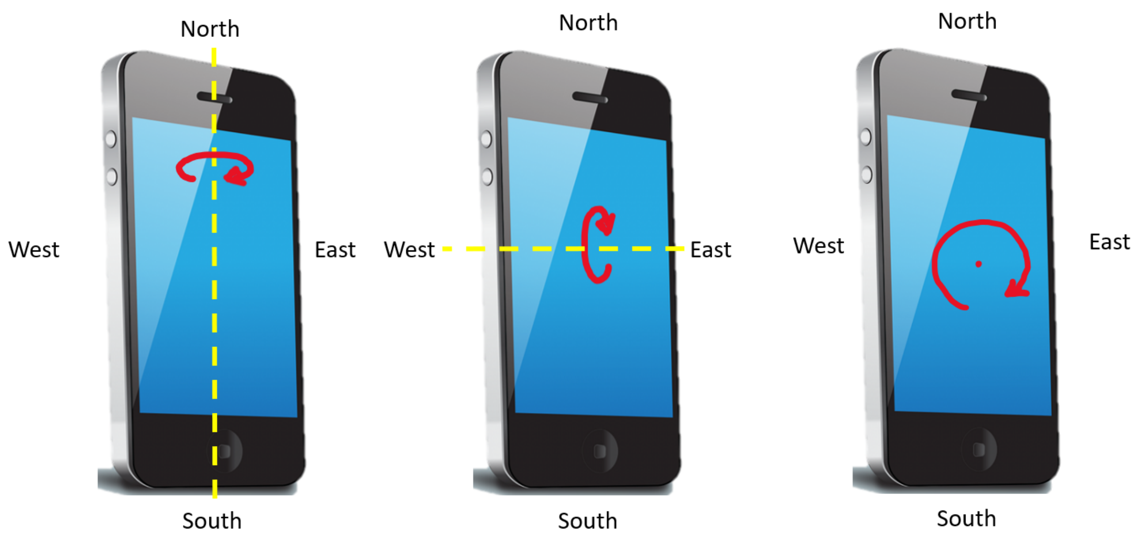
Programming the Raspberry Pi Pico and Writing Sensor Data to the SD Card

Due Jul 1 by 11:59pm **Points** 100 **Submitting** a text entry box or a file upload **File Types** pdf

The objective of this assignment is to program the Raspberry Pi Pico to write data to an SD card. The format of the data on the SD card should match the format of the data collected using the Multi Record feature on the app "**Physics Toolbox Sensor Suite**". I-phone users can use the "**phyphox**" app. In this assignment, you will tape or attach your flight controller to your cell phone to collect data. The data from your flight controller will need to match the data from the phone.



Writing Sensor Data to the SD Card

Connect your Raspberry Pi Pico, your 10 DOF IMU sensor, your GPS module, and your SD Card Module to your PCB (Printed Circuit Board). Program the Raspberry Pi Pico to create a .csv file and write the sensor data to the SD card. The .csv file should have the following headers:

time	gFx	gFy	gFz	wx	wy	wz	p	Bx	By	Bz	Azimuth	Pitch	Roll	Latitude	Longitude	Speed (m/s)

Program the Raspberry Pi Pico to write the time to the .csv file under the "time" column in units of seconds. Program it to write the accelerometer data under the gFx, gFy, and gFz columns in units of g-force. Write the gyrometer data under the wx, wy, and wz columns in units of rad/s. Write the pressure under the p column in units of mbar (hPa). Write the geomagnetic components under the Bx, By, and Bz columns in units of microteslas. Write the roll, pitch, and yaw orientation data under the Roll, Pitch, and Azimuth columns. Write latitude and longitude under the Latitude and Longitude columns in units of degrees. Write speed under the Speed (m/s) column in units of m/s. For example, the .csv file should look something like this:

time	gFx	gFy	gFz	wx	wy	wz	p	Bx	By	Bz	Azimuth	Pitch	Roll	Latitude	Longitude
0.00	0.0049	-0.0413	0.988	0.0037	0.0183	0.0031	866.8751	-16.26	-10.74	-44.4	337.0061	-32.7291	38.9241	41.69538	-112.296
0.001415	0.0049	-0.0413	0.988	0.0037	0.0183	0.0031	866.8751	-16.26	-10.74	-44.4	337.0061	-32.7291	38.9241	41.69538	-112.296
0.011223	0.0049	-0.0413	0.988	0.0202	0.0379	0.0024	866.8275	-16.26	-10.74	-44.4	341.9433	-32.2245	41.0611	41.69538	-112.296
0.014867	-0.0205	-0.0169	0.9657	0.0202	0.0379	0.0024	866.8275	-16.26	-10.74	-44.4	341.9433	-32.2245	41.0611	41.69549	-112.296
0.019046	-0.0205	-0.0169	0.9657	0.0202	0.0379	0.0024	866.8275	-16.26	-10.74	-44.4	341.9433	-32.2245	41.0611	41.69549	-112.296

0.020739	-0.0205	-0.0169	0.9657	0.0202	0.0379	0.0024	866.8275	-16.26	-10.74	-44.4	341.9433	-32.2245	41.0611	41.69549	-112.296
0.022488	-0.0205	-0.0169	0.9657	0.0202	0.0379	0.0024	866.8275	-16.26	-10.74	-44.4	342.3604	-31.7293	44.1415	41.69549	-112.296

Some Arduino Code To Get You Started

```
#include <SPI.h>
#include <SD.h>
#include <MPU9250_WE.h>                              //Include the MPU9250
library for Accel, Gyro, and Magnetometer
#include <BMP280_DEV.h>                              //Include the BMP280_DEV.h
library for altitude, pressure, and temperature
#include <function_CppKokSchonQuaternionEstimator.h> //Include the Kok Schon
Quaternion Estimator
#include <Quaternion2Euler.h>                        //Include the library to
convert quaternion to euler orientation
#include <SoftwareSerial.h>                          //Include the SofwareSerial
library for GPS communication
#include <TinyGPS.h>                                 //Include the GPS library

//name the file for the SD card
const String filename = "RollPitchYawData.csv";
//Say whether to collect GPS data or not
bool useGPS = true;

//######################Variables for GPS###########################//
TinyGPS gps;
const int rxPin = 1;
const int txPin = 0;
SoftwareSerial ss(rxPin, txPin);
static void readGPS(unsigned long ms);
float GPSlat = 0.0;  //declare GPS latitude and longitude variables
float GPSlon = 0.0;
float GPSspeed = 0.0;  //declare GPS speed (m/s)
//###################End ofVariables for GPS#######################//

//#####################Variables for SD Card Reader################//
#if !defined(ARDUINO_ARCH_RP2040)
#error For RP2040 only
#endif
#if defined(ARDUINO_ARCH_MBED)
#define PIN_SD_MOSI PIN_SPI_MOSI
#define PIN_SD_MISO PIN_SPI_MISO
#define PIN_SD_SCK PIN_SPI_SCK
#define PIN_SD_SS PIN_SPI_SS
#else
#define PIN_SD_MOSI PIN_SPI0_MOSI
#define PIN_SD_MISO PIN_SPI0_MISO
#define PIN_SD_SCK PIN_SPI0_SCK
#define PIN_SD_SS PIN_SPI0_SS
#endif
```

```
#define _RP2040_SD_LOGLEVEL_ 0
//################End of Variables for SD Card Reader##############//

//##############Variables for 10 DOF MPU9250 Sensor###############//
#define MPU9250_ADDR 0x68  //define the address for the MPU 9250
//Create the MPU9250 object and name it myMPU9250
MPU9250_WE myMPU9250 = MPU9250_WE(MPU9250_ADDR);
//Create the BMP280_DEV object, name it bmp280. I2C address is 0x77
BMP280_DEV bmp280;
//declare temperature, pressure, and altitude
float temperature, pressure, altitude;
//###########End of Variables for 10 DOF MPU9250 Sensor############//

//Additional variables
bool USBconnected = true;
double quaternion[4] = { 1.0, 0.0, 0.0, 0.0 };
double t_last = 0.0;

void setup() {
  // Start serial communication
  Serial.begin(9600);
  delay(3000);
  if (!Serial) {
    USBconnected = false;
  }
  if (useGPS) {
    //Start communication with the GPS module
    ss.begin(9600);
    unsigned long GPSdelay = 1000;  //Time delay in ms
    readGPS(GPSdelay);
  }
  Wire.begin();  //Begin I2C
  delay(2000);
  if (!myMPU9250.init()) {  //Start the MPU, if it fails, report an error
    if (USBconnected) {
      Serial.println("MPU9250 does not respond");
    }
  }
  if (!myMPU9250.initMagnetometer()) {  //Start the magnetometer, if it fails,
report
    if (USBconnected) {
      Serial.println("Magnetometer does not respond");
    }
  }
  bmp280.begin();  // Default initialization, place the BMP280 into SLEEP_MODE
```

```
  //bmp280.setPresOversampling(OVERSAMPLING_X4); // Set the pressure oversampling
to X4
  //bmp280.setTempOversampling(OVERSAMPLING_X1); // Set the temperature
oversampling to X1
  //bmp280.setIIRFilter(IIR_FILTER_4); // Set the IIR filter to setting 4
  bmp280.setTimeStandby(TIME_STANDBY_2000MS);  // Set the standby time to 2
seconds
  bmp280.startNormalConversion();              // Start BMP280 continuous
conversion in NORMAL_MODE

  if (USBconnected) {
    Serial.println("Position your MPU9250 flat and don't move it -
calibrating...");
  }
  delay(1000);
  myMPU9250.autoOffsets();  //Callibrate the accelerometer and gyro offsets
  if (USBconnected) {
    Serial.println("Done!");
  }

#if defined(ARDUINO_ARCH_MBED)
  if (USBconnected) {
    Serial.print("Starting SD Card ReadWrite on MBED ");
  }
#else
  if (USBconnected) {
    Serial.print("Starting SD Card ReadWrite on ");
  }
#endif

  if (USBconnected) {
    Serial.println(BOARD_NAME);
    Serial.print("Initializing SD card with SS = ");
    Serial.println(PIN_SD_SS);
    Serial.print("SCK = ");
    Serial.println(PIN_SD_SCK);
    Serial.print("MOSI = ");
    Serial.println(PIN_SD_MOSI);
    Serial.print("MISO = ");
    Serial.println(PIN_SD_MISO);
  }

  if (!SD.begin(PIN_SD_SS)) {
    if (USBconnected) {
      Serial.println("Initialization failed!");
```

```
    }
  }
  if (USBconnected) {
    Serial.println("Initialization done.");
  }

  //create the SD card file and open it for writing
  File dataFile = SD.open(filename, FILE_WRITE);

  //If it opened correctly, write the header to it
  if (dataFile) {
    String headerString = "time,";
    headerString +=
"gFx,gFy,gFz,wx,wy,wz,p,Bx,By,Bz,Azimuth,Pitch,Roll,Latitude,Longitude,Speed
(m/s)";
    dataFile.println(headerString);
    dataFile.close();
  } else {
    if (USBconnected) {
      Serial.println("Initialization failed to open the file");
    }
  }
  t_last = 0.0;
}

void loop() {
  //Get the values from the accelerometer
  xyzFloat accel = myMPU9250.getGValues();
  //Get the values from the Gyrometer
  xyzFloat gyro = myMPU9250.getGyrValues();
  //Get the values from the Magnetometer
  xyzFloat Mag = myMPU9250.getMagValues();

  //put the data into arrays
  double gyrometer[3];
  double gravity[3];
  double magnetometer[3];
  gyrometer[0] = gyro.x * 3.14159 / 180.0;
  gyrometer[1] = -gyro.y * 3.14159 / 180.0;
  gyrometer[2] = -gyro.z * 3.14159 / 180.0;

  gravity[0] = -accel.x;
  gravity[1] = accel.y;
  gravity[2] = accel.z;
```

```
    magnetometer[0] = (Mag.y - 40.3058);
    magnetometer[1] = -(Mag.x + 11.444);
    magnetometer[2] = (Mag.z + 33.2877);

    //get the time
    double t = (double)millis() / 1000.0;
    double dt = t - t_last;
    t_last = t;

    //Get the quaternion orientation
    function_KokSchonQuaternionEstimator(
      quaternion,
      quaternion,
      gyrometer,
      gravity,
      magnetometer,
      dt,
      67.0,
      1.0,
      0.3);

    double roll, pitch, yaw;
    Quaternion2Euler(&roll, &pitch, &yaw, quaternion);

    //Get the bmp280 measurements
    //Temperature in Celsius, pressure in hectopascals, altitude in meters
    bmp280.getMeasurements(temperature, pressure, altitude);

    //Get GPS measurements
    if (useGPS) {
      unsigned long GPSdelay = 1;  //Time delay in ms
      readGPS(GPSdelay);
      unsigned long age;
      //Read the latitude and longitude
      gps.f_get_position(&GPSlat, &GPSlon, &age);
    }

    String dataString = "";
    dataString += t;  //time
    dataString += ",";
    dataString += gravity[0];  //gFx
    dataString += ",";
    dataString += gravity[1];  //gFy
    dataString += ",";
    dataString += gravity[2];  //gFz
```

```
  dataString += ",";
  dataString += gyrometer[0];  //wx
  dataString += ",";
  dataString += gyrometer[1];  //wy
  dataString += ",";
  dataString += gyrometer[2];  //wz
  dataString += ",";
  dataString += pressure;  //p
  dataString += ",";
  dataString += magnetometer[0];  //Bx
  dataString += ",";
  dataString += magnetometer[1];  //By
  dataString += ",";
  dataString += magnetometer[2];  //Bz
  dataString += ",";
  dataString += yaw;  //Azimuth
  dataString += ",";
  dataString += pitch;  //Pitch
  dataString += ",";
  dataString += roll;  //Roll
  dataString += ",";
  if (useGPS) {
    char charArray[12];
    dtostrf(GPSlat, 1, 6, charArray);  //Latitude
    dataString += charArray;
    dataString += ",";
    dtostrf(GPSlon, 1, 6, charArray);  //Longitude
    dataString += charArray;
    dataString += ",";
    dataString += GPSspeed;  //Speed (m/s)
  } else {
    dataString += "0";  //Latitude
    dataString += ",";
    dataString += "0";  //Longitude
    dataString += ",";
    dataString += "0";  //Speed (m/s)
  }

  writeToSDcard(dataString);
}

void writeToSDcard(String dataString) {
  //create the SD card file and open it for writing
  File dataFile = SD.open(filename, FILE_WRITE);

  //If it opened correctly, write the header to it
  if (dataFile) {
    dataFile.println(dataString);
    dataFile.close();
  } else {
    if (USBconnected) {
      Serial.println("Failed to open the file");
    }
  }
}


static void readGPS(unsigned long ms) {
  unsigned long start = millis();
  do {
    while (ss.available())
      gps.encode(ss.read());
  } while (millis() - start < ms);
}
```

Some C++ Arduino Library Code To Get You Started

```
#include "Quaternion2Euler.h"

void Quaternion2Euler(
    double* roll,
    double* pitch,
    double* yaw,
    const double q[4]) {
    double e0 = q[0];
    double e1 = q[1];
    double e2 = q[2];
    double e3 = q[3];
    if (e0 < 0.0) {
        e0 = -e0;
        e1 = -e1;
        e2 = -e2;
        e3 = -e3;
    }
    *roll = atan2(2.0 * (e0 * e1 + e2 * e3), (e0 * e0 + e3 * e3 - e1 * e1 - e2 * e2));
    *pitch = asin(max(-1.0, min(1.0, 2.0 * (e0 * e2 - e1 * e3))));
    *yaw = atan2(2.0 * (e0 * e3 + e1 * e2), (e0 * e0 + e1 * e1 - e2 * e2 - e3 * e3));
}

```

```
#include "function_CppKokSchonQuaternionEstimator.h"

static double cosd(const double angleDeg) {
    double angleRad = angleDeg * 3.14159 / 180.0;
    return cos(angleRad);
}

static double sind(const double angleDeg) {
    double angleRad = angleDeg * 3.14159 / 180.0;
    return sin(angleRad);
}

//The Kok Schon Quaternion Estimator Algorithm
extern int function_KokSchonQuaternionEstimator(
    double quaternion_new[4], //OUTPUT: updated quaternion orientation
    const double quaternion[4], //INPUT: previous quaternion orientation
    const double gyrometer[3], //INPUT: [rad/s] 3-axis gyro data with bias removed
    const double gravity[3], //INPUT: [any] x-front, y-right, z-down body-frame gravity vector
    const double magnetometer[3], //INPUT: [any] x-front, y-right, z-down calibrated magnetometer signal
    const double dt, //INPUT: [s] time-step
    const double inclinationAngleDegrees, //INPUT: [degrees] the geomagnetic inclination angle
    const double alpha, //INPUT: [1,2] tuning parameter for magnetometer trust weighting
    const double beta //INPUT: [< 1] small-valued positive tuning parameter
) {
    double e0, e1, e2, e3;
    e0 = quaternion[0];
    e1 = quaternion[1];
    e2 = quaternion[2];
    e3 = quaternion[3];

    double RI2b[3][3] = { 0.0 };
    RI2b[0][0] = pow(e0, 2) + pow(e1, 2) - pow(e2, 2) - pow(e3, 2);
    RI2b[0][1] = 2.0 * (e0 * e3 + e1 * e2);
    RI2b[0][2] = 2.0 * (e1 * e3 - e0 * e2);

    RI2b[1][0] = 2.0 * (e1 * e2 - e0 * e3);
    RI2b[1][1] = pow(e0, 2) - pow(e1, 2) + pow(e2, 2) - pow(e3, 2);
    RI2b[1][2] = 2.0 * (e0 * e1 + e2 * e3);

    RI2b[2][0] = 2.0 * (e0 * e2 + e1 * e3);
    RI2b[2][1] = 2.0 * (e2 * e3 - e0 * e1);
    RI2b[2][2] = pow(e0, 2) - pow(e1, 2) - pow(e2, 2) + pow(e3, 2);

    double gG[3];
    for (int ii = 0; ii < 3; ii++) {
        gG[ii] = RI2b[ii][2];
    }

    double normM = sqrt(pow(magnetometer[0], 2)
        + pow(magnetometer[1], 2)
        + pow(magnetometer[2], 2));
```

```

    double m[3];
    for (int ii = 0; ii < 3; ii++) {
        m[ii] = magnetometer[ii] / max(0.1, normM);
    }

    double mG[3];
    for (int ii = 0; ii < 3; ii++) {
        mG[ii] = RI2b[ii][0] * cosd(inclinationAngleDegrees) +
            RI2b[ii][2] * sind(inclinationAngleDegrees);
    }

    double normA = sqrt(pow(gravity[0], 2)
        + pow(gravity[1], 2)
        + pow(gravity[2], 2));
    double gA[3];
    for (int ii = 0; ii < 3; ii++) {
        gA[ii] = gravity[ii] / max(0.1, normA);
    }

    double gAminusgG[3];
    double mMinusmG[3];
    for (int ii = 0; ii < 3; ii++) {
        gAminusgG[ii] = gA[ii] - gG[ii];
        mMinusmG[ii] = m[ii] - mG[ii];
    }

    double crossGravity[3];
    crossGravity[0] = gG[1] * gAminusgG[2] - gG[2] * gAminusgG[1];
    crossGravity[1] = -(gG[0] * gAminusgG[2] - gG[2] * gAminusgG[0]);
    crossGravity[2] = gG[0] * gAminusgG[1] - gG[1] * gAminusgG[0];

    double crossMag[3];
    crossMag[0] = mG[1] * mMinusmG[2] - mG[2] * mMinusmG[1];
    crossMag[1] = -(mG[0] * mMinusmG[2] - mG[2] * mMinusmG[0]);
    crossMag[2] = mG[0] * mMinusmG[1] - mG[1] * mMinusmG[0];

    double dJ[3];
    for (int ii = 0; ii < 3; ii++) {
        dJ[ii] = crossGravity[ii] + crossMag[ii];
    }

    double normJ = sqrt(pow(dJ[0], 2) + pow(dJ[1], 2) + pow(dJ[2], 2));
    double dw[3];
    for (int ii = 0; ii < 3; ii++) {
        dw[ii] = dJ[ii] / max(0.000000001, normJ);
    }

    double dGyro[3];
    for (int ii = 0; ii < 3; ii++) {
        dGyro[ii] = gyrometer[ii] - beta * dw[ii];
    }

    double S[4][3];
    S[0][0] = -quaternion[1]; S[0][1] = -quaternion[2]; S[0][2] = -quaternion[3];
    S[1][0] = quaternion[0]; S[1][1] = -quaternion[3]; S[1][2] = quaternion[2];
    S[2][0] = quaternion[3]; S[2][1] = quaternion[0]; S[2][2] = -quaternion[1];
    S[3][0] = -quaternion[2]; S[3][1] = quaternion[1]; S[3][2] = quaternion[0];

    double q[4];
    for (int ii = 0; ii < 4; ii++) {
        q[ii] = quaternion[ii] + dt / 2.0 * (
            S[ii][0] * dGyro[0] + S[ii][1] * dGyro[1] + S[ii][2] * dGyro[2]);
    }

    double normQ = sqrt(pow(q[0], 2) + pow(q[1], 2) + pow(q[2], 2) + pow(q[3], 2));
    for (int ii = 0; ii < 4; ii++) {
        quaternion_new[ii] = q[ii] / normQ;
    }

    return 0;
}

```

Some MATLAB Code To Get You Started

```
close all
clear all
clc
%% import the data from a .csv file
%pico data first
[matlabFile,path] = uigetfile('*.csv', ...
'Select The PICO data');
picoData = readtable([path,matlabFile]);
%phone data second
[matlabFile,path] = uigetfile('*.csv', ...
'Select The Phone data');
phoneData = readtable([path,matlabFile]);

%% extract the data
%Pico data
xM = picoData.Bx; %x-axis magnetometer data
yM = picoData.By; %y-axis magnetometer data
zM = picoData.Bz; %z-axis magnetometer data
xA = picoData.gFx; %x-axis accelerometer data
yA = picoData.gFy;
zA = picoData.gFz;
xG = picoData.wx; %x-axis gyrometer data
yG = picoData.wy;
zG = picoData.wz;
GPS_lat = picoData.Latitude; %GPS Latitude
GPS_lon = picoData.Longitude;
roll = picoData.Roll*180/pi; %Roll euler angle
pitch = picoData.Pitch*180/pi;
yaw = picoData.Azimuth*180/pi;
[~,ind] = max(abs(yG));
t = picoData.time - picoData.time(ind); %Shift so the time lines up

%Phone data
xMi = phoneData.By; %x-axis magnetometer data
```

```
yMi = phoneData.Bx; %y-axis magnetometer data
zMi = -phoneData.Bz; %z-axis magnetometer data
xAi = -phoneData.gFy;%x-axis accelerometer data
yAi = -phoneData.gFx;
zAi = phoneData.gFz;
xGi = phoneData.wy;%x-axis gyrometer data
yGi = phoneData.wx;
zGi = -phoneData.wz;
GPS_lati = phoneData.Latitude; %GPS Latitude
GPS_loni = phoneData.Longitude;
rolli = -phoneData.Roll; %Roll euler angle
pitchi = -phoneData.Pitch;
yawi = phoneData.Azimuth;
[~,indI] = max(abs(yGi));
ti = phoneData.time - phoneData.time(indI); %Shift so the time lines up

%% Plot the data

%Magnetometer
figure
subplot(311)
plot(t, xM, ti, xMi)
title('Magnetometer')
ylabel('x')
grid on
subplot(312)
plot(t, yM, ti, yMi)
ylabel('y')
grid on
subplot(313)
plot(t,zM, ti, zMi)
ylabel('z')
xlabel('Time (s)')
grid on
```

```
legend('Pico','Phone')

%Accelerometer
figure
subplot(311)
plot(t, xA, ti, xAi)
title('Accelerometer')
ylabel('x')
grid on
subplot(312)
plot(t, yA, ti, yAi)
ylabel('y')
grid on
subplot(313)
plot(t,zA, ti, zAi)
ylabel('z')
xlabel('Time (s)')
grid on
legend('Pico','Phone')

%Gyrometer
figure
subplot(311)
plot(t, xG, ti, xGi)
title('Gyrometer')
ylabel('x')
grid on
subplot(312)
plot(t, yG, ti, yGi)
ylabel('y')
grid on
subplot(313)
plot(t,zG, ti, zGi)
ylabel('z')
xlabel('Time (s)')
```

```
grid on
legend('Pico','Phone')

%GPS
figure
subplot(211)
plot(t, GPS_lat, ti, GPS_lati)
title('GPS')
ylabel('Latitude')
avgL = mean(GPS_lati(end-10:end));
ylim([avgL-0.001, avgL+0.001])
grid on
subplot(212)
plot(t,GPS_lon, ti, GPS_loni)
ylabel('Longitude')
avgL = mean(GPS_loni(end-10:end));
ylim([avgL-0.001, avgL+0.001])
xlabel('Time (s)')
grid on
legend('Pico','Phone')

%Orientation
figure
subplot(311)
plot(t, roll, ti, rolli)
title('Orientation')
ylabel('Roll')
grid on
subplot(312)
plot(t, pitch, ti, pitchi)
ylabel('Pitch')
grid on
subplot(313)
plot(t,yaw, ti, yawi)
ylabel('Yaw')
xlabel('Time (s)')
grid on
legend('Pico','Phone')
```

What to Submit

To get credit for this assignment, demonstrate your working code to the instructor. Also do the following:

1. Create a pdf document with all five graphs demonstrating that the sensor data from the phone and Raspberry Pi Pico match.

Submit the document