Lab 7: Rangefinders and Line Following Sensors

1. Introduction

In this lab, students will learn and practice the practical skills necessary to use IR rangefinders. You will create Arduino code to interface with each sensor and will investigate the functionality of each sensor. You will also learn how to use arrays of IR detector/emitters for line following. First you will develop interface code and then begin to create a line following algorithm that will be necessary for your project.

1.1 Lab Objectives

- Understand the practical use of IR rangefinders
- Understand the practical use of IR line following sensors
- Implement some basic digital filtering

1.2 Project Objectives

- Choose, mount, and calibrate rangefinders
- Develop a working line following algorithm

1.3 Lab Hardware

- Lab Robots with:
    - Sharp GP2Y0A41SK0F Infrared Rangefinder (Wall Following Sensor)
    - QTR-MD-08RC Reflectance Sensor Array (Line Following Sensor)
- Jumper wire kit
- Solderless breadboard

1.4 Project hardware

- Sharp GP2Y0AxxxK0F Infrared Rangefinder (Wall Following Sensor)
- QTR-xx-xxRC Reflectance Sensor Array (Line Following Sensor)

2. Laboratory Concepts

2.1 Digital Filtering

In this lab you will cover basic Finite Impulse Response (FIR) and Infinite Impulse Response (IIR) filters. FIR and IIR filters can be generalized with the following relationships,

$$Y_i = \sum_{j=0}^{N-1} X_{i-j} \times H_j \qquad \text{FIR Filter}$$

$$Y_i = \sum_{j=0}^{N-1} X_{i-j} \times H_j + \sum_{j=1}^{M} Y_{i-j} \times K_j \qquad \text{IIR Filter}$$

where Y_i is the filter output at the i^{th} time step, X_{i-j} is the sensor measurement at $(i\text{-}j)^{th}$ timestep, H_j and K_j are coefficients that represent how the different X_{i-j} and Y_{i-j} are weighted in the summations, N is the number of measurements used in the filters, and M is the number of prior filter outputs used by the IIR filter. Note that $H = [H_0 \dots H_{N-1}]$ and $K = [K_1 \dots K_M]$ are vectors of coefficients often referred to as the "filter kernel", which determine the specific behaviors of the filters.

Each new measurement acts as an impulse to the filter, which is where their names are derived from. As the filters operate, measurements are acquired at a sampling rate, f_s, and i increments with each new timestep. New measurements are added to the filter summation and prior measurements are dropped after $N-1$ samples are weighted and summed. The summation considers a finite amount of data, N point, over a finite period of time, $T_{sum} = N/f_s$. Each measurement effects the output of the filter for a period of T_{sum} seconds, which means that filter response is always delayed. Since the FIR only uses N measurements to calculate the output of the filter, a sensor measurement no longer effects the filter output after T_{sum} seconds. This is why it is named a Finite Impulse Response filter; each measurement only effects the filter output for a finite amount of time, T_{sum}.

The IIR filter, on the other hand, includes the weighted sum of past filter estimates, Y_{i-j}, via the coefficients in K, hence impulses contributed by past measurements always contribute to the filter output. Thus, past measurement effect IIR filter output for an infinite amount of time, which is why it is called an Infinite Impulse Response filter. Note that the IIR filter is akin to a transfer function, such as those used to describe the low-pass and high-pass filters in continuous time systems discussed in the prior labs. Those transfer function result in asymptotic responses to step inputs, which require an infinite amount of time to reach their steady state, but we usually approximate them as finite time using the 98% settling time.

While beyond the scope of this class, note that a transform called the "*z-transform*" can be used to transform a continuous time transfer function, $T(s)$, into the discrete domain to form the discrete time transfer function, $T(z)$. The coefficients of $T(z)$ are the filter kernels H and K mentioned above. Deriving $T(z)$ is studied in ME EN 52/6200 Classical Controls and ME EN 52/5210 State Space controls, which both provide a focus on the topic of "digital controls" used by microcontrollers in mechatronic systems.

It is important to be aware of these ideas since they tell us how the IIR and FIR relate to our prior labs and ME EN 3220 Dynamic Systems and Controls, but in this class we will focus on simple examples of the IIR and FIR filters that are commonly used in practice.

Moving Average Filter

One of the simplest examples of a FIR filter is the Moving Average filter. Your microcontroller will "sample" A/D at a sampling frequency, f_s, resulting in a series of samples X_i. To filter (e.g., smooth) data we can implement a moving average. A Moving Average filter is described by,

$$Y_i = \sum_{j=0}^{N-1} \frac{X_{i-j}}{N}$$

where N is the number of data points being averaged. Note that $H = [H_0 \dots H_{N-1}] = [1/N \dots 1/N] = \frac{1}{N}[1 \dots 1]$. If $N = 5$, then $H = \frac{1}{5}[1\ 1\ 1\ 1\ 1]$. As the number of data points, N, increases, the impulse caused by any sample is reduced since 1/N becomes smaller, which results in smoother filter output, but this comes at the expense of increased lag. An A/D measurement will remain in the filter summation for N/f_s seconds. Thus, if we suppose that $N = 5$ and $f_s = 1000Hz$, then a measurement will impact the filter output for $T_{sum} = 0.005$ sec. As N increases, T_{sum} will increase, which increases the lag of the filter.

The Moving Average filter essentially functions as a simple form of a low pass filter. The cutoff frequency of the FIR filter can be approximated by,

$$f_c \approx 0.44 * \frac{f_s}{N},$$

meaning that if a signal has high frequency content above f_c, then it filters that content out, resulting in smoothed output, like that provided by a low pass filter. As *N* increases, the cutoff frequency will decrease, and the filter output will be smoothed further, but at the cost of increased lag as data samples remain in the filter for a longer time.

Weighted Recursive Average

One of the simplest forms of the IIR filter is the Weighted Recursive Average filter, described by:

$$Y_i = \alpha X_i + (1-\alpha) Y_{i-1}$$

where α is a weighting factor with a value between 0 and 1. Note that $H = [H_0] = [\alpha]$, $K = [K_1] = [1-\alpha]$, *N=1*, and *M=1*. As α decreases, each new measurement has less impact on the filter output and the filter output, Y_i, is more dependent upon the prior filter output Y_{i-1}. As α decreases, the data is smoothed more, but again at the expense of phase lag.

Again, akin to the low pass filters discussed previously, we can characterize the filter behavior with a cutoff frequency,

$$f_c = \frac{\alpha}{(1-\alpha)2\pi} f_s$$

which is dependent upon the sampling frequency, f_s, and weighting factor α.

2.2 IR Rangefinders

The figure below shows a simple schematic of the Sharp infrared rangefinders in action. A frequency modulated (FM) pulse of infrared (IR) light is emitted by an LED in the device.

The light travels outward until an object obstructs its path and it is reflected to the sensor. The distance to the object will determine the angle of incidence of the reflected light when it returns to the sensor. The sensor measures the angle of incidence using a special lens and a one-dimensional position sensitive device (PSD). The output voltage of the sensor is determined by the position of the reflected light on the PSD, which corresponds to the distance to the object. The sensor uses a band-pass filter tuned to the frequency of the original pulse so it does not respond to IR light from other sources. This allows it to operate in environments with bright ambient lighting.

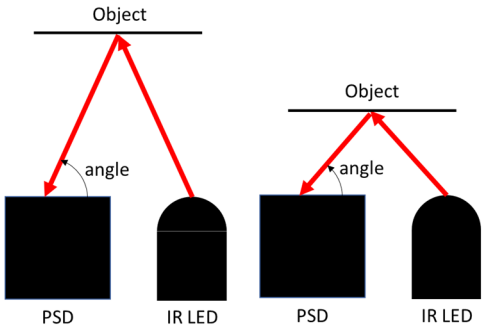

Figure 1. Triangulation method for determining the distance based on reflected angle.

Limitations of Sharp IR Rangefinders

While the Sharp IR rangefinders are simple to use, accurate, and much less noisy than a sonar sensor, they have their limitations. The most important limitation to know is their nonlinear response. Unlike a sonar sensor where distance and time of flight are linearly related, distance and angle of incidence have a nonlinear relationship and the sensor does not correct for this. They also have a minimum detection range, where below this distance objects will appear farther away than they really are. There is also the potential for interference if two or more sensors are pointed at each other. Another problem is with semi-transparent objects, which unpredictably reflect light. In this case, the range reading will be inaccurate, but the sensor will usually detect the presence of an object. Two other issues to be aware of are: (1) excess ambient IR light can saturate the PSD making it unable to detect the reflected pulses, and (2) the response curve has small temperature dependence. Details on these issues can be found in the sensors' datasheets.

Sharp GP2Y0A41SK0F Infrared Rangefinder

This rangefinder follows the concept above. It is comprised of an IR LED, IR detector, and a signal processing circuit. This device outputs an analog voltage corresponding to the detection distance. Because of the sensor's construction, the relationship between distance and voltage is not a linear relationship. The datasheet below shows how the sensor behaves. In this lab you will calibrate it exactly for experimental purposes. In the lab store we have different versions with different ranges.
https://www.pololu.com/file/0J713/GP2Y0A41SK0F.pdf

Figure 2. Sharp GP2Y0A41SK0F Infrared Rangefinder from Pololu.

We can model the relationship between the sensor output voltage V and the distance D as a power law:

$$V = aD^b$$

where a and b are constants to be experimentally calibrated via regression. In order to do a linear regression, we can take the natural logarithm of both sides:

$$\ln(V) = b \ln D + \ln a$$

Now when we plot $\ln V$ vs. $\ln D$, we can fit a straight line whose slope is b and y-intercept is $\ln a$, as shown below. Once a and b are known, we can invert the relationship to compute the distance D, given a voltage measurement V:

$$D = \left(\frac{V}{a}\right)^{1/b}$$

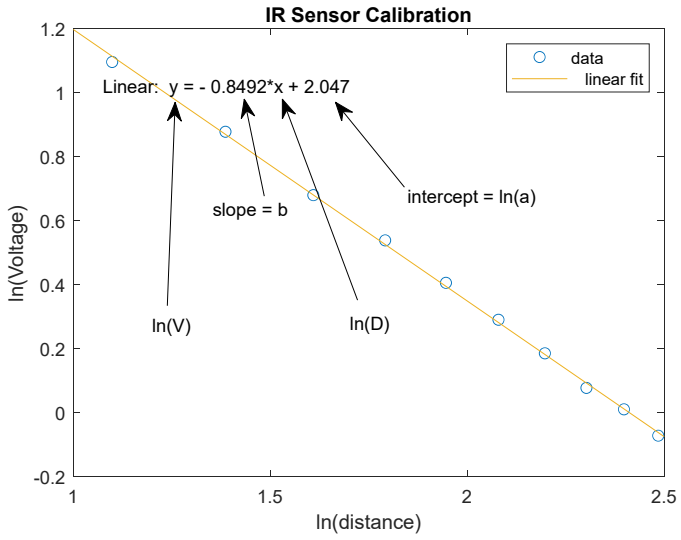

Figure 3. Experimental Calibration of Infrared Rangefinder

2.3 Line Following

Line following is an autonomous method for mobility and navigation. Usually, a robot will follow a black line in a white area or a white line in a black area. To do this, a robot needs a way to sense the line to follow. The best way to do this is using IR emitter/detector pairs, as shown in Fig. 4.

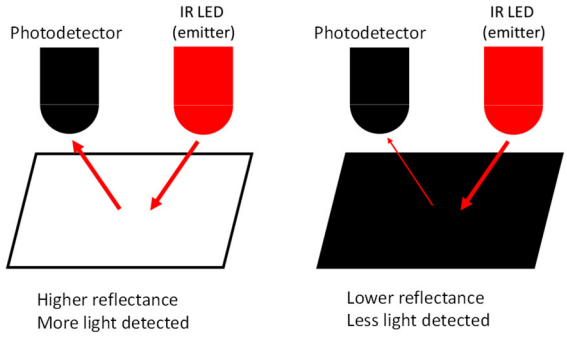

Figure 4. Reflectance sensor pairs made of Photo diodes(detector) and IR LEDs (emitter).

By sensing the surface reflectance, you can differentiate between two surfaces. Using multiple pairs (an array) of IR emitters/detectors allows you to sense where the line is in reference to the robot. This allows the robot to estimate if the line is shifting to the right or left. If the line shifts to the right, turn the robot right; if the line shifts to the left, turn the robot left. This is the simplest form of the line following algorithm.

Limitations of Line Following Sensors

The primary limitations of line following sensors are that the number of pairs determines how accurate and how robust your algorithm is. The wider the pairs are spread apart the larger range of area where lines can be detected. The spacing between pairs also determines how wide of lines you can detect. In most cases, these sensors need to be very close to the surface so the readings can be accurate.

Reflectance Sensor Arrays

The QTR-MD-08RC provides an array of 8 pairs of IR emitters/detectors evenly spaced at intervals of 8 mm. As shown in the circuit diagram in Fig. 5, each sensor in the array consists of an IR LED and a phototransistor in series with a capacitor. To use a sensor, you must first discharge the capacitor by applying a voltage to its OUT pin. You can then measure the intensity of the reflected light by withdrawing the externally supplied voltage and timing how long it takes the voltage to decay back to zero as the phototransistor recharges the capacitor. Shorter decay time is an indication of greater reflection. The advantage of this measurement technique is that it requires no A/D conversion. You just need an array of digital pins on your Arduino to set each OUT high, and measure when each OUT goes low. The Arduino library for this device will handle all this for you and return a range of values for each sensor that corresponds to the reflectance. These sensors also come in different sizes with different numbers of pairs and spacings. A separate emitter CTRL pin allows you to control the brightness of the LEDs, which are on with full brightness by default if the CTRL pin is not connected. To learn more about these sensors, visit: https://www.pololu.com/category/123/pololu-qtr-reflectance-sensors

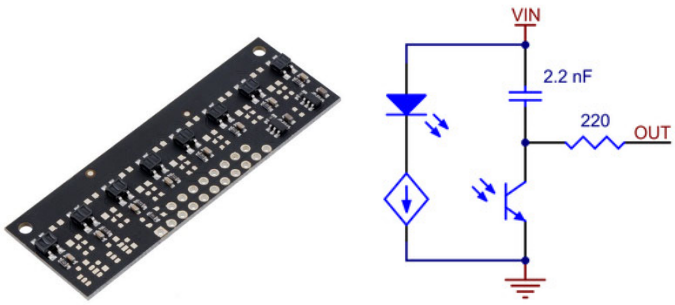

Figure 5. QTR-MD-08RC Reflectance Sensor Array from Pololu.

Line Localization

Each sensor on the reflectance array will provide a reading that should be a measurement of the reflectance underneath each IR Emitter/Detector pair. The best way to use this information to localize the line is shown in the figure below. The sensor readings, A_i, correlate to the reflectance measured at each sensor $S = i = 1, 2, 3, \ldots 6$. In this example, the line is centered just to the left of sensor 4. This example also assumes that the black line will output a larger number than white surfaces. This is just an example and does not match outputs of the sensor used in lab. In lab, the Arduino library will be used to read the A_i measurements for each emitter/detector pair and you will need to write your own code to perform the line location calculation.

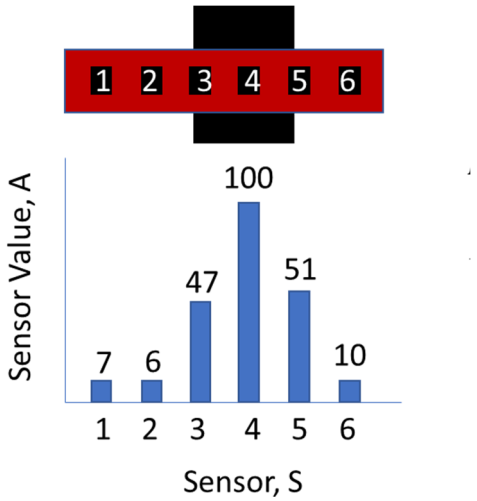

Figure 7. Line localization using an array of IR emitter/detector pairs.

If we let $\boldsymbol{d_i}$ be the locations (e.g. 0, 0.8, 1.6, 2.4, 3.2, 4.0 cm) of each of the 6 sensors, then the line location $\boldsymbol{d}$ will be given by:

$$d = \frac{\sum A_i d_i}{\sum A_i}$$

Proportional Line Following

Suppose that we define ***d*** to be the line location from the example above. In order to have a robot follow the line, we should first define a desired location d_0. In the example above, with a 6 sensor array spaced every 0.8 cm, setting d_0 = 2.0 cm would correspond to the sensor being centered over the line. We could then define a simple discrete line-following algorithm based on the error between the actual line location and the desired line location:

$$error = d_0 - d$$

For example: If error > 1, Turn left
Else if error < -1, Turn right
Else Go straight

This will work, but will lead to jerky line following. A better way is to define a ***proportional feedback law*** such that the degree to which the robot turns depends on the magnitude of the error, which will lead a smoother line following action. To do this, you can define a **base_speed** to set how fast you want the robot to move along the line, and then add/subtract a correction term to each motor based on the error:

$$LeftMotorCommand = base\_speed - Kp*error$$

$$RightMotorCommand = base\_speed + Kp*error$$

where ***Kp*** is a proportional gain factor. Note that depending on how you wire your two motors and calculate your error, you may need to flip signs on the terms in the above equations. Ideally you would wire up your motors so that a positive ***base_speed*** drives both motors forward. If you move the robot to the left of the line center, the error should be positive, the left wheel should speed up and the right wheel should slow down, causing the robot to turn to the right while continuing to move forward. If you move the robot to the right of the line center, the error should be negative, and the opposite should happen. The larger the error, the more sharply the robot will turn. The value for ***Kp*** will determine how sensitive the robot is to the error, and will need to be experimentally tuned to smoothly follow the line. In practice, start with a relatively low value for ***Kp*** (think about how big the ***error*** will be and how much you need to penalize it to make an impact on the motor commands relative to the base speed). The faster your base speed, the larger ***Kp*** will need to be in order to closely follow the line. If ***Kp*** is too large, then robot will tend to overshoot when correcting for the error and oscillate about the line. If you want high-speed line following, you may need to add a derivative term to your feedback law, and if you want your robot to stay exactly centered over the line, you may need to add an integral term to your feedback law, however that will be outside the scope of this lab.

3. Pre-Lab Exercises

1. What is the effective range of the Sharp GP2Y0A41SK0F Infrared Rangefinder? Sketch the theoretical calibration curve supplied in the datasheet. Do not take a screenshot of the original datasheet.

2. What is the optimal sensing distance (from the reflective surface) that the QTR-MD-08RC sensor should be?

3. Create two MATLAB functions to implement a moving average FIR filter (name it FIR_MA) filter and a weighted recursive average IIR filter (name it IIR_WA). The FIR function should take a data set (X) and number of samples (N) to average as inputs. The IIR function should take a data set (X) and weighting factor (α) as inputs. They should output the filtered data set (Y). Bring your functions to lab. Include screenshots of your code and plots in line with this question. Also submit your .m files to canvas. Do not use the Matlab "filter" function; you need to write your own code for these. Be sure to include your name and lab section in the comments of your code. Function definitions are below:

   Y = FIR_MA(X,N)
   Y = IIR_WA(X,alpha)

4. Using the provided signal file **Lab7_prelab4_noisydata.mat**, plot the original data and a filtered version of it on the same plot. Do this for each of the filters below using your functions from the previous problem. Attach these properly labeled plots to your prelab. Use a legend to designate filtered and unfiltered data. The X-axis should be time (sec) and the Y-axis should be voltage (V)). Include screenshots of your code and plots in line with this question. Also submit your .m files to canvas.
   a. IIR: $Y_i = 0.75\, Y_{i-1} + 0.25\, X_i$
   b. IIR: $Y_i = 0.9\, Y_{i-1} + 0.1\, X_i$
   c. FIR: $Y_i = \frac{X_{i-7}+X_{i-6}+X_{i-5}+X_{i-4}+X_{i-3}+X_{i-2}+X_{i-1}+X_i}{8}$
   d. FIR: $Y_i = \frac{\sum_{j=0}^{24} X_{i-j}}{25}$

4. Laboratory Exercises

4.1 Digital Filter

4.1.1 Download the Lab7_GUI and put your IIR_WA function from prelab in the same working directory. The Lab7_GUI will use your IIR_WA function.

- Run the Lab7_GUI and check to make sure that it correctly filters a signal from the function generator.
  - Connect the FG to the NI DAQ ACH0. The GUI will display both the raw unfiltered signal and the digitally filtered signal. Alter α between 1 and 0 to make sure the function works. An α=1 means no filtering. As you lower α, the cutoff frequency of the filter will decrease and you should see more attenuation of the signal.

4.1.2 Once you know the filter is working, disconnect the function generator and connect the output of the front IR rangefinder on your Lab Robot to the NI DAQ ACH0, using the breakouts labelled "RANGE 1" and "GND" on the Lab Robot (see the figure below).

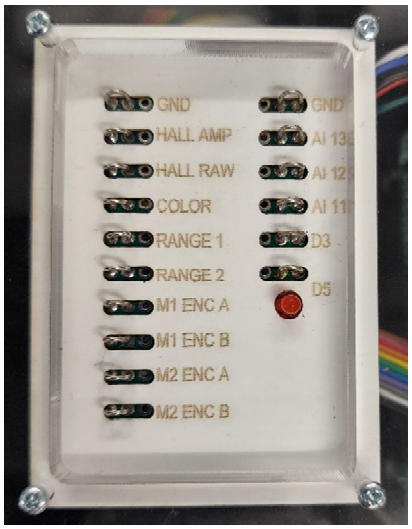

Figure 3. Lab Robot Breakouts

4.1.3 Design an IIR filter for the IR rangefinder.

- Place the Lab Robot on your lab bench and place a block in front of the sensor.
- You should design a filter (choose a value for α) that smooths the data without too much time lag. Move the block back and forth in front of the IR rangefinder. Try to get at least 3 cycles of a wave per 1 second of time to show up in your GUI oscilloscope. Alter your α parameter to find a desirable filter.

4.1.4 Save 3 sets of data using your GUI. Each set of data should have a different α. Choose α's so that you get a wide range of behavior can be seen (over-filtered, well-filtered, and under-filtered). Tip: Pause the oscilloscope by pushing the "Run Oscope" button *before* you hit the save button.

Have a TA check your progress

4.2 IR Rangefinder calibration

While the IR rangefinder on your Lab Robot is connected to the NI DAQ, you will now use the Lab7_GUI to calibrate the IR rangefinder from volts to centimeters.

4.2.1 Calibrate the IR rangefinder

- Referring to the distances in Table 4.2.1 below, use a meter stick to place the block at each of these distances from the face of the rangefinder sensor.
- Record the analog value (average voltage) of the sensor in section 4.4 Student work and Tables.
- In MATLAB, Plot this data with **sensor voltage on the y-axis** and the **distance on the x-axis**. Use linear scaling on both axes. This plot should resemble the data sheet. This plot will be used to answer a question in the post lab. Don't forget you should use data markers to plot sparse experimental data.
- Now create a second plot with **ln(Voltage) on the Y-axis** and **ln(distance) on the X-axis** as shown in Figure 3 (Section 2.2). Only include data from 4-30 cm. In the menu of your MATLAB Figure, select Tools→Basic Fitting, then select a linear fit and display the equation on your plot. Following the example in Figure 3 (Section 2.2), use the values for the slope and intercept to write an equation for Distance as a function of Voltage.
- Record your power calibration equation in section 4.4 Student work and Tables.

4.2.2 Verify the calibration equation.

- Referring to the distances in Table 4.2.2 below, use a meter stick to place the block at each of these distances from the face of the rangefinder sensor.
- Using the voltage readings from the sensor and the equation you derived, calculate the distances the sensor is reading. Record this calculated sensor distance in the table 4.2.2 below. You will use this data in the post lab.

4.2.3 Before you dismantle your sensor. Investigate how the orientation of your surface affects the sensor measurement. Also look at how the surface color/finish affects the measurement. You will want to keep this in mind while working on the project.

Have a TA check your progress

4.3 Reflectance Arrays

We will now experiment with line following using the QTR-MD-08RC sensor on your Lab Robot.

4.3.1 The QTR-MD-08RC Reflectance Array on your Lab Robot is already connected to the Arduino Mega on your Lab Robot according to the table below.

QTR-MD-08RC	Lab Robot Arduino pins
GND	GND
VCC	5V
CTRL (either)	Not used
Sensors 1-8	Digital pins 27-41 (odd pins only) respectively

4.3.2 Download the Arduino sketch *Lab7Template.ino* from Canvas. This template already has code written for you to create a reflectance array object, initialize it in the Setup function, and repeatedly read it in the Loop function. Upload this sketch to the Arduino Mega on your Lab Robot and test the sensor using the provided test piece consisting of a black line on a white surface.

- First place the Lab Robot such that the sensor array is above the white area of the test piece:
  - You should see relatively low values from the sensors, with significant variance across each sensor
  - Determine the average value for each sensor when the sensor is exposed to only white and record it in section Table 4.3 below. (These will be different for each sensor at different distances from the surface)

- When the Lab Robot is moved such the sensor array is straddling the black line, you should see relatively high values for the sensors in the array that are seeing the black line, and low values for those that are still seeing white.

4.3.3 Alter the Arduino sketch to remove the bias from the sensors. Do this in the Loop function where the template says "Put your Line Localization Code Here." For each sensor, you will need to apply the following formula, where the sensor_bias[i] is the average value for the i^{th} sensor that you recorded in Table 4.3.

- Sensor_value_unbiased[i] = sensor_value[i] - sensor_bias[i]

4.3.4 Now using the information discussed in the subsection on Line Localization in Section 2.3 of the lab documentation, add code to compute the line location in centimeters. Use your unbiased sensor values as your A_i and the sensor positions (in cm) as your d_i. HINT: Arduino does not have a "summation" command, so you can use a "for loop". Modify your Arduino sketch to output this error (in cm) to the Serial Monitor.

4.3.5 Finally, compute the error/difference between your actual line location and your desired line location (i.e. halfway between sensors 4 and 5). Test your sketch:

- If your robot is centered over the black line, you should see an error of approximately 0
- If you move the robot right or left the line, the error should be positive or negative, depending your sign convention.

Have a TA check your progress

4.4 Line Following

Now that you can compute your distance from the line, you will use the method described in Section 2.3 of the lab handout to implement proportional line following on the Lab Robot.

4.4.1 Working with your sketch from the prior section, implement a **proportional control law** (see equations in section 2.3) to determine the motor commands for your left and right wheels. Put this in your Loop function after your Line Localization code, right before the commands are sent to the motor driver.

- Put your control law in the Loop function immediately after your line localization code, right before the commands are sent to the motor driver.
- Also uncomment the code in the setup function that allows you to input Kp and base_speed parameters via the serial monitor.

4.4.2 First test your line following sketch by holding the robot over the line **without letting the wheels touch the surface**.

- Verify that when the robot is over the center of the line, both wheels are moving forward. If this is not the case, you might need to flip signs on one of your motor commands.
- Next verify that when the robot is to the right of the line, the right motor speeds up and the left motor slows down, and when the robot is to the left of the robot, the left motor speeds up and the right motor slows down. If not, you may need to flip your sign convention on your error terms.

4.4.3 Now test your line following sketch with the wheels on the ground. Do this experiment on the floor of the lab using the larger test pieces with U-turns and S-turns. Start with a relatively slow base speed and experimentally tune the proportional gain factor so that the robot smoothly follows the line. Then try increasing the base speed and retuning the proportional gain factor. Find three different sets of parameters (base speed and proportional gain) that work well together and record them in Table 4.4 below.

Have a TA check your progress

4.5 Student work and Tables

Name:_____

Table 4.2.1- Rangefinder Calibration

Actual Distance (cm)	Sensor Voltage (V)
2	
3	
4	
6	
8	
10	
14	
18	
22	
26	
30	
34	

Table 4.2.2-Rangefinder Verification

Actual Distance (cm)	Sensor Voltage (V)	Computed Distance (cm)
2.5		
3.5		
5		
7		
9		
12		
16		
20		
24		
28		
32		
36		

Calibration equation: D = f(V)

= _____

Table 4.3 – Reflectance Array Mean Values

Sensor	1	2	3	4	5	6	7	8
Mean (low) value								

Table 4.4 – Line Following Parameters

Base Speed			
Proportional Gain			

5. Post-Lab Exercises

1. Using MATLAB, plot your three data sets of IR rangefinder output from 4.1.4. You should have one plot with three subplots. Each subplot should have a different α value displayed. Show the raw and filtered data super imposed on one another. Distinguish between raw and filtered values. Include screenshots of your code and plots in line with this question. Also submit your .m files to canvas.

2. Using MATLAB, plot the data from table 4.2.1 with sensor voltage on the y-axis and the distance on the x-axis. Use linear scaling on both axes. Attach your properly labeled plot to the Post-lab. Discuss how your plot compares to the plot in the data sheet. Include screenshots of your code and plots in line with this question. Also submit your .m files to canvas.

3. Using MATLAB, plot your IR verification data from table 4.2.2. Fit a line to the Actual (cm) vs. Sensor (cm). A slope of 1 and a bias of zero would be a perfect match. Attach your properly labeled plot with linear fit equation displayed on it. Include screenshots of your code and plots in line with this question. Also submit your .m files to canvas. What does your fit say about the calibration equation you created in lab?

4. How does the shape of an object affect the IR range sensor's measurements? How does the material/surface finish of an object affect the IR range sensor's measurements?

5. Using pseudocode, show how you could check for 3-way intersections while line-following. In other words, what criteria would you place on the sensor values from your reflectance sensor to recognize that you've arrived at the intersection?

6. Project Milestone

The purpose of PM 7 is to take what you learn from Lab 7 and apply it to your project. Your project will likely require some form of range sensing. In Lab 7 you used IR rangefinders. Your project will also likely benefit from some form of line following functionality. In Lab 7 you learned how to use the Reflectance sensors for line following. During Lab 8 (PM7) each team will present the following in a 5 to 10-minute presentation. Upload a copy of your presentation on Canvas before Lab 8 begins. Your presentation should include the following:

- Discuss each team member's contributions for this week.

Rangefinders

Each team must implement at least one rangefinder on their robot for wall following or wall approaching (see below), provide a demonstration of wall following or wall approaching in lab, and discuss these in the PM7 presentation. Hint: Take videos of your robot doing demonstrations when preparing for the milestone just in case something goes wrong on the day of your lab.

- In your presentation:
  - Indicate your choice of rangefinder(s)
    - Discuss what purpose this sensor has in your autonomous method.
    - Indicate in your presentation what the intended operating range is.
  - Provide a photo of the sensor(s) installed on the robot.
  - Discuss any filtering or calibration of the sensor.
  - Discuss your algorithms for wall following or wall approaching. Explain your algorithm using pseudo code, equations, or flow charts. No code snippets.
- Demonstrate rangefinder operation by performing either of the following:
  - Wall following: Driving along a wall at a distance of your choice
  - Wall approaching: Drive towards a wall and stop at a distance of your choice

Reflectance arrays

Each team must implement at least one reflectance sensor for line following, demonstrate line following in lab, and discuss these in the PM7 presentation.

- In your presentation:
  - Indicate how many arrays will be used and where they will be placed. Provide a photo of the sensor(s) installed on the robot.
  - Discuss your line following algorithm. Explain your algorithm using pseudo code, equations, or flow diagrams. No code snippets.
- Demonstrate use of reflectance arrays.
  - Demonstrate your robot performing line following on the curved line of the competition course. The line is comprised of 0.75" wide black electrical tape on a white surface, and has a radius of curvature of about 6", if you would like to practice at home.

Update state transition diagram

- Now that you have included new sensors, update your state transition diagram to incorporate any changes.