Approach

In my approach to this project, I aimed for a user-friendly, versatile program. To start off, we need to perform some preliminary setup in order to have the most accurate reading possible. While this is going on, the red LED is lit up, to indicate that the user shouldn't do anything yet. The board should be lying motionless at this time because after setting up the gyroscope, we take 100 samples right away to calculate the average noise in each axis that the gyro picks up when not moving. After this is done, the red LED turns off and there is a 5-second delay to give the user time to affix the board to their leg. The board should be affixed such that it only moves along one axis, which is fairly easy to do. The green LED lights up when recording data for distance measurement and turns off when the recording is done. The code uses the x-axis for demonstration purposes, but this can easily be changed to the desired axis. 800 measurements are taken over 12 seconds. The average noise is subtracted from each one and it is converted to degrees per second (dps) using the following formula:

$$R_t = SC * (R_m - R_0)$$

where,

- R_t is the true angular rate given in dps
- R_m is the MEMS gyroscope measurement given in signed integer LSBs
- R_0 is the zero-rate level given in signed integer LSBs (the gyroscope output when no angular rate is applied)
- SC is the scale factor (or sensitivity) given in dps/LSB

SC is 0.0077 for our range of ±250dps. Then, every 10 samples are averaged into one data point. The resulting array of data is passed into the distance calculation function, along with the user's height in feet, inches. First, this function finds the number of strides taken by finding local peaks in the array. This is done by finding consecutive decreasing values where both values are still positive. Both values being positive ensures that we don't count the leg going back as a stride, because that's already included in the stride equations. At the end, the number of strides is multiplied by the average stride length in inches of the user's height[1]. I've adjusted this number based on my own observations. We also have to account for longer or shorter strides, which happen while walking slowly or running, respectively. Since we have 80 data points for 12 seconds, each data point is 150ms apart. We can multiply the difference in indices of where consecutive strides occur in the original array to find the length of that stride. Then we adjust the distance covered accordingly, subtracting one inches per 100ms over the average stride time[2] and adding two inches per 100ms under the average stride time. Once again, I've adjusted this average number based on my own observations. The total distance is calculated using the average string length and time. Finally, the distance covered in inches is returned.

[1] Stride Length
[2] Appendix 1

Problems

Unfortunately, I was unable to test and demo my program because of some issues I was having with my board. On the day before the due date, my board started giving me a hardfault exception and crashing when I tried to run any SPI instructions or printf instructions. It was working earlier that same day and I'm not sure what happened. The first error message is in this pastebin. I opened a fresh new project and started with very basic instructions to see what triggered the issue. I spent a few hours incrementally adding instructions and running them, until it was the same as my original one. I found that there was one function call on a certain line, which when included in the program, caused it to crash. This was the readAxes function call on line 202. I replaced the function call with the actual code in the function and then my program was able to run. Additionally, this function was called earlier in the program with no ill effects. Other weird things broke my program too, such as adding printf statements for certain variables and changing certain local variables to global. Also, printf didn't work anymore at all, even when it was called successfully, nothing appeared on my terminal. The most interesting part was that using the debugger, I was able to see when my code crashed, and it always crashed at spi.format() in the setupGyro() function, before it even got to the lines that were causing the problem.

~~In light of these late-occurring issues, the professor told me to submit without running any tests or having a demo. As such, I was unable to see if my distance calculation function works since I wasn't able to see any of the measurement data. I've given it my best try using just my theoretical idea of what the data would look like.~~

After being given some more time to work on it, as well as the information that my errors were likely caused by too much memory usage, I was able to get it working. I removed some unnecessary arrays to reduce memory usage and made some necessary adjustments to my distance calculation. I was able to get my program working and measuring distance traveled with an error of less than 10%.

Code

```
#include <mbed.h>

#define NOISE_SAMPLE_SIZE 100
#define NUM_SAMPLES 800

SPI spi(PF_9, PF_8, PF_7);
DigitalOut recordLight(LED1);
DigitalOut setupLight(LED2);
```

```
DigitalOut cs(PC_1);


//This function sets up the gyroscope to be read from using SPI
void setupGyro(){
  // Chip must be deselected
  cs = 1;

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

  // Select chip
  cs = 0;

  // Write to control register 1
  spi.write(0x20);

  // Mode = normal, enable x,y,z axes, default output data rate and
bandwidth
  spi.write(0x0f);

  cs = 1;
}


//This function reads all three axes from the gyroscope once.
//It takes arrays for the low and high bytes for each axis, as well as the
index to place the value into
void readAxes(uint16_t index, int16_t x_low[], int16_t x_hi[]){

  cs = 0;
  spi.write(0xa8);
  x_low[index] = spi.write(0x00);

  //The cs line must go high between each read, otherwise the same register
is read over and over again
  cs = 1;
  wait_us(10);

  cs = 0;
```

```
  spi.write(0xa9);
  x_hi[index] = spi.write(0x00);

  cs = 1;
  wait_us(10);

  cs = 0;
  /* Removed to reduce memory usage
  spi.write(0xaa);
  y_low[index] = spi.write(0x00);

  cs = 1;
  wait_us(10);

  cs = 0;

  spi.write(0xab);
  y_hi[index] = spi.write(0x00);

  cs = 1;
  wait_us(10);

  cs = 0;

  spi.write(0xac);
  z_low[index] = spi.write(0x00);

  cs = 1;
  wait_us(10);

  cs = 0;

  spi.write(0xad);
  z_hi[index] = spi.write(0x00);
  */
  cs = 1;
}

//This function calculates and returns distance covered in one direction in
inches.
//It takes as parameters an array of angular velocity values and its size,
and the person's height in feet, inches
//It uses the average stride length per height, as well as the average
```

```
stride time, and accounts for
//different stride times by adjusting the distance covered for each short
or long stride
//Average stride length per height formula from
https://www.verywellfit.com/set-pedometer-better-accuracy-3432895
//Average stride time is 0.89s-1.32s from appendix 1 of this book
https://www.sciencedirect.com/book/9780750688833/gait-analysis
//I adjusted the values for average stride length and time based on my own
observations
int calculateDistance(float values[], uint16_t size, uint8_t feet, uint8_t
inches){

  //Distance covered in inches
  int distanceCovered = 0;

  //We'll use this to find the number of strides and the amount of time
between them
  int strides[size];
  int numStrides = 0;

  for (int i = 3; i < size - 1; i++){

    //Compare consecutive values to see if we've hit a peak. A peak
(prev>next) signifies a stride
    float prev = values[i];
    float next = values[i + 1];

    //We don't want to count when the leg is moving back so we make sure
both values are positive
    if (prev > next && prev > 0){
      strides[numStrides] = i;
      numStrides++;

      //After encountering a stride, we won't encounter another one for at
least half a second, so to
      //ensure we don't measure noise as a stride, skip the next three
measurements
      i += 3;
    }
  }

  //Stride time in ms
  int strideTime;
```

```
  float extra = 0;

  //We sample every 15ms but average 10 samples into one data point,
meaning each data point is 150ms apart
  //We have 80 data points over a collection time of 12 seconds. We'll use
this information to tell if any
  //strides are shorter or longer than average, and adjust the distance
covered accordingly
  for (int i = 0; i < numStrides - 1; i++){

    //Stride time in ms, found by multiplying the difference in indices of
consecutive strides by 150
    strideTime = (strides[i + 1] - strides[i]) * 150;

    //I adjusted the values for average stride length and time based on my
own observations
    //Shorter stride time = shorter stride and less distance covered per
stride and vice versa
    if (strideTime < 1500){
      extra += 2 * ((1500 - strideTime) / 100); //Add 2 inches per 100ms
under the average range
    }
    else if (strideTime > 1500){
      extra -= 1 * ((strideTime - 1500) / 100); //Subtract 1 inch per 100ms
over the average range
    }
  }

  distanceCovered += extra;

  //Average stride length is 5 inches per foot of height plus 0.5 inches
per extra inch
  distanceCovered += numStrides * (7 * feet + 1 * inches);

  return distanceCovered;
}

int main(){

  //Indicator that setup is in progress and recording has not begun yet,
leave the board still to calibrate
  setupLight = 1;
```

```
  setupGyro();

  int16_t x_noise_low[NOISE_SAMPLE_SIZE];
  int16_t x_noise_hi[NOISE_SAMPLE_SIZE];
  /* Removed to reduce memory usage
  int y_noise_low[NOISE_SAMPLE_SIZE];
  int y_noise_hi[NOISE_SAMPLE_SIZE];
  int z_noise_low[NOISE_SAMPLE_SIZE];
  int z_noise_hi[NOISE_SAMPLE_SIZE];
*/
  for (int i = 0; i < NOISE_SAMPLE_SIZE; i++){
    //These samples will be used to approximate the average noise the gyro
detects when still
    readAxes(i, x_noise_low, x_noise_hi);
    //The gyro updates every 10ms so we'll sample every 15ms.
    wait_us(15000);
  }

  int16_t xNoise[NOISE_SAMPLE_SIZE];
  //int16_t yNoise[NOISE_SAMPLE_SIZE];
  //int16_t zNoise[NOISE_SAMPLE_SIZE];

  int16_t xAvgNoise = 0;
  //int16_t yAvgNoise = 0;
  //int16_t zAvgNoise = 0;

  for (int i = 0; i < NOISE_SAMPLE_SIZE; i++){

    //Convert the two 8 bit register values into one 16 bit number
    xNoise[i] = (x_noise_hi[i] << 8) + x_noise_low[i];
    xAvgNoise += xNoise[i];

    /* Removed to reduce memory usage
    yNoise[i] = (y_noise_hi[i] << 8) + y_noise_low[i];
    yAvgNoise += yNoise[i];
    zNoise[i] = (z_noise_hi[i] << 8) + z_noise_low[i];
    zAvgNoise += zNoise[i];
  */}

  xAvgNoise = xAvgNoise / NOISE_SAMPLE_SIZE;
  //yAvgNoise = yAvgNoise / NOISE_SAMPLE_SIZE;
  //zAvgNoise = zAvgNoise / NOISE_SAMPLE_SIZE;
```

```
  int16_t x_low[NUM_SAMPLES];
  int16_t x_hi[NUM_SAMPLES];
  /*Removed to reduce memory usage
  int y_low[NUM_SAMPLES];
  int y_hi[NUM_SAMPLES];
  int z_low[NUM_SAMPLES];
  int z_hi[NUM_SAMPLES];
*/

  //Setup is done, recording will start in 5 seconds
  setupLight = 0;

  while (1){

    wait_us(5000000);

    //Recording begins
    recordLight = 1;

    for (int i = 0; i < NUM_SAMPLES; i++){
      readAxes(i, x_low, x_hi);
      //The gyro updates every 10ms so we'll sample 750 times, every 15ms.
      wait_us(15000);
    }

    //Recording is done
    recordLight = 0;

    int16_t xValsRaw[NUM_SAMPLES];
    //float yValsRaw[NUM_SAMPLES];
    //float zValsRaw[NUM_SAMPLES];

    for (int i = 0; i < NUM_SAMPLES; i++){

      //Convert to 16 bit number and clean up the raw data by subtracting
the average noise
      xValsRaw[i] = (x_hi[i] << 8) + x_low[i];
      xValsRaw[i] = xValsRaw[i] - xAvgNoise;
/*
      yValsRaw[i] = (y_hi[i] << 8) + y_low[i];
      yValsRaw[i] = yValsRaw[i] - yAvgNoise;
      yValsRaw[i] = yValsRaw[i] * 0.00875;
```

```
      zValsRaw[i] = (z_hi[i] << 8) + z_low[i];
      zValsRaw[i] = zValsRaw[i] - zAvgNoise;
      zValsRaw[i] = zValsRaw[i] * 0.00875;
    */}

    float xVals[NUM_SAMPLES / 10];
    //float yVals[NUM_SAMPLES / 10];
    //float zVals[NUM_SAMPLES / 10];

    int i = 0;
    while(i<NUM_SAMPLES){

      //Take the average of every 10 samples and convert to dps
      float xVal = 0;
      for (int j = 0; j < 9; j++){
        xVal += (float)xValsRaw[i + j];
      }

      xVal = xVal * 0.0077;
      xVal = xVal / 10;
      xVals[i / 10] = xVal;
/* Removed to reduce memory usage
      float yVal = 0;
      for (int j = 0; j < 9; j++){
        yVal += yValsRaw[i + j];
      }

      yVal = yVal / 10;
      yVals[i / 10] = yVal;

      float zVal = 0;
      for (int j = 0; j < 9; j++){
        zVal += zValsRaw[i + j];
      }

      zVal = zVal / 10;
      zVals[i / 10] = zVal;
*/
      i = i + 10;
    }

    //This is my height, parameters can be customized
```

```
    uint8_t feet = 6;
    uint8_t inches = 0;

    //Calculate the distance traveled in inches for a person of given
height
    //Using only the x axis values because of the way I oriented my board
on my leg
    int distance = calculateDistance(xVals, (NUM_SAMPLES / 10), feet,
inches);

  }
}
```