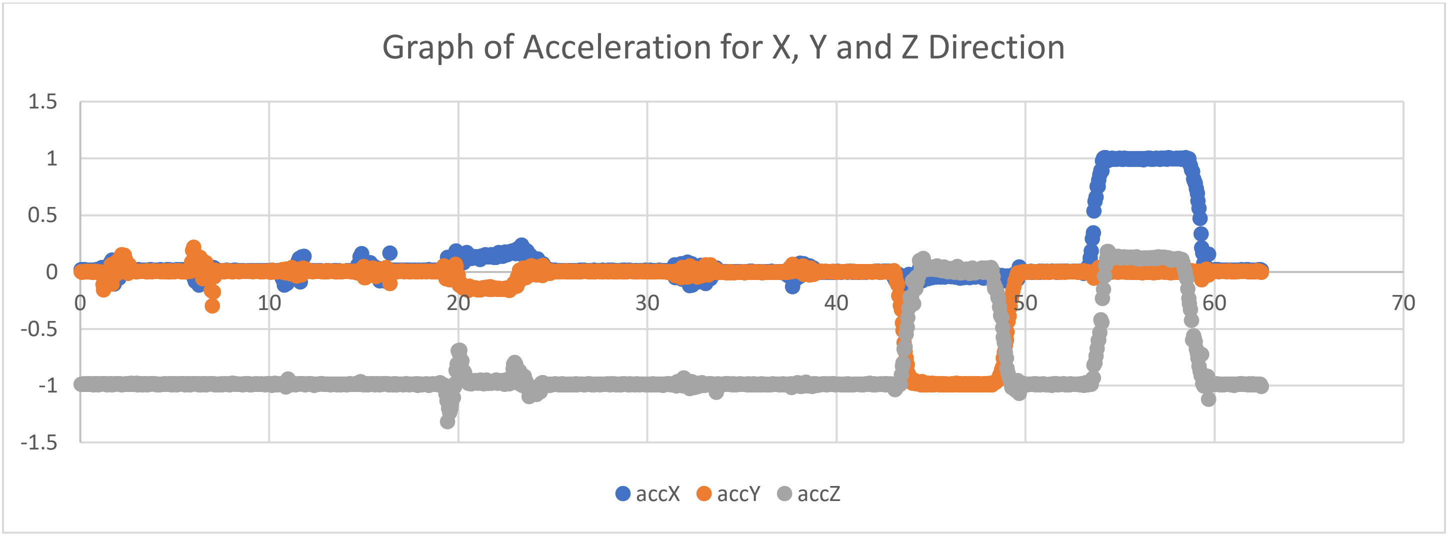
LAB 6 REPORT – Aaron Bruner

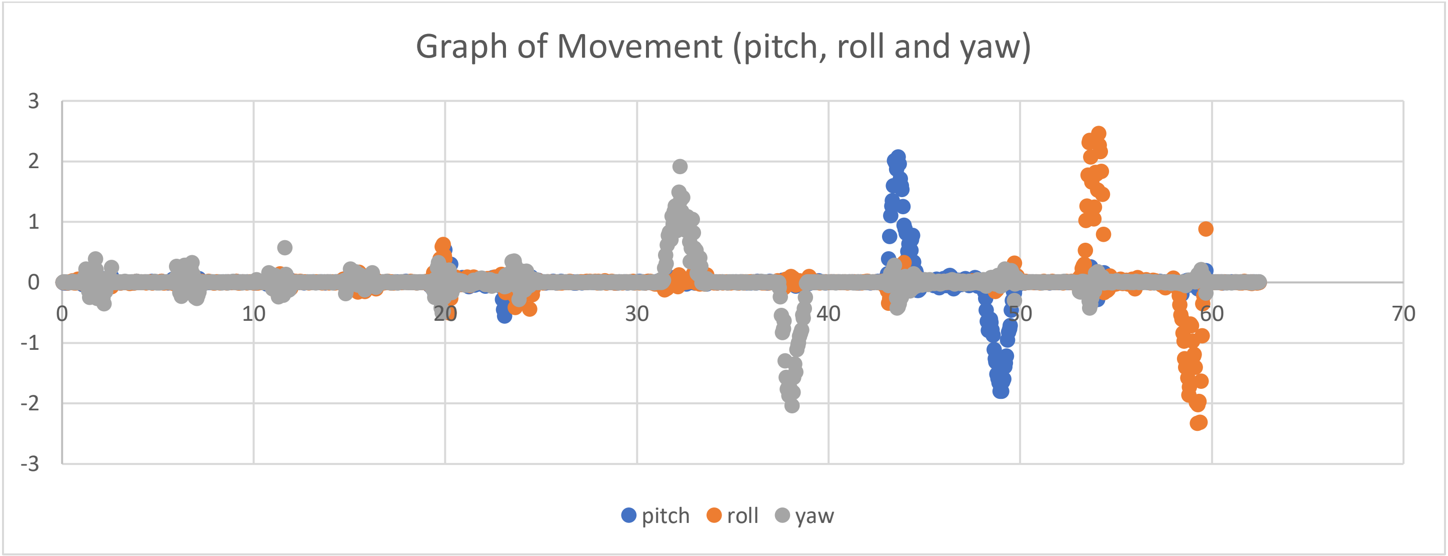
The purpose of this lab is to calculate motion using accelerometers and gyroscope data from an iPhone. Provided data is represented in 7 columns: time, x_acc, y_acc, z_acc, pitch, roll and yaw. We are looking to detect times of motion using our code. Below are the times where we detected motion:

```
 *----= Movement # [  1 ] =---------= Total Linear Distance =-------[ 0.70 - 2.65 ]------*
|     X     =  0.00896 (m)       Y    =  0.00391 (m)      Z    = -0.48490 (m)            |
|     Pitch = -0.01646 (rad)     Roll = -0.02574 (rad)    Yaw  = -0.00332 (rad)          |
*-Legend: XYZ are measured in meters (m) and Pitch,Yaw,Roll are measured in radians (rad)-*

*----= Movement # [  2 ] =---------= Total Linear Distance =-------[ 5.45 - 7.15 ]------*
|     X     =  0.01176 (m)       Y    =  0.00895 (m)      Z    = -0.90921 (m)            |
|     Pitch = -0.01487 (rad)     Roll = -0.00325 (rad)    Yaw  = -0.02546 (rad)          |
*-Legend: XYZ are measured in meters (m) and Pitch,Yaw,Roll are measured in radians (rad)-*

*----= Movement # [  3 ] =---------= Total Linear Distance =-------[ 10.45 - 11.75 ]------*
|     X     =  0.00933 (m)       Y    =  0.01100 (m)      Z    = -1.23598 (m)            |
|     Pitch = -0.01783 (rad)     Roll = -0.01174 (rad)    Yaw  = -0.03646 (rad)          |
*-Legend: XYZ are measured in meters (m) and Pitch,Yaw,Roll are measured in radians (rad)-*

*----= Movement # [  4 ] =---------= Total Linear Distance =-------[ 14.30 - 15.85 ]------*
|     X     =  0.01991 (m)       Y    =  0.01011 (m)      Z    = -1.62355 (m)            |
|     Pitch = -0.02400 (rad)     Roll =  0.02260 (rad)    Yaw  = -0.02759 (rad)          |
*-Legend: XYZ are measured in meters (m) and Pitch,Yaw,Roll are measured in radians (rad)-*

*----= Movement # [  5 ] =---------= Total Linear Distance =-------[ 18.90 - 20.40 ]------*
|     X     =  0.04307 (m)       Y    = -0.00165 (m)      Z    = -1.99690 (m)            |
|     Pitch =  0.13824 (rad)     Roll = -0.04863 (rad)    Yaw  =  0.07314 (rad)          |
*-Legend: XYZ are measured in meters (m) and Pitch,Yaw,Roll are measured in radians (rad)-*

*----= Movement # [  6 ] =---------= Total Linear Distance =-------[ 22.45 - 24.60 ]------*
|     X     =  0.11262 (m)       Y    = -0.02586 (m)      Z    = -2.52217 (m)            |
|     Pitch = -0.03486 (rad)     Roll =  0.00839 (rad)    Yaw  = -0.09359 (rad)          |
*-Legend: XYZ are measured in meters (m) and Pitch,Yaw,Roll are measured in radians (rad)-*

*----= Movement # [  7 ] =---------= Total Linear Distance =-------[ 30.90 - 33.40 ]------*
|     X     =  0.10689 (m)       Y    = -0.02730 (m)      Z    = -3.14102 (m)            |
|     Pitch =  0.01145 (rad)     Roll =  1.57449 (rad)    Yaw  = -0.09868 (rad)          |
*-Legend: XYZ are measured in meters (m) and Pitch,Yaw,Roll are measured in radians (rad)-*

*----= Movement # [  8 ] =---------= Total Linear Distance =-------[ 36.95 - 38.85 ]------*
|     X     =  0.10667 (m)       Y    = -0.02438 (m)      Z    = -3.61654 (m)            |
|     Pitch =  0.01028 (rad)     Roll =  0.03359 (rad)    Yaw  = -0.10964 (rad)          |
*-Legend: XYZ are measured in meters (m) and Pitch,Yaw,Roll are measured in radians (rad)-*

*----= Movement # [  9 ] =---------= Total Linear Distance =-------[ 42.60 - 44.50 ]------*
|     X     =  0.08443 (m)       Y    = -0.26451 (m)      Z    = -3.90930 (m)            |
|     Pitch =  1.61275 (rad)     Roll = -0.03473 (rad)    Yaw  = -0.20440 (rad)          |
*-Legend: XYZ are measured in meters (m) and Pitch,Yaw,Roll are measured in radians (rad)-*
```

```
*----= Movement # [ 10 ] =---------= Total Linear Distance =-------[ 47.80 - 49.75 ]------*
|    X    = 0.06450 (m)      Y    = -0.57744 (m)     Z    = -4.15206 (m)            |
|    Pitch =  0.03485 (rad)     Roll =  0.01801 (rad)    Yaw  = -0.15782 (rad)          |
*-Legend: XYZ are measured in meters (m) and Pitch,Yaw,Roll are measured in radians (rad)-*

*----= Movement # [ 11 ] =---------= Total Linear Distance =-------[ 52.60 - 54.60 ]------*
|    X    = 0.29529 (m)      Y    = -0.57716 (m)     Z    = -4.46685 (m)            |
|    Pitch =  0.03711 (rad)     Roll = -0.05617 (rad)    Yaw  =  1.51334 (rad)          |
*-Legend: XYZ are measured in meters (m) and Pitch,Yaw,Roll are measured in radians (rad)-*

*----= Movement # [ 12 ] =---------= Total Linear Distance =-------[ 57.75 - 59.75 ]------*
|    X    = 0.65226 (m)      Y    = -0.57814 (m)     Z    = -4.66209 (m)            |
|    Pitch = -0.00151 (rad)     Roll = -0.01146 (rad)    Yaw  = -0.09057 (rad)          |
*-Legend: XYZ are measured in meters (m) and Pitch,Yaw,Roll are measured in radians (rad)-*
```

We detected a total of 12 motions across the data set. The value in the top right of the movement is the time span of the movement. Graphing the initial data give us the following data outputs:





We can see the 12 movements in the combined graphs which represent our time values from our movements.

Source Code:
```
/* File  : lab6.c
   Author: Aaron Bruner
   Class : ECE - 4310 : Introduction to Computer Vision
   Term  : Fall 2022

   Description: The purpose of this assignment is to calculate motion using accelerometers and
gyroscopes.

   Required Files:
    * acc_gyro.txt

   Bugs:
    * Currently none
*/

#pragma region definitions

#define TRUE 1
#define FALSE 0
#define START 0
#define END 1
#define CONST_SEC_PER_ROW 0.05
#define CONST_SAMPLE_RATE 20 // Hz
#define CONST_GRAVITY 9.81  // m/s^2
#define WINDOW_SIZE 12
#define MIN_WINDOW 1
#define MAX_WINDOW 6
// Values lower than 0 means less detection
#define ACCELEROMETERS_ACCX_THRESHOLD 0.008
#define ACCELEROMETERS_ACCY_THRESHOLD 0.02
#define ACCELEROMETERS_ACCZ_THRESHOLD 0.008
#define GYROSCOPE_ROLL_THRESHOLD 0.003
#define GYROSCOPE_PITCH_THRESHOLD 0.04
#define GYROSCOPE_YAW_THRESHOLD 0.003

#define SQR(x) ((x)*(x))

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct motionData {
    double time;
    double accX;
    double accY;
    double accZ;
    double pitch;
    double roll;
    double yaw;
};

struct motionData* readCSV(char* motionDataDir, int* fileRows);
void integrate(struct motionData* data, int* index, double* outputArr);
void segmentData(int* isMoving, struct motionData* data, int index, int rows);

char* defaultMotionDataDir = "acc_gyro.txt";
char* resultDir = "movements.txt";
int isMoving = FALSE;

#pragma endregion
```

```
int main(int argc, char* argv[])
{
    FILE* resultFPT;
    struct motionData* motionData, *rover;
    int fileRows, isMoving = FALSE, movementCounter = 0, index[2];
    double distance[MAX_WINDOW], time[2];
    time[START] = time[END] = index[START] = index[END] = 0.0;

    // Input is only ./lab6
    argc == 1 ? (motionData = readCSV(defaultMotionDataDir, &fileRows)) : (fprintf(stdout, 
"Incorrect number of arguments...\nUsage: ./lab6\n"), exit(0));

    // Clear results file
    resultFPT = fopen(resultDir, "w"); resultFPT == NULL ? fprintf(stdout, "Failed to open 
%s...\n", resultDir), exit(0) : fclose(resultFPT);
    rover = motionData; // Point the rover at first element of motion data

    // For each data point
    for (int a = 0; a < fileRows; a++, rover++)
    {
        segmentData(&isMoving, motionData, a, fileRows);

        isMoving ? (time[START] == 0.0 ? time[START] = rover->time, index[START] = a : 0.0) : 
((time[START] != 0.0 && time[END] == 0.0) ? time[END] = rover->time, index[END] = a : 0.0);

        // If we've detected a start movement AND end movement then output
        if (time[START] != 0.0 && time[END] != 0.0)
        {
            movementCounter += 1;
            integrate(motionData, index, distance);

            resultFPT = fopen(resultDir, "a");
            fprintf(resultFPT, "*----= Movement # [ %2d ] =---------= Total Linear Distance =-----
--[ %0.2f - %0.2f ]------*\n", movementCounter, time[START], time[END]);
            fprintf(resultFPT, "|     X     = % 01.5f (m)       Y    = % 01.5f (m)      Z    = % 
01.5f (m)            |\n", distance[0], distance[1], distance[2]);
            fprintf(resultFPT, "|     Pitch = % 01.5f (rad)     Roll = % 01.5f (rad)    Yaw  = % 
01.5f (rad)          |\n", distance[3], distance[4], distance[5]);
            fprintf(resultFPT, "*-Legend: XYZ are measured in meters (m) and Pitch,Yaw,Roll are 
measured in radians (rad)-*\n\n");
            fclose(resultFPT);

            // Reset time variables after exporting a detected iPhone movement
            time[START] = time[END] = 0.0;
        }
    }

    return 0;
}

/// <summary>
/// Read in motion data values from CSV file. Except the delimiter is a space
///
/// Need to read the header row first.
/// </summary>
/// <param name="motionDataDir">File directory for CSV file</param>
/// <returns>An array of structures which contain the time, accX, accY, accZ, pitch, roll and 
yaw.</returns>
struct motionData* readCSV(char* motionDataDir, int *fileRows)
```

```c
{
    char header[0xff];
    size_t headerSize = 0xff;
    double time, accX, accY, accZ, pitch, roll, yaw;
    int i = 0;
    (*fileRows) = 0;
    struct motionData* data;
    FILE* FPT;

    // Open the file for reading
    FPT = fopen(motionDataDir, "r");
    FPT == NULL ? printf("Failed to open %s.\n", motionDataDir), exit(0) : FALSE;

    // Read in header data before reading rows
    fgets(header, headerSize, FPT);

    // Determine the number of rows in the file
    while ((i = fscanf(FPT, "%lf %lf %lf %lf %lf %lf %lf\n", &time, &accX, &accY, &accZ, &pitch,
&roll, &yaw)) && !feof(FPT))
        if (i == 7) (*fileRows) += 1;
    // Number of rows + 1 since last row isn't counted
    (*fileRows)++;

    // Allocate space for array of structures
    data = calloc((*fileRows), sizeof(struct motionData));

    // Return to the beginning of the file
    rewind(FPT);
    // Read in header data before reading rows
    fgets(header, headerSize, FPT);

    // Scan in all columns and rows
    for (i = 0; i <= (*fileRows) && !feof(FPT); i++)
        fscanf(FPT, "%lf %lf %lf %lf %lf %lf %lf\n", &data[i].time, &data[i].accX, &data[i].accY,
&data[i].accZ, &data[i].pitch, &data[i].roll, &data[i].yaw);

    fclose(FPT);

    return data;
}

/// <summary>
/// To segment the data, you should use a window and calculate the variance of the data
/// along all 6 axes. When those variances are all less than some threshold, the iPhone is at
/// rest, and when any of the 6 variances is greater than the threshold, the iPhone is in
/// motion.
///
/// https://www.geeksforgeeks.org/program-for-variance-and-standard-deviation-of-an-array/
/// </summary>
/// <param name="isMoving">Boolean value to determine if the iPhone is moving</param>
/// <param name="data">Data of iPhone moving</param>
/// <param name="index">Index value we're currently at in main process</param>
/// <param name="rows">Number of rows in data array</param>
void segmentData(int *isMoving, struct motionData* data, int index, int rows)
{
    int i;
    double meanx, meany, meanz, meanP, meanY, meanR;
    double varx, vary, varz, varP, varY, varR;
    meanx = meany = meanz = meanP = meanY = meanR = 0.0;
    varx = 0, vary = 0, varz = 0, varP = 0, varY = 0, varR = 0;
    // Calculate mean values
    int limit = index + WINDOW_SIZE > rows ? rows : index + WINDOW_SIZE;
    for (i = index; i < limit; i++)
    {
        meanx += data[i].accX;
        meany += data[i].accY;
        meanz += data[i].accZ;
        meanP += data[i].pitch;
        meanY += data[i].yaw;
        meanR += data[i].roll;
    }
    meanx /= (WINDOW_SIZE + 1);
    meany /= (WINDOW_SIZE + 1);
    meanz /= (WINDOW_SIZE + 1);
    meanP /= (WINDOW_SIZE + 1);
    meanY /= (WINDOW_SIZE + 1);
    meanR /= (WINDOW_SIZE + 1);
    // Calculate the variance using mean values
    for (i = index; i < limit; i++)
    {
        varx += SQR(data[i].accX - meanx);
        vary += SQR(data[i].accY - meany);
        varz += SQR(data[i].accZ - meanz);
        varP += SQR(data[i].pitch - meanP);
        varR += SQR(data[i].yaw - meanY);
        varY += SQR(data[i].roll - meanR);
    }
    varx /= (WINDOW_SIZE + 1);
    vary /= (WINDOW_SIZE + 1);
    varz /= (WINDOW_SIZE + 1);
    varP /= (WINDOW_SIZE + 1);
    varR /= (WINDOW_SIZE + 1);
    varY /= (WINDOW_SIZE + 1);

    // Check thresholds to determine if moving
    *(isMoving) = (varx > ACCELEROMETERS_ACCX_THRESHOLD ||
                   vary > ACCELEROMETERS_ACCY_THRESHOLD ||
                   varz > ACCELEROMETERS_ACCZ_THRESHOLD ||
                   varP > GYROSCOPE_PITCH_THRESHOLD     ||
                   varR > GYROSCOPE_ROLL_THRESHOLD      ||
                   varY > GYROSCOPE_YAW_THRESHOLD)         ? TRUE : FALSE;
}

/// <summary>
/// To calculate the motion, the data must be integrated (gyroscopes) or double integrated
/// (accelerometers).For the gyroscopes, this can be accomplished by multiplying the data
/// by the time between samples.For the accelerometers, you must calculate three values :
/// the velocity at the end of a sampling period, the average velocity during the sampling
/// period, and then the distance traveled during the sampling period.Assume the initial
/// velocity is zero.The velocity at the time of a data sample is equal to the velocity at the
/// time of the previous sample plus the acceleration reading multiplied by the time between
/// samples.This assumes constant acceleration, which is okay because the sampling
/// interval is small.The average velocity during the sampling period is the average of the
/// initial and final velocities.The distance traveled during the sampling period is the
/// average velocity multiplied by the time between samples.
/// </summary>
/// <param name="data">Data of iPhone moving</param>
/// <param name="index">Index value we're currently at in main process</param>
/// <param name="outputArr"></param>
void integrate(struct motionData *data, int *index, double *outputArr)
{
    double velo = 0.0, prevVelo = 0.0;

    // 0, 1 and 2 are single integrations for gyroscopes
    for (int i = index[START]; i <= index[END]; i++, prevVelo = velo)
    {
        outputArr[0] += (((data[i].accX * CONST_GRAVITY * CONST_SEC_PER_ROW) + prevVelo) / 2) *
CONST_SEC_PER_ROW;
        outputArr[1] += (((data[i].accY * CONST_GRAVITY * CONST_SEC_PER_ROW) + prevVelo) / 2) *
CONST_SEC_PER_ROW;
        outputArr[2] += (((data[i].accZ * CONST_GRAVITY * CONST_SEC_PER_ROW) + prevVelo) / 2) *
CONST_SEC_PER_ROW;
    }
    // 3, 4 and 5 are double integrals for accelerometers
    for (int i = index[START]; i <= index[END]; i++)
    {
        outputArr[3] += data[i].pitch * CONST_SEC_PER_ROW;
        outputArr[4] += data[i].yaw   * CONST_SEC_PER_ROW;
        outputArr[5] += data[i].roll  * CONST_SEC_PER_ROW;
    }
}
```