Week 4 Report – Tinkering Lab – PG20 – Project: Air Mouse

Members and corresponding contributions

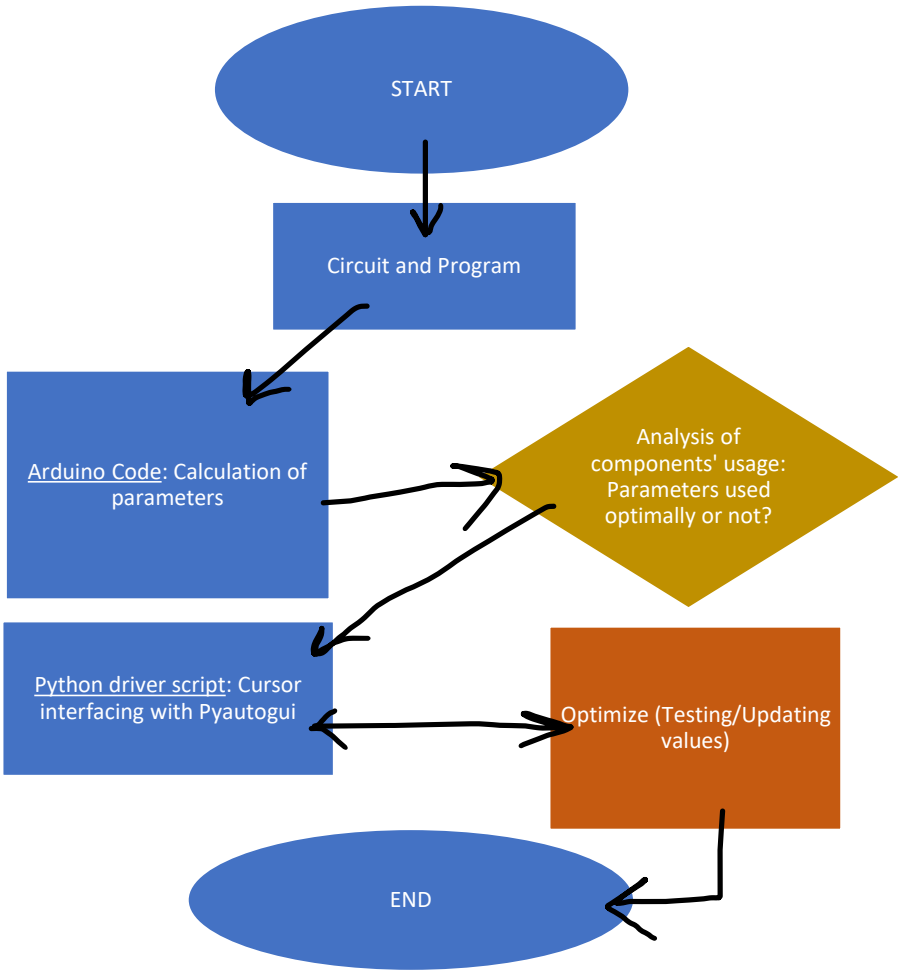
1. Karthik Kumar Pradeep (2022EPB1234)
    - *Contribution*: Programmed the accelerometer/Arduino and Python driver script. Undertook testing and troubleshooting to handle hardware and software issues
2. Khushi Tapariya (2022EPB1235)
3. Manasvi Swarnkar (2022EPB1237)
    - *Contribution of Members 2 and 3*: Surveyed the capabilities of the components used in the project, and prospective methods of usage
4. Mayank Sharma (2022EPB1238)
5. Nilesh Kumar (2022EPB1239)
6. Nishant Kumar (2022EPB1240)
    - *Contribution of Members 4, 5, and 6:* Surveyed the libraries used in the project, and prospective methods of usage

Progress until this week

Arduino Uno was used along with the accelerometer ADXL335 and 3 push buttons which act as trigger, left and right click buttons. The Air Mouse is now functional and can be used to perform almost all the tasks that a typical mouse can do. However, there was scope for improvement in the motion of the air mouse, particularly in terms of its smoothness.

Tasks taken up in Week 4

- Workflow diagram:



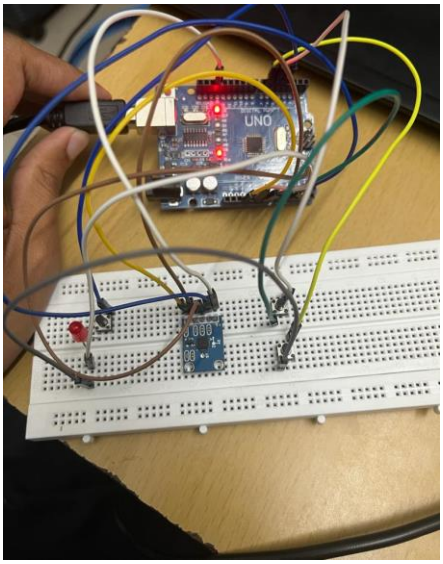
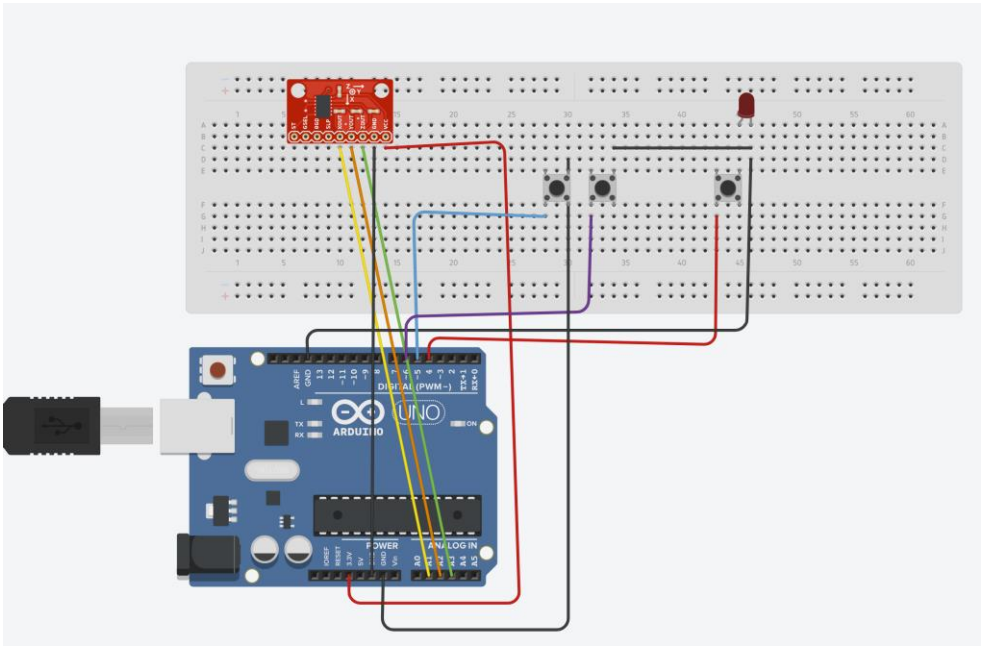
- Objective: To build towards a fully functional air mouse, and to make it work in a way befitting a decent alternative to a typical mouse.

- Problem Statement: To optimize the movement of the screen cursor. Test how the motion of the cursor varies with various parameters such as roll, pitch, time delay, etc. and finalize values of the parameters

- Components Used: Arduino UNO (with its USB cable), ADXL335 Accelerometer, 3 Push buttons, LED, Jumper Wires, Breadboard.

- Hardware: **No changes since last week (Week 3)**

    - Circuit: ***Note that the ADXL335 used has only 5 pins unlike the below Tinkercad diagram***

- 
- 

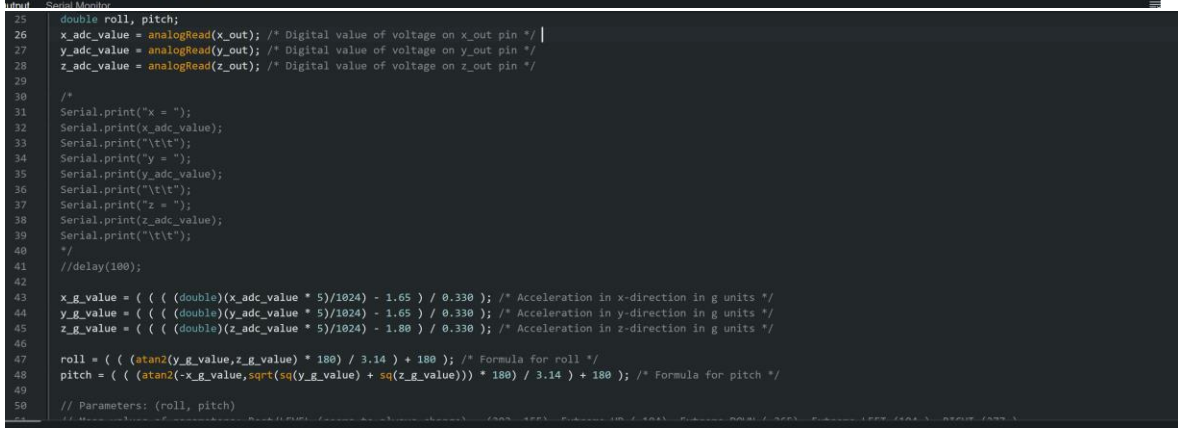
- Software: Just as has been implemented so far, the Arduino code programs the accelerometer to read out x, y, and z values which are then used to find roll and pitch values. The appropriately mapped roll and pitch values (for cursor motion) are printed onto the serial monitor. The code detects whether push buttons (for triggering the working of air mouse, left and right click buttons) are High (1) or Low (0).
    - ***This week's changes***: We have paid more attention to the roles of the various parameters in the code (both Arduino code and Python driver script). In particular, we have observed how much time is taken for each single cursor step (cursor motion is in discrete steps) through the Python driver script. Accordingly, we have adjusted the time delay before the start of the next loop in Arduino code. We also observed how the roll and pitch values are mapped to x, and y values using map(). We tuned the maximum and minimum values

of roll and pitch when using map(), by observing what kind of values roll and pitch take on, when we move the air mouse through space.

- Changes in Arduino code (apart from comments related to parameter testing):
  - Lines 55 and 56: The x, y values (for cursor motion) which are determined by appropriate scaling (mapping) of the roll and pitch values. This is done by observing the range of values taken by roll and pitch under different orientations of the air mouse. Roll is changed from (45, 300, -180, 180) to (74, 280, -80, 80), and Pitch is changed from (90, 250, -180, 180) to (100, 265, -80, 80).

  - Line 100: The delay is changed from 120 ms to 100 ms. This is the most optimal choice as far as responsiveness of the air mouse is concerned, as the movement of the cursor takes close to 0.1 seconds.

- Additional lines in Python driver script:
  - Line 3: The library 'time' is imported for measuring how much time the move() call takes to make the cursor
  - Lines 15, 17, 24 and 25: time() function from 'time' library is used to calculate the time interval between move() command and the movement of the cursor. This is printed onto the output terminal.
- <u>Code Snippets</u>:
- Arduino code:
- 

- 

-

-
```
// Parameters: (roll, pitch)
// Mean values of parameters: Rest/LEVEL (seems to always change) - (202, 155), Extreme UP (,104), Extreme DOWN (,265), Extreme LEFT (104,), RIGHT (277,)
// OLD Observed (favorable): Roll varies roughly from 45 to 300
// OLD Pitch from say 90 to 250

int x = map(roll, 74.00, 280.0, -80.0, 80.0);
int y = map(pitch, 100.0, 265.0, -80.0, 80.0);

/*
  Serial.print("Roll = ");
  Serial.print(roll);
  Serial.print("\t");
  Serial.print("Pitch = ");
  Serial.print(pitch);
  Serial.print("\n");
*/
  if (buttonState == LOW)
  {
      //Serial.println("Button is pressed");
      //Serial.print(roll);
      //Serial.print(",");
      //Serial.print(pitch);
      //Serial.print(",");
      Serial.print(x);
      Serial.print(",");
      Serial.print(y);
```
-
```
      Serial.print(y);
      //Serial.println("OK");

      Serial.print(",");
      if (leftState == LOW)
        Serial.print(1);
      else
        Serial.print(0);

      Serial.print(",");
      if (rightState == LOW)
        Serial.println(1);
      else
        Serial.println(0);
  }
  else
  {
      if (leftState == LOW && rightState == LOW)
        Serial.println("0, 0, 1, 1");
      if (leftState == HIGH && rightState == LOW)
        Serial.println("0, 0, 0, 1");
      if (leftState == LOW && rightState == HIGH)
        Serial.println("0, 0, 1, 0");
      //Serial.println("Button is not pressed");
  }
  delay(100);
}
```
-
- Python driver script:
```
import serial
import pyautogui
import time

# y is the mapped pitch val, x is the mapped roll val
# Do not go to extreme vertical orientations

ser = serial.Serial('COM6', 9600)

while True:
    try:
        # start = time.time()
        data = ser.readline().decode().strip().split(',')
        x, y, l, r = map(int, data)
        move_start = time.time()
        pyautogui.move(x, y, 0.1)
        move_end = time.time()
        if (l == 1):
            pyautogui.click()
        elif (r == 1):
            pyautogui.rightClick()
        print(f"{x}, {y}")
        # end = time.time()
        time_interval = move_end - move_start
        print(f"move time = {time_interval}")
    except ValueError:
        pass
```
-
- Output:

- 

- Result: The Python driver script prints out the mapped roll and pitch values (x, y) along with the time taken (move time) for each 'discrete step' of the screen cursor. The changes to our code have helped us optimize the user experience of the air mouse and have made the cursor motion as smooth as the limited hardware allows, along with the left and right click buttons. Thus, we have made an optimal and functional air mouse with the limited capabilities of the existing components. We can now move towards ways of making button usage more trivial and think of adding more features to the air mouse.

Challenges faced and future work

- The air mouse is now optimal and functional. The issue of smooth movement has been dealt with as much as the hardware allows. For improvements, there are still some challenges – concerning the few missing capabilities of the air mouse.
- Challenges:
  - The air mouse does not have an efficient way to *drag* the cursor on the screen. The idea of using the left click and trigger button simultaneously is unsuccessful due to the working of move() in discrete steps.
- Solutions (for future work):
  - Inclusion of another push button for 'dragging' (clicking, holding, and moving the cursor)
  - Utilizing the functions in the Pyautogui library more rigorously. In particular, the usage of the drag() function in Pyautogui.

- Future work: To utilize the relevant capabilities of the Pyautogui library fully. Implement push button for 'dragging'. Another major idea is to implement the usage of the push button as a 'toggle' button – which toggles a certain state of the air mouse, such as "movement", "dragging" , etc. This will greatly increase the ease of using the air mouse.

Summary

With the work done until this week (Week 4), we have successfully managed to implement an optimal and functional air mouse. The work is based on the idea of roll and pitch values of the accelerometer motion in space. We have tested the air mouse and concluded that it can perform various tasks that a typical mouse can do satisfactorily (Motion and clicking)

and have optimized it as far as the components involved can. This has made our air mouse a better candidate for replacing a typical mouse. We would like to now focus on improving the user experience by trying to implement usage of ‘toggle’ states and also make the air mouse capable of ‘dragging’ the cursor on the screen.