LAB 3 REPORT

By

Qais Qamhia

(NB-427)

18/04/2020, Bremen

Jacobs University Bremen

Introduction to Robotics and Intelligent Systems Lab

Dr. Fangning Hu, instructor

Introduction:

In this lab, I took a deeper look into Processing and its applications. Particularly it focuses on 3-dimensional transformations and how to implement and connect it with your Arduino circuit.

It also introduces the Accelerometer and shows some of its possible implementations in our intelligent system.

The second part of this lab talks about system, and circuit analysis using Matlab Simulink. It goes through some basic circuits enabling you to visualize the inputs and outputs of each without having to create any physical circuits in real life.

Execution:

Task 3.1:

The code will be executed as the following:
It defines a window that renders the frames in 3 dimensions using the P3D command. Then defines variables rotx and roty and sets them to 0.

In the draw loop, we make the background white (it's looped, so it will delete the traces of any following object translations in the loop). Then we use the function pushMatrix() which creates a matrix stack and pushes the following transformation matrices into the matrix stack, until it reaches popMatrix() where it'll pop these transformation functions out of the stack and start reading them (in their new order).

The transformation functions:

In translate() function, we have width and height, both of which are system variables that store the width\height of the display window. This value is set by the first parameter of the size() function.
So, basically this translation makes the object in the middle of the define window, and the -50 defines the z-coordinate (how close or far away the object is from us).

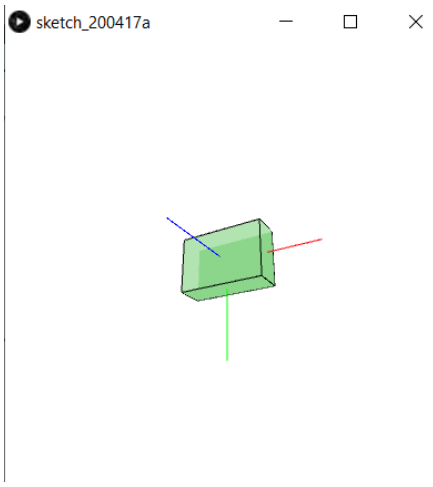
For rotateX() and rotateY(), they will rotate the object around the x-axis and y-axis respectively, by a specified angle as a function argument. This angle should be specified in radians (0 – 2Pi). Such that positive numbers rotate in a clockwise direction and negative numbers rotate in a counterclockwise direction.

rotateX() will take rotx as an argument, which gets a new value each time the mouse gets clicked and dragged. The new value will be the previous Y-coordinate of the mouse minus the new value multiplied by 0.01. This rate isn't arbitrary. We're basically dividing by a hundred so that the values resulted will be in the range of our rotateX() function (0 – 2Pi).

Same goes for rotateY() with the only difference that in the latter, the new roty value will be the

current X-coordinate of the mouse minus the previous value multiplied by 0.01.

As a result of these two, the movement will be as follows:



Movement on Y (rotation around X):

Up to down - negative value - CCW
Down to up - positive value - CW

Movement on X (rotation around Y):

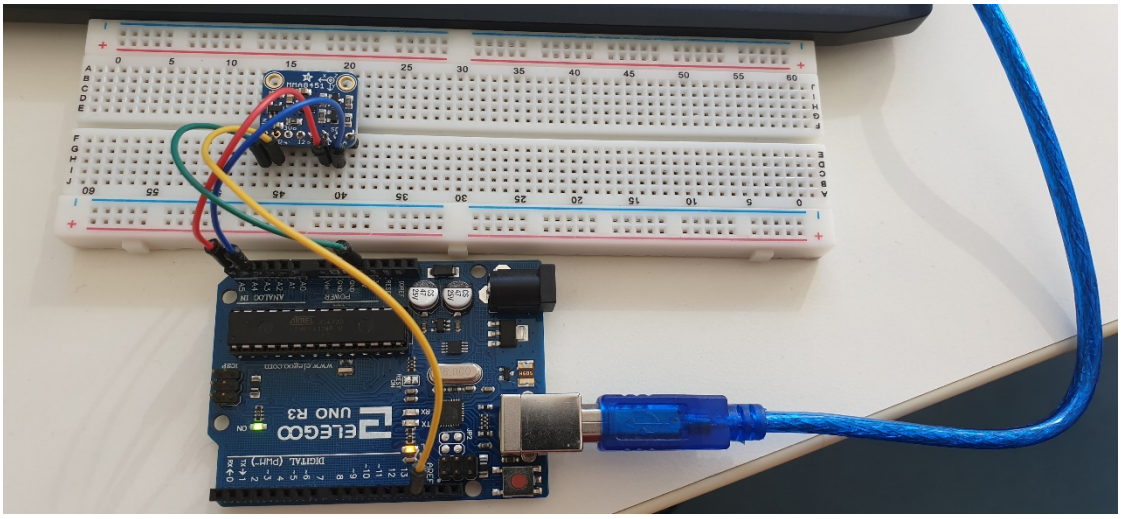
Right to left - negative value - CCW
Left to right - positive value – CW

Back to the execution:

PushMatrix() will store translate(), rotateX() and rotateY() in a transformation matrix stack and once it reaches draw_axes() it will go looking for the function, the function will draw a box, fill it will the color green and draw the x-y-z axis on it. After exiting that function, it will come back directly to popMatrix() which will pop the transformation matrices from the stack, and apply them on the box and axes. After that it'll go back to background(255), deleting the traces of the old objects, then pushMatrix() again, and so on…

Task 3.2:

This task introduces the Accelerometer. I had to download the required libraries first. Then I connected the Accelerometer to the Arduino as described in the manual (Vin. – 5V pin. GND – GND. SCL – A5 pin. SDA – A4).
I then used the provided demo from the Adafruit MMA8451 to test out the unit.



1. I tested out the circuit by moving it around while keeping track on the results shown on the serial monitor. I noticed that the signs of XYZ accelerations weren't as I expected, as X positive was to the right, not the left. And Y positive was upwards not downwards. However, Z was as expected (Towards me it positive. Away from me negative).

2. I managed to get all 8 orientations. However, I faced a problem where it switches randomly between some positions (Portrait Up Front and Portrait Back Front for instance). I found out that it's due to the fact that the unit's pins are tilted, which causes it not being completely horizontal for its normal state, and therefore, mixing the orientations up as the angle would be less than 30 degrees when it's not supposed to be.

3. My results weren't identical to results given in this task. I got 9.65 (due to the aforementioned tilt in pins), which changes to 9.8 after I try to align the unit as horizontally as possible. And that is indeed m/s^2 (The acceleration of gravity). Unlike the example given in the task (0.98 m/s^2), which isn't really in m/s^2.

```
X:      -48     Y:      866     Z:      4026
X:      -0.11   Y:      2.05    Z:      9.65    m/s^2
Portrait Up Front

X:      -66     Y:      850     Z:      4028
X:      -0.12   Y:      2.03    Z:      9.65    m/s^2
Portrait Up Front

X:      -52     Y:      848     Z:      4040
X:      -0.11   Y:      2.04    Z:      9.64    m/s^2
Portrait Up Front
```

Task 3.3:

In this example, I continued with the same Accelerometer circuit. The objective was to connect this circuit with a Processing program, which will visualize the changes in motion (of the Accelerometer) on our window, by applying the rotations of the unit onto a digital cube in the program.

1. For the Arduino code, I kept it as it is for the most part. I only had to change the output given and make each value of XYZ get printed in separate lines instead of all at the same line as it originally was. (I also deleted the redundant output just to make the program somewhat cleaner).

```
void loop() {
  // Read the 'raw' data in 14-bit counts
  mma.read();


  /* Get a new sensor event */
  sensors_event_t event;
  mma.getEvent(&event);

  /* Display the results (acceleration is measured in m/s^2) */
  Serial.print("X: "); Serial.print(event.acceleration.x/10); Serial.print("\n");
  Serial.print("Y: "); Serial.print(event.acceleration.y/10); Serial.print("\n");
  Serial.print("Z: "); Serial.print(event.acceleration.z/10); Serial.print("\n");


  /* Get the orientation of the sensor */
  uint8_t o = mma.getOrientation();


  delay(500);

}
```

2. For the processing program, I filled in the missing parts of code using the formulas given in the slides. The resulting code was the following:
   (**Note**: The norm is equal to the square root of the sum of the X, Y and Z components squared.)

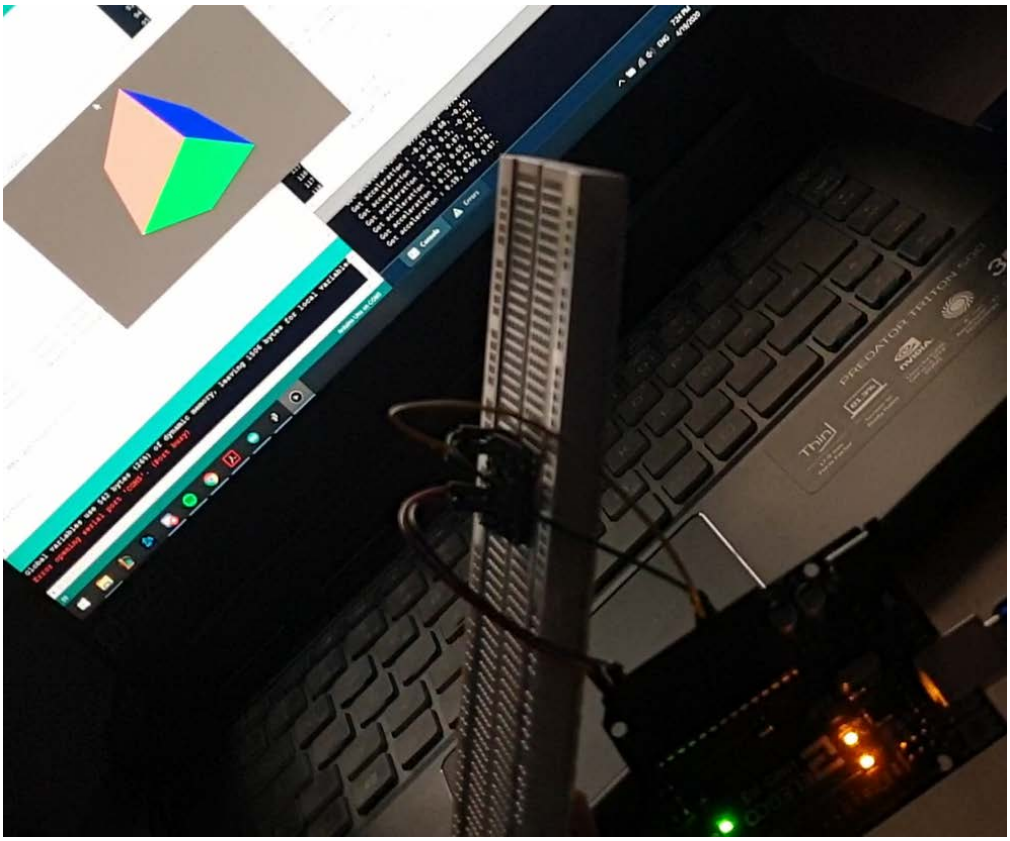
```
  // Compute the roll and pitch angles
float acc_norm = sqrt((acceleration [0])*(acceleration [0]) + (acceleration [1])*(acceleration [1]) + (acceleration [2])*(acceleration [2]));
float ax= acceleration [0]/ acc_norm ;
float ay= acceleration [1]/ acc_norm ;
float az= acceleration [2]/ acc_norm ;


// Fill -in: Compute roll and pitch
pitch = acc_norm*sin(az/acc_norm);
roll = acc_norm*tan(ax/cos(pitch));


 // Draw the rotated object here .
translate ( width /2.0 , height /2.0 , -50);
// Fill -in: rotate by roll and pitch
rotateX(pitch);
rotateY(roll);
draw_rgb_cube ();
}
```

3. I ran the Processing program, and by it being connected to the same Arduino serial, it receives the XYZ coordinates from the Accelerometer corresponding to its current orientation in space and uses those to apply Pitch and Roll transformation into the 3D cube. The result was getting the cube to rotate in similar direction to the one I rotated my breadboard in.

20200419_192355~2.mp4



4. parseStringForAccelerations function will check whether or not the received input from the Arduino port, which will be in the form of a string consisting of letter (X:… Y:… Z:..) and values, is suitable for being a valid acceleration. If yes, then it'll return a true value to be put in the newSample variable. If not, it will return false value (Zero), which will cause the program not to go on as planned, and the box not to rotate.
   This variable is needed to stop the program from crashing in case it receives in valid data. Because in this case it will just not accept it and wait for the next string of data to be received and try again. However, if we didn't use it, we'll be putting unsuitable data in the wrong functions, which will crash the whole system.

Task 3.4:

In this part of the Lab, I used Matlab Simulink to visualize the output of some basic circuits. In this task, it's about RL circuit.
RL circuit is a basic circuit consisting of an inductor, a resistor and a power source. We start the analysis with Ohm's law V = I.R for the total voltage of the circuit.
And for the inductor voltage, we use the general equation for inductors V = L.di/dt, which states that the voltage passing through the inductor is directly proportional to the change in current of the circuit (the inductor will fight the change in the current and generate an opposing current in the different direction), multiplied by the inductance of that inductor (a scalar for the relationship; a constant).

We then use Laplace transform to convert the resultant functions from a function of time into a function of s-domain (frequency).

The final equation between the input (voltage) and the output (current) of this circuit will be the following:

$$H(S) = \frac{I(S)}{V_1(S)} = \frac{1/S}{L + R/S}$$

To represent this on Simulink, we have to parts:

- **Numerator:**
  We use the Step block to represent the voltage, the Step block provides a step between two definable levels at a specified time. If the simulation time is less than the Step time parameter value, the block's output is the Initial value parameter value. For simulation time greater than or equal to the Step time, the output is the Final value parameter value. In this example we define both Step time, and the final value as 1. And the initial value as 0. In other words, the circuit will start with 0 voltage at first, and then after one second of execution it will become 1V.

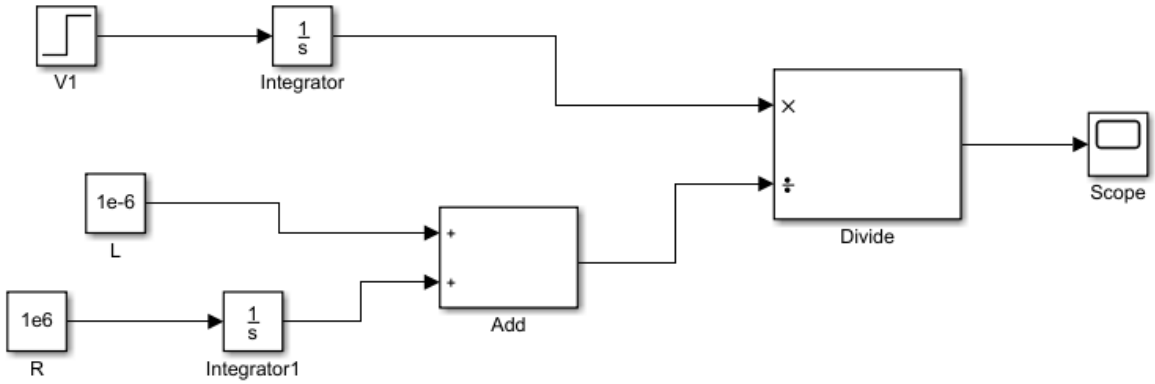
  We then Integrator block for the voltage. The Integrator block outputs the value of the integral (Laplace transform) of its input signal with respect to time. So, it will basically return the integral of 0 at first, which is 0. Then it will return the integral (Laplace transform) of 1, which is 1/S.

Signal Description	Time-Domain Waveform, f(t)	s-Domain Waveform, F(s)
Step	$u(t)$	$\frac{1}{s}$

- **Denominator:**
  We define the inductance as a constant L with the value 1e-6.
  Then we define the resistance as a constant R with the value 1e6. And then take the integral (Laplace transform) of R, which is a constant, so it goes outside of the integration, and the final result will be R multiplied by the integral of 1 => R/S

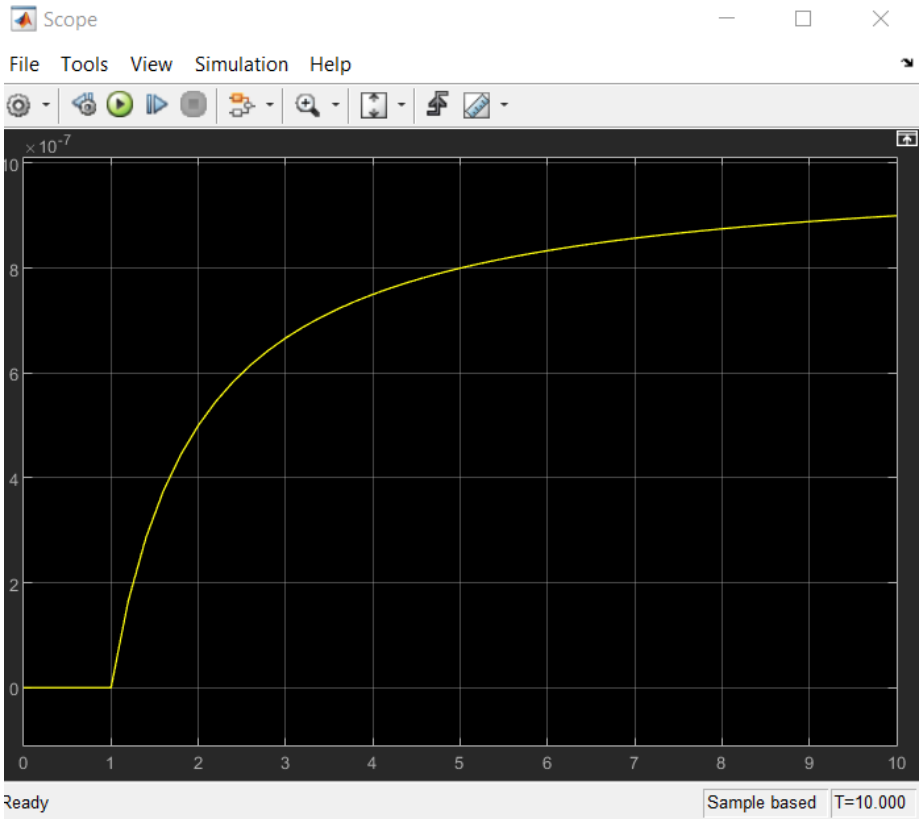
  After that we add both values together using Add block so we would get L + R/S.

We use Divide block to combine both the Numerator and Denominator. And after that we use the Scope block which displays the signals generated during the simulation.



Task 3.5:

After running the system, I notice that the system is stable, and the results are as expected. Because at the beginning (before step time (1 second)), the voltage is set to 0 (the circuit is open). After that we close the circuit and set the voltage to 1V (by using the step function) the inductor will fight against the change (the increase) of current, and try to slow it down as much as possible by generating an opposing current. Which explains the gradual increase in the current (not instant) that we can see in the following graph:



Task 3.6:

In this task we analyze RC circuit, which is a circuit that contains: A resistor, a capacitor and a power source. And the whole process of deriving the output to input function (Transform function), is really similar to the RL circuit process in the previous task, but here we start with the voltage-current relationship of the capacitor, which states that the current going through the capacitor is equal to a constant multiplied by the change in voltage.

Note: This is derived from the basic capacitor-to-voltage relationship: q = Cv. By taking the derivative of both sides, the change in charge versus change of time (current) is equal to C.dv/dt

And using Ohm's law, we know that I = V1-V2/R.

And the output we want to observe here is not the change of current, but the change in the capacitor's charge. Which at the beginning of the simulation (right after we close the circuit) will be zero volts (all charge will be in the resistor), and will slowly charge with time, until the capacitor is fully charged (resistor will have no charge at that point), then the current will stop.

In Simulink:

- **Numerator**:
  It's identical to what we did in the previous task's numerator

- **Denominator**:
  I defined a constant with the value 1, and integrated it to get 1/S.
  Then defined both constants R and C with the values: 1e6, 1e-6 respectively, and calculated the product of these two constants to get RC.

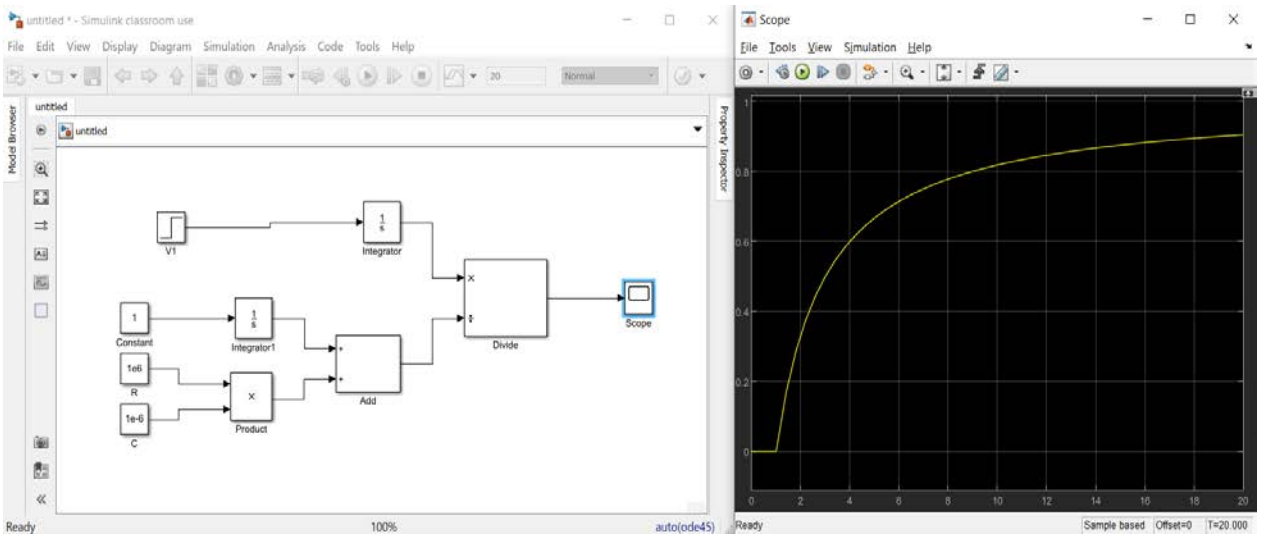
  After that I used Add block to add 1/S with RC.

All what's left after that is merging the numerator and the denominator using the Divide block, and then using Scope to visualize the results. Which were as expected:

Right after the circuit is closed, the maximum current is running through the capacitor, but there is still no charge on the capacitor (Voltage V2 is 0).

The capacitor will start storing charge gradually until its fully charged, hence the increase in V2, which slows down a bit when it's almost fully charged and has the maximum voltage (the voltage of the battery V1).

The following picture shows both the circuit and the results of the simulation:



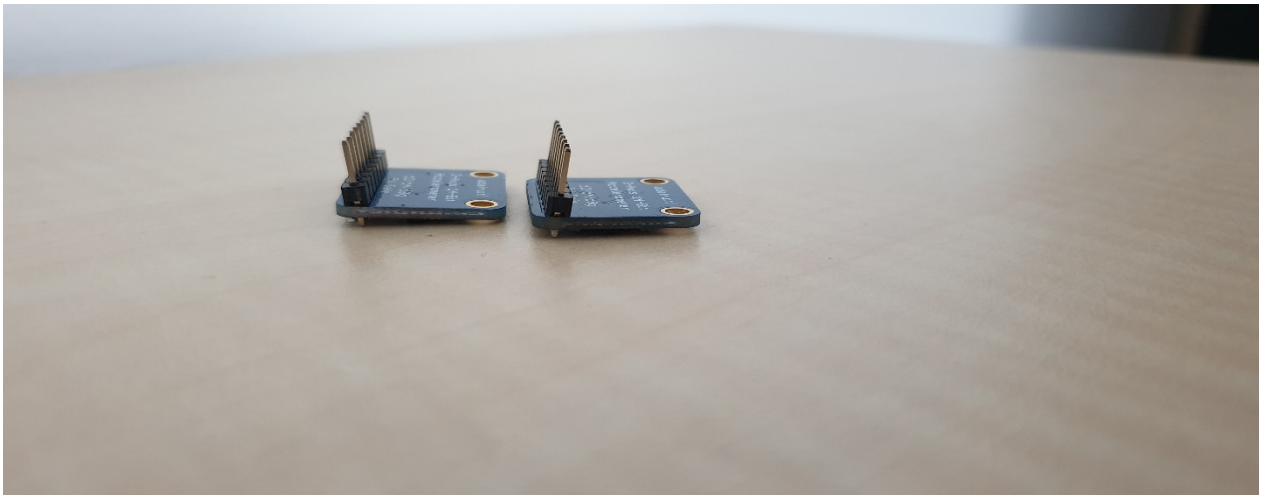
Conclusion:

This lab was more complicated than both the first two. It was so time consuming and required a lot of research in order to solve its tasks.

The Accelerometer part was rather confusing, and all the codes weren't properly explained, especially Task 3.3. Most of the transformations were confusing and weren't properly explained anywhere. I solved them by asking the TA's and doing some intensive research about the subject.

Also, the Adafruit library didn't work on my device at first. I tried deleting it and trying to reinstall it but that didn't work either. So, I had to reinstall the whole Arduino IDE, which fixed the issue.

On top of that, my Accelerometer unit had aa broken pin the first time (bad welding). I only noticed that after I kept getting "Couldn't start" in my serial monitor.
And the second time, the pins were bent, which caused some abnormal results to occur.



Finally, the Simulink part, which was a totally new subject for me that nobody tried explaining. And this also, naturally, took me a lot of work and research in order to understand the circuits first, and then each of the blocks and their functions.

References:

https://processing.org/reference/
https://processing.org/reference/rotateX_.html
https://processing.org/reference/pushMatrix_.html
https://processing.org/reference/fill_.html
https://processing.org/reference/box_.html
https://processing.org/examples/accelerationwithvectors.html
*https://www.dummies.com/education/science/science-electronics/analyze-a-first-order-rl-circuit-using-laplace-methods/
https://www.khanacademy.org/science/electrical-engineering/ee-circuit-analysis-topic/ee-natural-and-forced-response/a/wmc-inductor-in-action
https://www.khanacademy.org/science/electrical-engineering/ee-circuit-analysis-topic/ee-natural-and-forced-response/a/ee-rc-natural-response
https://www.mathworks.com/help/simulink/math-operations.html
https://www.mathworks.com/help/simulink/sources.html
https://www.mathworks.com/help/simulink/continuous.html
https://www.mathworks.com/help/simulink/sinks.html