Final Project:

IoT Motion Pattern Recognition with Accelerometer

Overview:

In this final project, you will develop a system that recognizes motion patterns based on the capabilities of the EDUMKII onboard accelerometer. You will use a level surface to avoid the effects of gravitational acceleration on the accelerometer measurements. Within this planar motion, you will design a system to detect a proper L-shaped motion. Additionally, you will send an indication of a valid gesture to Adafruit IO along with registered accelerometer data in the planar field of motion.

Motion Detection Overview:

The approach for detection of motion patterns relies on the measurement of **displacement and time** of events. A state machine method provides an ideal approach for construction of algorithms that enable motion pattern recognition. Typical conditions of a state machine include:

- Absence of motion corresponding to an idle state.
- Detection of motion exceeding a threshold for a specific axis direction.
- Detection of the conditions associated with a specific motion pattern corresponding to time and amplitude.

Our objective in this assignment is to develop a state machine that first detects motion in the Y-axis direction followed by the detection of motion in the negative X-Axis direction. The axis orientation for the EDUMKII accelerometer is shown below, where accelerometer **decreases** as you move the LaunchPad in the **positive** direction of one of these axes.

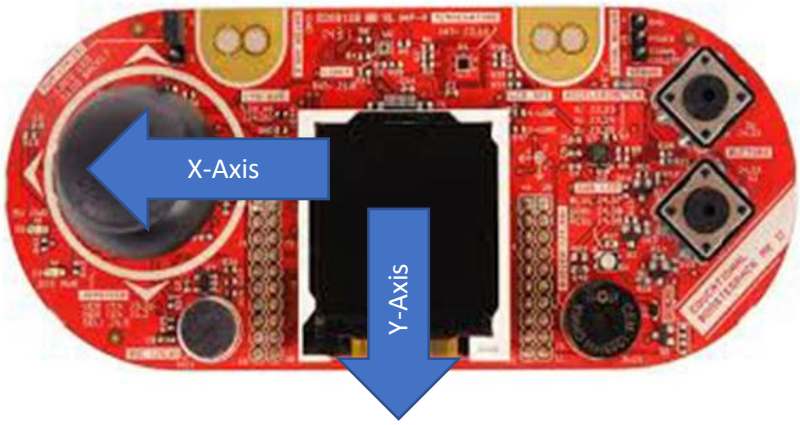

Figure 1: EDUMKII X-axis and Y-axis Orientation

Part 1: Gesture Detection

In the first section of this lab, you should design the logic to detect the distance moved by your microprocessor while on a flat surface. We will discuss a potential implementation below using concepts of displacement. In order to do this, we must refresh our memory on the relationship between acceleration and position.

In order to do this, we use fundamental physics to describes the relationship between acceleration, velocity, and displacement. Velocity is determined by performing an integral over time of acceleration. Displacement is determined by performing an integral over time of velocity.

$$v(t) = \int_0^t a(\tau)\, d\tau + v(t=0)$$
$$d(t) = \int_0^t v(\tau)\, d\tau + d(t=0)$$

Since our acceleration data is a discrete time sampled signal, where the sampling frequency is the clocking frequency of our ADC (see bullet point below for details). A discrete trapezoidal integration is computed using the relationship below. Δt references the sampling period (1 / sampling frequency).

$$v(1) = v(0) + \left[a(0) + \frac{a(1)-a(0)}{2}\right] * \Delta t = v(0) + \frac{a(1)+a(0)}{2} * \Delta t$$
$$d(1) = d(0) + \left[v(0) + \frac{v(1)-v(0)}{2}\right] * \Delta t = d(0) + \frac{v(1)+v(0)}{2} * \Delta t$$

Notice that in order to calculate the expressions above, you need to store the acceleration data of the current and previous cycle. Because floating point arithmetic is slow, you should only use integers in your code. Additionally, subtracting previous velocity and displacement values will essentially yield 0. To fix this assume the previous reference reading is always 0. The equations are modified accordingly below, where Δf is the sampling frequency.

$$v(1) = a(0) + \frac{a(1)-a(0)}{2 * \Delta f} = \frac{a(1)+a(0)}{2 * \Delta f}$$
$$d(1) = v(0) + \frac{v(1)-v(0)}{2 * \Delta f} = \frac{v(1)+v(0)}{2 * \Delta f}$$

Before you continue in this assignment, successfully sample accelerometer data and display instantaneous velocity and displacement on the EDUMKII LCD screen. The necessary steps for achieving this are described below.

- You must make an approximation for the value of Δt. The best way to do this is to display the value of a counter that increments each time the ADC14 loop executes. Using a timer on your phone, you can find the approximate time period for each loop of the ADC.

- For best results, set the ADC resolution to 10 bits. This will reduce some of the quantization noise of your ADC readings.

```
ADC14_setResolution(ADC_10BIT);
```

Figure 2: Function to set resolution of the ADC

- You must implement a 10-value wide smoothing filter to reduce the noise from the ADC14 accelerometer raw values. This will help prevent your velocity and displacement calculations from drifting. **Implement this in the ADC14 Interrupt Handler.**

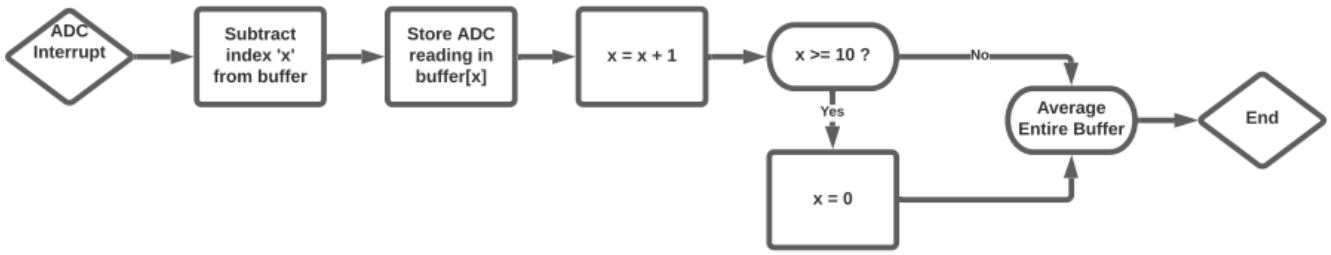

Figure 3: Smoothing Filter Logic

- Once you filter the accelerometer data calculate velocity and displacement using the equations mentioned above. Display acceleration, velocity, and displacement on the LCD. Make sure the values receive make sense intuitively.

- Once you get consistent results from your ADC, move onto the next part of this assignment.

Part 2: Gesture Detection State Machine

Below is a description of the logical approach for detecting the pattern of an alphabetical letter "L." Of course, this approach can be abstracted to accommodate for any set of motion.

- A potentially valid gesture is first detected by a large displacement in the positive Y-axis direction with a correspondingly small displacement in any X-axis direction.

- Next the pattern includes a time buffer. Specifically, a valid motion must end after the Y-motion and remain motionless for some time. Only after a period can the X-motion occur.

- Now, the microprocessor must experience a large displacement in the X-axis direction with a correspondingly small displacement in the Y-axis direction.

- The minimum displacement value for a valid gesture can vary depending on how you implement your calculations. Regardless, it is important to set a reasonable threshold that gives consistent results.

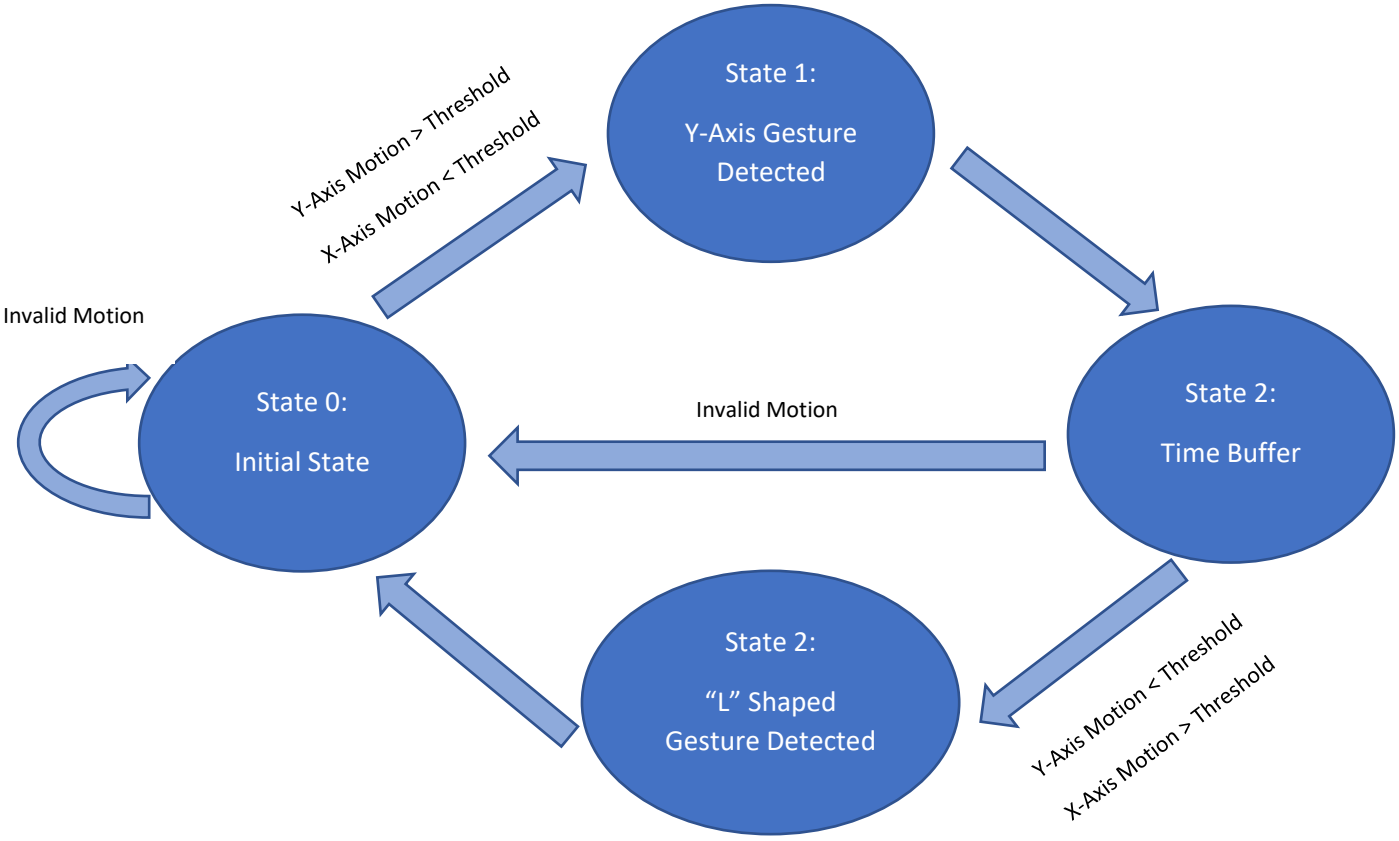

Figure 4: Gesture Detection State Machine

When you implement a state machine it is beneficial to encode each state with a certain LED color or display the current state on the LCD. This will give you some visual feedback regarding what state you are currently in and how your next state logic is behaving. While sometimes implementing a state machine can be grueling, remember that the implementation of a state machine is only a string of well-designed IF statements.

Part 3: Cloud Connectivity

In the final section of this assignment, you will create an Adafruit IO dashboard to monitor the functionality of your design. You will need to create the following feeds to send data to the cloud.

- X-Axis Feed: Used to publish real-time displacement data from the x-axis.
- Y-Axis Feed: Used to publish real-time displacement data from the y-axis.
- Gesture Detection Feed: Used to indicate that an "L" shaped gesture is recognized.
- In Topic Feed: Used to allow communication from the cloud to your LaunchPad.

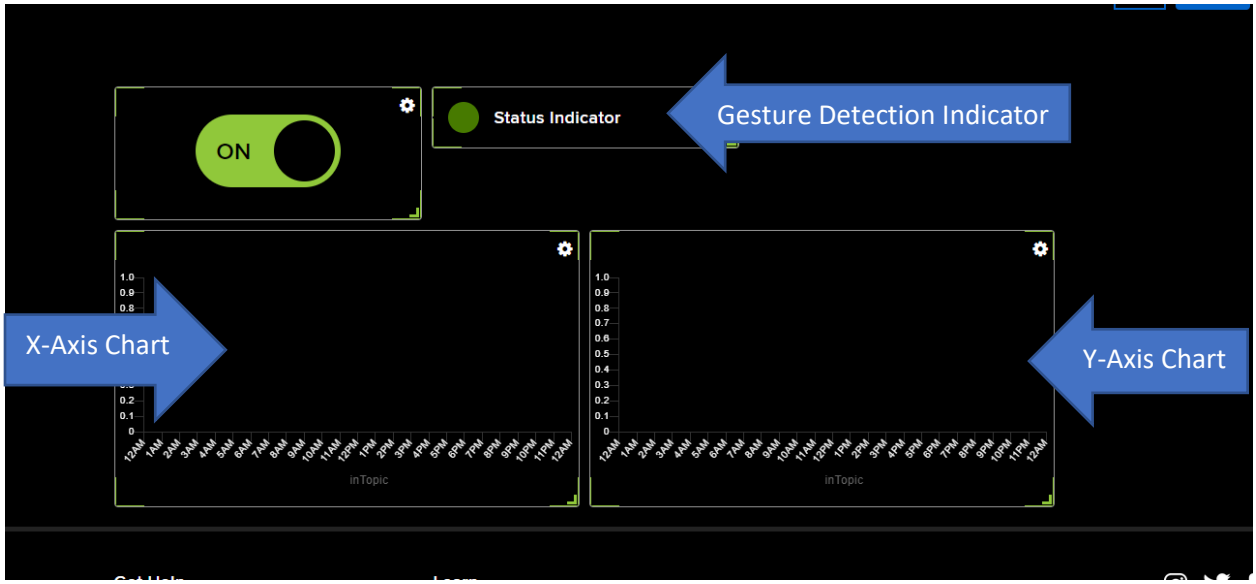

Figure 5: Adafruit IO Dashboard

In order to visualize these feeds, you will also need to create a dashboard to incorporate all the data from these feeds. The layout for this dashboard is displayed above. You will need to include two linecharts to plot the data for each axis feed. Additionally, the status indicator should remain green for at least 1 second when the gesture detection feed is true. **Since Adafruit IO limits the publishing rate to 1 sample every 2 seconds, you should only publish to one axis feed continuously at a time.** The purpose of the toggle switch is to change which axis feed your LaunchPad publishes to.

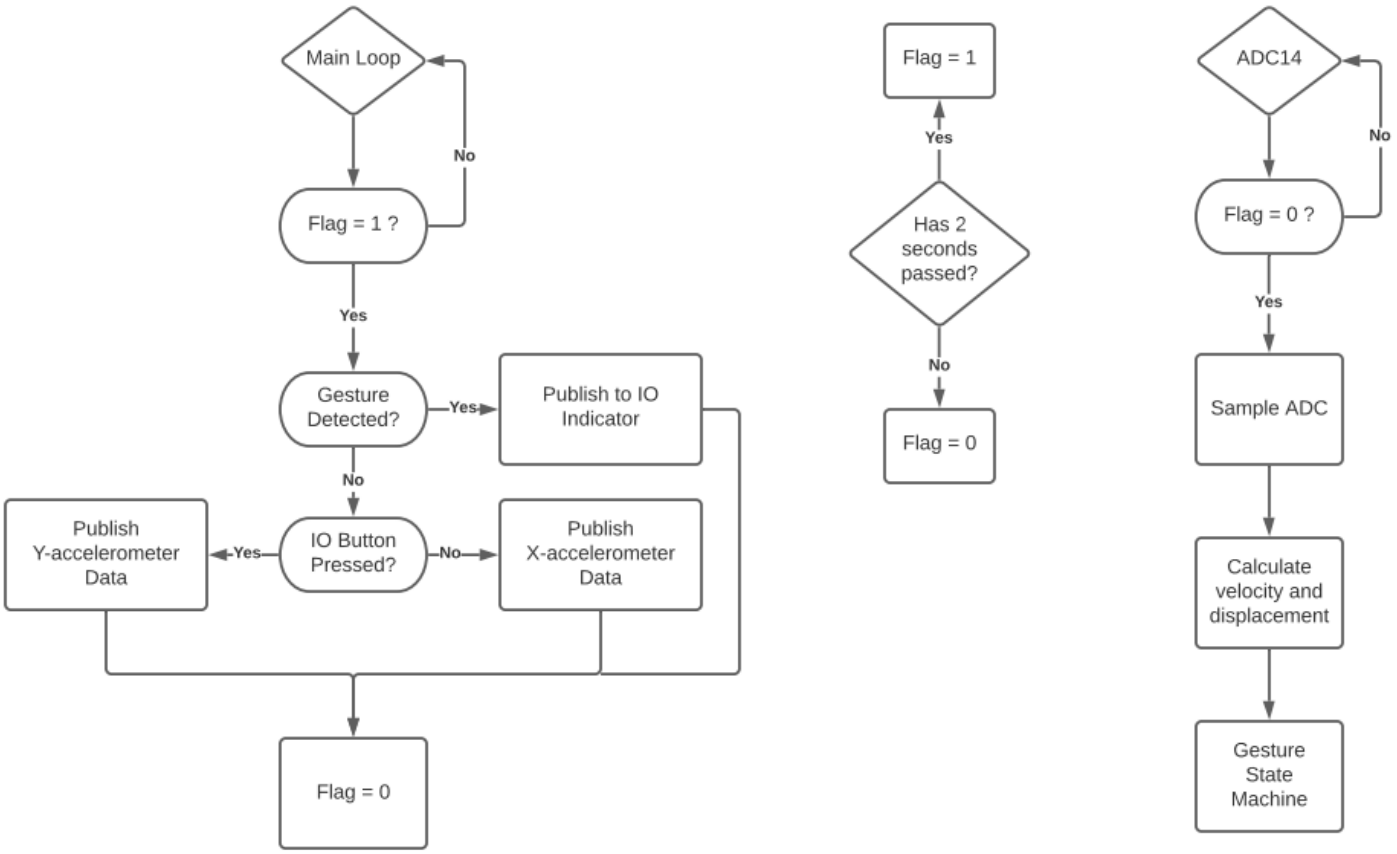

Figure 6: MQTT Gesture Detection Logic Overview

- Above is a description of the logic for publishing to your cloud feeds. You should have some flag variable that stalls all the MQTT publishing and polling tasks. If you do not properly implement this flag, the MQTT logic and ADC interrupt will interfere with each other, and the code will stall.

- The easiest way to implement a 2 second timer is to count the cycles passed on the ADC14 interrupt. This value can be fine-tuned through trial and error (the computational tasks within the ADC14 will reduce its execution rate)

- Once again, take care to only publish data to your MQTT server at a maximum rate of 1 data point every two seconds.