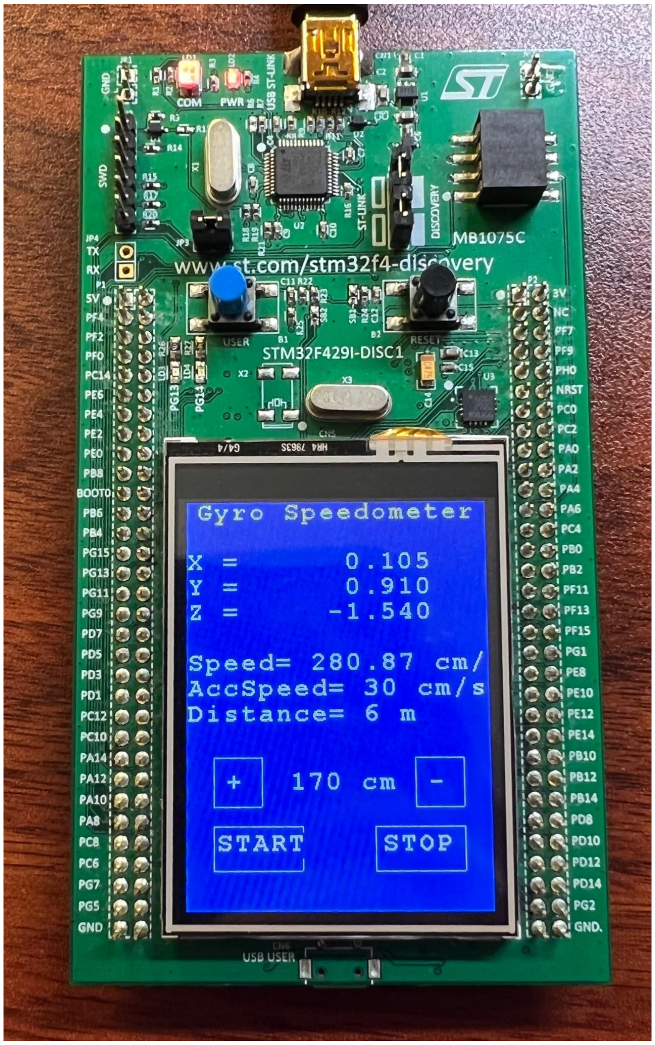
The Embedded Gyrometer “The Need for Speed”



Yanjie Xu (yx2304)

Puda Zhao (pz2078)

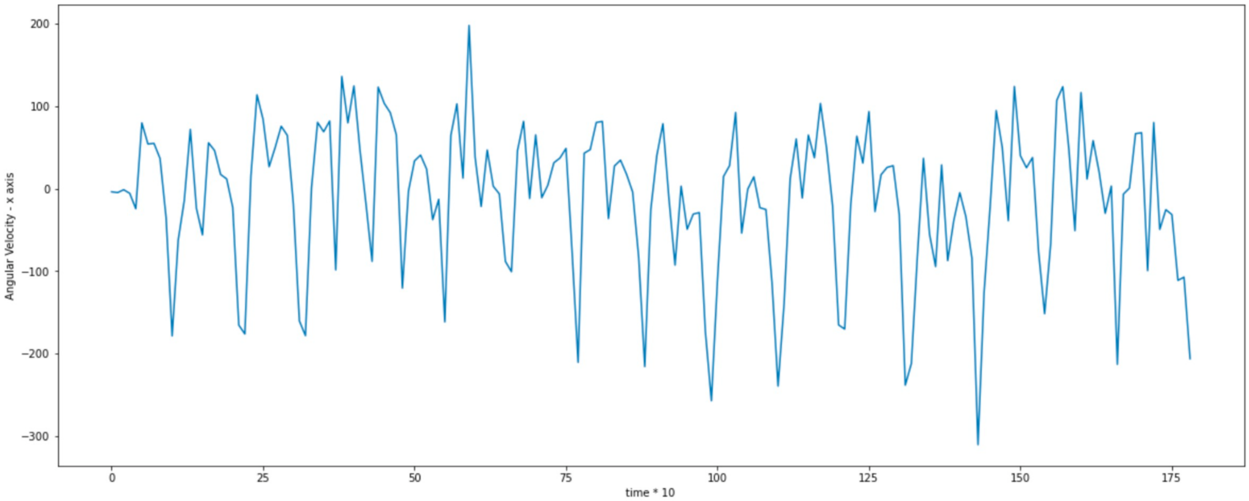
Dec 22 2021

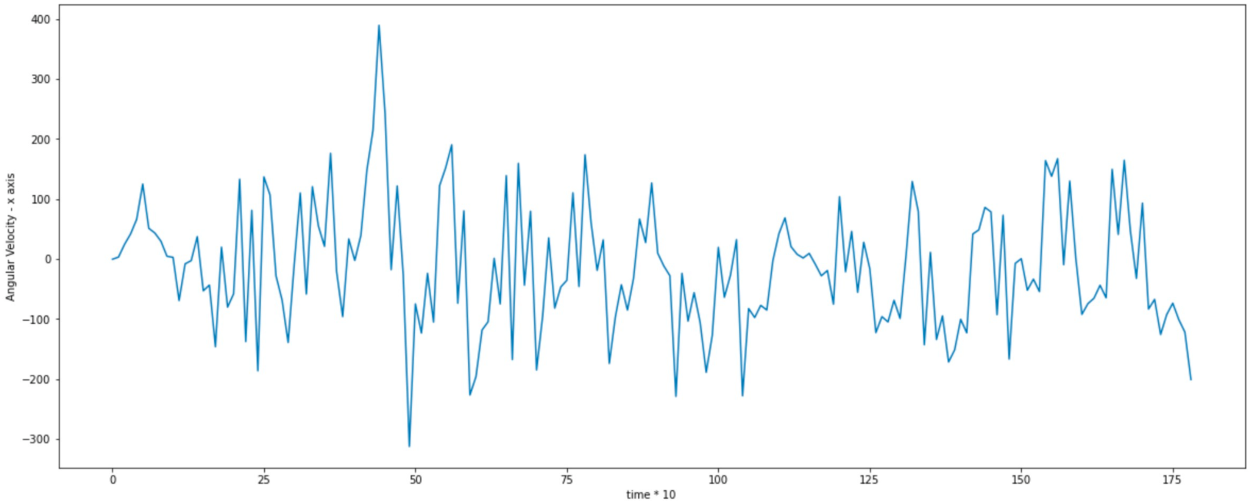
Brief Description

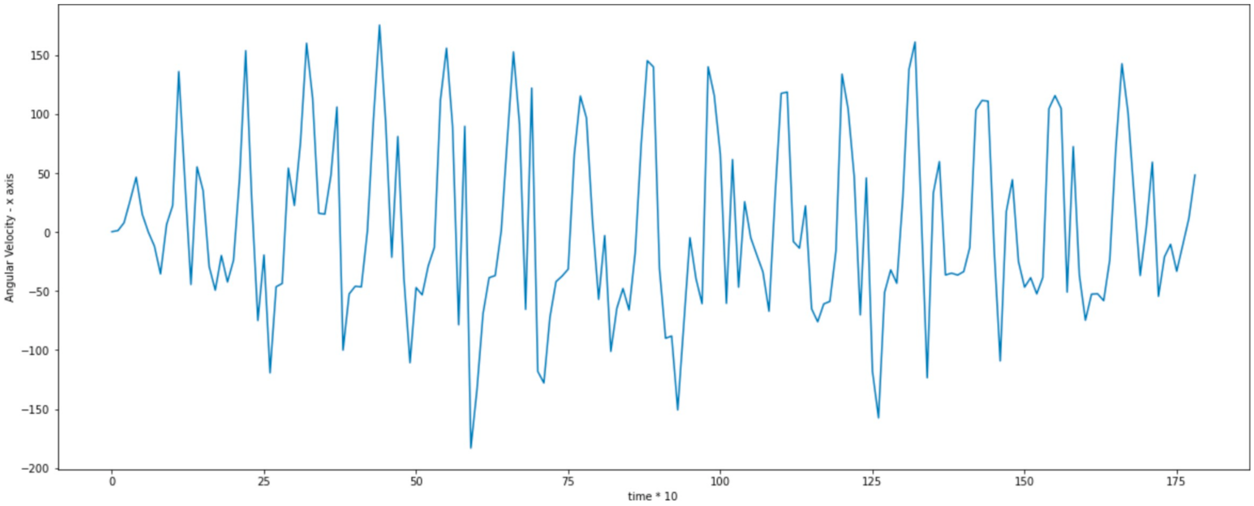
Our approach is to use gyro on STM32F429-DISC1 to get the angular velocity of leg movement. We then find the peak points when the user walks, and set one peak point as a flag of 1-step walking movement.

To get total walking distance, we calculate the product of the user's stride and steps count.
To calculate his/her stride, we let the user input his/her height on the touch screen. And then by a height stride formula, we can get the corresponding stride. By checking the flags of walking movement, we can get the time when the user move leg from back to front. and then we can get the instant movement status(fast /slow) by dividing the user's stride and the time it took during two walking flags. By counting the flags of walking movement, we can get step count. And then we can get the total distance by using the step count. We can also get the average speed by the same method. Counting steps in 20s movement, and then computing the distance and average speed.

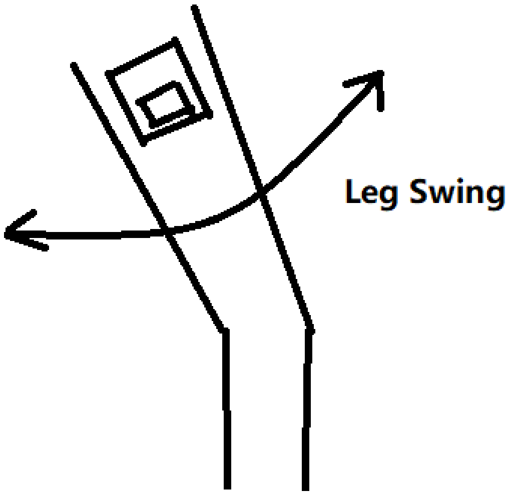
Illustration Figures and Data Plots





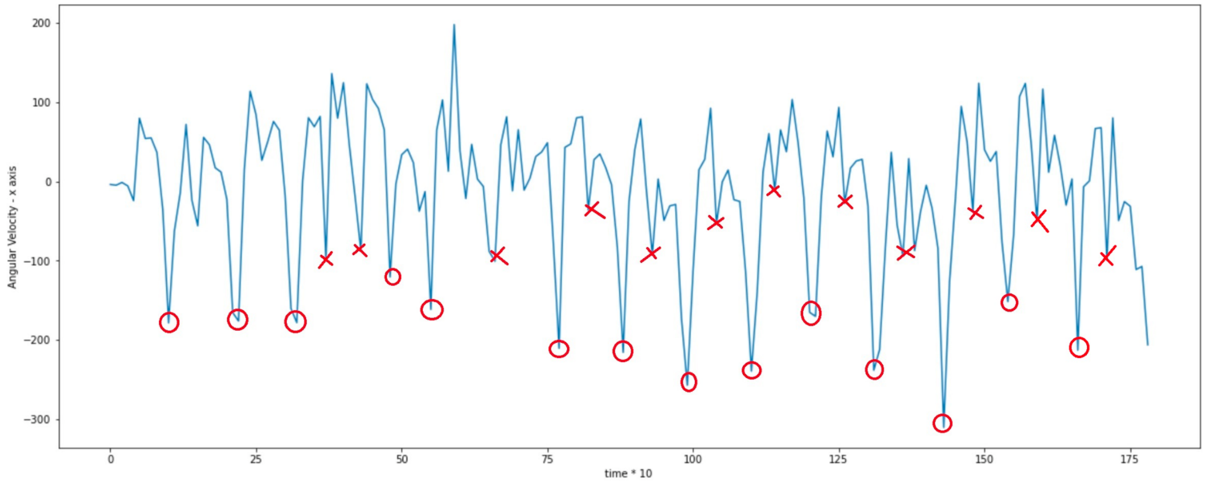


Here are 3 angular velocity-time plots drawn by a set of 20s data, corresponding to the x y z axis in order.（There are more than a dozen data sets used to analyze the pattern of motion.）



When we put the stm32 in our pants pocket in this way, we can see from the images that the angular velocity-time images of the x and z axis are more smooth. The plot of the x-axis is better, it does not have too much noise and shakes.

Its one-time local minimum usually represents a swing mid-end of the leg. By filtering and counting the number of peak points that meet the conditions, we can know the number of steps taken by the tester in the 20s.



To filter peak points, the following conditions must be met:
1.(local minimum) x[i] <= x[i-1] and x[i] <= x[i+1]
2.(timespan)The fastest human movement is 0.25s one step. In this project, because the sampling frequency is 0.5s, this condition does not play a role in filtering.
3.(Threshold) Remove peak points caused by jitter and abnormal motion.

Code

Platform: mbed, Platform IO, VS-Code
Libraries:

- BSP_DISCO_F429ZI
    - Need to modify wait_ms to wait_us in .pio/libdeps/disco_f429zi/BSP_DISCO_F429ZI/Drivers/BSP/STM32F429I-Discovery/stm32f429i_discovery.c .pio/libdeps/disco_f429zi/BSP_DISCO_F429ZI/Drivers/BSP/STM32F429I-Discovery/stm32f429i_discovery_sdram.c
- GYRO_DISCO_F429ZI
- LCD_DISCO_F429ZI
- TS_DISCO_F429ZI

We checked the GYRO_DISCO_F429ZI lib is using SPI communication, found on:
.pio/libdeps/disco_f429zi/BSP_DISCO_F429ZI/Drivers/BSP/STM32F429I-Discovery/stm32f429i_discovery.c
line 986, line 1010

```
//Author Yanjie Xu, Puda Zhao
//Dec 22 2021
//Gyro Speedometer Project
//Use Gyro to calculate walking speed

#include "mbed.h"
#include "rtos.h"
#include "GYRO_DISCO_F429ZI.h"
#include "LCD_DISCO_F429ZI.h"
#include "TS_DISCO_F429ZI.h"
#include <queue>
#include <list>

#define BUFFER_SIZE 30

void getGyroData(float* GyroBuffer);
void LCD_Setup();
void LCD_printGyro(float* GyroBuffer, uint8_t* text, uint8_t text_length);
void TS_Setup();
void TS_thread(TS_StateTypeDef* TS_State);
void updateStride(TS_StateTypeDef& TS_State);
void StartStopCheck(TS_StateTypeDef& TS_State);
void updateStride(TS_StateTypeDef& TS_State);
void writeData(queue<float *>& data, float* GyroBuffer);
void exportData(queue<float *>& data);
void calculateInstantVelocity(queue<float *>& data);
void calculateDistance(queue<float *>& data);

Thread ts_thread;
Mutex data_lock;
void TS_thread(TS_StateTypeDef& TS_State);

static uint8_t USER_HEIGHT = 170;
bool FLAG_START = false;

//Calculate Distance and Speed variables
```

```
static float timeCounter = 0;
const int32_t angularUpperLimit = -90;
static float gap = 0;
static float InstantVelocity = 0;
static float Last_InstantVelocity = 0;
static float Last_Last_InstantVelocity = 0;

GYRO_DISCO_F429ZI gyro;
LCD_DISCO_F429ZI lcd;
TS_DISCO_F429ZI ts;
DigitalOut led1(LED4);

int main()
{
   float GyroBuffer[3];
   uint8_t text[BUFFER_SIZE];
   TS_StateTypeDef TS_State;
   queue<float *> saved_data;
   printf("Gyroscope started\n");
   LCD_Setup();
   TS_Setup();
   ts_thread.start(callback(TS_thread, &TS_State));

   while(1) {
       getGyroData(GyroBuffer);
       LCD_printGyro(GyroBuffer, text, 30);
       // updateStride(TS_State);
       // StartStopCheck(TS_State);

       writeData(saved_data, GyroBuffer);
       calculateInstantVelocity(saved_data);
       exportData(saved_data);

       wait_us(1000 * 500);
   }
}

void TS_thread(TS_StateTypeDef* TS_State){
   while(1){
       updateStride(*TS_State);
       StartStopCheck(*TS_State);
       ThisThread::sleep_for(200);
   }
};


void getGyroData(float* GyroBuffer){
   gyro.GetXYZ(GyroBuffer);
   printf("%f, %f, %f\n", GyroBuffer[0]/500, GyroBuffer[1]/500, GyroBuffer[2]/500);
}

void LCD_Setup(){
   uint8_t test[20];
   BSP_LCD_SetFont(&Font20);
   lcd.Clear(LCD_COLOR_BLUE);
   lcd.SetBackColor(LCD_COLOR_BLUE);
   lcd.SetTextColor(LCD_COLOR_YELLOW);
   lcd.DisplayStringAt(0, LINE(0),  (uint8_t *)"Gyro Speedometer", CENTER_MODE);
   lcd.SetTextColor(LCD_COLOR_WHITE);
   lcd.DrawRect(21, 202, 38, 38);
   sprintf((char *)test, "+");
   lcd.DisplayStringAt(30, 213, (uint8_t *) test, LEFT_MODE);
   //print stride length
   sprintf((char *)test, "%d cm", USER_HEIGHT);
   lcd.DisplayStringAt(81, 213, (uint8_t *) test, LEFT_MODE);

   lcd.DrawRect(180, 202, 38, 38);
   sprintf((char *)test, "-");
   lcd.DisplayStringAt(190, 213, (uint8_t *) test, LEFT_MODE);

   lcd.DrawRect(21, 255, 70, 34);
   sprintf((char *)test, "START");
   lcd.DisplayStringAt(23, 260, (uint8_t *) test, LEFT_MODE);

   lcd.DrawRect(148, 255, 70, 34);
   sprintf((char *)test, "STOP");
   lcd.DisplayStringAt(153, 260, (uint8_t *) test, LEFT_MODE);
}

void LCD_printGyro(float* GyroBuffer, uint8_t* text, uint8_t text_length){
   sprintf((char *)text, "X = %10.3f", GyroBuffer[0]/500);
   // board foward/backword rotate
   lcd.DisplayStringAt(0, LINE(2), (uint8_t *)text, LEFT_MODE);
   sprintf((char *)text, "Y = %10.3f", GyroBuffer[1]/500);
   // board left/right rotate
   lcd.DisplayStringAt(0, LINE(3), (uint8_t *)text, LEFT_MODE);
   sprintf((char *)text, "Z = %10.3f", GyroBuffer[2]/500);
   // board clockwise/counter-clockwise rotate
   lcd.DisplayStringAt(0, LINE(4), (uint8_t *)text, LEFT_MODE);
}




void updateStride(TS_StateTypeDef& TS_State){
   uint8_t char_buffer[20];
   ts.GetState(&TS_State);
   if ((TS_State.TouchDetected) && (TS_State.X > 21) && (TS_State.X < 59) &&
(TS_State.Y > 202) && (TS_State.Y < 240)){
       //printf("+ btn pressed");
       if(USER_HEIGHT < 255){
           USER_HEIGHT += 5;
       }
   }
   else if ((TS_State.TouchDetected) && (TS_State.X > 180) && (TS_State.X < 218) &&
(TS_State.Y > 202) && (TS_State.Y < 240)){
       //printf("- btn pressed");
       if (USER_HEIGHT > 0){
           USER_HEIGHT -= 5;
       }
   }
   else{
       lcd.ClearStringLine(5);
   }
   sprintf((char *)char_buffer, "%d cm", USER_HEIGHT);
   lcd.DisplayStringAt(81, 213, (uint8_t *) char_buffer, LEFT_MODE);
}

void StartStopCheck(TS_StateTypeDef& TS_State){
   ts.GetState(&TS_State);
   if ((TS_State.TouchDetected) && (TS_State.X > 21) && (TS_State.X < 91) &&
(TS_State.Y > 255) && (TS_State.Y < 289)){
       //printf("start btn pressed");
       lcd.ClearStringLine(7);
       lcd.ClearStringLine(8);
       FLAG_START = true;
       lcd.DisplayStringAt(0, LINE(7), (uint8_t *)"START Pressed", LEFT_MODE);
   }
   else if ((TS_State.TouchDetected) && (TS_State.X > 148 ) && (TS_State.X < 218) &&
(TS_State.Y > 255) && (TS_State.Y < 289))
   {
       FLAG_START = false;
       lcd.DisplayStringAt(0, LINE(7), (uint8_t *)"STOP Pressed", LEFT_MODE);
   }
}

void TS_Setup(){
   ts.Init(240, 320);
}

void writeData(queue<float *>& data, float* GyroBuffer){
   if (FLAG_START){
       float* temp_array = new float[3];
       for (int i = 0; i < 3; i ++){
           temp_array[i] = GyroBuffer[i];
       }

       data_lock.lock();
       data.push(temp_array);
       if (data.size() > 40){
           data.pop();
       }
       data_lock.unlock();

       printf("Queue Size: %d\n", data.size());
   }
}

void exportData(queue<float *>& data){
   if (!FLAG_START and data.size() > 0){
       uint8_t cnt = 0;
       float X_axis_data[40];
       data_lock.lock();
       printf("--------------------CSV DATA-------------------\n");
       while (data.size()){
           float* temp_array = data.front();
           X_axis_data[cnt] = temp_array[0]/500;
           cnt += 1;
           printf("%f, %f, %f\n", temp_array[0]/500, temp_array[1]/500,
temp_array[2]/500);
           data.pop();
       }
       data_lock.unlock();

       //output accumulate average speed and total distance
       u_int16_t velocity = 0;
       u_int16_t distance_in_20_seconds = 0;
       uint8_t count = 0;
       uint8_t char_buffer[20];
       for(int i=1; i<39; i++)
       {
           if(X_axis_data[i] <= X_axis_data[i-1] && X_axis_data[i] <= X_axis_data[i+1]
&& X_axis_data[i] <= -80)
           {
               count ++;
           }
       }

       velocity = 4 * count * USER_HEIGHT * 0.45 / 20;
       distance_in_20_seconds = velocity * 20 / 100;
       sprintf((char *)char_buffer, "AccSpeed= %d cm/s", velocity);
       lcd.DisplayStringAt(0, LINE(7), (uint8_t *)char_buffer, LEFT_MODE);
       sprintf((char *)char_buffer, "Distance= %d m", distance_in_20_seconds);
       lcd.DisplayStringAt(0, LINE(8), (uint8_t *)char_buffer, LEFT_MODE);
       printf("--------------------SUMMARY-------------------\n");
       printf("The average velocity is: %d cm/s, total distance: %d m\n\n", velocity,
distance_in_20_seconds);
   }

}

void calculateInstantVelocity(queue<float *>& data){
   uint8_t char_buffer[20];
   Last_Last_InstantVelocity = Last_InstantVelocity;
   Last_InstantVelocity = InstantVelocity;
   if( data.size() > 1)
   {
       float x_value = data.back()[0];
       x_value /=500;
```

```
        if (x_value < angularUpperLimit ){
            gap = timeCounter;
            timeCounter = 0;
        }

        timeCounter += 0.5;
        printf("timeCounter = %3.1f, gap = %3.1f\n", timeCounter, gap);

        if(gap == 0){
            gap = 0.5;
            InstantVelocity = USER_HEIGHT * 0.9 / gap + Last_InstantVelocity +
Last_Last_InstantVelocity;

        }
        else{
            InstantVelocity = USER_HEIGHT * 0.9 / gap + Last_InstantVelocity +
Last_Last_InstantVelocity;
        }

        if(timeCounter >= 4)
        {
            Last_Last_InstantVelocity = 0.0;
            Last_InstantVelocity = 0.0;
            InstantVelocity = 0.0;
        }

        InstantVelocity /= 3;
        sprintf((char *)char_buffer, "Speed= %5.2f cm/s", InstantVelocity);
        lcd.DisplayStringAt(0, LINE(6), (uint8_t *)char_buffer, LEFT_MODE);
        printf("The instant velocity is: %5.2f cm/s\n\n", InstantVelocity);
    }
}
```