Term Project
Need for Speed Challenge

Real Time Embedded Systems

Shantanu Kumar
Sk9698

NYU

Objective:

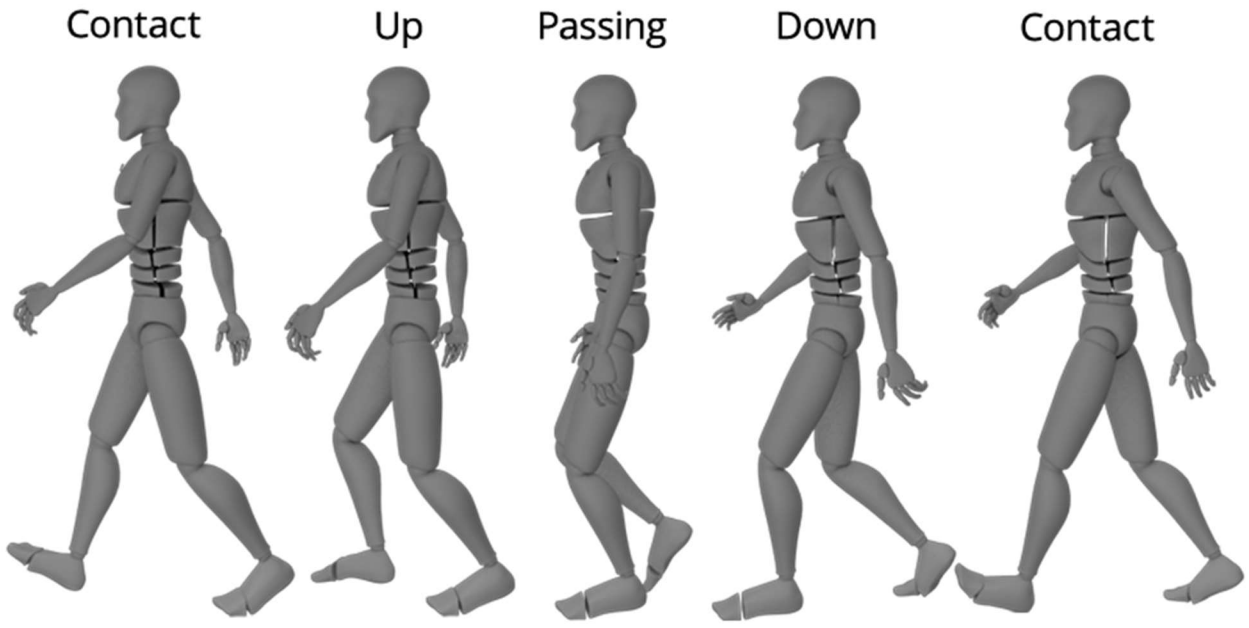
The objective of this semester's embedded challenge is to build a wearable speedometer which can calculate velocity by measuring angular velocities available from our built-in gyroscope (L3GD20) - without a GPS.

Approach:

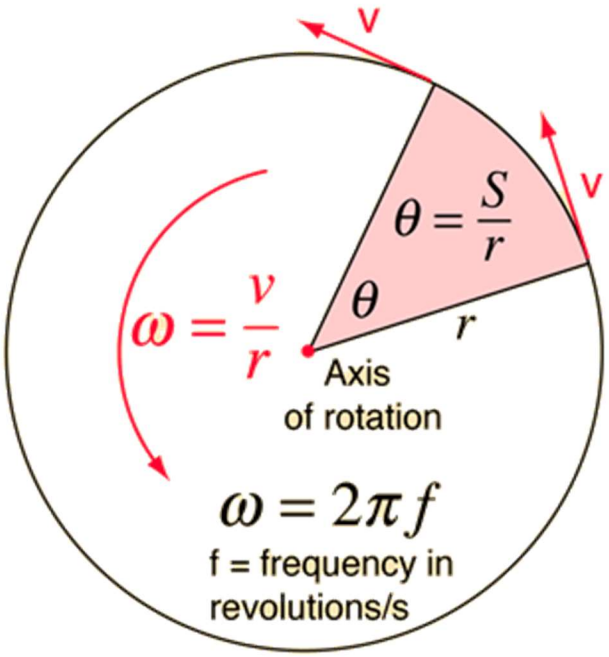
Understanding the problem, the gyroscope only gives us the angular velocity of an object generally denoted as omega. For an object rotating about an axis, every point has same angular velocity. The tangential velocity or the linear component of angular velocity is proportional to the distance from the angle of rotation.

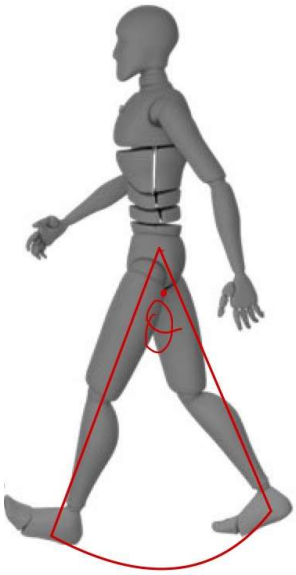
$$\vec{v} = \vec{\omega} \times \vec{r}$$

So, I decided to put the board on the ankle as legs generate most movement while walking. Let's see how we can interpret the movement in terms of angular velocity using the image given below.



We can infer that our legs move in a pendulum like movement, if we assume that the length from our hip to the ankle as the radius of a circle then we can assume the linear component to be omega multiplied by the radius. For me the length is about 3 feet from my hip joint to my ankle. The circumference of the circle covered by each step can be substituted as the linear distance travelled.





Interfacing with gyroscope:

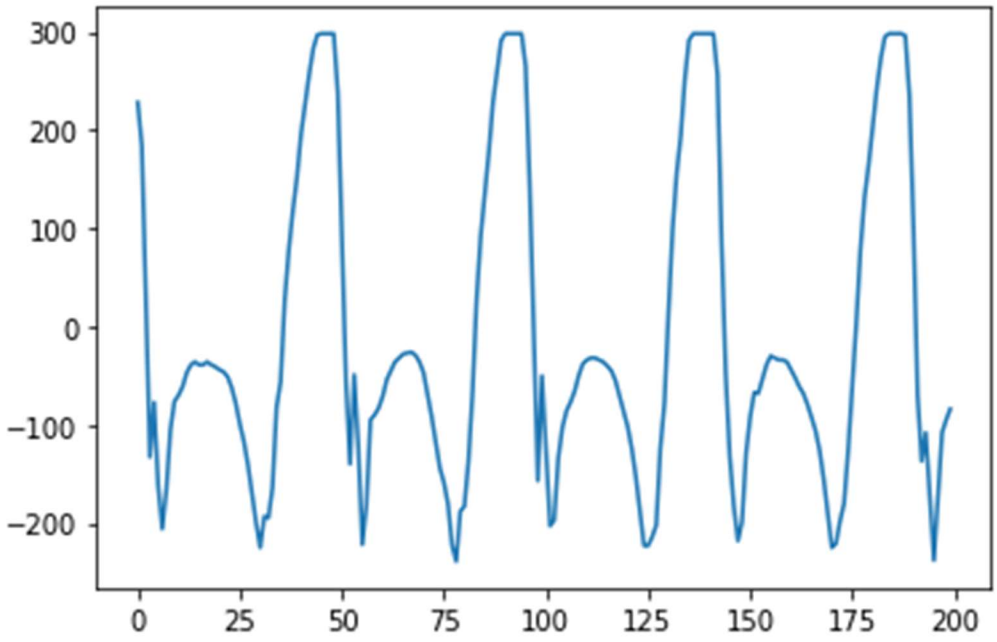
SPI communication was used to gather data from the gyroscope, I wrote a program which read the z high and low register for the gyroscope readings and passed the values by reference to the main function. I used int_16t as the data provided from the gyroscope was signed.

16-bit gyroscope's output data are in 2's complement format (signed integer) and the typical sensitivity at ±250 dps is 0.00875 dps/LSB from the datasheet.

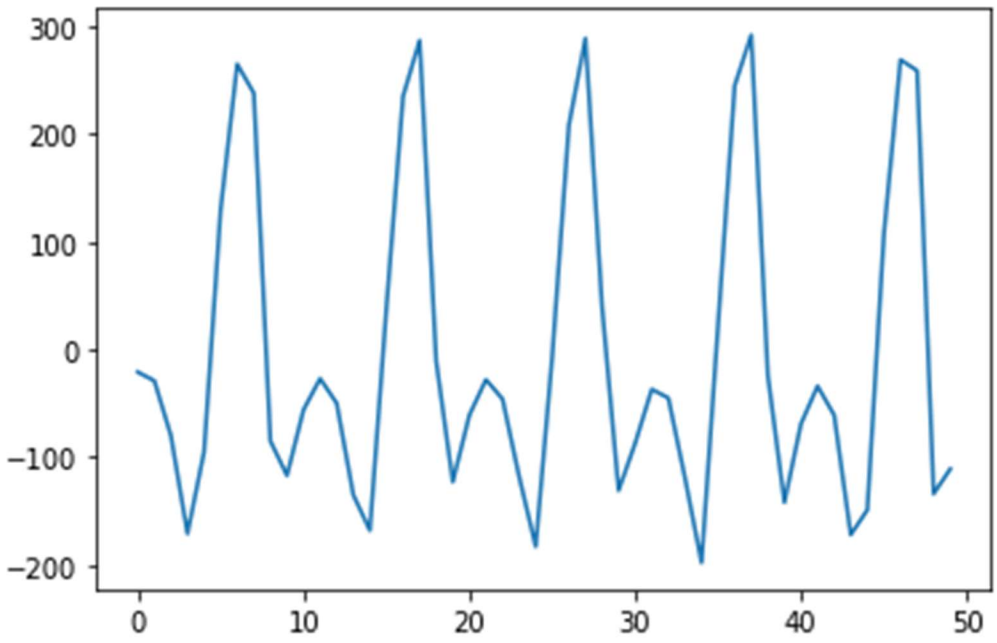
Z-axis: FF96 LSBs = -106 LSBs = -106 * 0.00875 = -0.93 dps

Frequency of Sampling:

I sampled the gyroscope at 100Hz and took an average of 10 readings, so the final sampling frequency was 10Hz. It shown in the plots below that sampling at 10Hz would still maintain the resolution of the data.



Sampling angular velocity of walking at 100Hz

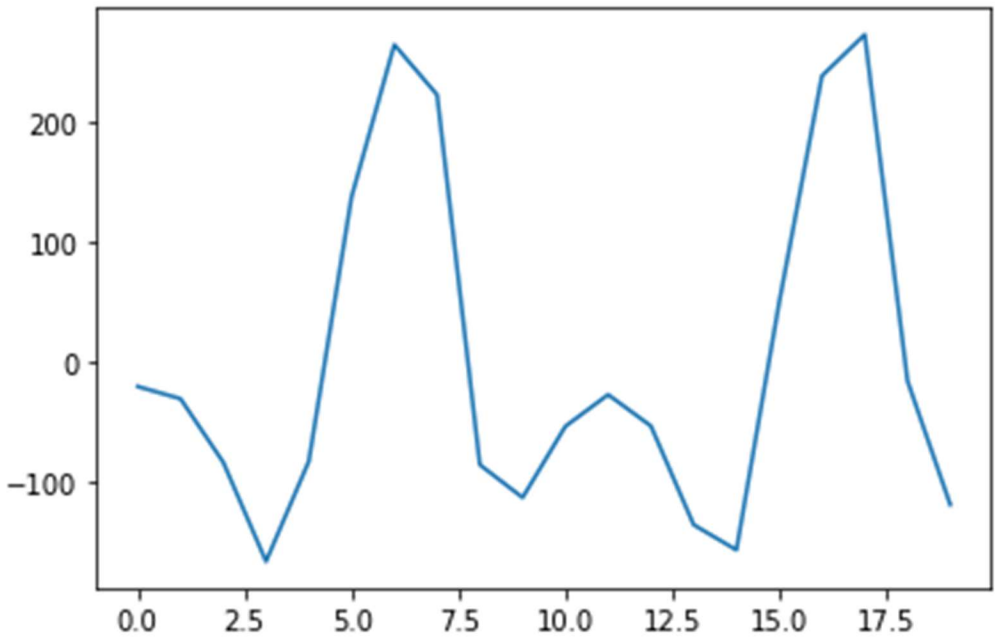


Sampling the angular velocity of walking at 10Hz

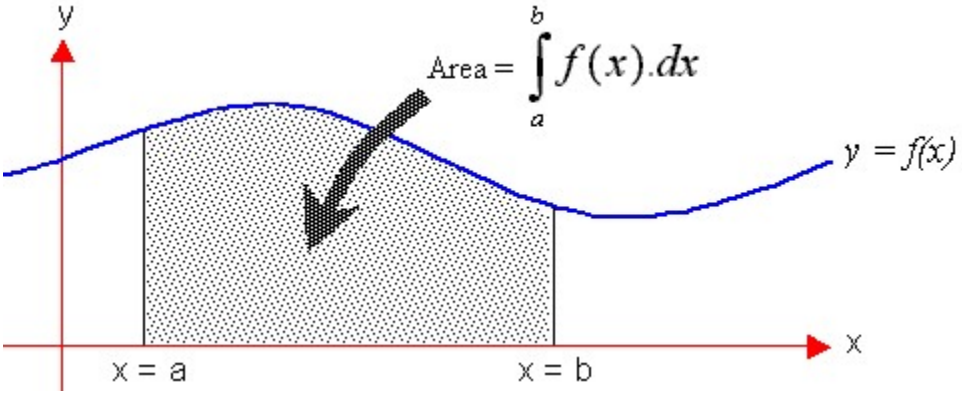
Filtering the Values:

I used a simple low pass filter with equation:

test[i] = (1-alpha).test[i-1] + alpha*(test1[i]+test[i-1])/2



Calculating the Distance:

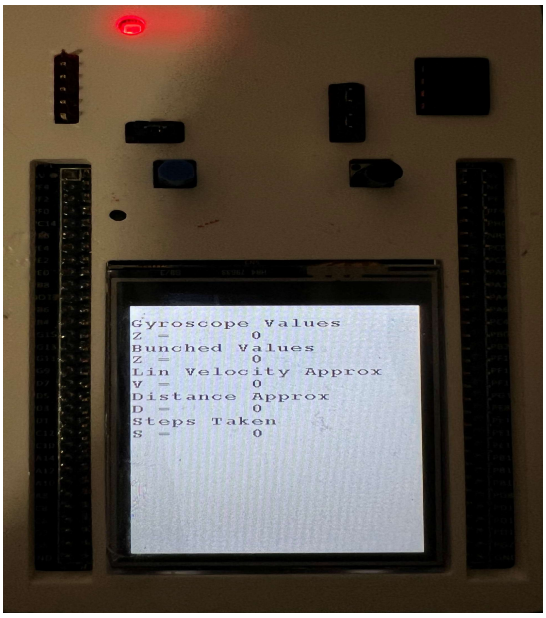


Integrating the area under the curve of the velocity graph would give us the displacement, I am taking a sample every 10ms and integrating it to find the distance travelled during it and adding it overtime to find the total distance.

Only positive values of the angular velocity were taken into consideration as the negative rotations were created by the twisting of ankle.

Displaying the Values:

I have used the onboard LCD Display to display the values measured over time. The library provided by STM was used.



Decreasing Idle Error:

As we don't walk most of the time, its important to decrease the error of the device while we are doing other things or actions such as sitting. I introduced an idle state to the device. I created a buffer to store the velocity values and took an average of it. I created a threshold value for this acceleration, below which the device would not count towards the distance travelled.

Pedometer Algorithm:

Counting the number of steps can be helpful as it can help us measure the distance travelled. I used a flag and velocity values to count towards the number of steps taken. Very slow steps cannot be detected by this algorithm, but we can use reinforced learning to correct such issue.

Testing:

I used the street outside my house to test the device. I used my iPhone to measure the distance which was about 50 feet.

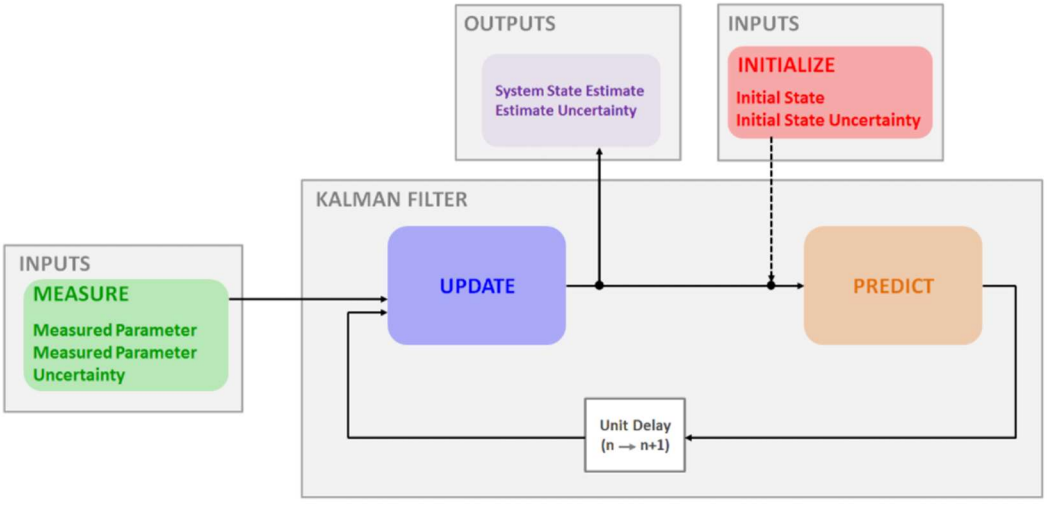
I conducted four test runs on camera and walked with different speed, the results were:

Serial Number	Distance(feet)	Measured Distance(feet)	Percentage Error
1	50	47	6%
2	50	50	0%
3	50	48	4%
4	50	51	2%

On an average of four readings, an average error of 3% was observed. The device gave accurate reading for different speeds of walking as the speed of walking increase so does the angular velocity.

Closing Notes and Future Aspects:

We have created a device which can track people activity and influence them to a better lifestyle. Adding IoT or Bluetooth to connect to an app would be convenient for the user where they can track their friend's activity. Adding an accelerometer would give us more accurate readings in terms of linear acceleration and movement. We can also use Kalman filter for increasing accuracy of the system with existing components.



Code:

Main.cpp

```
#include "mbed.h"
#include "LCD_DISCO_F429ZI.h"
#include "gyro.h" // contains values for Gyro
using namespace std::chrono;

SPI spi(PF_9, PF_8, PF_7); // mosi, miso, sclk
DigitalOut cs(PC_1);  // chip select pihn for Gyroscope

LCD_DISCO_F429ZI lcd;       //Create a LCD_DISCO_F429ZI object

Ticker ticker;

Timer t;

volatile bool is_sampleFlag = true;

void setSampleFlag() {

is_sampleFlag = true;

}


void getvalues(int16_t *z){
  cs = 0;

  // Send the adress of the z register to read along with write
  spi.write(GYRO_ZOUT_H | READFLAG);

  // Send a dummy byte to receive the contents of the register
  *z = spi.write(DUMMYBIT);
```

```
  spi.write(GYRO_ZOUT_L | READFLAG);

  *z |= (spi.write(DUMMYBIT) << 8 );

  // Deselect the device
  cs = 1;

}



int main()
{


    cs = 1; // Chip must be deselected

    wait_us(1000);


    spi.format(8,3);  // Setup the spi for 8 bit data, high steady state clock,
    spi.frequency(1000000);  // second edge capture, with a 1MHz clock rate


    int16_t GyroBuffer_sum=0;
    char buf[20];           //temporary string variable

    lcd.SetFont(&Font16);

    cs = 0;
    spi.write(CONTROL_REG);
    spi.write(GYRO_CONFIG);
    cs = 1;
    wait_us(1000);
```

```
    int16_t z;  //Varables for the program
    int buffer=0;
    uint8_t foot_length = 3;
    int16_t velocity;
    float delta_distance = 0;
    float total_distance = 0;
    int round_distance;
    int16_t GyroBuffer[3]= {0,0,0};    //array to store the results for  Z from the
gyro
    uint8_t steps = 0;
    bool stepset = false;
    float time_count;
    uint8_t average_speed = 0;


    t.start();
    //ticker.attach(&setSampleFlag, 1ms);

    while (1)
    {
        GyroBuffer_sum=0; //setting the average to 0

        for(int j = 0;j<10;j++)
        {
     //if(is_sampleFlag == true)
        //{
        //is_sampleFlag = false;

        getvalues(&z);
        z = z*0.00825;  //multiplying for 250 senstivity
        buffer +=z;
        wait_us(10000); //sampling at 100Hz
     //}
        }
        buffer/=10; //getting an average of 10 values
```

```
        //printf("%d\n", buffer);

        GyroBuffer[2]=GyroBuffer[1]; //shifting values for buffer
        GyroBuffer[1]=GyroBuffer[0];
        GyroBuffer[0]=buffer;
        for(uint8_t k = 0; k<3;k++)
        GyroBuffer_sum +=GyroBuffer[k];
        GyroBuffer_sum/=3; //buffer average

        buffer = (0.8)*GyroBuffer[1] + 0.2*(buffer+GyroBuffer[1])/2; // low pass
filter



        velocity = buffer * foot_length * 0.0174; //degree to radian conversion

        delta_distance = velocity * 0.1;  //distance for small time

        if (delta_distance > 0 && GyroBuffer_sum >50) // filter for the idle values
        total_distance+=delta_distance;

        round_distance = (int)total_distance;

        if (buffer > 150 && stepset == false) //pedometer algo
        {
            steps+=1;
            stepset = true;
        }
        if(buffer < 150)
        stepset = false;

        time_count = duration_cast<milliseconds>(t.elapsed_time()).count() / 1000;
//total time in seconds


        average_speed = round_distance/time_count; //calculating average speed since
active
```

```
        GyroBuffer_sum=0;



        lcd.DisplayStringAtLine(1, (uint8_t *) "Gyroscope Values"); //code to display
values via LED
        sprintf(buf, "Z = %6d", z);
        lcd.DisplayStringAtLine(2,(uint8_t *) buf);
        lcd.DisplayStringAtLine(3, (uint8_t *) "Bunched Values");
        sprintf(buf, "Z = %6d", buffer);
        lcd.DisplayStringAtLine(4,(uint8_t *) buf);
        lcd.DisplayStringAtLine(5, (uint8_t *) "Lin Velocity Approx");
        sprintf(buf, "V = %6d", velocity);
        lcd.DisplayStringAtLine(6,(uint8_t *) buf);
        lcd.DisplayStringAtLine(7, (uint8_t *) "Distance Approx");
        sprintf(buf, "D = %6d", round_distance);
        lcd.DisplayStringAtLine(8,(uint8_t *) buf);
        lcd.DisplayStringAtLine(9, (uint8_t *) "Steps Taken");
        sprintf(buf, "S = %6d", steps);
        lcd.DisplayStringAtLine(10,(uint8_t *) buf);
        wait_us(1000);
    }

}
```

Gyro.h

```
#define CONTROL_REG  0x20
#define GYRO_CONFIG  0xFF
#define GYRO_ZOUT_H  0x2D
#define GYRO_ZOUT_L  0x2C
#define WRITE        0x6A
#define DUMMYBIT     0x00
#define READFLAG     0x80
```