Main Code (C++)

```
//=====================================================================
//=====          Scooby basic Data Logger code.              =====
//=====  * Monitor loop runs at 59kHz (17usec) in RPi pico.     =====
//=====     > can be adjusted to anything slower (like 44.1kHz) =====
//=====  * Data capture triggered when any peripheral mic is    =====
//=====     sees large swing.Peripheral mic means Mic0 to Mic3  =====
//=====  * Trigger filtered by expected sequence in other       =====
//=====     peripheral mics. Bad sequence rejects trigger.     =====
//=====  * Center mic (Mic4) connected to ADC2 and data taken   =====
//=====     as part of data log for later analysis.            =====
//=====  * Data captured to RPi pico memory, transferred to SD  =====
//=====     after high speed captured.                         =====
//=====  * Monitor output is disabled if USB is not plugged in  =====
//=====     > Scooby will run without USB in final form        =====
//=====     > R Pi processor hangs if Serial is used without USB=====
//=====     > We detect USB connection on A1 input and disable  =====
//=====        serial IO if it is not present                  =====
//=====  * Versions:
//=====     > 1v01 - Add LCD 4x20 display support.
//=====     > 1v02 - Use trig functions for angle calculation
//=====     > 1v03 - Add Ring LED WS2812B support
//=====     > 1v04 - Bug fix: wouldn't write SD unless USB connected
//=====            LCD: format change - shows last 3 hits
//=====            Time: added seconds to LCD/SD
//=====            BUG: 'claps' give weird results ...
//=====     > 1v05 - 'claps' fix - check for consistent hits after
//=====              first hit - 7/10 for acceptable waveform
//=====     > 1v06 - hack led ring so it shows last four hits
//=====              color coded. Last one is white, then green,
//=====              then blue, then red.
//=====             Adding "char checkSignals()" to validate the
//=====              last hit.  Shd check mic4 for reasonable
//=====              waveform.
//=====             Also adding "void wait4echos()" to wait until
//=====              reverberation dies down to look for next trigger
//=====     > 1v07  - Reorg startup so that fancy ring is in setup
//=====              Add last 4 angles to ring display
//=====     > 1v08  - Add warning to splash screen if SD isn't connected
//=====     > 1v09  - Add BLE support - write to BLE same as SD drive
//=====               Running at 9600 baud takes too long - need
//=====               to fix this - increase baud rate a lot I hope.
//=====               Anyway, this version disables the waveform
//=====               output to BLE to keep response time short. b
```

```
//=====      > 1v10  -  Shut off BLE, setup just for Serial1.output
//=====                 to Hub via RS485/232.  Also add waveform
//====                 to data output and set Serial1 for 115200 baud
//=====      > 1v11  -  Fix serial problem. setupSerial() only executed
//=====                 when IfUSB was TRUE, and it did BOTH Serial and
//=====                 Serial1.  So Serial1 was hosed.  Fixed in this
//=====                 version.  Serial1 (com port) always is initialized.
//=====                Also: clearTrigTime() initializes to 9999 so we
//=====                 we can see when a mic(s) is not triggered. Had
//=====                 change getDataRecord() tests so they looked for 9999.
//=====                 Failure to trigger all mics and bad sequences are
//=====                 now reported on LCD.  All data still recorded on SD,
//=====                 monitor page, and com port output to hub.  I2C has
//=====                 been sped up to 400k (from 100k) to help cut latency.
//=====      >1v12      Add Edwards waveform algorithm ( tested in Python ) to
//=====                 CheckSignals() function.
//=====================================================================
//  SD card datalogger - for Rasberry Pi Pico processor
//
//  SPI pins for RPi Pico
//   SD card attached to SPI bus as follows:
//=====  SPI connections
// .  ** MOSI - GPIO19 - chip pin 25
// .  ** MISO - GPIO16 - chip pin 16
// .  ** SCLK - GPIO18 - chip pin 24
// .  ** CS   - GPIO17 - chip pin 22
//=====================================================================
//=====  I2C connector
//  . ** VCC - 3.3V for RTC - 5.0V wired for external connector for LCD
//  . ** GND - connected to plane
//  . ** SDA - GP4 - chip pin 6
//  . ** SCL - GP5 - chip pin 7
//=====================================================================
//=====  External LED connections - Proto 1
//  . **LED0 -GPIO 12 - on LED connector
//  . **LED1 -GPIO 15 -
//  . **LED2 -GPIO 14 -
//  . **LED3 -GPIO 13 -
//----------------------
//=====  External LED connections - Proto 2
//  . **LED0 -GPIO 15 - on LED connector
//  . **LED1 -GPIO 14 -
//  . **LED2 -GPIO 13 -
//  . **LED3 -GPIO 12 -
```

```
//=====================================================================
//=====  External MIC connections -Proto 1
//  . **MIC0 -GPIO 6
//  . **MIC1 -GPIO 8
//  . **MIC2 -GPIO 7
//  . **MIC3 -GPIO 9
//----------------------
//=====  External MIC connections -Proto 2
//  . **MIC0 -GPIO 8
//  . **MIC1 -GPIO 7
//  . **MIC2 -GPIO 6
//  . **MIC3 -GPIO 9

//=====================================================================

#define  pinMic0 6  //GP6 => pin 12    ... Pins connected to Mic connectors
#define  pinMic1 8  //GP7 => pin 11
#define  pinMic2 7  //GP8 => pin 10   ... rearrange as needed for hardware variations
#define  pinMic3 9  //GP9 => pin 9


#include "ScoobyDL_0v2.h"
void setup() {  // As in all arduino code, setup() runs only once
              // setup() is for code that gets various functions
              //    ready to be used.  Functions may be defined,
              //    variables/constants declared, pins assigned ...
              //    stuff like that.
  //---------------- Setup ADC - thats A0 to A2 analog inputs ---------
  //----- Analog inputs are for inputs that vary gradually over a
  //-----   certain voltage range. They are converted to an integer
  //-----   that varies from, for example, 0 to 1023 (10 bit) or
  //-----   0 to 4095 (12 bit).
  //----- ADCs (A0 to A2): use ADCs to grab data from Mic4, and to sense
  //-----   the presence of USB cable. For best data we'd like them
  //-----   to be highest resolution possible.  In RPi Pico thats
  //-----   12 bits.  The Arduino default is 10 bits which is what
  //-----   older processors support.
  analogReadResolution(12);     // Set the ADCs to 12 bits:
                          //   That means results are 0 to 4095
                          // RPi ADCs (analog to digital converters)
                          //    can be 12 bits, but we have to tell
                          //    it, otherwise it defaults to 10
                          //    which would be 0 to 1023
  //----- Now we read the USB pin to see if cable is plugged in.
```

```
  USBIn=analogRead(pinUSBIn);   // Read the ADC connected to USB pin
                          //    If USB present it will be about
                          //    4095 (set in hardware), so we
                          //    test for "greater than 3500"

  //----- Next we test to see if we should get the Serial going
  setupSerial(); // "IfUSB" is a MACRO that tests for USB presence
                 // Everywhere compiler sees "IfUSB" it substitutes
                 //   if(USBIn > 3500)

  //----- The RTC (real time clock) uses the I2C bus, as does the LCD display.
  //-----   so we start it up here.  Wire.begin() is a function defined in the
  //-----   Wire library that we include in our library ScoobyDL_0v2.h.
  //----- Late addition: Wire.setClock(780000); speeds up by 4x, makes LCD faster
  //-----   Online comments suggest that 780000 actually sets to 400k ... need to check
  Wire.setClock(780000); // Speed up LCD and RTC
  Wire.begin();       // Start I2C: setup functions and variables.

  //----- Now we assign pins to various functions defined in hardware.
  //-----   GPIO is general term for General Purpose Input Outputs.
  //-----   In RPi Pico they are called GPs and range from GP0 to GP28
  setupMicPins();     // Assign GPIOs to Mics and declare as INPUTS.
  setupLedPins();     // Assign GPIOs to LEDs and declare as OUTPUTS.
  setupAnalogPins();  // Assing ADCs:  A2: Mic4, A1: USB sense
  setupSdSpi();       // Assign Pins and names for SD/SPI library.
  setupLCD();         // Initialize LCD screen.
  splashLCD();        // Welcome screen on LCD.
  setupRing();        // Make fancy colors spin
//  setupBLE();         // Setup Bluetooth module
                  // VVV Set datafile name with date VVV
  sprintf(datafileName, "Scby%02d%02d.txt", myClock.getMonth(century),myClock.getDate());
  wait4trigger();     // Spin colors until theres a trigger then go to loop
}

//================================

void loop() {
  inPort = getPort();   // Sample ports
  if(getData()) {       // if inPort != 0 grab Data (samples, bits plus Mic4 waveform)
    clearRing();        // turn off all LEDs in ring
    clearTrigTimes();   // Set trigger bits to zero.
    openSDFile();       // If SD connected open the data file
    printLCD(0,0,lcdString0 );
```

```
    getDataRecord();    // Read 'samples' from mics
    printMics();        // Print mics in index order, not triggered order
    getMicOrder();      // Put mics in order of trigger times
    print1stMic();      // Print first mic
    getQuad();          // Figure out quadrant/angle of trigger
    reverbTime = wait4Echos(); // Listen and wait for echos to stop
    updateLCD();        // Print out to LCD
    updateRing();       // Light up appropriate LED on ring.
    printParams();      // Print results to monitor page (Serial) and SD
    if(checkSignals() == 0) {
      if(badSeq)        printLCD(0, 0, "* Bad Mic Sequence *");
      else              printLCD(0, 0, "* Mic Sequence Ok  *");
      if(micFail)       printLCD(1, 0, "* Mic Trig FAILURE *");
      else              printLCD(1, 0, "* All Mics Trigger *");
      if(wfmError != 0) {
        sprintf(wfmMsg, "* Wfm Code:%07X *\r\n", wfmError);
        printLCD(2, 0, wfmMsg );
      } else {
        printLCD(2,0,"*     Wfm Okay     *");
      }
    }
    IfUSB { Serial.print(wfmMsg);
      sprintf(wfmMsg, "Times: Pt1= %d, Pt2 =%d, Pt3 =%d, Pt4 =%d\r\n",
              timePoint1,timePoint2,timePoint3,timePoint4);
      Serial.print(wfmMsg);
      sprintf(wfmMsg, "Leading Edge=%d, Peak Duration=%d, Zero Crossing=%d\r\n",
              leadingEdge, peakLength,zeroCrossing);
      Serial.print(wfmMsg);
    }
    if(dataFile) dataFile.close();  // If SD is enabled close the file
//    delay(1000);
  }
}
```

Library (C++)

```
//===== NOTE: Mic numbers go from 0 to 4 for 5 microphones.
//=====       0 - 3 are directional mics, 4 is waveform mic
//=====  Libraries
 #include <FastLED.h>     // Basic led stripe library
 #include <SPI.h>         // This is for SPI bus: Serial Peripheral Interface
 #include <RP2040_SD.h>   // THis is for the SD (Secure Digital) card routines
 #include <DS3231.h>      // Real time clock, chip is DS3231
 #include <Wire.h>        // I2C standard interface library
 #include <LiquidCrystal_I2C.h> // Support for LCD screen
//================================================================
```

```
//===== LCD Declaration
LiquidCrystal_I2C lcd = LiquidCrystal_I2C(0x27,20,4); // instantiate lcd for I2C addr, #cols, #rows
//=====================================================================
//===== Real Time Clock parameters
DS3231 myClock;         // Instantiate a real time clock called myClock
bool century = false;   // This only changes in 2100
bool h12Flag;           // We don't use this - bug found in Snoopy, we just calcute 12hr clock
bool pmFlag;            // We drive this with our own code - chip messes this up
const int chipSelect = 2;   // CS: chip select is so we can have more than one SPI device
//=====================================================================
//===== Macros
// --- Turn off serial outputs if USB is not connected
// #define IfUSB if(1)
#define IfUSB if(USBIn > 3500)  // This is a macro - piece of code substituted by compiler
//==== Ring led stuff

#define bleNewName 1
#define DATA_PIN     21
#define NUM_LEDS    60
#define BRIGHTNESS0  2
#define BRIGHTNESS1  64
#define LED_TYPE    WS2812B
#define COLOR_ORDER RGB
#define echoThresh 170  // echoThresh determines when echo has stopped
                        // 343 = (4096/3.3V) * 1.66V/6 <= this is trip point of trigger mics
                        //   referred to mic4. We set threshold at less than half for safety.
CRGB ring[NUM_LEDS];

//===== Useful definitions
#define samples 150     // Number of samples taken when triggered keep small for debug
#define TRUE 1          // Just because I sometimes use TRUE for true
#define FALSE 0         // Just because I sometimes use FALSE for false
#define VerboseSerial   // If defined print a lot of stuff for debug

#if !defined(ARDUINO_ARCH_RP2040) // Make sure this is going to right processor
  #error For RP2040 only
#endif
#if defined(ARDUINO_ARCH_MBED)    // Special definitions for SPI for RPi Pico

  #define PIN_SD_MOSI       PIN_SPI_MOSI  // MOSI => phys pin 25, GPIO19 aka SPIO TX
  #define PIN_SD_MISO       PIN_SPI_MISO  // MISO => phys pin 21, GPIO16 aka SPIO RX
  #define PIN_SD_SCK        PIN_SPI_SCK   // SCK => phys pin 24, GPIO 18
  #define PIN_SD_SS         PIN_SPI_SS    // CSN => phys pin 22, GPIO 17 aka CSel

#else
```

```
  #define PIN_SD_MOSI       PIN_SPI0_MOSI   // Some processors allow more SPI busses
  #define PIN_SD_MISO       PIN_SPI0_MISO   //   so theres 0.. n Pins assigned
  #define PIN_SD_SCK        PIN_SPI0_SCK
  #define PIN_SD_SS         PIN_SPI0_SS

#endif
/*===== Uncomment for Proto 1 ================================================ */
#define pinLED0 15 // GP12  ... Pins connected to LED header
#define pinLED1 12 // GP15
#define pinLED2 14 // GP14  ... rearrange as needed for hardware variations
#define pinLED3 13 // GP12

#define  pinMic0 8  //GP8     ... Pins connected to Mic connectors
#define  pinMic1 9  //GP9
#define  pinMic2 6  //GP6    ... rearrange as needed for hardware variations
#define  pinMic3 7  //GP7

#define NameString "SCOOBY: PROTO 1 1v12"
/*============================================================================
=*/

/*===== Uncomment for Proto 2 ================================================ /
#define pinLED0 15 // GP12  ... Pins connected to LED header
#define pinLED1 14 // GP15
#define pinLED2 12 // GP14  ... rearrange as needed for hardware variations
#define pinLED3 13 // GP12

#define  pinMic0 8  //GP6 => pin 12    ... Pins connected to Mic connectors
#define  pinMic1 7  //GP7 => pin 11
#define  pinMic2 6  //GP8 => pin 10   ... rearrange as needed for hardware variations
#define  pinMic3 9  //GP9 => pin 9

#define NameString "SCOOBY: PROTO 2 1v12"
/*============================================================================
=*/


int LEDpins[] = { pinLED0,pinLED1,pinLED2,pinLED3 };


#define pinMic4 A2  //Analog ADC input 2
#define pinUSBIn A1 //Analog ADC input 1


#define _RP2040_SD_LOGLEVEL_ 0    // Hmm ... not sure what this is - it was in sample code
//============================================================
```

```
//===== General program variables
int i,j,k;                  // generic indices ... for local loops and such
char bit0, bit1, bit2, bit3;  // bits read from hardware comparators for each mic
char micBits[33];             // string to hold bits for display
byte micSig[samples];         // data array of mic bits
unsigned char inPort = 0;     // byte read from port
int Mic4In;                 // Analog input read from Mic4 pin
char dataline[60];            // string for display
unsigned int startTime;       // Start time of data loop: micros() result
unsigned int reverbTime;      // Time (usec) of reverbs after hit
int Waveform[samples];        // ADC2 results for Mic4
                        // critical wfm points taken from waveform data.
                        //    zeroth element is time, 1th is amplitude
int timePoint0 = 0;           // Start point at trigger time, nominally zero in amplitude and time.
int timePoint1 = 0;           // Start time of leading edge
int timePoint2 = 0;           // Start time of peak: {time, amplitude}
int timePoint3 = 0;           // End time of peak: {time, amplitude}
int timePoint4 = 0;           // Zero crossing after peak
const int ampLimit0 = 50;     // Amplitude limit on point 0: start of data, trigger point
const int ampLimit1 = 150;    // Amplitude limit on point 1: start of leading edge
const int ampLimit2 = 1800;   // Amplitude limit point 2: end of leading edge
const int ampLimit3 = 1800;   // Amplitude limit point 3: end of peak
const int ampLimit4 = 0;      // Amplitude limit point 4: zero crossing
// const int leTime[2] = {300, 375};   // leading edge 50% point time: {min, max}
const int leTime[2] = {200, 375};   // leading edge 50% point time: {min, max}
// const int pkTime[2] = {100, 600};   // peak time (length): {min, max};
const int pkTime[2] = {80, 600};   // peak time (length): {min, max};
// const int zeroTime[2]={300, 1500}; // zero crossing time: {min, max};
const int zeroTime[2]={200, 1500}; // zero crossing time: {min, max};
int wfmCursor = 0;
int leadingEdge;
int peakLength;
int zeroCrossing;
int wfmError;
char wfmMsg[21];


int sampTime[samples];        // sample time = micros() for each sample - timeline
char sdInit = 0;              // Flag for successful SD startup
unsigned long portDelay;      // Calculate port latency in setup()- for debug
File dataFile;                // data file for SD
char lcdString0[30];          // Buffers for LCD
char lcdString1[30];          // Buffers for LCD
char lcdString2[30];          // Buffers for LCD
char lcdString3[30];          // Buffers for LCD
```

```
char bleName[]="Scooby2";
char bleString[60];


char datafileName[14];
int iRing=-1;               // Points to next update on ring
int iRing1=-1;              // Previous ring location 1 hit old
int iRing2=-1;              // Previous ring location 2 hits old
int iRing3=-1;              // Previous ring location 3 hits old
int Triggered =0;           // Logic flag, goes TRUE after first trigger
int iFring=0;               // Same as iRing but for fancy startup display
int loopCount=0;            // Loop Counter to time fancy ring
int fBlack=0;               // If true, paint ring black, else use color (YEL)
int fRed = 255;             // Colors for fancyRing()
int fGrn = 0;
int fBlu =  0;
int fRingCount = 0;

int micTrigTime[4],micIndex[4];     //Arrays for mic trigger times and index of each mic
int micFail=0;                  // Error Flag: at least 1 mic didn't trigger
int wfmFail=0;                  // Error Flag: waveform failed shape tests
int mic1st, mic2nd;             // First and second mic triggers
int micQuad[2];                 //      ditto in array
int micAngle;                   // Computed angle from first mic.
int dT;                         // Delta time between mic0 and mic1
int quadAngle, quadSign, absAngle;  // Angle within quad, sign of quad, and final angle
int badSeq;                     // Error Flag = sequence of trigger times is weird
int sumHits;                    // Sum hits in first 10 samples.  Less than 7 is not a 'pop'
int USBIn;                      // Analog input read from USBIn pin
int echoLpf=0;
const float delTime= 576.3;   // ((7.78"/12.0")/1125.0ft/sec)*1e06 -- max time between first mics in
usec
const float pi = 3.1415926;   // define pi for trig operations
//================================================================
//===== Useful Functions
int limit(int x, int xlo, int xhi){   // Restrict range of x: xlo <= x <= xhi
  if(x>xhi) return xhi;
  if(x<xlo) return xlo;
  return x;
}

char getPort() {                // get data from comparators on Mics 0 to 3
  bit0 = digitalRead(pinMic0);    // and shift them into a single byte
  bit1 = digitalRead(pinMic1);
  bit2 = digitalRead(pinMic2);
  bit3 = digitalRead(pinMic3);
```

```
  return (~(bit0 | bit1<<1 | bit2<<2 | bit3<<3))&0x0f;
}

void fancyRing(){
 if(Triggered != 0) return;
  FastLED.setBrightness(BRIGHTNESS0);
  fRingCount += 1;
  if(fRingCount >= 300) fRingCount =0;
  iFring += 1;
  if(iFring > 59) iFring = 0;
  if(fRingCount < 50)       {fRed = 0; fGrn = 0; fBlu = 128;}
  else if(fRingCount < 100) {fRed = 0; fGrn = 128; fBlu = 128;}
  else if(fRingCount < 150) {fRed = 0; fGrn = 128; fBlu = 0;}
  else if(fRingCount < 200) {fRed = 128; fGrn = 128; fBlu = 0;}
  else if(fRingCount < 250) {fRed = 128; fGrn = 0; fBlu = 0;}
  else if(fRingCount < 300) {fRed = 128; fGrn = 0; fBlu = 128;}
  ring[iFring] = CRGB(fRed,fGrn,fBlu);
  FastLED.show();
}

void wait4trigger(){
  loopCount = 0;
  while(TRUE) {
    inPort =  getPort();    // Sample ports
    if(inPort != 0) {
      Triggered = TRUE;
      return;
    }
    loopCount += 1;         // Keep track of times through loop
    if(loopCount >= 2000) { // If we get to 2000 reset loopCount and update Ring display
      loopCount = 0;
      fancyRing();          // Ring display rotates color pattern on startup
    }
  }
}

int wait4Echos(){
  // add code to wait until levels at mic4 quiet down before continuing
  // echoes are interesting, but occassionally only the echo triggers
  //    and yields wierd waveforms.  So we'll try to ignore them.
  //  Maybe report the length of reverberation
  int echoStart = micros();
  echoLpf = 0;
  for(i=0;i<20; i++) echoLpf = 0.9*echoLpf + 0.1*abs(analogRead(pinMic4) - 2048);
  while(echoLpf > echoThresh){
    echoLpf = 0.9*echoLpf + 0.1*abs(analogRead(pinMic4) - 2048);
```

```
    delayMicroseconds(15);
  }
  return (int)micros() - echoStart;
}

char checkSignals(){
  // add code to check validity of latest trigger
  // check mic4 output re trigger mic, and other trig mics too
  // if not valid, send back a false
  // in main loop should either print full results or indicate false trigger rejected
  timePoint0 = 0;           // Set all times to zero.  If searches fail, they stay at zero.
  timePoint1 = 0;
  timePoint2 = 0;
  timePoint3 = 0;
  timePoint4 = 0;
  leadingEdge = 0;
  peakLength = 0;
  zeroCrossing = 0;

  wfmCursor = 0;
  micFail = FALSE;      // Make sure all mics have triggered
  wfmError = 0;
  for(i=0;i<4;i++){
    if(micTrigTime[i]==9999) micFail = TRUE; // 9999 means a mic didn't trigger.
  }

//===== Following is based on waveform drawing in the Scooby presentation doc =====
// -------------------- Point 1 Search: start of leading edge ------------
  while( (Waveform[wfmCursor] ) <= ampLimit1) { // Search for point 1.
    wfmCursor += 1;
    if(wfmCursor >= samples) { wfmError = 8; return FALSE; }
  }
  timePoint1 = sampTime[wfmCursor]; // get the time of point 1
// -------------------- Point 2 Search: start of peak  --------------------
  while( (Waveform[wfmCursor] ) <= ampLimit2) {  // Search for point 2.
    wfmCursor += 1;
    if(wfmCursor >= samples) { wfmError = 16; return FALSE; }
  }
  timePoint2 = sampTime[wfmCursor]; // get the time of point 2
// -------------------- Point 3 Search: end of peak ----------------------
  while( (Waveform[wfmCursor] ) >= ampLimit3) {  // Search for point 3.
    wfmCursor += 1;
    if(wfmCursor >= samples) { wfmError = 32; return FALSE; }
  }
  timePoint3 = sampTime[wfmCursor]; // get the time of point 3
// -------------------- Point 4 Search: zero crossing ---------------------
```

```
  while( (Waveform[wfmCursor] ) >= ampLimit4) {  // Search for point 4.
    wfmCursor += 1;
    if(wfmCursor >= samples) { wfmError = 64; return FALSE; }
  }
  timePoint4 = sampTime[wfmCursor]; // get the time of point 4

// --------------Calculate Critical Features ----------------------
  leadingEdge = (timePoint1 + timePoint2)/2;  // Calc midpoint of leading edge
  peakLength = (timePoint3 - timePoint2);     // Calc length in time of peak
  zeroCrossing = timePoint4;                  // This will be 0 if point 4 search fails.

// ---------------Test and Generate Error Flag ---------------------
// Waveform error flag: 0 means Okay, lower 3 bits flag following errors
  if ( (leadingEdge < leTime[0]) || (leadingEdge > leTime[1]) ) wfmError = 1;
  if ( (peakLength < pkTime[0]) || (peakLength > pkTime[1]) ) wfmError += 2;
  if ( (zeroCrossing < zeroTime[0]) || (zeroCrossing > zeroTime[1]) ) wfmError += 4;

  if(micFail || badSeq || (wfmError !=0) ) return FALSE; // Return False if any error.
  else return TRUE;
}

void print1stMic(){
      sprintf(dataline,"First Mic hit: %d\r\n", micIndex[0]);
      IfUSB Serial.print(dataline); //print first mic hit
      dataFile.print(dataline);
      Serial1.print(dataline);
}

void printMics(){ // print mics in index order (not in trigger time order)
    sprintf(dataline,"Index, Trig Time (index order)\r\n"); // Assemble header
    dataFile.print(dataline); // Print to SD

    IfUSB Serial.print(dataline); // Print to monitor page if USB connected
    Serial1.print(dataline); // Print to BLE
    for(i=0;i<4;i++){   // Print 4 mics in index order (0,1,2,3), indexes and trigger time
      sprintf(dataline,"%d %d\r\n", i, micTrigTime[i]); // Index, Trigger Time
      dataFile.print(dataline); // print to SD
      Serial1.print(dataline);  // print to BLE
      IfUSB {
        Serial.print(dataline); // print to monitor page
      }
    }
}

void printParams(){
      sprintf(dataline, "Quad = [%1d, %1d]\r\n",mic1st,mic2nd);
```

```
    dataFile.print(dataline);
    Serial1.print(dataline);
    lfUSB Serial.print(dataline);

    sprintf(dataline, "Mic Angle = %d\r\n", micAngle);
    dataFile.print(dataline);
    Serial1.print(dataline);
    lfUSB Serial.print(dataline);

    sprintf(dataline, "Mic Trig 1st 2nd :%d, %d\r\n",micTrigTime[0],micTrigTime[1]);
    dataFile.print(dataline);
    Serial1.print(dataline);
    lfUSB Serial.print(dataline);

    sprintf(dataline, "Total angle: %d\r\n", absAngle);
    dataFile.print(dataline);
    Serial1.print(dataline);
    lfUSB Serial.print(dataline);

    if(badSeq) {
      sprintf(dataline, "!!! BAD SEQUENCE !!!\r\n");
      dataFile.print(dataline);
      Serial1.print(dataline);
      lfUSB Serial.print(dataline);
    }

    sprintf(dataline, "Reverberation Time: %d\r\n", reverbTime);
    dataFile.print(dataline);
    Serial1.print(dataline);
    lfUSB Serial.print(dataline);

    sprintf(dataline, "========END DATA EVENT========\r\n");
    dataFile.print(dataline);
    Serial1.print(dataline);
    lfUSB Serial.print(dataline);


}

void printLCD(int row, int col, char pstring[]){ // Print entire string at a location on screen
  i=0;
  lcd.setCursor(col,row);
  while (pstring[i] != 0) {
    lcd.print(pstring[i]);
    i++;
    if(i >= 20) return;
```

```
  }
  return;
}

void updateRing(){
  FastLED.clear(); // Save previous ring locations
  iRing3 = iRing2;
  iRing2 = iRing1;
  iRing1 =iRing;
  if(iRing3 >=0){
  ring[iRing3] = CRGB(0, 4, 0);
  ring[(iRing3 +1)%60] = CRGB(0, 4, 0);
  }
  if(iRing2 >=0){
  ring[iRing2] = CRGB(0, 0, 8);
  ring[(iRing2 +1)%60] = CRGB(0, 0, 8);
  }
  if(iRing1 >=0){
  ring[iRing1] = CRGB(32, 0, 0);
  ring[(iRing1 +1)%60] = CRGB(32, 0, 0);
  }
  iRing = limit(((360-absAngle)/6),0,59);
  ring[(iRing)%60] = CRGB(128,128,128);
  ring[(iRing+1)%60] = CRGB(128,128,128);
  FastLED.show();
}

void clearRing(){
  FastLED.setBrightness(BRIGHTNESS1);
  for(iFring=0; iFring<60; iFring++){
    ring[iFring] = CRGB(0,0,0);
  }
  FastLED.show();
}

void updateLCD(){
//----- Print trigger event to LCD -----

//  sprintf(lcdString0, "Trigger: %02d/%02d
%02d:%02d",myClock.getMonth(century),myClock.getDate(),
//        myClock.getHour(h12Flag,pmFlag),myClock.getMinute());

  lcdString0[20] = 0;
  printLCD(0,0,lcdString0);

//----- Print mic order and angle to LCD -----
```

```
  strcpy(lcdString3,lcdString2);
  strcpy(lcdString2,lcdString1);
  sprintf(lcdString1, "S%1d%1d%1d%1d A%03d
T%02d:%02d:%02d",micIndex[0],micIndex[1],micIndex[2],
                    micIndex[3],absAngle,myClock.getHour(h12Flag,pmFlag),myClock.getMinute(),
                    myClock.getSecond());
  lcdString1[20] = 0;

  printLCD(1,0,lcdString1);
  printLCD(2,0,lcdString2); //Print most recent two hits
  printLCD(3,0,lcdString3);

}

void openSDFile(){    // Open file and write header for current data capture
  if(sdInit != 0)  {     // Open SD file if SD setup was succesful
       // vvv Print data header to dataline string
    sprintf(dataline, "\r\nData Capture: %02d/%02d/20%02d
%02d:%02d:%02d\r\n",myClock.getMonth(century),myClock.getDate(),

myClock.getYear(),myClock.getHour(h12Flag,pmFlag),myClock.getMinute(),myClock.getSecond());
    IfUSB Serial.println(dataline); // print to serial monitor for reference
    Serial1.println(dataline); // print to BLE to label data with date/time
    dataFile = SD.open(datafileName, FILE_WRITE); //Open the file for write
    if(dataFile) dataFile.print(dataline); // and print header to SD
  } else {
    IfUSB Serial.println("SD Init Failed!");
  }
}


void clearTrigTimes(){ for(i=0;i<4;i++) micTrigTime[i] = 9999; }  // reset trigger times to zero to start

void getDataRecord(){
      sprintf(dataline, "Time  Vec   Wfm\r\n");
      dataFile.print(dataline);
/*---------------Commented out until we increase baud rate on BLE */
      Serial1.print(dataline);
/*-------------------------------------------------------------- */
      IfUSB Serial.print(dataline);
      for(i=0;i<samples;i++){
        itoa(micSig[i]+0x10,micBits,2); // Convert micSig[] to ascii base 2 string
                                  //    and force upper bit to 1 so zeros
                                  //    don't get suppressed
        micBits[0] = ' ';              // now blank upper bit in string - its done its job
                                  // Above forces leading zeros into micBits[] string for display
```

```
      sampTime[i] = sampTime[i] - startTime;
      sprintf(dataline, "%d   %s   %d\r\n",(int)sampTime[i], micBits, Waveform[i]);
//====== Now find first trigger signal in each mic.  Once TrigTime is set it isn't revisited. =====
      if(((micSig[i] & 0x01) !=0) & (micTrigTime[0] == 9999)) micTrigTime[0]=sampTime[i];
      if(((micSig[i] & 0x02) !=0) & (micTrigTime[1] == 9999)) micTrigTime[1]=sampTime[i];
      if(((micSig[i] & 0x04) !=0) & (micTrigTime[2] == 9999)) micTrigTime[2]=sampTime[i];
      if(((micSig[i] & 0x08) !=0) & (micTrigTime[3] == 9999)) micTrigTime[3]=sampTime[i];

//====== Print out this line of data record to SD file and Serial Monitor if possible,  =====
#if defined(VerboseSerial)
      IfUSB Serial.print(dataline);
#endif
/*---------------Commented out until we increase baud rate on BLE */
      Serial1.print(dataline);
/* ---------------------------------------------------------------- */
      if(dataFile)  dataFile.print(dataline);
      }
}
char getData() {                          // get 'samples' number of data points
  startTime = (unsigned int) micros();          // save the start time
  micSig[0] = inPort;
  sampTime[0] = startTime + 1;                  // make sure first sample isn't at zero
  Waveform[0] = analogRead(pinMic4) - 2048;     // Grab first waveform sample from Mic4
  sumHits = 0;
  if( ((inPort & 0b1111) != 0) )    {           // Test if any port is triggered
    for(i=1;i<samples;i++){                     // If triggered, grab the rest of samples
      micSig[i] = getPort();                    // Get mic bits for mics 0 to 3
      sampTime[i] = ((unsigned int) micros());  // Get the sample time for this sample
      Waveform[i] = analogRead(pinMic4) - 2048; // Get the Mic4 waveform sample
      if(micSig[i] != 0) sumHits +=1;           // Keep track of number of hits
      if((i == 10) && (sumHits < 7))            // If we get to 10 with less than 7 hits
        return FALSE;                           //   then this isn't a 'pop' waveform.
    }
    return TRUE;                                // Succesful data grab: return TRUE
  } else {
    return FALSE;                               // Nothing triggered: return FALSE
  }
}

void getQuad(){         // Figure out what quadrant the trigger is in
  mic1st = micIndex[0]; // Mic1st is mic zero.
  mic2nd = micIndex[1];
  micQuad[0] = mic1st;
  micQuad[1] = mic2nd;
  badSeq = FALSE;
  switch (mic1st){    // error check 2nd mic, set to zero if err quad
```

```
    case 0:          // can be either 1 or 3 else assume directly in front of mic1
      quadAngle = 0;
      if(mic2nd == 2)  { badSeq = TRUE; quadSign =0;    } // Something wrong with trigger order
      if(mic2nd == 1)  { quadSign = -1; quadAngle = 45; } // Mic2nd is 1 so add computed angle
      if(mic2nd == 3)  { quadSign =  1; quadAngle = 315;} // Mic2nd is 3 so subtract computed angle
    break;

    case 1:    // can be either 0 or 2 else assume directly in front of mic2
      quadAngle = 90;
      if(mic2nd == 3)  { badSeq = TRUE; quadSign =0;      } // Something wrong with trigger order
      if(mic2nd == 2)  { quadSign = -1; quadAngle = 135;  } // Mic2nd is 1 so add computed angle
      if(mic2nd == 0)  { quadSign = 1;  quadAngle = 45;  } // Mic2nd is 3 so subtract computed angle
    break;

    case 2:    // can be either 1 or 3 else assume directly in front of mic3
      quadAngle =180;
      if(mic2nd == 0)  { badSeq = TRUE; quadSign = 0;     } // Something wrong with trigger order
      if(mic2nd == 3)  { quadSign = -1; quadAngle = 225;  } // Mic2nd is 1 so add computed angle
      if(mic2nd == 1)  { quadSign =  1; quadAngle = 135;  } // Mic2nd is 3 so subtract computed angle
    break;

    case 3:    // can be either 2 or 4 else assume directly in front of mic4
      quadAngle = 270;
      if(mic2nd == 1)  { badSeq = TRUE; quadSign = 0;     } // Something wrong with trigger order
      if(mic2nd == 0)  { quadSign = -1; quadAngle = 315;  } // Mic2nd is 1 so add computed angle
      if(mic2nd == 2)  { quadSign =  1; quadAngle = 225;  } // Mic2nd is 3 so subtract computed angle
    break;
  }
  dT = micTrigTime[1] - micTrigTime[0];
  if(dT > delTime) dT = delTime;
  micAngle = (int) ((180.0/pi)*asin( (double) dT / delTime ) ); // compute relative angle

  micAngle = limit(micAngle, 0, 45);
  absAngle = quadAngle + quadSign*micAngle;
  if (absAngle <0) absAngle += 360;
}


void getMicOrder(){          // Gets first trigger point for each mic and puts them in order
  int i,j,tmpTime,tmpIndex;
  for(i=0;i<4;i++){
    digitalWrite(LEDpins[i], LOW);    // Set all LEDs to OFF
    micIndex[i] = i;                  // Initialize mic indexes
  }
  for(i=0; i<3; i++){     // Bubble sort the mic times
    for(j=i+1;j<4;j++){   // Two nested loops compare one mic to remaining mics
```

```
      if(micTrigTime[j] < micTrigTime[i]) {     // and swap if [jth] < [ith]
        tmpTime = micTrigTime[j];     // save the trigger time for jth mic
        tmpIndex = micIndex[j];       // save the index for jth mic
        micTrigTime[j] = micTrigTime[i];    // put ith Time into jth slot
        micIndex[j] = micIndex[i];          // put ith index into jth slot
        micTrigTime[i] = tmpTime;           // put saved jth Time into ith slot
        micIndex[i] = tmpIndex;             // put saved jth index into ith slot
      }
    }
  }
  digitalWrite(LEDpins[micIndex[0]], HIGH);

  sprintf(dataline,"Turning ON LED #%d\n",micIndex[0]);
  dataFile.print(dataline);
  IfUSB Serial.print(dataline);

  sprintf(dataline,"Index  Trig Time (Trig order)\r\n");
  dataFile.print(dataline);
  Serial1.print(dataline);
  IfUSB Serial.print(dataline);
  for(i=0;i<4;i++){
     sprintf(dataline,"%d   %d\r\n",micIndex[i],micTrigTime[i]);
     dataFile.print(dataline);
     Serial1.print(dataline);
     IfUSB Serial.print(dataline);
  }
}


//=============================================================
//===== Setup functions
//-------------------------------------------------
//==== Define mic input pins
void setupMicPins(){
  pinMode(pinMic0, INPUT_PULLUP); // GP6 => phys pin 9
  pinMode(pinMic1, INPUT_PULLUP); // GP7 => phys pin 10
  pinMode(pinMic2, INPUT_PULLUP); // GP8 => phys pin 11
  pinMode(pinMic3, INPUT_PULLUP); // GP9 => phys pin 12
}
void setupRing(){
  iRing = -1;
  iRing1 = -1;
  iRing2 = -1;
  fBlu = 0;
  fRed = 0;
  fGrn = 0;
```

```
  FastLED.addLeds<WS2812B, DATA_PIN, RGB>(ring, NUM_LEDS);
  FastLED.setBrightness(BRIGHTNESS0);
}
void setupLedPins(){
  pinMode(pinLED0, OUTPUT);    //LED0 to LED3 pins defined above
  pinMode(pinLED1, OUTPUT);
  pinMode(pinLED2, OUTPUT);
  pinMode(pinLED3, OUTPUT);
}
void setupAnalogPins(){
            pinMode(pinMic4, INPUT); // Center Mic
    pinMode(pinUSBIn, INPUT); // VSys sense - is USB plugged in?
    analogReadResolution(12); // change resolution to 12 bits
    Mic4In=analogRead(pinMic4);
    IfUSB {
      Serial.print("VBus = ");
      Serial.print(USBIn);
      Serial.println("  ... USB connected");
    }
}
//-----------------------------------------------------
//==== LCD splash  ====
void setupLCD(){
  lcd.init();
  lcd.backlight();
  strcpy(lcdString0, "                ");
  strcpy(lcdString1, "                ");
  strcpy(lcdString2, "                ");
  strcpy(lcdString3, "                ");
}
void splashLCD(){
    sprintf(lcdString0, NameString);
    printLCD(0,0,lcdString0);
    sprintf(lcdString1,"  CHS STEM PROJECT  ");
    printLCD(1,0,lcdString1);
    sprintf(lcdString2,"Ms T's Class 2023-24");
    printLCD(2,0,lcdString2);
    if(sdInit == 0)   sprintf(lcdString3,"!!!!! SD FAILED !!!!");
    else              sprintf(lcdString3,"Scooby keeps us safe");
    printLCD(3,0,lcdString3);
}
//-----------------------------------------------------
//==== SD/SPI initialize ====
void setupSdSpi(){
  sdInit = (SD.begin(PIN_SD_SS) != 0);
```

```
  IfUSB {
    Serial.println("Starting SD Card ReadWrite on ");
    Serial.println(BOARD_NAME);
    Serial.println(RP2040_SD_VERSION);

    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 (sdInit == 0)
    {
      Serial.println("Initialization failed!");
      return;
    }
      Serial.println("Initialization done.");
  }
}
//-----------------------------------------------------
void setupBLE() {
  Serial1.begin(9600);        // On RPi Pico Serial1 is on GPIO 0 - TX, and GPIO 1 - RX
  while (!Serial1) delay(10); //    RPi TX must connect to BLE RX and RPi RX to BLE TX

  IfUSB Serial.println("BLE interface started"); // Just let us know we got this far

  Serial1.write("AT+RESET\r\n");  // Reset to start clean
  delay(10);                    // BLE seems more stable if we give it some time

  Serial1.write("AT\r\n");              // Send null AT command - should show up on BLE monitor
  if(!Serial1.available()) delay(10);
  IfUSB Serial.println(Serial1.readString()); // and send to monitor page.
  delay(100);
  IfUSB Serial.write("Sent Command: AT\r\n"); // Let us see this on monitor page
  delay(10);                          // Wait a bit just because.

  if(bleNewName) {                            // If new name flag !=0 do this stuff
    sprintf(bleString, "AT+NAME%s\r\n", bleName); // Assemble command with newname
    Serial1.write(bleString);                   // Write to Serial1
    delay(10);                                     // Pause a bit
    IfUSB Serial.println(Serial1.readString()); // Read back echo if there
  }

  delay(10);
}


//=================================================
```

```
//===== Calculate port delay
void calcPortDelay(){
          IfUSB Serial.println("Starting PICO code");
          startTime = (unsigned int) micros();
          for(i=0;i<100;i++) {j=getPort(); Mic4In=analogRead(Mic4In);}
          portDelay = ((float) (micros() - startTime))/100.0;
          IfUSB {
            Serial.print("Port Delay = ");
            Serial.println(portDelay);
            Serial.print("Mic4 analog = ");
            Serial.println(Mic4In);
          }
}

void setupSerial(){
  IfUSB {
    Serial.begin(9600);       // Start Serial monitor
    while(!Serial);
    Serial.println("Serial Started");
  }
  Serial1.begin(115200);  // use for Com port
}
```

Triangulation

```
#################################################################################
#This is my attempt at making SCOOBY's triangulation code
#It will be based on the slides that Mr David provided and the 30 minutes
#walkthrough we did on 2/27, Wish me luck!
#ok so I noticed that the math in the code does not give the correct
#answer always so I have some instances where I have to change the
#sign of the number to get the correct location
#################################################################################

#only thing you need to change is what file the code will read
#or just make it read the serial port

#libraries
import math
from matplotlib import pyplot as plt
import re

#varibles
pi = math.pi
r270 = math.pi * 1.5
```

```
r180 = math.pi
r90 = math.pi / 2

signFlag = 0

units = 'ft'
Fdis = 14.5

planeAxis = [-60, 60, -60, 60]

#in the real code we would instead use the right functions to read the serial port


################################################################################
# Open the text file

def Triangulation(output1, output2):
  def getAngleSCB1(string):
        arr = string.split('\n')
        for i in range(len(arr)):
        if 'Total angle:' in arr[i]:
        angle1 = int(arr[i].replace("Total angle: ", ""))
        return angle1


  def getAngleSCB2(string):
        arr = string.split('\n')
        for i in range(len(arr)):
        if 'Total angle:' in arr[i]:
        angle2 = int(arr[i].replace("Total angle: ", ""))
        return angle2


  #calculations
  def calculateXandYcase1(phi, theta, Fdis, units):
        print("Case 1 Calcuation:")
        print()

        x = math.tan(theta) * Fdis / (math.tan(theta) - math.tan(phi))
        y = math.tan(theta) * (x - Fdis)

        x = round(x, 2)
        y = round(y, 2)
```

```
        coordinateChecker(phi, x, y)
        if signFlag == 0:

        print("The gunshot coordinates are at: (", x, units, ",", y, units, ")")

        xScbys = [0, x, Fdis]
        yScbys = [0, y, 0]

        plt.clf()

        plt.plot(xScbys, yScbys, 'b')  #'m*')

        x1 = [0, Fdis]
        y1 = [0, 0]

        plt.plot(x1, y1, 'r*')

        # Displaying grid
        plt.grid()

        # Controlling axis
        plt.axis(planeAxis)
        # Adding title
        plt.title('Where is the gunshot?')
        plt.xlabel(units)
        plt.ylabel(units)

        plt.pause(1.25)

        # Displaying plot
        plt.show(block=False)


def calculateXandYcase2(phi, theta, Fdis, units):
        print("Case 2 Calculation:")
        print()

        x = math.tan(theta) * Fdis / (math.tan(theta) - math.tan(phi))
        y = math.tan(phi) * x

        x = round(x, 2)
        y = round(y, 2)

        coordinateChecker(phi, x, y)
```

```
        if signFlag == 0:

        print("The gunshot coordinates are at: (", x, units, ",", y, units, ")")

        #this draws the line between the 3 points
        xScbys = [0, x, Fdis]
        yScbys = [0, y, 0]

        plt.clf()

        plt.plot(xScbys, yScbys, 'b')  #'m*')

        x1 = [0, Fdis]
        y1 = [0, 0]

        plt.plot(x1, y1, 'r*')

        # Displaying grid
        plt.grid()

        # Controlling axis
        plt.axis(planeAxis)
        # Adding title
        plt.title('Where is the gunshot?')
        plt.xlabel(units)
        plt.ylabel(units)

        plt.pause(0.25)

        # Displaying plot
        plt.show(block=False)


##########################################################################
#Now we check for errors
def coordinateChecker(phi, x, y):
        #We have this to make sure the coords make sense, IE in the
        #correct quadrant of the coordinate plane
        global signFlag

        if phi > r270:
        if x < 0 and y > 0:
        print("The lines did not intersect at the right point! CALCULATION ERROR")
        signFlag += 1
```

```
        #print(signFlag)
        elif phi > r180:
        if x > 0 and y > 0:
        print("The lines did not intersect at the right point! CALCULATION ERROR")
        signFlag += 1
        #print(signFlag)
        elif phi > r90:
        if x > 0 and y < 0:
        print("The lines did not intersect at the right point! CALCULATION ERROR")
        signFlag += 1
        #print(signFlag)
        elif phi > 0:
        if x < 0 and y < 0:
        print("The lines did not intersect at the right point! CALCULATION ERROR")
        signFlag += 1
        #print(signFlag)


def errorDetector(phi, theta):
        #these are the two cases where the calculation could fail
        if math.tan(phi) == 0 and math.tan(theta) == 0:
        print("The sound is along the X axis")
        return True
        elif math.tan(phi) == math.tan(theta):
        print("The two lines are parallel, very far away from the two units!")
        return True


################################################################################
#now scooby does its thing
#"reads the ports"
phi = getAngleSCB1(output1)
print(output1)
theta = getAngleSCB2(output2)

#this is to imitate reading the serial port
#print(theta,phi)
print("The angle of the first unit is: ", phi, "degrees")
print("The angle of the second unit is: ", theta, "degrees")
print()
print("Phi =", phi)
print('Theta =', theta)
print()
print("The units are", Fdis, units, "apart")
```

```
  print()
  phi = phi * pi / 180  #we have to convert it to radians first
  theta = theta * pi / 180  #i could make a function but I dont feel like it

  #print("Phi:",phi,"Theta:",theta)
  if errorDetector(phi, theta) == True:
          print("Error, please try again")
  elif abs(math.tan(theta)) < abs(math.tan(phi)):
          calculateXandYcase1(phi, theta, Fdis, units)
  elif abs(math.tan(theta)) > abs(math.tan(phi)):
          calculateXandYcase2(phi, theta, Fdis, units)
```

Sound Discrimination

```
## Balloon pop vs common loud sounds for our project SCOOBY
from os import write
import pandas as pd
import csv
import os

##This function looks for when the sound reaches its peak, loops till it
##Finds a number greater than 2000 or just the greatest number
def startOfPeak(t, wfm):
  for s in wfm:
      if s >= t:
        return s
  return max(wfm)

##This function looks for the end of peak:
def endOfPeak(t, slice):
  for e in slice:
      if e < t:
          print(e)
          return e

#This function looks for when the sound starts to curve up, leading edge start
def lookForLES(wfm):
  for les in wfm:
      if les > 150:
        return les
```

```
#TESTS
#this test checks if the leading edge (LE) time is correct
def leadingEdgeTimeTest(wfm, pk, time):
  print()
  leStart = lookForLES(wfm)
  print("Leading Edge Start:",leStart)

  print("Start of Peak:",pk)
  leStartTime = time[wfm.index(leStart)]
  pkTime = time[wfm.index(pk)]
  print("Leading Edge Start Time:",leStartTime,"us")
  print("Peak Time:",pkTime,"us")
  leAvgTime = (leStartTime + pkTime)//2
  leAvgTimes.append(leAvgTime)
  print("Leading Edge Average Time:",leAvgTime,"us")
  if leAvgTime >= 250 and leAvgTime <= 420:
      print("Leading Edge Average Time has correct length")
      return True
  elif leAvgTime < 250:
      print("Leading Edge Average Time is too short in length, too quiet")
      return False
  elif leAvgTime > 420:
      print("Leading Edge Average Time is too long in length, too loud")
      return False
  else:
    print("something is not right")
    return False

#This test calculates the duration of the peak:
def testDurationPeak(time, start, end):
  print()
  print("Start of Peak Time",time[start],"us")
  print("End of Peak Time:",time[end],"us")
  durationOfPeak=(time[end]-time[start])
  print("Duration of the Peak:",durationOfPeak,'us')
  peakDurations.append(durationOfPeak)
  if durationOfPeak > 100 and durationOfPeak < 700:
    print("Peak has correct length.")
    return True
  elif durationOfPeak < 100:
    print("Peak is too short, maybe not loud enough. Could be balloon.")
    return False
  elif durationOfPeak > 700:
```

```
    print("Peak was too long, maybe too loud. Could be balloon.")
    return False

#This part of code will see if the soundwave goes ZeroCrossing
def testForZeroCrossing(wfm, endIndex):
  print()
  for n in wfm:
    if n < 0:
      negIndex = wfm.index(n)
      if negIndex > endIndex:
        ZeroCrossingWFMs.append(n)
        print("The sound goes crosses zero")
        print(n)
        return True
  print("Sound does not past zero, either too loud or too quiet")
  ZeroCrossingWFMs.append(n)
  return False

def testForPeak():
  print()
  if startPeak >= threshold:
    print('The peak is',startPeak)
    print("It's loud enough for a balloon!")
    return True
  else:
    print('The peak is only',startPeak)
    print("It is not loud enough for a balloon! Test failed.")
    return False

#########################################################################
#this list is used to create like a data log of the soundwaves we have tested
fileNames = [] #use this list to append the name of each file


startPeakValue = [] #use this list to append each peak

endPeakValue = []

leAvgTimes = [] #use this list to append each leading edge average time


peakDurations = [] #use this list to append each duration of the peak
```

```
ZeroCrossingWFMs = [] #use this list to append either 0 (for non ZeroCrossing values) or 1 (for ZeroCrossing values)


fileName = input("Enter the name of the csv file you want to use: ")
df = pd.read_csv(fileName)
#print("Printing data frame from Scooby's file:")
#print(df)
fileNames.append(fileName)
time = []
for j,row in df.iterrows():
  time.append(row["Time"])
#print(time)

wfm = []
for i,row in df.iterrows():
  wfm.append(row["WFM"])
#print(wfm)

threshold = 1800
startPeak = startOfPeak(threshold, wfm)
startIndex = wfm.index(startPeak)

endPeak = endOfPeak(threshold, wfm[startIndex :]) #list slicing starts at startIndex and goes to the end (:) of list
endIndex = wfm.index(endPeak)

startPeakValue.append(startPeak)
endPeakValue.append(endPeak)


print("\nResults:")

balloon = []



isPeak = testForPeak()
isTestDurPeak = testDurationPeak(time, startIndex, endIndex)
isZeroCross = testForZeroCrossing(wfm, endIndex)
isLeadingEdgeTT = leadingEdgeTimeTest(wfm, startPeak, time)
if isPeak == True and isTestDurPeak == True and isZeroCross == True and isLeadingEdgeTT == True:
  print()
```

```
  print("The soundwave is probably a balloon!")
  print("Test Passed!")
  balloon.append(1)
else:
  print()
  print("Test Failed!")
  print("The soundwave is not a balloon")
  balloon.append(0)



#This part of the code writes the file name and important values into another csv file called
"data.csv", which is our data log of the soundwaves we have tested.
rows = list(zip(fileNames, startPeakValue, endPeakValue, leAvgTimes, peakDurations,
ZeroCrossingWFMs, balloon))

#Test to see if we already created a file and if we did we write to it
file_exists=os.path.exists("data.csv")
csvfile=open('data.csv','a',newline='')
csv_writer=csv.writer(csvfile)

#if it does not exist, we create a file with appropiate headers
if not file_exists:
  headers=["File Names","Start Peak Value", "End Peak Value", "Leading Edge Average Time",
"Peak Duration", "Zero Crossing Values", "Balloon"]
  csv_writer.writerow(headers)

#We write the data to the file after it has been created or it exist already
csv_writer.writerows(rows)

csvfile.close()
```

Data Collection

```
# import modules
import serial, os, string, random
import matplotlib.pyplot as plt
import string

# i copied this code from geeks4geeks
def rand():
        # initializing size of string
        N = 14
```

```
        # using random.choices()
        # generating random strings
        res = ''.join(random.choices(string.ascii_uppercase + string.digits, k=N))

        return res

# thanks stackoverflow for this code
def clearConsole():
    os.system('cls' if os.name=='nt' else 'clear')

# self-explainatory code.
clearConsole()

# scooby unit as a serial link
# we are using 115200 baud rate because of the pico's limitations
# within our current implementation, we also use a 0.2s timeout
# just in the case of a small failure.
unit1 = serial.Serial('/dev/serial/by-id/usb-UTEK_USB__-__Serial_Cable_FT8862LR-if00-port0',
115200, timeout=0.2)

# Global variables to store data
data = ""
startLogging = False
isDone = False

while True:
        try:
                # Get incoming data from serial, decode it into UTF-8 and replace all inserted
newlines with blank characters.
                incomingData = unit1.readline().decode('utf-8').replace('\n', '')
                print(incomingData)

                # if new csv header in data, stop recording
                if "Index  Trig Time" in incomingData:
                        startLogging = False
                        isDone = True

                # if logging enabled, log data into a data string buffer.
                if startLogging == True:
                        data = data + incomingData

                # if data csv header is present, trigger data recording.
                if "Time  Vec" in incomingData:
                        startLogging = True
```

```
                # if data recording is finished.
                if isDone == True:

                        # create empty arrays to fill with data buffers
                        waveformArray = []
                        dataArray = []

                        # fill array with collected data
                        initialArr = data.split('\r')

                        # for loop to parse (clense) the data
                        for i in range(len(initialArr)):

                                # if csv separator present in data
                                if "    " in initialArr[i]:
                                        # add waveform and time (microseconds) into arrays.
                                        waveformArray.append(int(initialArr[i].split("    ")[2]))
                                        dataArray.append(int(initialArr[i].split("    ")[0]))


                        # create a csv file with a random name to collect data.
                        f = open('./data/' + rand() + '.csv', 'w');

                        # clean data and replace bugs with fixed values
                        f.write('Time,Vec,Wfm\n' + data.replace('  ', ',').replace('    ', ',').replace('     ',
',').replace(',,', ','));

                        # close the file :D
                        f.close()

                        # clear the plot and display the new plot
                        plt.clf()
                        plt.plot(dataArray, waveformArray, 'b+')
                        plt.plot(dataArray, waveformArray, 'r')

                        # we are pausing for 1/4th of a second just so that our hack to display a
graph actually works.
                        plt.pause(0.25)

                        # this is a hack, please do not use.
                        plt.plot(block=False)

                        # clear variables to original values
```

```
                finalArr = []
                initialArr = []
                data = ""
                isDone = False

        except:
                # sad code :(
                print(':(')

                # reset data in case of triggered failure
                data = ""
                startLogging = False
                isDone = False
```

Machine Learning Model

```
import tensorflow_decision_forests as tfdf
import tensorflow as tf
from tensorflow_decision_forests.keras import core
from datetime import date
import pandas as pd

# Load a dataset in a Pandas dataframe.
train_df = pd.read_csv("data/train.csv")
test_df = pd.read_csv("data/test.csv")

# Convert the dataset into a TensorFlow dataset.
train_ds = tfdf.keras.pd_dataframe_to_tf_dataset(train_df, label="Balloon")
test_ds = tfdf.keras.pd_dataframe_to_tf_dataset(test_df, label="Balloon")
model = tfdf.keras.RandomForestModel()
model.fit(train_ds)
model.compile(metrics=["accuracy"])
# Configure the tuner.
tuner = tfdf.tuner.RandomSearch(num_trials=1_000_000)
tuner.choice("num_candidate_attributes_ratio", [1.0, 0.95, 0.9])
tuner.choice("use_hessian_gain", [True, False])

tuner.choice("growing_strategy", ["BEST_FIRST_GLOBAL"])
tuner.choice("max_num_nodes", [16, 32, 64, 128, 256, 512, 1024])


model2 = tfdf.keras.GradientBoostedTreesModel(
    task=core.Task.CLASSIFICATION,
    tuner=tuner,
    max_depth=100
```

```
)

model2.fit(train_ds)
model2.compile(metrics=["accuracy",tf.keras.metrics.Precision(),tf.keras.metrics.Recall()])
```

Asynchronous SCOOBY Communication (Triangulation)

```
# Raul here, this code took a few hours to debug.
# Most of the problems were mainly class and
# asyncio problems, they should be fixed now however.

# Imports
import asyncio
import serial_asyncio
from modules.edwardsTriangulation import *

# Global Variables (Will Need this for class to class communication.)
scby1Data = ""; scby1Ready = False; scby2Data = ""; scby2Ready = False


# serial asyncio classes.
class SCBY1(asyncio.Protocol):
        # Initializes data variable.
        # I know, there were better ways to do this...
        def __init__(self) -> None:+
        super().__init__()
        self.data = ""

        # once the serial communication is made, redirect the transport to our class.
        # this is mainly for serial metadata / connection info.
        def connection_made(self, transport):
        self.transport = transport

        # once data is received, parse data.
        def data_received(self, data):
        # get global variables in case match is made.
        global scby2Data, scby2Ready, scby1Data, scby1Ready

        # parse data from a byte buffer into a string.
        self.data = self.data + data.decode().replace("\\r", '\r').replace("\\n", '\n')

        # if our match is within the data, set the scby1Ready variable to true (indicates that
scooby 1 has been triggered), and check if scby2Ready is true.
        if "Reverberation" in self.data:
```

```
        scby1Data = self.data
        scby1Ready = True

        # clear data to prepare for next buffer
        self.data = ""

        if scby1Ready == True and scby2Ready == True:
                # if match is successful, call triangulation function
                Triangulation(scby1Data, scby2Data)

        # once port is closed, stop asyncio loop.
        def connection_lost(self, exc):
        print('port closed')
        self.transport.loop.stop()

class SCBY2(asyncio.Protocol):
        # Initializes data variable.
        # I know, there were better ways to do this...
        def __init__(self) -> None:
        super().__init__()
        self.data = ""

        # once the serial communication is made, redirect the transport to our class.
        # this is mainly for serial metadata / connection info.
        def connection_made(self, transport):
        self.transport = transport

        # once data is received, parse data.
        def data_received(self, data):
        # get global variables in case match is made.
        global scby2Data, scby2Ready, scby1Data, scby1Ready

        # parse data from a byte buffer into a string.
        self.data = self.data + data.decode().replace("\\r", '\r').replace("\\n", '\n')

        # if our match is within the data, set the scby2Ready variable to true (indicates that
scooby 2 has been triggered), and check if scby1Ready is true.
        if "Reverberation" in self.data:
        scby2Data = self.data
        scby2Ready = True

        # clear data to prepare for next buffer
        self.data = ""
```

```
        # if match is successful, call triangulation function
        if scby1Ready == True and scby2Ready == True:
                Triangulation(scby1Data, scby2Data)

        # once port is closed, stop asyncio loop.
        def connection_lost(self, exc):
        print('port closed')
        self.transport.loop.stop()

# get the asyncio event loop
loop = asyncio.get_event_loop()

# SCOOBY unit access via serial (linux / unix only for now, since windows is a hassle to get
working.)
scc1 = serial_asyncio.create_serial_connection(loop, SCBY1,
'/dev/serial/by-id/usb-UTEK_USB__-__Serial_Cable_FT8862LR-if00-port0', baudrate=115200)
scc2 = serial_asyncio.create_serial_connection(loop, SCBY2,
'/dev/serial/by-id/usb-UTEK_USB__-__Serial_Cable_FT8862LR-if01-port0', baudrate=115200)

# define transports (serial metadata) and protocols (data RX / TX) for each event loop.
transport, protocol = loop.run_until_complete(scc1)
transport, protocol = loop.run_until_complete(scc2)

# run the loop forever.
loop.run_forever()

# (when ports closed) stop event loop.
loop.close()
```