School of Engineering, Discipline of Mechanics and Mechatronics, pp. 1–15

Lab 4: Hardware Calibration

ENGG1500
Semester 1 2020

Nicholas O'Dell, Brenton Schulz, Alexander Fairclough, Dylan Cuskelly.

---

1 Introduction

In this lab you will work to develop engineering solutions to some of the hardware problems you have encountered in the project thus far. For example, you should have discovered that when you send equal signals to your left and right motors the vehicle does not drive straight. There are numerous causes for this, such as: initial state of the castor wheel, inconsistencies in the plastic gearboxes, different electrical resistance of each motor winding, motor mounting orientation and motor misalignment.

Be assured that problems such as asymmetric motor properties, sensor calibration and *lack* of available pins are all part of the project, and are to be solved by your team. Often a solution to these problems can be developed through running a simple experiment, in this lab you will run several small experiments to obtain calibration data. The experiments you run will assume certain properties of the robot under test conditions and operation, for each of these experiments you will define these assumptions and determine under what conditions they are valid.

There are three exercises to complete in this lab:

1. Motor PWM to speed calibration.
2. Line position estimation using multiple line sensors.
3. RGB distance sensor calibration.

It is recommended that you choose and attempt one of the three exercises during each lab in weeks 4 and 5.

2 Motor Calibration

Hint

This task will be useful for Lab 5.

Tip

Before doing this task you should complete the extension task from Lab 2 to ensure your encoders are functioning correctly.

Task: Record the motors' response to different PWM duty cycles. This will be achieved by looping through different PWM values while using the encoder counts to calculate the average angular velocity of the motors. This data will then be exported and plotted in Excel (or Matlab etc).

The Arduino `for()` loop syntax is as follows:

Listing 1: For loop introduction.

```
for(initial value; exit condition; loop increment)
{
    // do_the_things();
}
```

For example: (NB `x++` is shorthand for `x = x + 1`)

Listing 2: For loop example.

```
int x;
for(x = 0; x < 100; x++)
{
    // Code in here is executed 100 times
    // x changes value each time the loop occurs
}
```

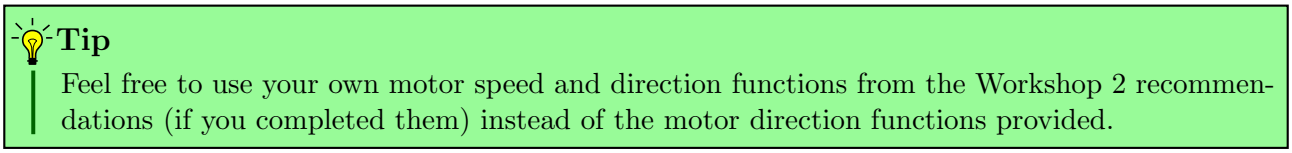
Implement the following pseudo code in Arduino:

Listing 3: Pseudo code algorithm.

```
BEGIN
  FOR pwm = 0 to 255
    Zero encoders
    Set pwm value on both motors
    Delay 1 second
    Read encoders
    Print pwm and encoder values to serial port
  ENDFOR
END
```

Remember to include the `ENGG1500Lib` header file. To clear the encoder counts use `enc_clear();` while `enc_getLeft();` and `enc_getRight();` return the left and right counts respectively.

Begin with the following template available on Blackboard:



Listing 4: Motor Calibration Template.

```
#include <ENGG1500Lib.h>
#define ENA 5
#define ENB 6
#define IN1 8
#define IN2 9
#define IN3 10
#define IN4 11

void setup() {
  // put your setup code here, to run once:
  pinMode(ENA, OUTPUT); //set ENA as an output
  pinMode(ENB, OUTPUT); //set ENB as an output
  pinMode(IN1, OUTPUT); //set IN1 as an output
  pinMode(IN2, OUTPUT); //set IN2 as an output
  pinMode(IN3, OUTPUT); //set IN3 as an output
  pinMode(IN4, OUTPUT); //set IN4 as an output
  enc_init(); //initialise encoders
  delay(2000);
  Serial.begin(9600);
  Serial.println("PWM,ENC_L,ENC_R");
```

```
  //TODO: set motor direction
}

void loop() {
  // put your main code here, to run repeatedly:
  unsigned int pwm = 0;
  for (pwm = 0; pwm < 256; pwm += 5)
  {
    //TODO:
    //zero encoders
    //set left and right motor PWM
    //delay for 1 second
    //read the encoders
    //print pwm and encoder values to the serial monitor in CSV format:
    Serial.print(pwm);
    Serial.print(",");
    Serial.print(enc_getLeft());
    Serial.print(",");
    Serial.println(enc_getRight());
  }
}
void leftForwards(void) //This function sets IN1 = LOW and IN2 = HIGH in order to set
     //...the direction to forwards for motor 1
{
  digitalWrite(8, LOW); //IN1
  digitalWrite(9, HIGH); //IN2
}
void leftBackwards(void) //This function sets IN1 = HIGH and IN2 = LOW in order to set
     //... the direction to backwards for motor 1
{
  digitalWrite(8, HIGH); //IN1
  digitalWrite(9, LOW); //IN2
}
void rightForwards(void) //This function sets IN3 = LOW and IN4 = HIGH in order to set
     //... the direction to forwards for motor 2
{
  digitalWrite(10, LOW); //IN3
  digitalWrite(11, HIGH); //IN4
}
void rightBackwards(void) //This function sets IN3 = HIGH and IN4 = LOW in order to
     //...set the direction to forwards for motor 2
{
  digitalWrite(10, HIGH); //IN3
  digitalWrite(11, LOW); //IN4
}
```

After your code has been executed, the serial monitor's output can be copied (highlight, **Ctrl+C**) then pasted directly into Excel[1]. A graph of duty cycle Vs speed data is very useful for your own reference when designing different aspects of your project.

You may consider doing this experiment with the robot on the ground and using a long USB cable

[1] or imported into MATLAB for plotting and analysis.

or laptop to obtain the data. This will more accurately determine the response of the motors under load. Also consider performing this experiment starting at the maximum PWM value and stepping down to see if there is any *hysteresis*[2].

3 Line Position Estimation

Hint

Consider fixing the *slop* in your motor mounts to help raise the sensors off the ground a little.

So far in the labs we have only explored sensors that use digital signals. The line sensors you were provided have a fourth pin which we have not yet used, labelled **A0**. Instead of providing a `HIGH` or `LOW` signal it provides a voltage that varies between 0 and 5 V, where a higher voltage indicates a darker surface (less reflection). Just as we can read digital signals with `digitalRead()` we can read analogue ones with a similar function: `analogRead()`. The `analogRead()` function returns a value in the range [0,1023] for a voltage in the range [0,5]. For example, to read the voltage present on pin `A0`:

```
int val = analogRead(A0);
```

Suppose that instead of using the digital information from the sensors which could provide information such as: the line is between sensor (a) and sensor (b); we instead use the continuous values obtained from `analogRead()` to predict a likely position of the line. If we have several sensors connected we can use the information gathered from all of them to estimate the position of the line. Figure 1 depicts some sensor data collected from a robot that is sitting over a line.

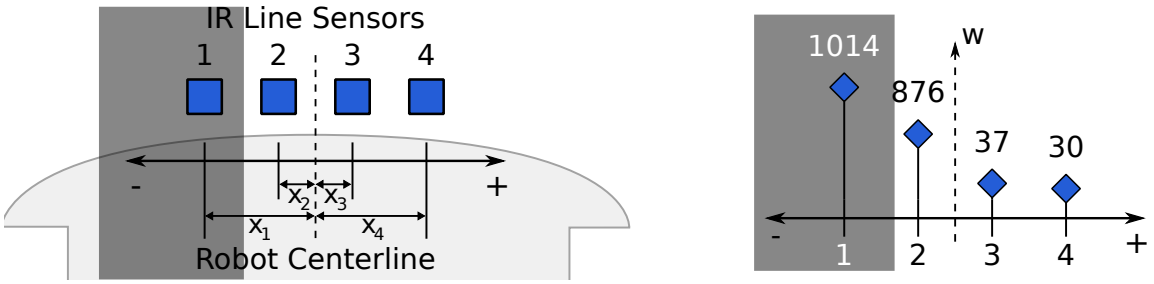


Figure 1: Example data collected from an array of four line sensors.

One method for interpolating the data shown in Figure 1 is to use the *weighted arithmetic mean*[3]

$$\overline{x} = \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i} = \frac{w_1 x_1 + w_2 x_2 + \ldots + w_n x_n}{w_1 + w_2 + \ldots + w_n}, \tag{1}$$

where each w_i is the i-th sensor reading and each x_i is the distance to centre of the i-th sensor from the centroid of the robot. For example, suppose $x_1 = -30$, $x_2 = -10$, ,$x_3 = 10$ and $x_4 = 30$, when we evaluate Equation 1 for the data shown in Figure 1, where $w_1 = 1014$, $w_2 = 876$, $w_3 = 37$ and $w_4 = 30$,

[2] https://en.wikipedia.org/wiki/Hysteresis
[3] https://en.wikipedia.org/wiki/Weighted_arithmetic_mean

$$\overline{x} = \frac{\sum_{i=1}^{4} w_i x_i}{\sum_{i=1}^{4} w_i} = \frac{w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4}{w_1 + w_2 + w_3 + w_4} = \frac{(1014 \times -30) + (876 \times -10) + (37 \times 10) + (30 \times 30)}{1014 + 876 + 37 + 30} = -19.4 \text{ mm}. \tag{2}$$

This is obviously not perfect, but can provide a reasonable estimate for the line position.

Task: Estimate the position of a line using analogue information obtained from 4 line sensors, and determine assumptions of this measurement method.

1. Mount 4 IR line sensors to the front of your robot using *10 mm stand-offs*, shown in Figure 2.

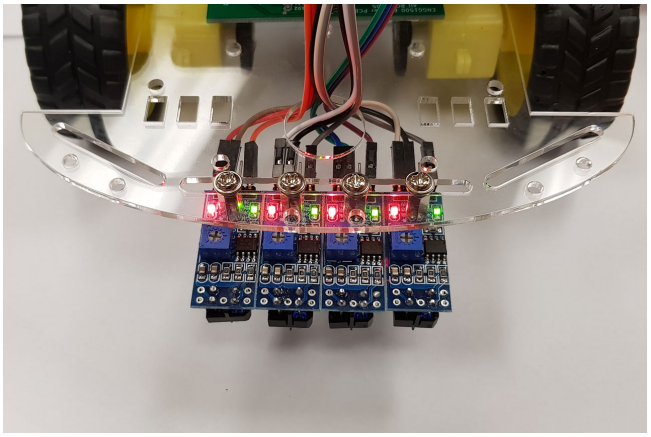


Figure 2: Mounting 4 line sensors

> **Hint**
>
> If you mount your sensors identically to Figure 2 you won't need to change the array of sensor positions in the code template.

2. Connect each of the line sensors to **A0**, **A1**, **A2**, **A3**. Connect their pins in the same order as they are physically mounted.
3. Implement Equation 1 to compute an estimate of the line position, use the template provided in Listing 5.
   a) Use `analogRead()` to read the data into the elements of the variables `w1`, `w2`, `w3`, `w4`.
   b) Calculate the numerator of Equation 1 and store it in the variable `num`.
   c) Calculate the denominator of Equation 1 and store it in the variable `den`.

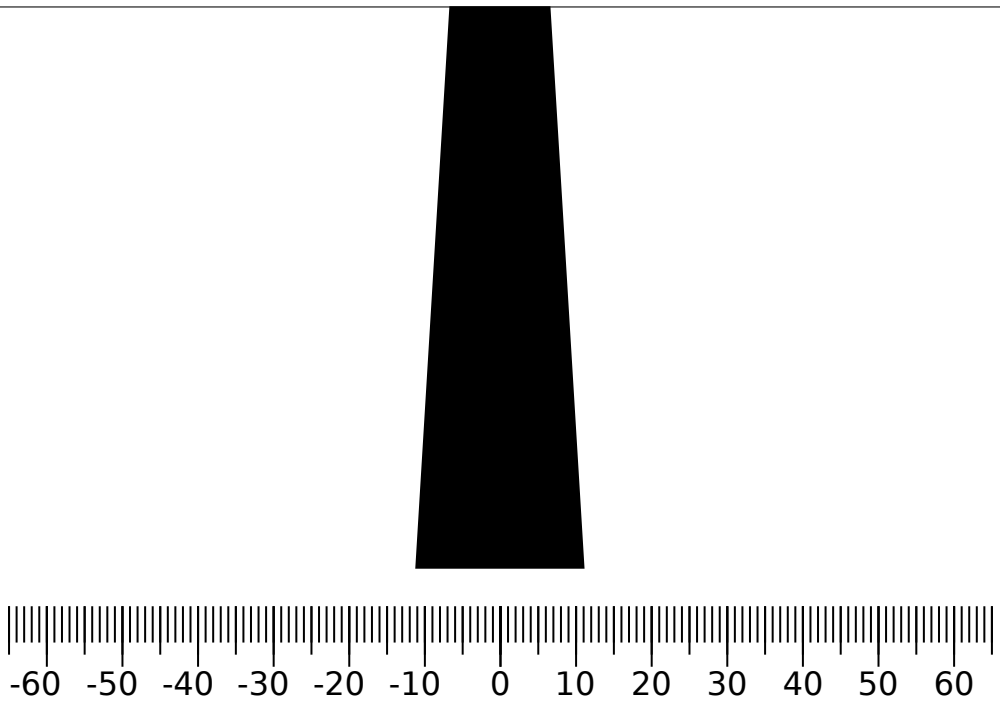
Listing 5: lineCalibrate.ino

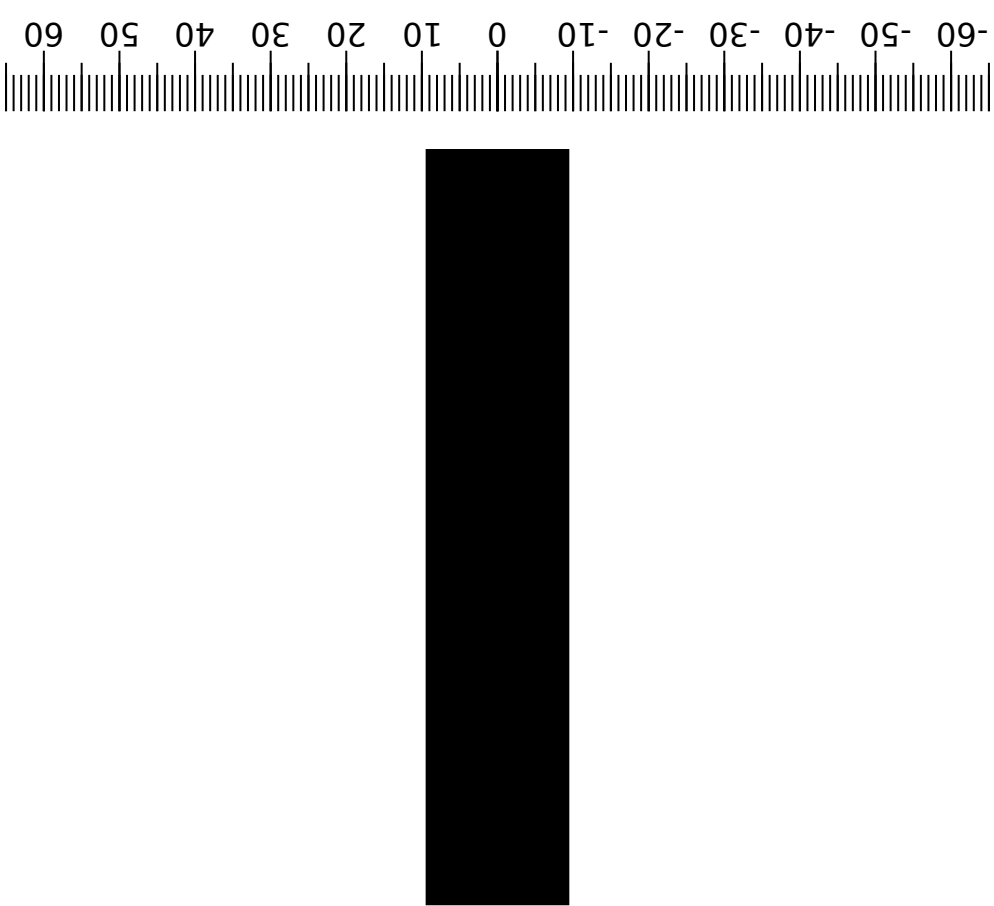
```
//Distances from the centroid of the robot to the centre of each sensor in mm
float x1 = -22.5;
float x2 = -7.5;
float x3 = 7.5;
```

```
float x4 = 22.5;

//Variables to store each data point in
float w1;
float w2;
float w3;
float w4;

//Variables for storing the numerator and denominator of Equation 1
float den = 0;
float num = 0;

void setup() {
  // Initialise pins for line sensors
  pinMode(A0, INPUT);
  pinMode(A1, INPUT);
  pinMode(A2, INPUT);
  pinMode(A3, INPUT);

  Serial.begin(115200);
}

void loop() {
  //  TODO: Take sensor measurements using "w1 = analogRead(pin)" (store sensor
       //...data in w1, w2, w3, w4)
  //  TODO: Calculate numerator of weighted average, store in num
  //  TODO: Calculate denominator of weighted average, store in den

  float lineDist = num/den;

  Serial.print("Distance from line = ");
  Serial.println(lineDist);
  delay(50);  //  50ms delay
}
```

4. Upload the code to your Arduino.
5. Use page 9 (obtain a hard-copy from your lab demonstrator) to assess the accuracy this method for estimating the position of a line. List some assumptions of this method and consider the following:
   - How accurate is it under ideal conditions?
   - Is the method invariant to lighting conditions?
   - How does the method perform for lines of varying thickness?
   - How does the method perform for lighter coloured lines?
   - What happens when no line is present?
6. Discuss with your team how this method of reading a line could be beneficial to your project.
   - How could this be used to create a more accurate line following algorithm.

- Could using this method have a benefit to the passenger (🐤)?


-60 -50 -40 -30 -20 -10 0 10 20 30 40 50 60


60 50 40 30 20 10 0 -10 -20 -30 -40 -50 -60

4 RGB distance sensor calibration

Included in your kit is an APDS9960 RBG colour and gesture sensor, shown in Figure 3. This multi-purpose sensor has several useful functions for your project:

- RGB colour sensor, which might be useful detecting the illuminated yellow pedestrians.
- Ambient light information, which could be used to calibrate your light bases sensors (IR sensors) as a way of avoiding false positive and false negative sensor readings.
- Distance sensor, which could be used instead of the ultrasonic sensor or used for accurate close range distance measurements.

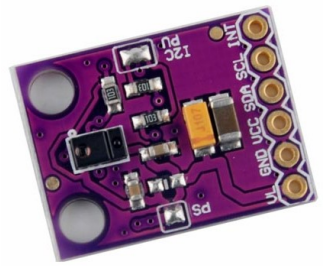


Figure 3: APDS9960 RGB colour and gesture sensor.

For this lab exercise we will focus on the distance sensing capabilities of the sensor. Unlike the ultrasonic sensor which uses time of flight information to produce a metric measurement, the APDS9960 uses light intensity information to produce a measurement with no apparent units.

In this lab exercise you will perform an experiment to calibrate the unit-less APDS9960 distance data against the known metric distance obtained by using the ultrasonic sensor. This data will then be displayed as a scatter plot to obtain a mapping from the sensor data to a metric distance.

4.1 Using the sensor

Warning

In this lab you will use a new sensor, it is only sensor in your kit that requires 3.3 V instead of 5 V. Do NOT apply 5 V to this sensor, there are no spares.

An example of how to connect the sensor to the Arduino is shown in Figure 4 with connections corresponding to Table 1. No mounting hardware for this sensor has been provided or suggested for the project, this is up to your team.

Sparkfun produce an identical sensor to the one you have received as well as a software library that can be used to read gestures, colours, proximity etc. The library can be installed through the library manager in the Arduino IDE.It can be installed by selecting: **Sketch → Include Library → Mange Libraries** and searching for **sparkfun apds9960**.

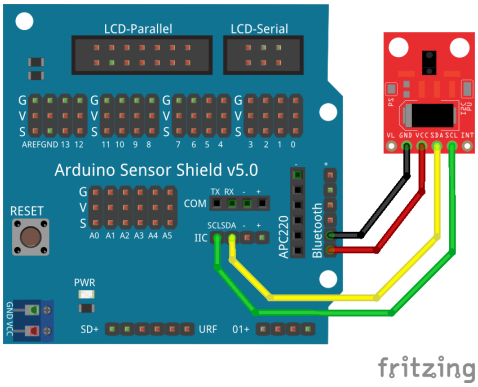


Figure 4: APDS-9960 connection.

Table 1: RGB-gesture sensor pin assignment.

RGB-gesture sensor	Arduino
VCC	3.3 V (not labelled, is located on the Bluetooth rail)
GND	GND
SDA	SDA/A4
SCL	SCL/A5

4.2 Calibration Experiment

In this experiment you will place your robot on a flat surface and move it slow towards or away from a hard flat obstacle, e.g., your kit box or the desk divider. The collected data will be used to interoperate the sensor information.

Task: Obtain a mapping from the APDS9960 sensor data to a distance in millimetres.

Follow the instructions below to perform the experiment.

1. Install the SparkFun-APDS-9960 library mentioned above.
2. Connect the APDS-9960 as shown in Figure 4.
3. Open one one of the examples from the library to ensure the sensor is working correctly, open **Files → Examples → SparkFun APDS9960 RGB and Gesture Sensor → Proximity-Sensor**.
4. Connect the ultrasonic sensor identically to Lab 3.
5. Mount the APDS-9960 directly below your ultrasonic sensor. No hardware has been provided for this, but BluTac or a cable tie should suffice.
6. Implement the pseudo code shown in Listing 6 in Arduino. Complete the `TODO` comments in the template provided on Blackboard, also provided in Listing 7.

Listing 6: APDS-9960 distance sensor calibration.

```
BEGIN
  PRINT DATA HEADER
  FOR i = 1 to 100 (collect 100 data points)
    READ ULTRASONIC
    READ APDS9960
    PRINT CSV DATA
    SMALL DELAY (0.2-0.5 seconds)
  ENDFOR
END
```

7. Complete the `TODO` in the code, by calling the `sonar_mm` function and storing the result in `distance_mm`.
8. **Slowly** move an object toward/away from the two sensors as you begin to log the data using the serial monitor, keeping in mind that the ultrasonic sensor will only work at a minimum distance of ~ 50 mm.
9. Copy and paste the data into Excel.
10. Generate a scatter plot of the APDS-9960 data against the metric data obtained from the ultrasonic sensor to obtain a characteristic curve of the sensor.
11. What assumptions did your experiment assume when you collected the data?
12. Compare your characteristic curve against the one located in the sensor's data-sheet. How do they compare[4]?

This experiment can be improved by running it multiple times with different coloured obstacles and by changing the sensor gain[5] to obtain a better model of the sensor's behaviour.

Listing 7: DistanceCalibration.ino - Distance sensor calibration template

```
/*************************************************************
  Adapted from
  ProximityTest.ino
  APDS-9960 RGB and Gesture Sensor
  SparkFun Electronics
*************************************************************/

#include <Wire.h>
#include "SparkFun_APDS9960.h"

#define ECHO 12
#define TRIG 7

// Global Variables
SparkFun_APDS9960 apds = SparkFun_APDS9960();
uint8_t proximity_data = 0;
unsigned int distance_mm = 0; //This variable will hold the distance

void setup() {
  // Initialize Serial port
  Serial.begin(9600);

  // Initialize APDS-9960 (configure I2C and initial values)
  if ( apds.init() ) {
    // Serial.println(F("APDS-9960 initialization complete"));
  } else {
    Serial.println(F("Something went wrong during APDS-9960 init!"));
  }

  // Adjust the Proximity sensor gain
  if ( !apds.setProximityGain(PGAIN_2X) ) {
```

[4] You will obtain a careful mistrust of hardware data-sheets through your engineering studies.
[5] Change `PGAIN_2X` to `PGAIN_1X`,`PGAIN_4X` or `PGAIN_8X`.

```
    Serial.println(F("Something went wrong trying to set PGAIN"));
  }

  // Start running the APDS-9960 proximity sensor (no interrupts)
  if ( apds.enableProximitySensor(false) ) {
    // Serial.println(F("Proximity sensor is now running"));
  } else {
    Serial.println(F("Something went wrong during sensor init!"));
  }
  // Initialise the Ultrasonic sensor
  pinMode(ECHO, INPUT); //Initialise pin 12 as an input
  pinMode(TRIG, OUTPUT); //Initialise pin 7 as an output
}



void loop() {
  Serial.println("time,APDS9960,HCSR04_mm");
  for (int i = 0; i < 100; ++i)
  {
    // Read the proximity value
    if ( !apds.readProximity(proximity_data) ) {
      Serial.println("Error reading proximity value");
      break;// breaks out of the for loop early
    }
    //Read ultrasonic data
    //TODO: call the function sonar_mm and store the result in distance_mm

    //Print the CSV data
    Serial.print(i);
    Serial.print(",");
    Serial.print(proximity_data);
    Serial.print(",");
    Serial.println(distance_mm);

    // Wait 200 ms before next reading
    delay(200);
  }
  while (1) {}; //infinite loop to stop the experiment
}

unsigned int sonar_mm(void) {
  long duration = 0;
  const float speed_sound = 340.29;// m/s, "const" makes the compiler able to optimise
       //... the program where this variable is used, cool!
  // Read in a distance from the ultrasonic distance sensor:
  // The ultrasonic burst is triggered by a HIGH pulse of 10 microseconds.
  digitalWrite(TRIG, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG, LOW);
  //read length of time pulse
  duration = pulseIn(ECHO, HIGH); //This function measures a pulsewidth and returns
       //...the width in microseconds
```

```
  // convert the time into a distance
  // the code "(unsigned int)" turns the result of the distance calculation
  // into an integer instead of a floating point (decimal or fractional) number.
  return (unsigned int)(0.5 * duration * 1e-6 * speed_sound * 1e3);
  //"unsigned" ensures we are returning an unsigned number, remember that there is no
       //...such thing as negative distance.
}
```

5 Recommendations

Some recommendations to consider for your project that follow directly from this lab include:

- Repeating the PWM to speed calibration experiment under load with the aid of a long USB cable for logging the data, or perhaps a mock dynamometer for your robot to sit in during the experiment.
- Consider how you could use the PWM to speed curve for your project. Could it simply aid in choosing PWM values more wisely? How else could it be useful?
- Experiment with sensor placement for estimating line position.
- Use the algorithm shown in section 3 to develop a line following solution. You could consider using the measurement to determine a difference in left and right motor PWM values.

```
float linePosition = Equation 1;
float error = 0 - linePosition;//This defines the difference between where we
     //...would like the line to be (0, the centroid of the robot) and where the
     //...line actually is (linePosition).
float Kp = 1; //This variable defines the aggressiveness of the line following
     //...algorithm, i.e., the larger this value the more the robot will react to
     //... a small change in the line position.

//Suppose the line position is -10mm, slightly left of the desired position. If
     //...we compute the following for Kp = 1...
int leftPwm  = 100 + Kp*error; //leftPwm = 100+1*(-10) = 90
int rightPwm = 100 - Kp*error; //rightPwm = 100-1*(-10) = 110
//causing the robot to veer left, bringing the line back under the center of the
     //...robot.
```

  This simple algorithm can work well for gentle line geometry, extensions can be made to allow it to handle harsher geometry, e.g., sharp corners.

- Decide how you may use the ultrasonic sensor and APDS9960 sensor with the most utility. Should you use the APDS9960 or ultrasonic sensor exclusively, or a combination of them both? How?
- Formulate conceptual solutions to overcome the various sections of the track where there are no lines to follow, obstacles to avoid, etc. In the next workshop we will show you how to tie together your individual solutions in a modular structure. Until then focus on a working solution to each individual section.

- Allocate different navigation algorithms to different team members (or pairs in your group), read the section entitled *Teamwork in Programming* in the Project Reference Manual for advice.