801,28.39600165,28.33442974,28.27312057,28.21207246,28.1512
326902,23.91519427,23.87127872,23.82752152,23.78392181,23.7404
0.64109122,20.60819811,20.57540768,20.54271945,20.51013295,20.
.14408305,18.118529,18.09304528,18,18,18,18,18,18,18,18,18,18,
```

### 2.2.1.3 Rover PID code and rangedata.h - Fix using Shaft Encoder

by Paul Zelaya

In this version of the Lab, shaft encoders will be used in order to determine distance rover traveled. We do not need the IR sensors to determine how far away an obstruction is. Therefore we can comment out the IR function in our GyroCalRev1.ino file that we upload to the Arduino. This code has already been commented out in the code included in Section 3 of this lab.

## 2.3 Correct for Drift

Requirements

- Wire.h
- L3G.h
- High baud rate (57600 or 115200)
- Gyro connected
- Stable surface to put gyro on.

Every gyro has drift, some more than others. To correct for drift manually you can subtract out the observed drift in your code as shown here.

```
sensorReading = gyro.g.z -9 // replace 9 with your observed drift.
```

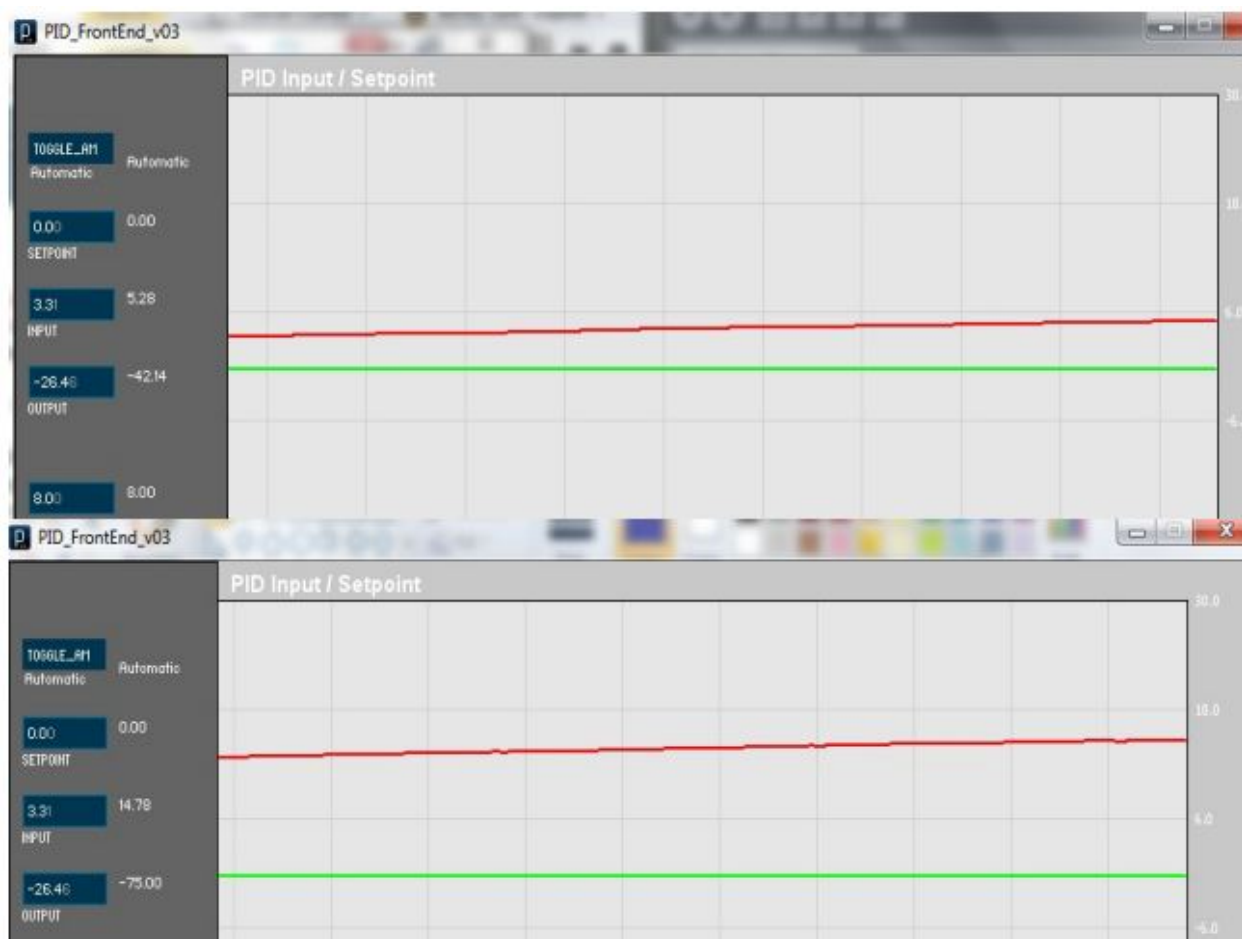
**NOTE:** Depending if your gyro is mounted horizontally or vertically you may need to change gyro.g.z to gyro.g.x.

Code has been provided to eliminate the guesswork of what this calibration constant should be.

GyroCaliRev3 will read the angle from the gyro and automatically adjust the `int GyroCali` until it gets a certain number of Zeros in a row. Then it displays 17 values that worked and an average of all the values. Keep increasing `int Zeros` to 11 or 12 to get better results. Once you get a calibration value plug it into `int GyroCali`. In the `loop` comment out `caliGyro()`, and `readGyro2()`. Uncomment `readGyro()` and run the code. The drift should have decreased significantly.

Yet there is still drift

In your lab report note how this drift compares with the drift observed in the gyro lab using Matlab.



### 2.3.1 Eliminate Drift (Design Challenge)

As you discovered in the Gyro Lab, over time and temperature your drift will change meaning the manual solution is not a good long term one. Update your Arduino or Processing C++ code to automatically calculate for and correct for drift before you tune your PID (next section).

First, consider the values angle that are being read. If you send these to the serial port you will see all 0.00s if rover is stationary. When adding up these 0s we are getting a positive drift? This is because the angle being read is actually 0.0001-0.0020. We want to filter out these values by creating an if statement that looks for significant values. However if we set this significant value to high (ex 0.1 degree) then it will not detect small movements and too low (ex 0.0010 degree) will still let in some drift. If we are actually turning this means the values around it will be significant as well. So if our if statement is activated multiple times then we should add up the last few significant values and start recording the angle.

When the rover is stationary we can set our threshold at 0.0010 degrees but when the rover is moving we will be getting more noise. Try moving closer to a threshold of 0.0100 when moving.

Variable `angle` has a data type of `double`, this means you will have to multiply it by 100 to actually see the hidden values and compare it to 0.1 instead of 0.0010. Check out the function `readGyro2( )`

## 3 PID Control Code

Now upload the code `ForwardPID` to the rover. This will set the rover to go forward continuously and send the current angle of the rover out to be monitored in processing using `readGyro2( )`. In the processing software you can manipulate the PID constants and observe how the rover corrects its direction. Keep changing these values until your rover can recover to its original course after manually being turned in place.

### 3.1 GyroCaliRev3

```
#include <Wire.h>
#include <L3G.h>

L3G gyro;
const double sensitivity = 297; // gyro sensitivity setting for calculation

unsigned long newTime;
unsigned long dt;
double sensorReading;
double angle = 0;
unsigned long oldTime;
int GyroCali = 225;
int Zeros = 8 //Hit zero 8 times in a row before saving a calibration value
{ //try starting at 8 and moving up to 11 or 12!
double gyroAngle1 = 0;
double gyroAngle2 = 0;
int counter = 0;
int tracker = 1;
int i=0;
int hold = 0;
int holdGyro = 0;
int value1 = 0;
int value2 = 0;
int value3 = 0;
int value4 = 0;
int value5 = 0;
int value6 = 0;
int value7 = 0;
int value8 = 0;
int value9 = 0;
int value10 = 0;
int value11 = 0;
int value12 = 0;
int value13 = 0;
int value14 = 0;
int value15 = 0;
int value16 = 0;
int value17 = 0;
int value18 = 0;

void setup()
{
  Serial.begin(115200);
  Wire.begin();
  gyro.init();
  gyro.enableDefault();
  analogReference(EXTERNAL); // 3.3v Analog Ref Voltage
}

void loop()
{
  //readGyro();
  //readGyro2();
  caliGyro();
}
/*****/
// -----
// Read Gyro - returns current angle
// Make sure readGyro is the only function uncommented in the loop
// -----
float readGyro()
{
```

```

gyroAngle2=gyroAngle1; //Save old angle value
newTime = millis();
dt = newTime - oldTime;
oldTime = newTime;
gyro.read();
sensorReading = gyro.g.x+GyroCali; // get new angle difference value
angle = (sensorReading/32768)*sensitivity*(float(dt) / 1000.0); //angle in degrees
gyroAngle1 += angle; //keep adding on angle if it is significant
Serial.println(gyroAngle1);
return gyroAngle1;
}

```

```

/*****/
// -----
// Read Gyro - returns current angle
// Make sure readGyro is the only function uncommented
// in the loop. This code discards
// insignificant values and checks to see if significant
// values are next to other significant values before
// deciding what the actual angle is.
// -----
float readGyro2()
{
gyroAngle2=gyroAngle1; //Save old angle value
newTime = millis();
dt = newTime - oldTime;
oldTime = newTime;
gyro.read();
sensorReading = gyro.g.x+GyroCali; // get new angle difference value
angle = (sensorReading/32768)*sensitivity*(float(dt) / 1000.0); //angle in degrees
Serial.print(" "); //print in 2nd column
Serial.println(abs(angle*100));
if (abs(angle*100)>0.10) //might be a significant value
{
if (hold>=10) //if 10 significant values then angle has definitely changed
{
if (hold==10)
{
gyroAngle1 += holdGyro; //add the 10 saved values that weren't added before
}
else
{
gyroAngle1 += angle; //keep adding on angle if it is significant
}
}
holdGyro += angle;
hold += 1;
Serial.print("Hold:");
Serial.println(hold);
}
else //not a significant value
{
if (hold!=0)
{
// Serial.println("filtered");
}
hold = 0;
holdGyro = 0;
}
Serial.println(gyroAngle1);
return gyroAngle1;
}

```

```

/*****/
// -----

```

```

//
// Calibrate Gyro - Adds in a constant (int GyroCali) to cancel out drift
// Make sure readGyro() is commented out of setup/loop and to uncomment
// caliGyro(). Make sure gyro is not moving during test.
// When test is complete enter in a new GyroCali value at top.
// -----
float caliGyro()
{
gyroAngle2=gyroAngle1; // save old angle value
newTime = millis();
dt = newTime - oldTime;
oldTime = newTime;
gyro.read();
sensorReading = gyro.g.x+GyroCali; // get current angle difference value
angle = (sensorReading/32768)*sensitivity*(float(dt) / 1000.0);
gyroAngle1 += angle; // get current angle value
if (angle>0) //if +angle difference then turn calibration down
{
GyroCali -= 1;
Serial.println("-1");
}
if(angle<0) //if -angle difference then turn calibration up
{
GyroCali += 1;
Serial.println("+1");
}
if(angle==0) //keep track of how many times we are dead on 0
{
Serial.print("same");
for (i=0; i<tracker; i++) //everytime we get a viable calibration value we want to
{ //add a star in the serial port to tell us our progress
Serial.print("*");
}
Serial.println("");
counter += 1;
}
else //if not dead on then reset
{
counter=0;
}
if(counter==Zeros)
{
if (tracker==1)
{
value1=GyroCali;
}
else if (tracker==2)
{
value2=GyroCali;
}
else if (tracker==3)
{
value3=GyroCali;
}
else if (tracker==4)
{
value4=GyroCali;
}
else if (tracker==5)
{
value5=GyroCali;
}
else if (tracker==6)
{
value6=GyroCali;
}
else if (tracker==7)

```

```

{
value7=GyroCali;
}
else if (tracker==8)
{
value8=GyroCali;
}
else if (tracker==9)
{
value9=GyroCali;
}
else if (tracker==10)
{
value10=GyroCali;
}
else if (tracker==11)
{
value11=GyroCali;
}
else if (tracker==12)
{
value12=GyroCali;
}
else if (tracker==13)
{
value13=GyroCali;
}
else if (tracker==14)
{
value14=GyroCali;
}
else if (tracker==15)
{
value15=GyroCali;
}
else if (tracker==16)
{
value16=GyroCali;
}
else if (tracker==17)
{
value17=GyroCali;
}
else if (tracker==18)
{
Serial.println();
Serial.print(value1);
Serial.print(",");
Serial.print(value2);
Serial.print(",");
Serial.print(value3);
Serial.print(",");
Serial.print(value4);
Serial.print(",");
Serial.print(value5);
Serial.print(",");
Serial.print(value6);
Serial.print(",");
Serial.print(value7);
Serial.print(",");
Serial.print(value8);
Serial.print(",");
Serial.print(value9);
Serial.print(",");
Serial.print(value10);
Serial.print(",");
Serial.print(value11);

```

```

Serial.print(",");
Serial.print(value12);
Serial.print(",");
Serial.print(value13);
Serial.print(",");
Serial.print(value14);
Serial.print(",");
Serial.print(value15);
Serial.print(",");
Serial.print(value16);
Serial.print(",");
Serial.print(value17);
Serial.print(",");
Serial.println(value18);
Serial.printf("Average value:")
Serial.println((value1+value2+value3+value4+value5+value6+value7+value8+value9+value10+value11+value12+value13+value14+value15+value16+value17)/17);
while(1)
{
//test complete, values displayed
}
}
tracker++;
}
Serial.println(gyroAngle1);
return gyroAngle1;
}

```



## 3.2 ForwardPID

```
#include <Wire.h>
#include <L3G.h>
#include <AFMotor.h>
#include <PID_v1.h>
#include <avr/pgmspace.h>
#include <rangedata.h>
// DC MOTORS
int m_ref = 150; // Initial motor duty cycle (0 to 255)
#define min_dutycycle 75
#define max_dutycycle 250
double maxOutput = 75;
AF_DCMotor motor1(1, MOTOR12_64KHZ); // create motor #1, 64KHz pwm
AF_DCMotor motor2(2, MOTOR12_64KHZ); // create motor #2, 64KHz pwm
AF_Stepper motorS(200, 2);
double stepper = 0; // keep track of stepper location (in degrees)
// TURN ENCODER
const int pin2 = 2; // Reads shaft encoder 1
const int pinA0 = A0; // Reads shaft encoder 2
int turnEncoder1 = 0; // use these variables to keep track how many steps
int turnEncoder2 = 0; // the rover has taken
boolean lastStateA1; // Channel A of shaft encoder 1
boolean lastStateA2; // Channel A of shaft encoder 2
int NumbPulse=800; //12in (1click/0.15in)=80 clicks 2ft=160clicks etc
int x=0; //pulses encoder1
int y=0; //pulses encoder2
int z=0; //no change
//GYRO
L3G gyro;
const double sensitivity = 297; // gyro sensitivity setting for calculation
unsigned long newTime;
unsigned long dt;
double sensorReading;
double angle = 0;
unsigned long oldTime;
int GyroCali = 225;
double gyroAngle1 = 0;
double gyroAngle2 = 0;
int counter = 0;
int hold = 0;
int holdGyro = 0;
double drift = 0;
double turn = 0;
int direction10 = 1;
int Turnfinal = 175;
//PID
/**** Experimental ****
#define PIDLimit (max_dutycycle - min_dutycycle)/2
*/
#define PIDLimit 75 // PID output limited to +/- 75
double Setpoint, Input, Output;
//Specify the links and initial tuning parameters
PID myPID(&Input, &Output, &Setpoint,8,0,0, DIRECT);
unsigned long serialTime; // this will help us know when to talk

void setup()
{
  Serial.begin(115200);

  // setup gyro
  Wire.begin();
  if (!gyro.init())
  {
```

```

    Serial.println("Failed to autodetect gyro type!");
    while (1);
}
gyro.enableDefault();
analogReference(EXTERNAL); // 3.3v Analog Ref Voltage

//setup motors
pinMode(pin2, INPUT);      //Define this pin as a input
digitalWrite(pin2, HIGH);  //Get this pull up resistor fired up
pinMode(pinA0, INPUT);     //Define this pin as a input
digitalWrite(pinA0, HIGH); //Get this pull up resistor fired up
motor1.setSpeed(200);      // LEFT set the speed to 200/255
motor2.setSpeed(200);      // RIGHT
//Serial.println("Motor setup complete");

//setup PID monitor
Input = readGyro2();
Setpoint=0;
myPID.SetOutputLimits(-PIDLimit, PIDLimit);
// turn the PID on
myPID.SetMode(AUTOMATIC);
// Stepper Motor Speed
motorS.setSpeed(40); // Stepper set to 40 rpm
}

void loop()
{
    motor1.run(FORWARD);      // turn it on going forward
    motor2.run(FORWARD);

    //*****
    // Move forward
    //*****
    // while neither motor has reached their last step, continue to run forward
    // and keep track of what step you are on
    while(1)
    {
        Input = readGyro2();
        myPID.Compute();
        motor2.setSpeed(constrain(m_ref + int(Output), min_dutycycle, max_dutycycle));
        motor1.setSpeed(constrain(m_ref - int(Output), min_dutycycle, max_dutycycle));
        if(millis()>serialTime)
        {
            SerialReceive();
            SerialSend();
            serialTime+=250;
        }
    }
}

//*****
// -----
// Read Gyro - returns current angle
// Make sure caliGyro() is commented out of setup/loop
// and that readGyro is uncommented. This code discards
// insignificant values and checks to see if significant
// values are next to other significant values before
// deciding what the actual angle is.
// -----

float readGyro2()
{
    gyroAngle2=gyroAngle1;    //Save old angle value

```

```

newTime = millis();
dt = newTime - oldTime;
oldTime = newTime;
gyro.read();
sensorReading = gyro.g.x+GyroCali; // get new angle difference value
angle = (sensorReading/32768)*sensitivity*(float(dt) / 1000.0);

//Serial.print(" ");
//Serial.println(abs(angle*100));
if (abs(angle*100)>1) //might be a significant value
{
    if (hold>=10) // angle has definitely changed
    {
        if (hold==10)
        {
            gyroAngle1 += holdGyro;
        }
        else
        {
            gyroAngle1 += angle;
        }
    }
    holdGyro += angle;
    hold += 1;
    //Serial.print("Hold:");
    //Serial.println(hold);
}

else //not a significant value
{
    if (hold==1)
    {
        // Serial.println("filtered");
    }
    hold = 0;
    holdGyro = 0;
}
//Serial.println(gyroAngle1);
return(gyroAngle1);
}

// -----
// Read IR Sensor
// -----
/*double readIR()
{
    IRValue = analogRead(IRPin);
    distance = pgm_read_word_near(range_data + IRValue); //LUT Distance
    // Serial.print("DN: "); // debugging
    // Serial.println(IRValue);
    return distance;
}

*/
/*****
* Serial Communication functions / helpers
*****/
union { // This Data structure lets
byte asBytes[24]; // us take the byte array
float asFloat[6]; // sent from processing and
} // easily convert it to a
foo; // float array
// getting float values from processing into the arduino
// was no small task. the way this program does it is
// as follows:

```

```

// * a float takes up 4 bytes. in processing, convert
// the array of floats we want to send, into an array
// of bytes.
// * send the bytes to the arduino
// * use a data structure known as a union to convert
// the array of bytes back into an array of floats
// the bytes coming from the arduino follow the following
// format:
// 0: 0=Manual, 1=Auto, else = ? error ?
// 1: 0=Direct, 1=Reverse, else = ? error ?
// 2-5: float setpoint
// 6-9: float input
// 10-13: float output
// 14-17: float P_Param
// 18-21: float I_Param
// 22-245: float D_Param
void SerialReceive()
{
  // read the bytes sent from Processing
  int index=0;
  byte Auto_Man = -1;
  byte Direct_Reverse = -1;
  while(Serial.available() && index<26)
  {
    if(index==0) Auto_Man = Serial.read();
    else if(index==1) Direct_Reverse = Serial.read();
    else foo.asBytes[index-2] = Serial.read();
    index++;
  }
  // if the information we got was in the correct format,
  // read it into the system
  if(index==26 && (Auto_Man==0 || Auto_Man==1) && (Direct_Reverse==0 ||
  Direct_Reverse==1))
  {
    Setpoint=double(foo.asFloat[0]);
    //Input=double(foo.asFloat[1]); // * the user has the ability to send the
    // value of "Input" in most cases (as
    // in this one) this is not needed.
    if(Auto_Man==0) // * only change the output if we are in
    { // manual mode. otherwise we'll get an
      Output=double(foo.asFloat[2]); // output blip, then the controller will
    } // overwrite.
    double p, i, d; // * read in and set the controller
    p = double(foo.asFloat[3]); // tunings.
    i = double(foo.asFloat[4]); //
    d = double(foo.asFloat[5]); //
    myPID.SetTunings(p, i, d); //
    if(Auto_Man==0) myPID.SetMode(MANUAL); // * set the controller mode
    else myPID.SetMode(AUTOMATIC);
    // * set the controller Direction
    if(Direct_Reverse==0) myPID.SetControllerDirection(DIRECT);
    else myPID.SetControllerDirection(REVERSE);
  }
  Serial.flush(); // * clear any random data from the serial buffer
}
// unlike our tiny microprocessor, the processing ap
// has no problem converting strings into floats, so
// we can just send strings. much easier than getting
// floats from processing to here no?
void SerialSend()
{
  Serial.print("PID ");
  Serial.print(Setpoint);
  Serial.print(" ");
  Serial.print(Input);
  Serial.print(" ");
  Serial.print(Output);

```

```
Serial.print(" ");
Serial.print(myPID.GetKp());
Serial.print(" ");
Serial.print(myPID.GetKi());
Serial.print(" ");
Serial.print(myPID.GetKd());
Serial.print(" ");
if(myPID.GetMode()==AUTOMATIC) Serial.print("Automatic");
else Serial.print("Manual");
Serial.print(" ");
if(myPID.GetDirection()==DIRECT) Serial.println("Direct");
else Serial.println("Reverse");
}
```