Final Project Report: Pedometer (Steps, Distance, Calories)

Aziq Furqan **Swostik Pati** **Saamia Shafqat**

1. Introduction

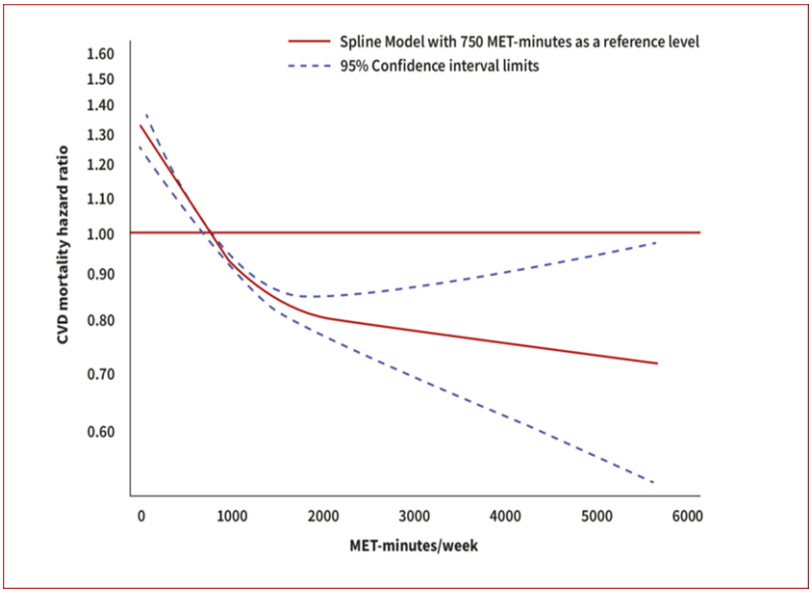
Aim of the Project:

The objective of our program is to develop a Pedometer application using the M5 Stack device that performs the following functions:

- Record Footsteps taken by the user
- Calculate Distance covered by the user
- Compute the Calories Burnt by the user

Significance of Our Project:

- In a health conscious world like ours, a step-tracker pedometer proves to be really helpful to track daily health goals.
- It motivates the user to maintain a healthy lifestyle by giving them a sense of accountability.
- Research has shown that cardiovascular diseases are greatly reduced with increase in physical activity.



The graph above shows the relation between number of MET-minutes (Metabolic Equivalents of Work) per week and Cardiovascular Disease Mortality hazard ratio. The graph shows that with an increase in the number of MET minutes per week (achieved by increasing physical activity like walking), cardiovascular diseases are less likely to occur in people. Therefore, the pedometer device designed by us using the M5 can help the user achieve a disease-free healthy life.

2. Project Development

Device Used: M5 Stack Core 2 - The device used for the purpose of our project development is the M5 Stack Core 2 device which is compatible with Arduino.

Software Used: Arduino - The entire code was written on the Arduino IDE and later uploaded to the device using the COM5 port.

Libraries Installed - M5 Core 2

Github Repository Link - *https://github.com/swostikpati/CPE-Term-Project*

Implementation:

The implementation of the entire project is detailed below. It has explanations about every function along with pictures and graphs(wherever necessary), the algorithm design for each, and snippets of well commented code.

Headers And Definitions:

Description:

The code begins by including the required libraries, defining several symbolic constants, and initializing several global variables to be used throughout the program.

The symbolic constants include:

- wtmin - stores the minimum time taken by an average user for completing a step in seconds
- wtmax - stores the maximum time taken by an average user for completing a step in seconds
- dmin - stores the minimum distance travelled by an average user for completing a step in meters
- dmax - stores the maximum distance travelled by an average user for completing a step in meters
- MET - stores the metabolic equivalents for the physical activity - in our case it is walking

The variables include:

- aX, aY, aZ - stores the accelerometer values of the three axis
- alpha - defines the extent of filtration
- aXFilt, aYFIlt, aZFilt - stores the accelerometer values after filtration of the three axis
- threshold - defines the threshold for acceleration to detect steps
- netmag - stores the net magnitude of acceleration
- flag, f, and f1 - flags that keep are set to true and false in several places in the program to perform desired executions
- count - stores the number of steps taken
- weight - stores the weight of the user
- dist - stores the distance traveled by the user
- wtime - stores the walk time of the user
- cal - stores the calories burnt by the user
- sH and sW - used to store the screen height and screen width of the device

Algorithm Design:

Include the M5Core 2 Library
Define symbolic constant wtmin by assigning it 0.5
Define symbolic constant wtmax by assigning it 1.0
Define symbolic constant dmin by assigning it 0.4
Define symbolic constant dmax by assigning it 0.9
Define symbolic constant MET by assigning it 2.8

Declaring variables aX , aY, and aZ all of float datatype and initialize them with 0.0
Declare variable alpha of integer datatype and set it to 0.1
Declare variables axFilt, aYFilt, and aZFilt of float datatype and initialize them with 0.0
Declare variable threshold of float datatype and initialize it to 0.05
Declare variable netmag of float datatype

Declare variables flag, f, and f1 of boolean datatype and set them to true
Declare variable count and weight of integer datatype and set them to 0
Declare variables dist, wtime, and cal of float datatype and set them to 0.0

Declare variables sH and sW of float datatype

Code:

```
//Importing dependencies
#include <M5Core2.h>
//Defining symbolic variables
#define wtmin 0.5 //minimum walk time per step
#define wtmax 1.0 //maximum walk time per step
#define dmin 0.4 //minimum distance per step
#define dmax 0.9 //maximum distance per step
#define MET 2.8 //MET value of walking

//Initializing variables

//Accelerometer data variables
float aX = 0.0f;
float aY = 0.0f;
float aZ = 0.0f;

//Acceleration filter variables
int alpha = 0.1; //Extent of filtration
float aXFilt(0.0f), aYFilt(0.0f), aZFilt(0.0f);
float threshold = 0.05f; //threshold for step detection
float netmag;

//flags
```

```
bool flag = true; //flag to detect if the person is already in the middle
of a step
bool f = true;    //flag to detect if screen is already pressed
bool f1 = true;   //flag to decide whether to display the initial screen

int count(0); //count of steps
float dist = 0.0f; //distance
float wtime = 0.0f; //walk time
int weight = 0; //weight
float cal = 0.0f; //calories

double sH, sW; //screenheight and screenwidth
```

Setup Function

Description:

The setup function is the part of the main program that runs once and sets up the device. It starts the M5 stack unit, initializes the accelerometer sensors of the Inertial Measurement Unit(IMU). It sets up the device background, text color, and the text size

Algorithm Design:

Setup Function

Begin the M5 stack
Initialize the IMU sensor of the M5 stack

Fill the screen of M5 stack with black background
Set the text color to blue with a black background
Set the text size to 2 units

Store the screen height value of M5 stack to sH
Store the screen width value of M5 stack to sW

Set the seed of the random number generator to system time.

Code:

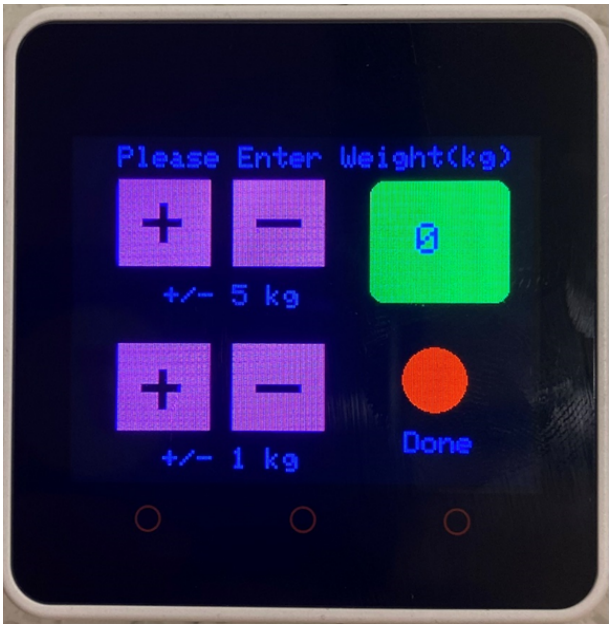
```
//setup function - to be run once
```

```
void setup() {
  //startup M5 core 2 device
  M5.begin();
  M5.IMU.Init();

  M5.Lcd.fillScreen(BLACK);
  M5.Lcd.setTextColor(BLUE, BLACK);
  M5.Lcd.setTextSize(2);

  //stores screenheight and screenwidth
  sH = M5.Lcd.height();
  sW = M5.Lcd.width();

  srand(time(0));
}
```

Initial User Interface

Description:

This part of the code defines the initial screen that the user screen. Here, the user is asked to input their weight using 4 buttons (+/-5 and +/-1). When the user has finally finished putting the user's weight, he clicks the done button. The code for taking input from the user is two-fold - creating the static interface for the user to interact and taking input through touch. The following code highlights the creation and organization of the interface elements like the rectangle buttons. An image of the look of the initial user interface is given below:



Initial UI

Algorithm Design:

Define the iUI function of void return type

*Set the cursor of the screen at the coordinates (0.1*sW, 0.03*sH)*
Set the text size to 2
Print "Please Enter Weight(kg)" on the screen

*Make a rectangle at coordinates (0.1*sW, 0.125*sH) of length (0.2*sW) and width(0.25*sH) of Pink color*
*Make a rectangle at coordinates (0.35*sW, 0.125*sH) of length (0.2*sW) and width(0.25*sH) of Pink color*
*Make a rectangle at coordinates (0.1*sW, 0.6*sH) of length (0.2*sW) and width(0.25*sH) of Pink color*
*Make a rectangle at coordinates (0.35*sW, 0.6*sH) of length (0.2*sW) and width(0.25*sH) of Pink color*

*Make a rectangle at coordinates (0.40*sW, 0.24*sH) of length (0.1*sW) and width(0.025*sH) of Black color*
*Make a rectangle at coordinates (0.40*sW, 0.72*sH) of length (0.1*sW) and width(0.025*sH) of Black color*

*Make a rectangle at coordinates (0.19*sW, 0.195*sH) of length (0.02*sW) and width(0.12*sH) of Black color*
*Make a rectangle at coordinates (0.15*sW, 0.24*sH) of length (0.09*sW) and width(0.025*sH) of Black color*

*Make a rectangle at coordinates (0.19*sW, 0.675*sH) of length (0.02*sW) and width(0.12*sH) of Black color*
*Make a rectangle at coordinates (0.15*sW, 0.72*sH) of length (0.09*sW) and width(0.025*sH) of Black color*

*Make a rounded rectangle at coordinates (0.65*sW, 0.13*sH) of length (0.30*sW), width(0.55*sH), and radius 10 of Green color.*

*Make a circle at coordinates (0.79*sW, 0.7*sH) and radius 0.07*sW of Red color*

Set the text size to 2
*Set the cursor of the screen at the coordinates (0.2 * sW, 0.43 * sH)*
Print "+/- 5kg" on the screen
*Set the cursor of the screen at the coordinates (0.2 * sW, 0.9 * sH)*
Print "+/- 1kg" on the screen
*Set the cursor of the screen at the coordinates (0.728 * sW, 0.85 * sH)*
Print "Done" on the screen
*Set the cursor of the screen at the coordinates (0.7 * sW, 0.4 * sH)*
Print the value of weight on the screen

Code:

```
//function to build the initial UI
void iUI(){
  M5.Lcd.setCursor(0.1 * sW, 0.03 * sH);
  M5.Lcd.setTextSize(2);
  M5.Lcd.printf("Please Enter Weight(kg)");
  //+ and - boxes
  M5.Lcd.fillRect(0.1*sW, 0.125*sH, 0.20*sW, 0.25 * sH, PINK);  //+5 box
  M5.Lcd.fillRect(0.35*sW, 0.125*sH, 0.20*sW, 0.25 * sH, PINK); //-5 box
  M5.Lcd.fillRect(0.1*sW, 0.6*sH, 0.20*sW, 0.25 * sH, PINK);    //+1 box
  M5.Lcd.fillRect(0.35*sW, 0.6*sH, 0.20*sW, 0.25 * sH, PINK);   //-1 box

  //-symbol
  M5.Lcd.fillRect(0.40*sW, 0.24*sH, 0.1*sW, 0.025*sH, BLACK);
  M5.Lcd.fillRect(0.40*sW, 0.72*sH, 0.1*sW, 0.025*sH, BLACK);

  //+symbol
```

```
M5.Lcd.fillRect(0.19*sW, 0.195*sH, 0.02*sW, 0.12*sH, BLACK);
M5.Lcd.fillRect(0.15*sW, 0.24*sH, 0.09*sW, 0.025*sH, BLACK);

M5.Lcd.fillRect(0.19*sW, 0.675*sH, 0.02*sW, 0.12*sH, BLACK);
M5.Lcd.fillRect(0.15*sW, 0.72*sH, 0.09*sW, 0.025*sH, BLACK);

//Weight Display Box
M5.Lcd.fillRoundRect(0.65*sW, 0.13*sH, 0.30*sW, 0.35 * sH,10, GREEN);

//Done box
M5.Lcd.fillCircle(0.79*sW, 0.7*sH, 0.07*sW, RED);

//+/-5 symbol
M5.Lcd.setTextSize(2);
M5.Lcd.setCursor(0.2 * sW, 0.43 * sH);
M5.Lcd.printf("+/- 5 kg");

//+/-1 symbol
M5.Lcd.setCursor(0.2 * sW, 0.9 * sH);
M5.Lcd.printf("+/- 1 kg");

//Done text
M5.Lcd.setCursor(0.728 * sW, 0.85 * sH);
M5.Lcd.printf("Done");
```

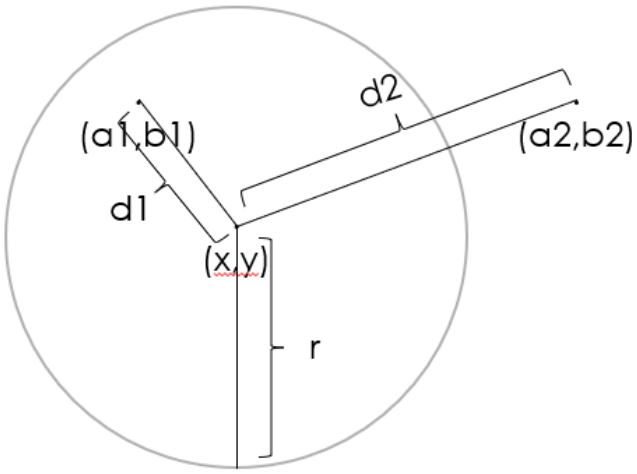
User Input Through Touch

Description:

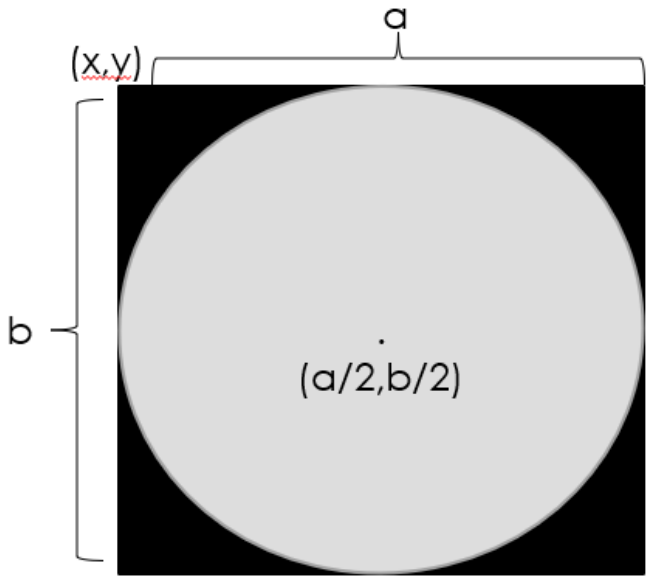
The most important element of the initial UI is to incorporate the mechanism to receive input from the user in order to get the weight of the user. The static user interface guides the user to interact with the screen based on which we have designed the mechanism to receive input through touch. The touch feature is implemented by using the getPressPoint() function of the Touch class in M5 stack. The details are given below:

- The pressed function takes in the details of a circle (called the touch area) - the coordinates of the center and radius) where the touch needs to be enabled.

- The getPressPoint function is called within the function which returns an object which contains the coordinates of the screen where the device detected a touch.
- The distance between the detected press point and the center of the touch area is computed.
- This distance is compared to the radius of the touch area. It is evident that if the radius is greater than or equal to the distance, then the pressed point lies in the touch area and as a result a touch is detected in the touch area.
- If the radius is less than the distance, the touch detected was outside of the touch area and hence the function returns false.



- In the above figure, the circle represents a touch area with center coordinates (x,y) and radius r. d1 is the distance between the point with coordinates (a1,b1) and the center, and d2 is the distance between the point with coordinates (a2,b2).
- It is evident that since, r is less than d2, (a2,b2) point lies outside the touch area; whereas since r is greater than d1, (a1,b1) point lies inside the touch area.
- In the case of our interface, since most of the elements are rectangles instead of squares, we create circular touch areas within these rectangle elements using the information of the rectangle objects.



- The above figure shows a rectangle starting with coordinates (x,y) with length a units and width b units. The coordinates of the center of the corresponding circular touch area will be (a/2,b/2) and its radius is a/2 or b/2.

Algorithm Design:

Define the pressed function of bool return type with x, y, and r as arguments of double datatype
Create a object coordinate of the TouchPoint_t class which receives the value of getPressPoint() function of the Touch class of M5 Stack
Declare a variable dist of double datatype and assign it the value of the square root of [(x-(x point of coordinate object))^2 + (y -(y point of coordinate object)^2)]
If the x point of the coordinate object or the y point of the coordinate object is less than 0
Set flag to true
Return false
Else
If dist is less than r and f is true
Set f to false
Return true

Else
Return false

Code:

```
//function to detect press - returns if pressed or not
bool pressed(double x, double y, double r){
```

```
  //gives the coordinate of the pressed point
  TouchPoint_t coordinate = M5.Touch.getPressPoint();
  //gives us the distance between the pressed point and the box(circle)
          double     dist     =     sqrt(pow((x-(coordinate.x)),2)     +
pow((y-(coordinate.y)),2));

  //checks if there is no touch detected
  if(coordinate.x <0 or coordinate.y <0){
    f = true; //turns true if screen is not pressed
    return false;
  }
  //if touch is detected
  else{
    //if touch inside the box(circle)
    if(dist < r && f){
       f = false; //turns false as soon as screen is pressed - so as to
avoid continuous presses
      return true;
    }
    //if touch is outside the box
    else{
      return false;
    }
  }
}
```

Display Weight

Description:

This function is used to display the weight of the user on the screen. It is called inside the set_weight function to keep printing the updated weight of the user.

Algorithm Design:

Define the wprint function of void return type

Set the text size to 3 units
Set the text color to blue with a green background
*Set the cursor of the screen to the coordinates (0.75 * sW, 0.25 * sH)*
Print the weight on the screen

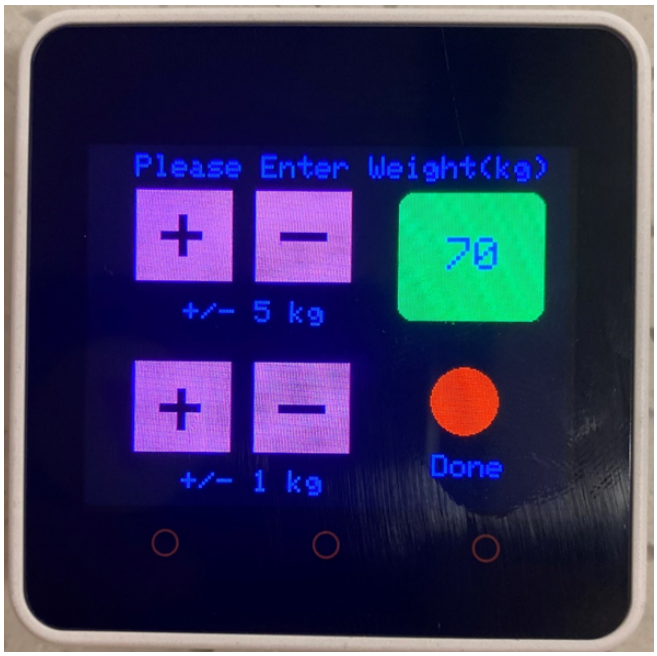
Code:

```
void wprint(){
     M5.Lcd.setTextSize(3);
     M5.Lcd.setTextColor(BLUE, GREEN);
     M5.Lcd.setCursor(0.75 * sW, 0.25 * sH);
     M5.Lcd.print(weight);
}
```

Update Weight

Description:

The pressed function is called for each of the four buttons (+ and -) to increment and decrement weight. The update weight function increments or decrements the weight based on the button clicked. It then calls the wprint function to display the updated weight. The image below shows the initial user interface with updated weights.



Algorithm Design:

Define the set_weight function of void return type

*If pressed function returns true with the arguments 0.1*sW+0.5*0.20*sW, 0.125*sH+0.5*0.25 * sH, 0.5*0.25 * sH*

Increment weight bt 5

Call the wprint function

*If pressed function returns true with the arguments 0.35*sW+0.5*0.20*sW, 0.125*sH+0.5*0.25 * sH, 0.5*0.25 * sH and weight>0*

Decrement weight bt 5

Call the wprint function

*If pressed function returns true with the arguments 0.1*sW+0.5*0.20*sW ,0.6*sH+0.5*0.25 * sH, 0.5*0.25 * sH*

Increment weight bt 1

Call the wprint function

*If pressed function returns true with the arguments 0.1*sW+0.5*0.20*sW, 0.125*sH+0.5*0.25 * sH, 0.5*0.25 * sH and weight>0*

Decrement weight bt 1

Call the wprint function

Code:

```
//updation of weight on button press by the user
void set_weight(){
  //+5 box
  if(pressed(0.1*sW+0.5*0.20*sW,0.125*sH+0.5*0.25 * sH,0.5*0.25 * sH)){
     weight+=5;
     wprint();
  }
  //-5 box
   if(pressed(0.35*sW+0.5*0.20*sW,0.125*sH+0.5*0.25 * sH,0.5*0.25 * sH) &&
weight>0){
     weight-=5;
     wprint();
  }
  //+1 box
  if(pressed(0.1*sW+0.5*0.20*sW,0.6*sH+0.5*0.25 * sH,0.5*0.25 * sH)){
     weight+=1;
     wprint();
  }
```

```
  //-1 box
    if(pressed(0.35*sW+0.5*0.20*sW,0.6*sH+0.5*0.25  *  sH,0.5*0.25  *  sH)&&
weight>0){
      weight-=1;
      wprint();
  }
}
```

Filtration of Raw Sensor Data

Description:

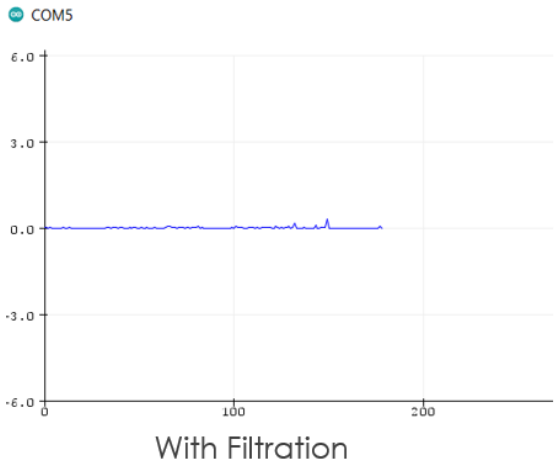
The data from the IMU accelerometer sensors is raw and unfiltered. This data needs to be filtered before being used for calculations as it has a lot of noise. We incorporate a 1st degree Low-pass filter called Exponential Moving average filter to remove the excess noise.

The filter is applied based on the equation below:

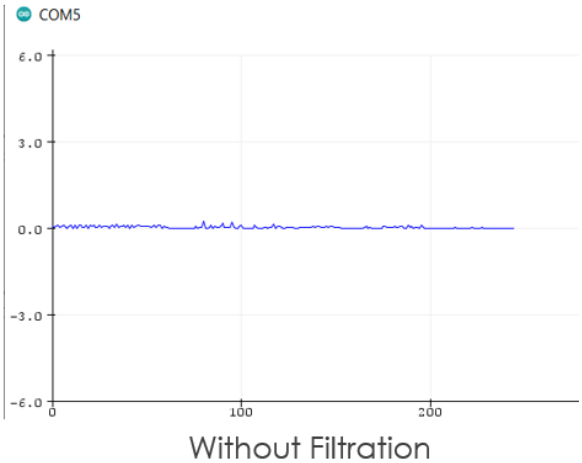
$$aFilt = ((1-\alpha) * aFilt(old)) + (\alpha * aRaw)$$

aFilt is the value of accelerometer data after filtration and aRaw is the value of the accelerometer data before filtration. 'α' depicts the extent of filtration which is between 0 and 1 (0 depicts the greatest extent of filtration).

The graphs below show the difference in net magnitude of acceleration values when the M5 stack device is in rest position.



With Filtration



Without Filtration

As it is evident from the graphs the graph without filtration shows much more noise in the signal than the one with filtration applied (the single peak in towards the end of the graph was due to unintentional movement of the device). Therefore it is important to apply filtration to the raw data in order to get a more smooth reading.

Algorithm Design:

Define the filter function of void return type taking X, Y, and Z as arguments passed by reference

*Compute [((1-alpha)*aXFilt) +alpha*X] and store it in aXFilt*
*Compute [((1-alpha)*aYFilt) +alpha*Y] and store it in aYFilt*
*Compute [((1-alpha)*aZFilt) +alpha*Z] and store it in aZFilt*

Code:

```
//Low Pass Filter Function
void filter(float &X, float &Y, float &Z){
  aXFilt = (1-alpha)*aXFilt + alpha * X;
  aYFilt = (1-alpha)*aYFilt + alpha * Y;
  aZFilt = (1-alpha)*aZFilt + alpha * Z;
}
```

Computing Acceleration

Description:

Calculating the final magnitude of the acceleration is a multi-fold process. We used the approach highlighted in the research paper "Step Detection Algorithm For Accurate Distance Estimation Using Dynamic Step Length" for computing acceleration. The steps are highlighted below:

- The M5stack comes with a built-in accelerometer sensor present in the Inertial Measurement Unit of the M5stack. It takes in three variables, each corresponding to the three axis of acceleration.
- After the acceleration values are obtained, they are filtered using the filter function.
- The magnitude of the acceleration is calculated using the formula:

$$\text{Magnitude} = \sqrt{x^2 + y^2 + z^2}$$

Where x, y and z correspond to the acceleration values along the three axes.

- First, an initial acceleration is computed. Then an average acceleration is computed for 50 trials.
- The final magnitude is the absolute difference between the initial and average magnitude.
- This process ensures the removal of the force of gravity from the final magnitude and makes the value desirable for step detection.

Algorithm Design:

Define the getVal function of float return type

Call the getAccelData function of the IMU module of M5 stack using aX, aY and aZ variables passed as arguments by reference
Call the filter function with aX, aY, and aZ passed as arguments
Declare a variable init_mag of float datatype and assign the square root of (aX^2 +aY^2+aZ^2) to it
Declare a variable magsum of float datatype and initialize it with 0.0
Declare a variable i of int datatype and initialize it with 0
Repeat while i is less than 50

Call the getAccelData function of the IMU module of M5 stack using aX, aY and aZ variables passed as arguments by reference
Call the filter function with aX, aY, and aZ passed as arguments
Declare a variable mag of float datatype and assign the square root of (aX^2 +aY^2+aZ^2) to it
Increment magsum by mag
Increment i by 1

Declare a variable magavg of float datatype and initialize it with the value of magsum/50
Return the absolute value of magavg and init_mag

Code:

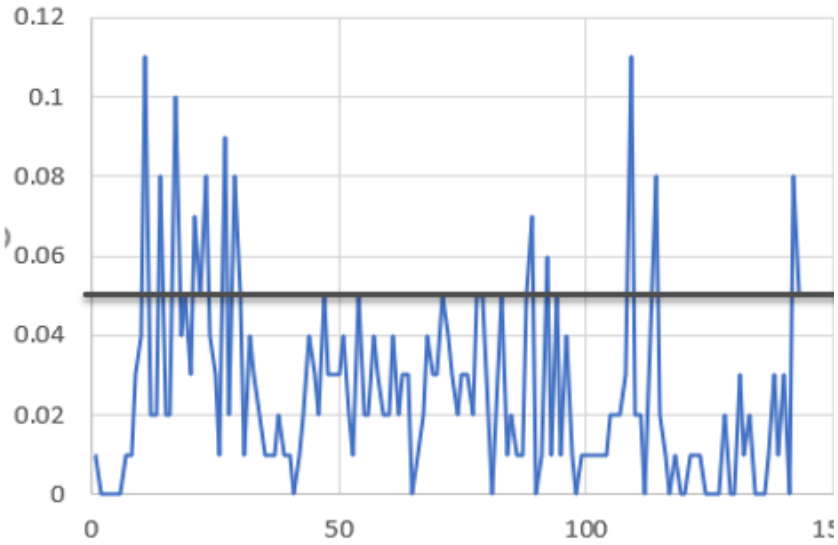
```
//gives final value of acceleration magnitude for each pass
float getVal(){
      M5.IMU.getAccelData(&aX,&aY,&aZ); //retrives acceleration data from device
    filter(aX, aY, aZ); //filters noise from the data
     float init_mag = sqrt(pow(aX,2) + pow(aY,2) + pow(aZ,2)); //magnitude of initial acceleration

    //finding average of 50 runs of the accelerometer
    float magsum = 0.0f;
```

```
    for(int i=0; i<50; i++){
      M5.IMU.getAccelData(&aX,&aY,&aZ);
      filter(aX, aY, aZ);
      float mag = sqrt(pow(aX,2) + pow(aY,2) + pow(aZ,2));
      magsum += mag;
    }
    float magavg = magsum/50;

       //returning the magnitude excluding the force of gravity (by
subtracting the average)
    return abs(magavg - init_mag);
}
```

Detecting Steps

Description:

Steps are detected with the help of a simple algorithm. First a threshold for acceleration is set. When the net magnitude of the acceleration of the device goes beyond this threshold, a step is detected. The threshold was set by first graphing the points of net acceleration magnitude from the serial monitor. We observed the trend of peaks in our data when we took a step or did an action that mimicked a step (jerks, rotations, etc.). The graph showing the values of net magnitude and the threshold line is given below.



Based on the peaks, we decided on setting 0.05 G of acceleration as the threshold value for step detection.

The function for step detection also updates the walk time and distance. Every time a step is detected, the distance is incremented by a random value between minimum and maximum step length (in meters) of an average human being. The walk time is set in a similar way by incrementing it by a random value between minimum and maximum time taken by an average human being to take a step. Since the seed of the random number generator was set to system time, it always produced new sets of random numbers. Finally the calories function is also called inside this function to update the total calories burnt.

Algorithm Design:

Define the stepDetect function of void return type

If netmag is greater than equal to threshold and flag is true

Flag is false

Increment count by 1

Increment dist by a random value between dmin and dmax

Increment wtime by a random value between wtmax and wtmin

Call the calories function and assign the value returned to cal

Else if netmag is less than threshold

Set flag to true

Code:

```
//function to detect the steps, update distance, and calories
void stepDetect(){
  //detecting a step
  if(netmag >= threshold && flag){
     flag = false; //flag turns false to avoid repetitive step count for the same step
    count++; //count of step increases
     dist+= ((double)rand() / RAND_MAX) * (dmax - dmin) + dmin; //distance per step randomised between dmax and dmin (m)
    wtime += ((double)rand() / RAND_MAX) * (wtmax - wtmin) + wtmin; //walk time per step randomised between wtmax and wtmin (sec)
    cal = calories(); //calories are updated
```

```
  }
  //if no step
  else if (netmag < threshold){
    flag = true; //flag turns true again only after the acceleration comes
below the threshold
  }
}
```

Calculating Calories

Description:

Calories are calculated based on the weight of the user, the walk time (in minutes), and the MET value of the physical activity (walking). MET stands for Metabolic Equivalents. One MET is defined as the energy you use when you're resting or sitting still. An activity that has a value of 4 METs means you're exerting four times the energy than you would if you were sitting still.

The formula for number of calories burned calculation is given below:

Calories burned =

Total Walk Time (in mins) x METs x 3.5 x (your body weight in kilograms) /200

The MET value for walking is defined in the section of symbolic constants.

Algorithm Design:

Define the calories function of float return type

Define the variable c of float dataype and assign it the value of [(wtime/60)(MET*3.5*weight)/200]*

Return the value of c

Code:

```
//calorie calculation
float calories(){
  float c = (wtime/60) * (MET *3.5 * weight)/200;
  return c;
}
```

Final User Interface

Description:

The final user interface shows the user the number of steps they took, the distance they traveled and the number of calories they burnt during the period. The screen keeps getting updated every 200 ms as the function gets called in the loop function. The image below shows how the final UI looks like.

Before walking

After walking

Algorithm Design:

Define the fUI function with void return type

Set the text size to 2
Set the text color to blue with black background
Print the value of netmag on the serial monitor
Set the cursor of the screen at coordinates (125,20)
Print "STEPS" on the screen
Set the cursor of the screen at coordinates (145,52)
Print the value of count on the screen
Set the cursor of the screen at coordinates (110,95)
Print "DISTANCE" on the screen
Set the cursor of the screen at coordinates (111,127)
Print the value of dist on the screen
Set the cursor of the screen at coordinates (110,172)
Print "CALORIES" on the screen

Set the cursor of the screen at coordinates (103,200)
Print the value of cal on the screen

Code:

```
//final UI to display the step count, distance and calories
void fUI(){
  M5.Lcd.setTextSize(2);
  M5.Lcd.setTextColor(BLUE, BLACK);
  Serial.println(netmag);
  M5.Lcd.setCursor(125, 20);
  M5.Lcd.printf("STEPS");
  M5.Lcd.setCursor(145, 52);
  M5.Lcd.print(count);
  M5.Lcd.setCursor(110, 95);
  M5.Lcd.printf("DISTANCE");
  M5.Lcd.setCursor(111, 127);
  M5.Lcd.printf("%6.2fm", dist);
  M5.Lcd.setCursor(110, 172);
  M5.Lcd.printf("CALORIES");
  M5.Lcd.setCursor(103, 200);
  M5.Lcd.printf("%6.2fcal", cal);
}
```

Loop Function

Description:

The second part of the main function is the loop function. It keeps calling all the functions called in its body repeatedly to perform all the desired actions. After the setup function, the loop function displays the initial user interface. The set_weight function is also called to keep updating the weight given as input by the user. When the user presses the “Done” button, the code replaces the initial UI with the final UI and keeps refreshing the screen after a delay of 200 ms.

Algorithm Design:

Loop Function

If f1 is true

Call the function iUI

Call the function wprint

Repeat while f1 is true

Call the function set_weight

*If the pressed function returns true with the arguments 0.79*sW, 0.7*sH, 0.07*sW*

Set f1 to false

Open new screen on the M5 Stack with black background

Call the getVal function and store its value in netmag

Call the stepDetect function

Call the fUI function

Give a delay of 200 ms before looping again

Code:

```
//keeps looping
void loop() {
  //to initially set up the screen
  if(f1){
    iUI();
    wprint();
  }
  //keeps updating the weight until the done button is pressed
  while(f1){
    set_weight();
    if(pressed(0.79*sW, 0.7*sH, 0.07*sW)){
      f1 = false; //flag turns false as soon as the done button is pressed
      M5.Lcd.fillScreen(BLACK);
    }
  }

  //net magnitude of the acceleration is calculated
  netmag = getVal();

  //detecting a step
  stepDetect();
```

```
  //final UI is displayed
  fUI();

  //delay of 200 ms
  delay(200);
}
```

Contribution of each member

- Swostik Pati: Implementing the entire code, including the step-detection algorithms, mechanism for distance measurement and calorie calculation, building of initial and final user interface. Tested each functionality of the project. Assisted Saamia in making the presentation slides. Wrote the complete final project report.

- Saamia Shafqat: Helped in designing the Initial User Interface and creating the slides for presentation. Helped in testing functionalities.

- Aziq Furqan: Helped in creating the Final User Interface. Helped in testing functionalities.

3. Results and Evaluation

Initial User Interface and User Input

Testing

The Initial User Interface was tested by user interaction. Each of the four buttons were individually pressed to make sure they were touch enabled. The change in weight value validates the user input. Finally, as soon as the user presses the done button, they are taken into the final UI screen.

Challenges and Solutions

Figuring out a desirable way of taking the weight as input was a big hassle. Making a number-pad for such a trivial task didn't seem wise. So we devised a better mechanism to take the weight as input from the user. We provide the user with four buttons, each giving the option to increment or decrement the weight by either 1 or 5. This allows the user to set the desired weight. Finally the user presses done and is taken to the next screen. Another challenge was to decide the placement of buttons and text on the screen. But with several trial runs, we were able to figure out an intuitive UI for the user.

The most important challenge that we faced while taking user input was caused due to the excessively high looping rate (200ms). Due to this even when the user pressed a button once, it got registered as several presses(based on the time for which the finger was pressed), and incremented/decremented the weight by massive amounts. To prevent this, we introduced a flag (boolean expression). This flag switched between true and false values. For a button to be registered as a press, apart from the press point being in the region of the touch area, the flag must also have a true value. Also as soon as the press is registered, the flag turns false thereby eliminating chances of a double press and will only turn true when the user removes his finger off the screen. So a button will only register as a second press when the user lifts his finger from the screen and presses again.

Computing Acceleration

Testing

The evaluation of the final value of the magnitude of acceleration is done by using the serial plotter and serial monitor functions in Arduino IDE. The values that we get from the serial monitor match the desired values of acceleration magnitude (in G) that we should get. Also, on plotting these values in the serial plotter and moving the M5 stack device, we observe corresponding shifts in the graph (in form of peaks).

Challenges

There were two big challenges while calculating the net magnitude of acceleration - filtering data and excluding force of gravity.

- As discussed earlier, the accelerometer data is filled with a lot of noise and therefore needs to be filtered using a low pass filter to allow only the low frequencies to travel excluding the excess noise. This was employed using the filter function which used the exponential moving average method to build a 1st degree low pass filter.
- To exclude the force of gravity, we followed the method mentioned in the paper “Step Detection Algorithm For Accurate Distance Estimation Using Dynamic Step Length”. We computed an average magnitude of acceleration using 50 trials and defined the net magnitude as the absolute difference between the average value and each observation. This facilitated the process of excluding all the force due to gravity data that might cause bias in our observations.

Step Detection

Testing

Each one of us walked several steps with the M5 stack tied to their wrists, or in their pockets(mimicking a mobile phone). The number of steps that the user took were compared to the final step count shown in the screen. It was found that for short distances the step counter gave almost accurate results with an error rate of +/-2%. For long distances, the results were within the error rate of step calculation was +/-5%.

To verify our device with other similar devices, we compared the step calculation by our device to that of Google fit and the observations are laid down below:

The left image shows the initial step count and the right-most one shows the final step count in Google Fit. The center one shows the calculation of steps in the M5 stack for the same walk. As we can see the step count in our device (286 steps) is very close to the one calculated by Google Fit (2774 - 2474 = 300 steps).

Challenges

There were two big challenges while developing the algorithm to calculate steps - setting a reasonable threshold for step detection and avoiding multiple step counts for the same step.

- Setting a reasonable threshold - We tried with a range of acceleration values for setting the step calculation. Initially we used to test our product by manually walking and checking for different threshold values. Even though this helped us get closer to the threshold values, it was very time consuming and couldn't give us the most accurate value for the threshold. On further research we learnt the use of serial plotter and serial monitor in the arduino IDE. Using the values from the serial monitor and plotting them in excel, we were able to figure out an accurate threshold for step detection. Our observations were further validated during the testing phase.

- Avoiding false/multiple step counts - To avoid false step counts(false peaks), we observed the false peaks from the data from the serial monitor and decided to keep our threshold above the false peaks. One more problem that we encountered was that as the program refreshed much faster than a user could complete a full step, multiple steps were calculated for the same step or during random jerks during which the net acceleration stayed above the threshold. This led to incorrect step counts. To counter this flaw we introduced another flag which was a boolean expression varying between true and false.

To detect a step, the acceleration must go above the threshold set and the flag also must be true. As soon as the acceleration goes beyond the threshold value, the step count is incremented and the flag turns false. The flag will only turn true again when the net acceleration comes below the threshold set. This will present the calculation of multiple steps and give us an extremely accurate reading of steps. It will also counter the problem of random jerks or motion in a vehicle.

Distance Measurement

Testing

We validated the distance measurement by walking for certain distances and then physically mapping the distance travelled (using applications from the internet). We compared our values and found them to be quite close to the actual measurement.

Challenges

Initially it was difficult to devise a mechanism to measure the distance traveled by the user. There are several methods that we studied. Some were beyond the scope of our project and others couldn't provide the distance measurement in real time. One of the methods that we looked into was multiplying the step count by the average step length (a fixed number) of a human being. We figured this could work and implemented it in our project, but on testing we figured out that it was far from accurate for both short and long distances. The real step length differed while walking (sometimes more and sometimes less than the average step length). This caused discrepancies in the calculation of total distance.

To counter this problem, we devised a modification to the method. Instead of multiplying the steps by a fixed number(average step length), we thought of incrementing the distance by a random length between maximum and minimum step length of an average user. Even though the real time additions to the distance may not be the real distance covered by the user, the aggregated final distance turned out to be very close to the real distance traveled by the user hence making our estimate much more accurate.

Calorie Calculation

Testing

Initially we tested to see if our formula was running properly, based on the given weight and walk time. On manually calculating the calories based on the walk time and weight, we found the values to be almost the same as the one shown in the device. This way we were sure the calorie calculation was close to accurate. We further went to check where our device stood in terms of apps like Google Fit. The comparison is shown in the images below.

The left image shows the initial calorie count and the right-most image shows the final calorie count in the Google Fit application. The center image shows the calorie count in the M5 Stack device. As we can see the calorie count in our device (11.8 cal) is very close to the one calculated by Google Fit (1125 - 1115 = 10cal).

Challenges

Even though the amount of calories burnt could be calculated using a simple formula, obtaining the information required for the formula was a difficult task. We had to design an entire initial interactive user interface to be able to take the weight of the user as input from them. The challenges for designing this interactive interface are already highlighted above. Further, we also required the walk time of the user (in minutes). Initially we thought of using the system time but we realized there was a big flaw in using this approach. There might be extended periods of time

when the user won't be moving but the system time would continue to increment. This will cause an extremely inaccurate measurement of the user's walk time.

To counter this flaw, we resorted to a similar method we used for measuring the distance traveled by the user. Whenever the user took a step we incremented the walk time by a random value between the minimum and maximum step time of an average user. Again, even though the real time additions to the walk time may not be the real time taken by the user to walk a step, the aggregated final walk time will be very close to the real walk time traveled by the user, hence making our estimate very accurate. This walk time was then used to calculate the calories using the formula and the final value was displayed to the user in real time.

Final User Interface

Testing

The final user interface shows the real time values of steps taken, distance travelled, and calories burned. We tested this component by observing the screen while walking. The steps, distance, and the calories got updated in real time without any lag. The measurements were also accurate.

Challenges

Since the final user interface required us to just display the information provided by other functions, there weren't many challenges to this. The only challenge was to display all the information in a well formatted way so as to make it easy for the user to read from the screen. We divided the screen into three horizontal sections, each showing the step count, the distance travelled, and the calories burnt, respectively.

4. Conclusion and Future Work

Project Achievements

The following are the milestones that we have successfully achieved in our project:

- Build a fully functional working prototype of our project - Pedometer.
- Configure the M5 stack to allow the user to input their weight.
- Filtering raw data, creating interactive user interfaces, and implementing many other small applications which have big implications for the final project.
- Calculate the number of steps taken by the user, the distance travelled by the user, and the calories burnt by the user.
- Display the information in a clear and concise manner with real-time updates
- Figure out an adequate threshold for step detection, and a mechanism to calculate distance travelled and calories burnt.
- Eliminate the problem of multiple counting of steps, excessive increments of weight values, and many other problems.
- Extensive testing of our product to make sure it provides as accurate values as possible in the scope of our project.

Potential Developments

The following improvements can be made in our product to enhance its value even further:

- Using better and more efficient methods of step counting, distance measurement, and calorie calculations.
- Using better mechanisms of filtering data to give even more accurate information.
- Generalizing the algorithm to accommodate more forms of physical activity apart from walking and map out these activities separately.
- Using GPS sensors and Google maps API to map out the route taken by the user while performing the activities so as to make the interface more creative and interactive

- Calculating BMI using data obtained from the user and the sensors which would be very helpful for several applications.
- Giving users the option to set daily goals for themselves therby increasing their sense of accountability.
- Curating personalized pathways to help the user achieve a better and healthy lifestyle.

5. Reflection on Learning

The project taught us a lot of important skills. It helped us to use the methods of experiential learning to build further on our theoretical knowledge and pushed us to challenge our boundaries. Some of the most important highlights of our learning experience were:

- Working with the Arduino IDE and the M5 Stack Core 2 device.
- Understanding the process of self-learning syntax and functions by reading documentation, running example codes, and contributing in forums.
- Understanding the process of research by reading several papers and articles about the project.
- Learning the importance of algorithm design in making such complex projects.
- Learning the process of designing a robust step-detection algorithm, mechanism for distance calculation, and computing calories.
- Learning the process of filtration of raw data using Exponential Moving Average filter
- Designing an interactive user interface and creating touch buttons to take input from the user.
- Learning the use of MET values for calorie calculation.
- Understanding the process of creating a project proposal for an idea.
- Realizing our limits to build the idea, making modifications to the initial idea, and finally building a working prototype from scratch.
- Presenting our working prototype with all its functionalities.
- Understanding the future scope of our project.
- Understanding the value of teamwork and coordination.

References:

- Who.int. 2021. Cardiovascular diseases (CVDs). [online] Available at: <*https://www.who.int/news-room/fact-sheets/detail/cardiovascular-diseases-(cvds)*>
- MET values of 800+ Activities. *https://golf.procon.org/met-values-for-800-activities/*
- MATLAB Arduino Tutorial 4 - Filtering Noise out of 3-axis Accelerometer Data in Real-time. *https://www.youtube.com/watch?v=TeKk3DjN_gs*
- Step Detection Algorithm For Accurate Distance Estimation Using Dynamic Step Length. *https://arxiv.org/ftp/arxiv/papers/1801/1801.02336.pdf*
- Calorie Calculation using MET values. *https://www.healthline.com/health/what-are-mets*
- M5 Docs: *https://docs.m5stack.com/en/api/core2/touch*
- Acceleration Formula: *https://physics.stackexchange.com/questions/41653/how-do-i-get-the-total-acceleration-from-3-axes*