LAB 9

Problem

Engineering problems come in many shapes and sizes and so do their solutions. As you've seen in this class, creating a program using Visual Studio and C# is an effective way to engage a computer to perform important problem-solving tasks. More times than not, the computers that do the lion's share of the work in today's innovations do not have a monitor, keyboard, nor a mouse. They are embedded within products and technologies and are less visible but responsible for all of the "brains" in engineered products ranging from smart coffee makers and wireless routers to automation in our airplanes and smart systems within autonomous vehicles. These embedded systems are also written in a language similar to C# and they generally involve obtaining data and processing that data often resulting in an action.

In this lab you use an infrared distance sensor and a servo motor actuator to create and calibrate a distance meter that will display the distance of a moving object. In part A, you'll get started by wiring the distance sensor to the Arduino chip on the Arduino Uno board and acquiring data from the sensor and sending the data through the Serial port to the computer. Within the Arduino development environment is a Serial Monitor through which you will be able to receive the data sent by your system and view them on the computer screen for testing purposes. In part B, you'll write a MATLAB script to receive the data, instead of the Serial Monitor, and plot the distance of the object as it moves. In part C, you'll differentiate the distance data to calculate the velocity and acceleration of the object and plot them on subplots along with the distance using MATLAB. In part D, you'll derive a first order fit of the distance data collected that will depict the trend of the data and create a best-fit polynomial of the data. Finally, in part E, you'll extend your program to control a servo motor that will move an indicator needle that will show the distance, in inches, as a meter. You will need to calibrate the meter.

Setting up the Arduino Development Environment

You will need to install the version of Arduino studio which does not require an "admin privilege" in the lab because of the security restrictions on installing software on the lab computers. The installation file can be find in the following link:

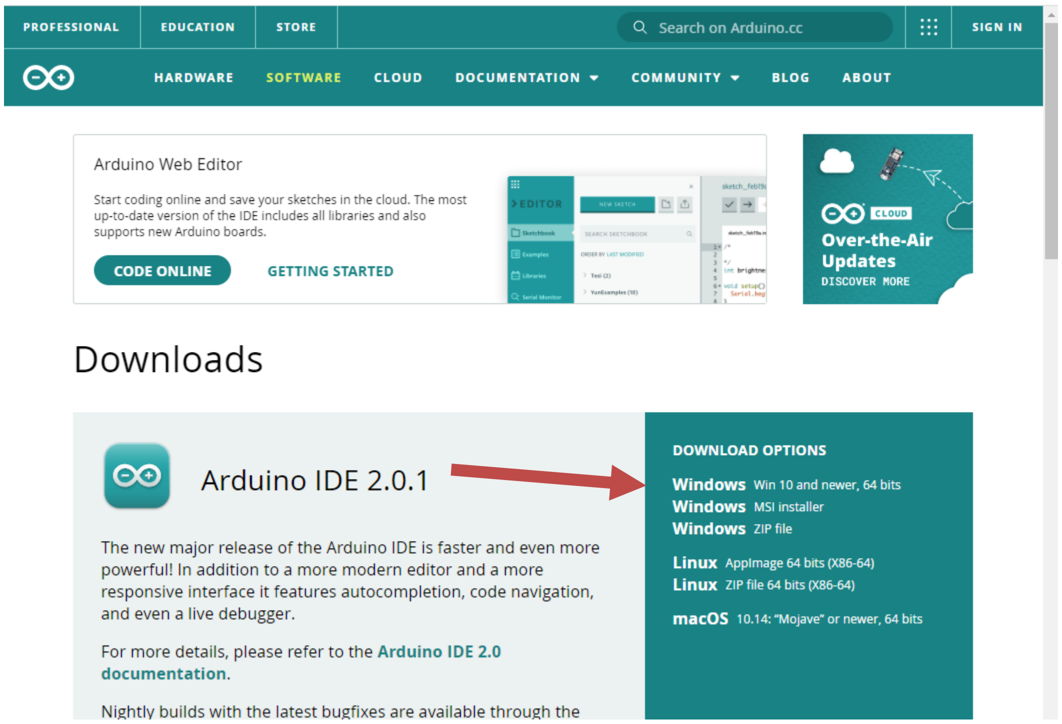
https://www.arduino.cc/en/Main/Software

Figure 1 shows the webpage and the option to choose labeled "ZIP file for non-admin install". Download this ZIP file and unzip the folder contained in the file to your Desktop on the lab computer.

Plug your Arduino Uno development board into a USB port on your computer. This will power up your board and the green light labelled "ON" will be lit.

Double-click the file "Arduino.exe" to run the Arduino Development environment. If you receive a Firewall notice, click "Allow Access".

In the Arduino development environment, select Tools from the menu bar and ensure that the Board selection is "Arduino/Genuino Uno" and that the Port selection is set to the COM port that your board is connected to. Usually the board setting is automatically set correctly but you must select the port setting as shown in Figure 2.



Arduino Web Editor

Start coding online and save your sketches in the cloud. The most up-to-date version of the IDE includes all libraries and also supports new Arduino boards.

Downloads

Arduino IDE 2.0.1

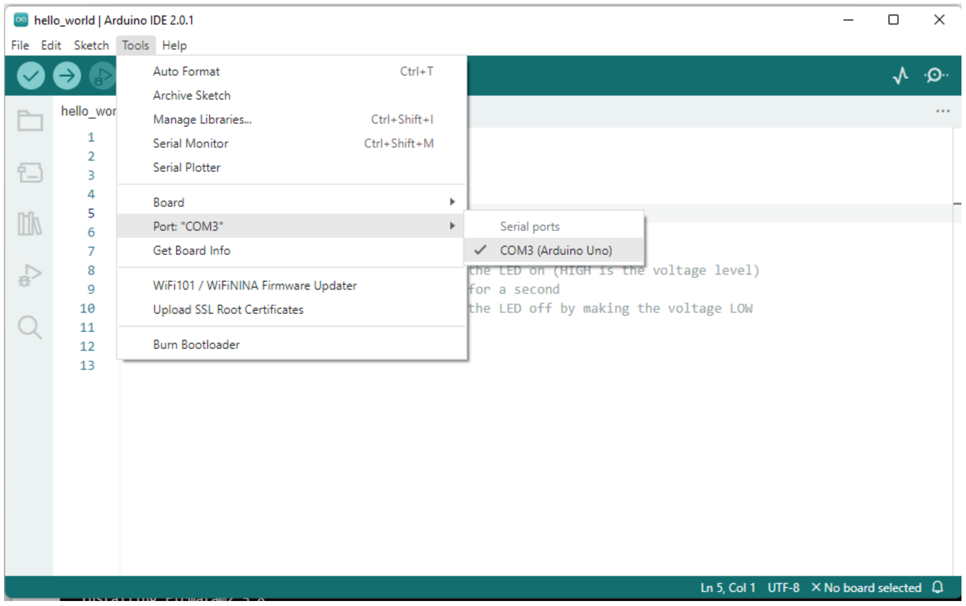
The new major release of the Arduino IDE is faster and even more powerful! In addition to a more modern editor and a more responsive interface it features autocompletion, code navigation, and even a live debugger.

For more details, please refer to the Arduino IDE 2.0 documentation.

Nightly builds with the latest bugfixes are available through the

Windows Win 10 and newer, 64 bits
Windows MSI installer
Windows ZIP file
Linux AppImage 64 bits (X86-64)
Linux ZIP file 64 bits (X86-64)
macOS 10.14: "Mojave" or newer, 64 bits

Figure 1



```
the LED on (HIGH is the voltage level)
for a second
the LED off by making the voltage LOW
```

Figure 2

PART A

In order to connect the infrared distance sensor to the board, we will need to connect the three wires from the sensor to the board header inputs. The three wires are red = power, black = ground, and white = analog signal from the distance sensor. First, plug the wire cable into the sensor head as shown in Figure 3. Next, strip the end of each of the three wires carefully using a pair of wire strippers. You'll want to strip off about 3/4 of an inch from the end. When you strip the end of a wire, you will notice multiple strands of wire will be exposed. Twist these wires into a "single" strand.

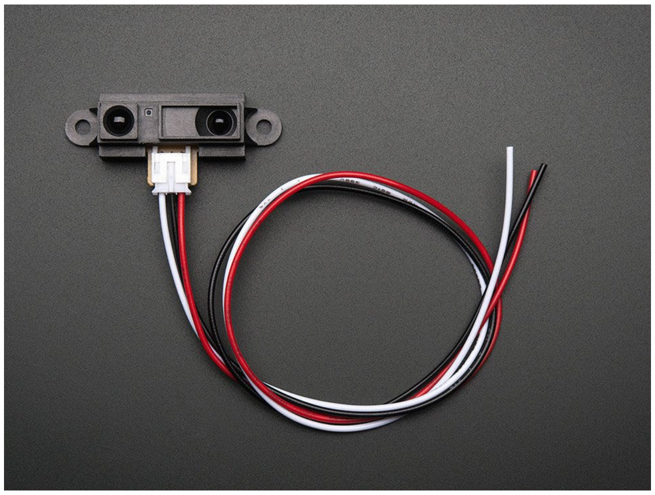


Figure 3

Placing this wire into the header directly would be a bad idea; you would not get a clean electrical contact and the "single" strand that you made would likely unwind and wires would be sticking out everywhere. Instead, take a Male-to-Male wire connector and *carefully* wind one of the wires from the infrared sensor around one of the ends of the connector as shown in Figure 4. Finally, using electrical tape, tape the wire and connector together. It's important that the tape touches the wires and isn't just wrapped around the connector in order to make a solid adhesion.

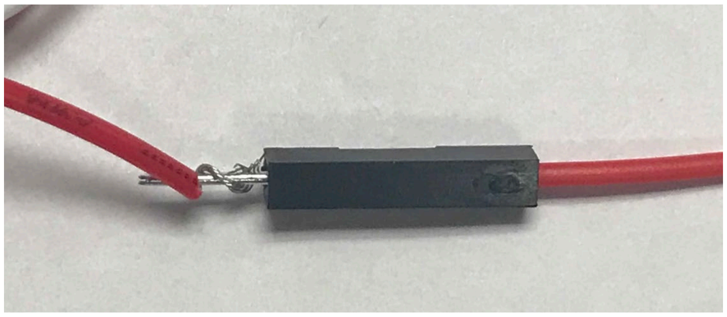


Figure 4

Repeat this process for the remaining wires on the sensor so that all 3 are connected to wire connectors.

Now we are ready to plug these wires into the wire header on the Arduino board. However, you must shut the board off before changing electrical connections. If you do not, you risk damaging something on the board. The Arduino Uno board does not have a power switch. So, you will need to unplug the USB cable from the board temporarily to power the board down. Once you have unplugged it, find the "5V', "GND", and "A0" positions in the header and plug the red, black, and white wires into those positions, respectively. Before powering on your board, double check that the red is plugged into "5V", the black is plugged into "GND", and the white is plugged into "A0" as shown in Figure 5, on the next page. In Figure 5, the white wire is connected to a yellow wire connector so in the picture the white analog data line is actually yellow. If you accidentally switch them and power the board up, you risk damaging the infrared sensor and/or board. After verifying the connections, plug the USB cable back into the board.

Using the Arduino Uno development environment, create a program (Arduino refers to a program as a "Sketch") that will read a value from the sensor using *analogRead*, map the value read from the Analog-to-Digital converter from the infrared sensor from a range of 0 to 1023 to a range of 255 to 0. Note that it is not a range of 0 to 255, instead, it is 255 to 0. The reason for this is that the infrared sensor values increase as the moving object gets closer to the sensor. By mapping the 0 – 1023 from the sensor onto 255 – 0, we can reverse the trend so that farther objects will be closer to 255 and closer objects will be near 0.

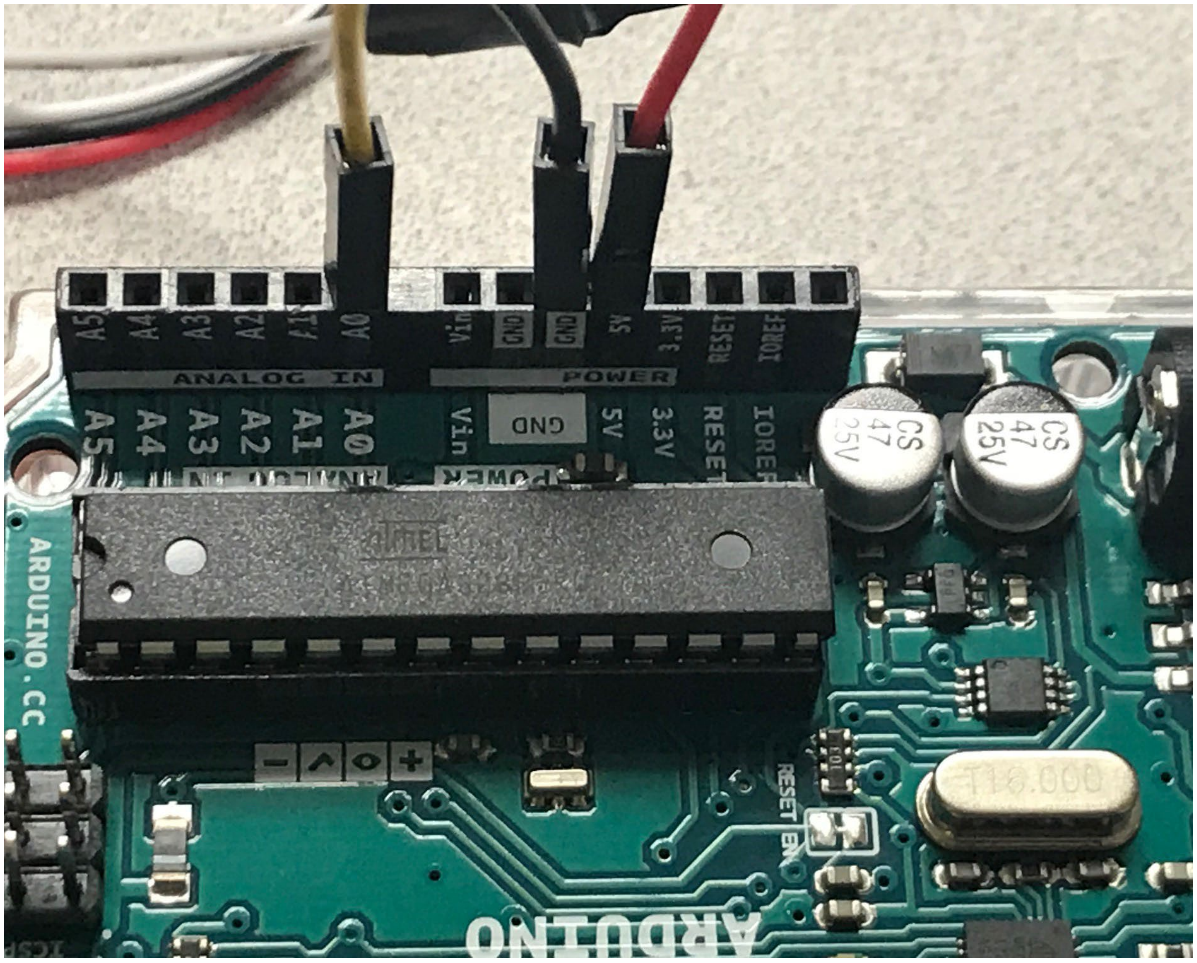


Figure 5

In an embedded system, there is not computer screen on which to view the output of your program. Instead, we will send the mapped values out of the chip through the serial port. The serial port is the USB port that is also providing power to the board. You will need to setup the serial port using the *Serial.begin* method to a baud rate of 9600 in the *setup* part of the sketch:

Serial.begin(9600);

In the *loop* part of the Sketch, write your main program that will repeat. Since *loop* is automatically repeated "forever" as long as the chip has power, you do not need to write a loop explicitly. Your program should obtain the digitized data from the infrared sensor, map it as described above, and output it to the serial port. Don't forget to add a delay using the *delay* method at the end of your loop to give a little time for the system to pause before looping. The *delay* method takes one parameter, that is the time to delay in milliseconds. For this part, you can use 10, for example, to delay 10 ms before looping and reading the next sensor value. This will give time for the Analog-to-Digital converter to convert the next sensor value before reading it again, otherwise, you may end up reading the same value that you did in the previous iteration.

When you have finished writing your program, you can verify your sketch doesn't have any errors by selecting *Sketch → Verify/Compile* from the menu bar. If there are no errors, you are ready to upload it to the board using *Sketch → Upload*. After uploading your program to the embedded system, open the serial monitor that is built-in to the Arduino development environment by selecting *Tools → Serial Monitor*. Move your hand closer and farther away from the sensor and verify that the values on the serial monitor are changing as they should.

PART B

The serial monitor is useful for verifying that your program is working correctly but you cannot do anything more with the data other than view them. We will use MATLAB to manipulate and plot the data obtained from the sensor. Write a MATLAB script that will read data from the Serial port, convert the data from a string to a number, concatenate the number into a vector in MATLAB, and using the *plot* function in MATLAB, plot the data. Repeat this for 100 data points.

There are two important things that you will need keep in mind when writing your programs. In your Arduino program, you should increase the delay to 500 ms (half of a second) otherwise the 100 points will be taken very quickly and you won't have time to move your hand closer and further. To illustrate this point, consider the following calculation:

If we left the delay at 5 ms from Part A, 100 points would take

$$\frac{100\ points}{}\frac{5\ ms}{1\ point} = 500\ ms$$

This means that when you run your MATLAB program, in half of a second it would receive 100 points to plot. This would not give you much time to move your hand in front of the sensor. So, if we increase the delay to 500 ms, MATLAB will collect 100 points in

$$\frac{100\ points}{}\frac{500\ ms}{1\ point} = 50000\ ms = 50\ seconds$$

which gives you plenty of time to move your hand closer to and further away from the sensor.

The second thing that you will need to keep in mind is on the MATLAB side. Since you are collecting a point, converting it, concatenating it to a vector variable, then plotting that vector within a loop, you will need to use the *drawnow* MATLAB command at the end of your loop so that MATLAB will update the plot. If you do not use this, MATLAB will not actually update the plot until your script has completed; that is, until after it has collected all 100 points. We want it to plot the points as they are received so to do so, we need to us *drawnow*.

Remember to close the serial port using *fclose* in MATLAB. Also remember that you must close the serial monitor within the Arduino environment, otherwise MATLAB will not be able to open the serial port because it will already be open by the serial monitor.

When you have finished modifying your Arduino program, upload it to the board, then run your MATLAB script. Move your hand closer to and further from the sensor and verify that your plotted data reflects your movements. A sample plot is shown in Figure 6. Remember not to move your hand too quickly since we are only collecting data from the sensor every 0.5 seconds.

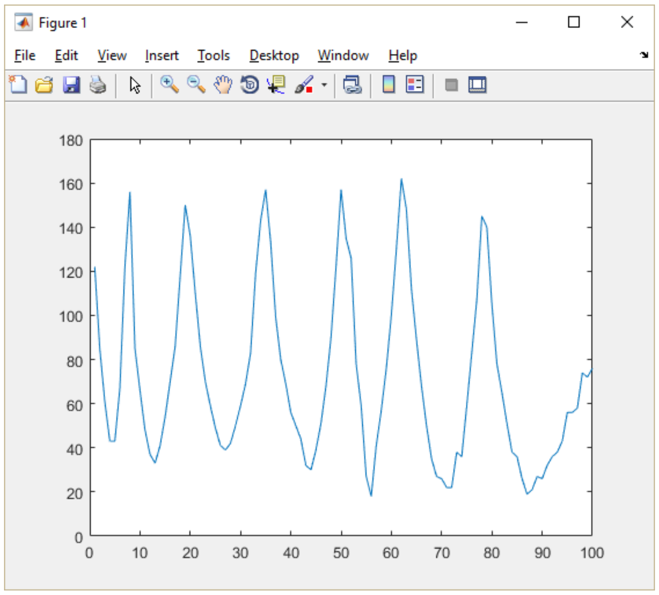


Figure 6

PART C

Now that we've successfully collected data from the sensor and plotted it in MATLAB, let's differentiate the data to calculate the velocity and acceleration of the moving object. Recall that given p(t) is the position of the object, v(t) = p'(t) and a(t) = v'(t), where *v* is velocity and *a* is acceleration. To accomplish this, we do not need to change the Arduino program on the embedded side, we need to change the MATLAB script to calculate the velocity and acceleration and plot those. Additionally, we want to create subplots in MATLAB so that we can plot the position (distance), velocity, and acceleration within one figure window on separate graphs as shown in Figure 7.

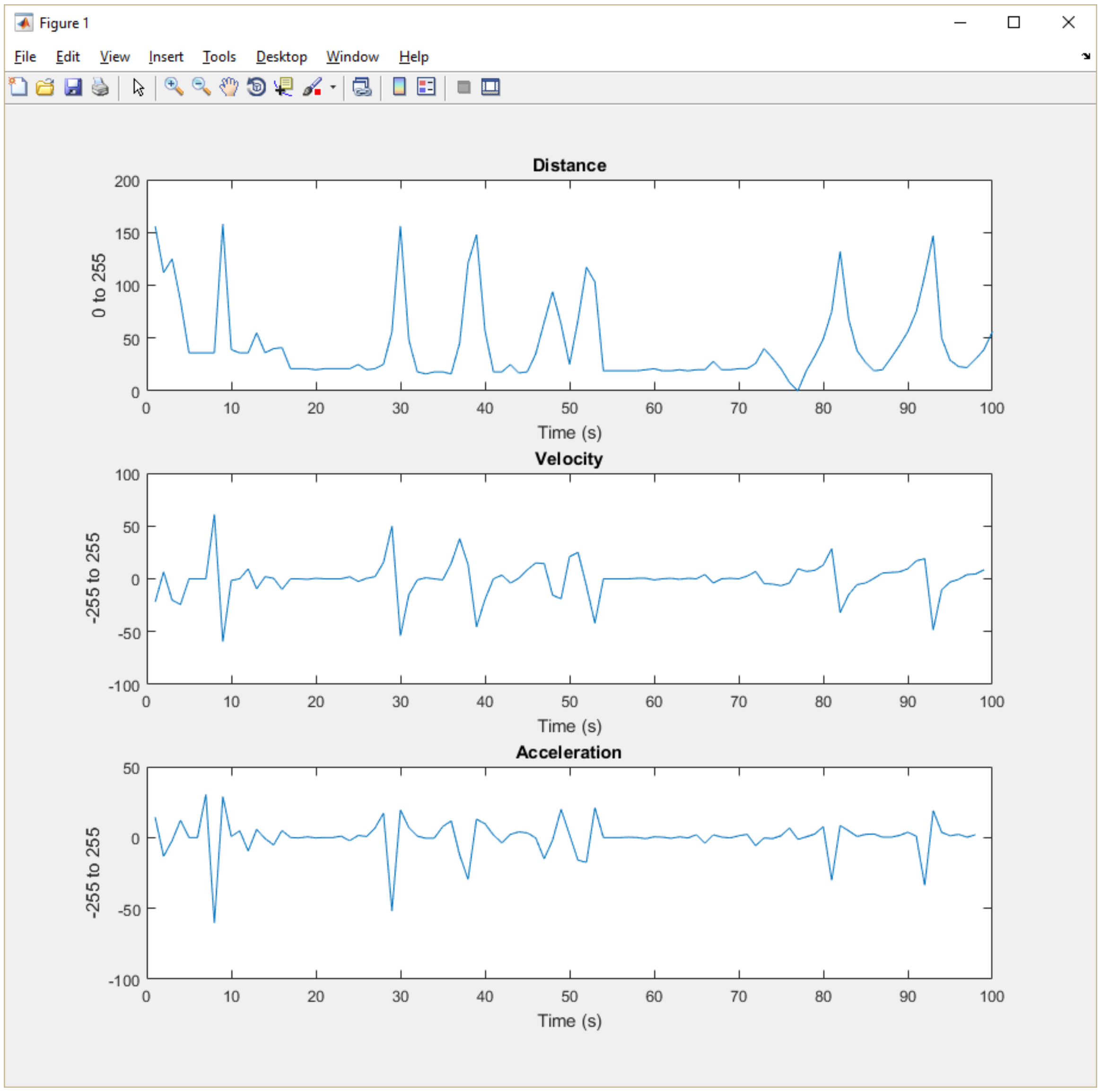


Figure 7

Calculating the velocity and acceleration from the position can be done by subtraction. Consider the vector *p* containing position data:

P = [23 34 36 37 39 41 44 46 47 …]

The first three values in the velocity vector would be calculated by:

V = [P(2) – P(1)   P(3) – P(2)   P(4) – P(3) …..]

and the first three values in the acceleration vector would be calculated by the difference of velocities:

A = [V(2) – V(1)   V(3) – V(2)   V(4) – V(3) …..]

Given these equations, when collecting data, you would need to have at least 2 position points to compute the first velocity to concatenate to the velocity vector and you would need to have at least 2 velocity points to compute the first acceleration to concatenate to the acceleration vector.

In other words, in the loop in MATLAB where you are collecting data, you can not calculate a velocity point during the first iteration of the loop, you must wait until the loop counter is at least 2. Likewise, you can not calculate an acceleration point during the first nor second iteration of the loop, you must wait until the loop counter is at least 3.

To create subplots on one plot, you can use the *subplot* function in MATLAB. Consider the example below:

```
subplot(3,1,1)
plot(p)
title('Distance')
xlabel('Time (s)')
ylabel('0 to 255')

subplot(3,1,2)
plot(v)
title('Velocity)
xlabel('Time (s)')
ylabel('-255 to 255')


subplot(3,1,3)
plot(a)
title('Acceleration)
xlabel('Time (s)')
ylabel('-255 to 255')
```

This will plot position, velocity, and acceleration on a single graph that contains three subplots. The *subplot* function has 3 parameters. The first parameter is the total number of subplots vertically that will be in the plot, the second parameter is the total number of subplots horizontally that will be in the plot, and the last parameter is where the next *plot* function should plot the data. In our case, we want 3 subplots vertically, 1 horizontal column, and we want to set the current subplot (the third parameter) to 1 then plot the position, then 2, then plot the velocity, then 3, then plot the acceleration.

The *title*, *xlabel*, and *ylabel* functions will apply to the plot that was just made. So, you would put those after the *plot* statement to label the current subplot.

Run your MATLAB script and move your hand closer to and further from the distance sensor at varying and constant rates to verify that the distance, velocity, and acceleration plots change accordingly.

PART D

Now that you have collected 100 points of data from the distance sensor, you can derive a least-squares fit of the data, this will show a trendline of the data. Using MATLAB, compute a_0 and a_1 coefficients for a first-order least squares fit. Notice that after your script finished executing, the 3 subplots remained on the screen along with the vectors of data from your script. You can see these in your workspace variables on the right-hand side in MATLAB. You can continue to work with the data collected from your script since it is still active in the workspace by typing commands in the command window.

Recall that in problem-solving assignment #3, we found the coefficients for a least-squares fit to data for both 1st- and 2nd-order polynomials – by hand and using Excel.

FIRST-ORDER APPROXIMATION: A TRENDLINE

We derived the formulas for a1 and a0, the two coefficients for a 1st-degree polynomial fit by minimizing the squared error:

$$a_1 = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{n\sum x_i^2 - (\sum x_i)^2} \qquad \text{and}$$

$a_0 = \bar{y} - a_1 \bar{x}$ where $\bar{x}$ represents the average of x and $\bar{y}$ represents the average of y

The MATLAB commands to perform each of these sums is given below in Listing 1. This assumes that your position vector is *P*. You may have to change *P* to whatever you named your position vector if it is different. You will also need to create an *X* vector that represents time going from 1 to 100. Listing 1 shows this in the first line of MATLAB commands. The *plot* function already assumed that the 100 points have an x-axis of 1 2 3 … 100, but for computing the least squares approximation for the data, we will need an *X* vector explicitly.

Listing 1: MATLAB commands for *a1* and *a0*

```
X = 1:100;   % CREATE A VECTOR FOR X (TIME, 1 THRU 100) TO USE IN THE CALCULATION
Y = P;       % SET Y TO P, THE POSITION VECTOR WITH 100 DATA POINTS COLLECTED
XSUM = SUM(X,2)
YSUM = SUM(Y,2)
N = SIZE(X,2)
XSQUARES = X .* X
SUMOFXSQUARES = SUM(XSQUARES,2)
XTIMESY = X .* Y
SUMOFXTIMESY = SUM(XTIMESY,2)
A1 = (N*SUMOFXTIMESY - XSUM * YSUM)/(N*SUMOFXSQUARES - XSUM^2)
A0 = (YSUM/N - A1 * XSUM/N)
```

Remember, .* (dot star) is an element-by-element multiplication and the star without the dot (*) is a standard matrix multiplication. Also recall that the sum function will give a vector of column totals, but *sum*(*matrix*,2) will give you a row vector of totals which means in this case a single number since there is only one row of data in X and Y.

Run the MATLAB commands shown in Listing 1 to obtain *a0* and *a1*.

Use *PolyFit* to determine the 1st-order coefficients. If you did this correctly, *a1* and *a0* obtained from direct computation and using the *PolyFit* functions should be the same. Print out the Command History and write the values obtained for both coefficients on the printout.

Use *PolyVal* to create a vector called *Trend* of predicted values (trendline) and plot the trendline on the subplot along with the position:

```
subplot(3,1,1)
plot(X,p,X,Trend)
```

This will plot position *p* and trendline *Trend* on the same subplot with *x*, which is simple 1:100, as the x-axis for both.

SECOND-ORDER APPROXIMATION

Recall that using partial differentiation we derived the equations for a2, a1, and a0 for a 2nd-order polynomial:

$$\begin{bmatrix} n & \sum x_i & \sum x_i^2 \\ \sum x_i & \sum x_i^2 & \sum x_i^3 \\ \sum x_i^2 & \sum x_i^3 & \sum x_i^4 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \sum y_i \\ \sum x_i y_i \\ \sum x_i^2 y_i \end{bmatrix}$$

that can be solved by inverting the matrix as shown:

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} n & \sum x_i & \sum x_i^2 \\ \sum x_i & \sum x_i^2 & \sum x_i^3 \\ \sum x_i^2 & \sum x_i^3 & \sum x_i^4 \end{bmatrix}^{-1} \begin{bmatrix} \sum y_i \\ \sum x_i y_i \\ \sum x_i^2 y_i \end{bmatrix}$$

Refer to the MATLAB commands in Listing 1 to create MATLAB commands to determine the coefficients *a0, a1,* and a2 for a second-order fit:

Enter these data into X and Y matrices (vectors)

Compute the XSum and YSum using Matlab's sum function. Remember to use help sum if you need help using the function.

Compute the XSquares using element-by-element multiplication

Compute the XCubes using element-by-element multiplication

Compute the XFourths using element-by-element multiplication

Compute XtimesY using element-by-element multiplication

Compute XSquaresTimesY using element-by-element multiplication

Compute the SumofXtimesY

Compute the SumofXSquaresTimesY

Compute the SumofXSquares

Compute the SumofXCubes

Compute the SumofXFourths

Set n to the correct value for this problem

Build the Matrices A and B using these computed values according to:

$$\underbrace{\begin{bmatrix} n & \sum x_i & \sum x_i^2 \\ \sum x_i & \sum x_i^2 & \sum x_i^3 \\ \sum x_i^2 & \sum x_i^3 & \sum x_i^4 \end{bmatrix}}_{A} \underset{\text{Coeff}}{\underset{\uparrow}{\begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix}}} = \underbrace{\begin{bmatrix} \sum y_i \\ \sum x_i y_i \\ \sum x_i^2 y_i \end{bmatrix}}_{B}$$

(hint: A = [val1,1 val1,2 val1,3 ; val2,1 val2,2 val2,3 ; val3,1 val3,2 val3,3]; remember that the semi-colon is used to start a new row)

Invert A

Multiply matrices $A^{-1}B$ to solve for the coefficients

Use *PolyFit* to check your answers for this 2nd-order polynomial fit.

Use *PolyVal* to create a vector called *Trend2* of predicted values (trendline) and plot the trendline on the subplot along with the position:

```
subplot(3,1,1)
plot(X,p,X,Trend2)
```

This will plot position *p* and trendline *Trend2* on the same subplot with *x*, which is simple 1:100, as the x-axis for both.

BEST FIT APPROXIMATION

By inspecting your position data, determine the order of polynomial that would best fit your data. Remember that an n-order fit can have n-1 points of inflection. That is, it can “change direction” n-1 times. Think of a parabola, for example. A parabola is a polynomial of degree 2 and it changes direction 1 time. Likewise, a polynomial fit of degree 6, for example, can change direction 5 times.

Once you have determined the best fit for your data, use *PolyFit* to determine the coefficients and *PolyVal* to create a *TrendX* vector. Plot your *TrendX* with your position data on the position subplot:

```
subplot(3,1,1)
plot(X,p,X,TrendX)
```

Show your lab mentor your best fit polynomial approximation to the position data that you collected in this lab.

PART E

An embedded system is used often to solve problems where data are collected, processed, then an action is taken (actuation). So far, we have read the data from the infrared distance sensor and processed it (mapping it to 255 to 0 in our case), but we have not caused any action based on the data collected. In this section, you will connect a Servo to the Arduino development board and you will control the servo to turn to different angles depending on whether the moving object is closer to or further from the distance sensor. You will create a "distance meter" where the servo turns a pointer (toothpick) pointing to the distance shown in inches.

The servo, shown in Figure 8, already has a female header port connected to it. So, we just need to connect 3 male-to-male cable connectors directly from this header to the Arduino board. There is one problem, however. The board only has one 5-volt supply pin and one 3.3-volt supply pin. Both the infrared sensor and the servo must be powered by a 5-volt supply. So, we will need to add the red power cable connector connected to the servo's red power port to the taped power junction that we made earlier. This will connect both the infrared distance sensor and the servo to our single power line.

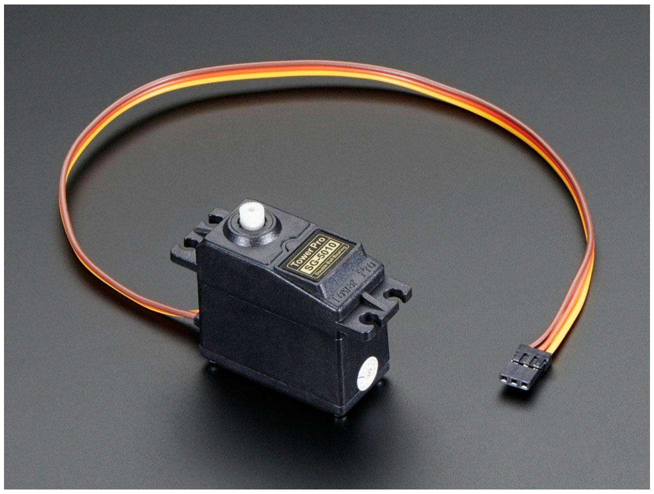


Figure 8

Remember to unplug the USB cable (power) from your Arduino board when changing the electronic configuration of the devices connected to it to avoid risking damage to the board or components.

Other than the power, we can simply plug the male cable connector for the black/brown ground cable into the second GND port on the Arduino board and the servo's yellow data *input* to the header port labelled ~9 on the Arduino board. This pin will output a pulse-width modulated signal (PWM) that will control the servo.

Once the connections have been made, verify that they are correct, and plug the board back into the USB power cable.

Another challenge to this setup is with the power required to run both of these components at the same time. This is often the case in embedded systems; power has to be carefully managed. The USB cable can supply a maximum current of 0.5 A. This is more than enough to power the sensor, in fact, it could power several sensors. But when you add a motor to the design like the servo, more current (and therefore power) is needed to drive a motor. The Arduino board has a secondary power plug port to which you can plug a DC power adapter to add more current and power input to the board. You have been given a DC power adapter, be sure to plug that into the board and the other end into a power strip or wall plug. This will provide enough power to properly operate both the distance sensor and the servo motor.

Now that the hardware is connected and powered correctly, add to your Arduino program the ability to control the servo based on the mapped distance value (255 to 0). To do this, you will need to add an include statement for *Servo.h*. This is equivalent to the *imports* statement in C#. The include statement needs to be added to the top of the program in the "global" area:

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

You will also need to declare a "global" variable of type *Servo*:

```
Servo myservo;  // create servo object to control a servo
```

In the setup part of the program, you will need to use myServo's *attach* method to attach the servo to a particular pin on the board. In our case, we have connected it to pin 9:

myservo.attach(9); // attaches the servo on pin 9 to the servo object

delay(15);

along with a short delay to give some time to the Ardiuno chip to set the connections. Fifteen milliseconds is enough.

Finally, in the "loop" part of the program, use the Servo's *write* method to write values to the servo. These values will change the angular position of the servo.

Attach the plastic component that has 4 plastic structures on it to the white servo rotator. You can do this simply by pressing it on, there is no need to press hard nor use tools.

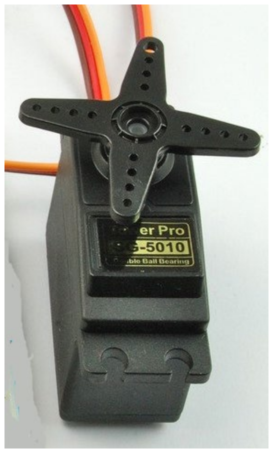


Figure 9

Now tape a toothpick to one of the plastic structures to make a "meter pointer". Place the servo on top of a white piece of plain paper. Upload your Arduino program to the board and by moving your hand closer to and further from the distance sensor, estimate the distance in inches and write those distances on the paper as the toothpick points to it to create a labeled distance meter that will show the distance of the object from the sensor in inches.

Demonstrate this to your lab mentor.

STEPS FOR SUBMITTING YOUR LAB:

On Moodle, there is an assignment for the EXTRA CREDIT ARDUINO LAB.

For each section that you completed, paste all of the items in a Word Document labelled by section (i.e. PART A, PART B, etc..) with the items shown for those sections below and submit the entire zip file at once under the EXTRA CREDIT ARDUINO LAB ASSIGNMENT on Moodle.

PART A – Getting started with the Arduino and the Serial Monitor

1. Screenshot of your Arduino Sketch (code)
2. Screenshot of the Serial Monitor with numbers from the sensor
3. Picture of your board wired up with the sensor (use your camera phone)

PART B – Plotting data in MATLAB using the serial port

1. Screenshot or paste your MATLAB code
2. Screenshot of the plot

PART C – Computing and plotting position, velocity, and acceleration

1. Screenshot or paste your MATLAB code
2. Screenshot of the plot containing the subplots

PART D – Trendline using *Polyfit* and *Polyval*

1. Screenshot or paste your MATLAB code
2. Screenshot of the plot with trendline

PART E – Using the Servo

1. Screenshot of the Arduino Sketch (code)
2. Picture of the Servo wired up to the board