# Description

The Drone IMU project is an extension of the Drone Project that adds Inertial Measurement Unit (IMU) capabilities to the aircraft.

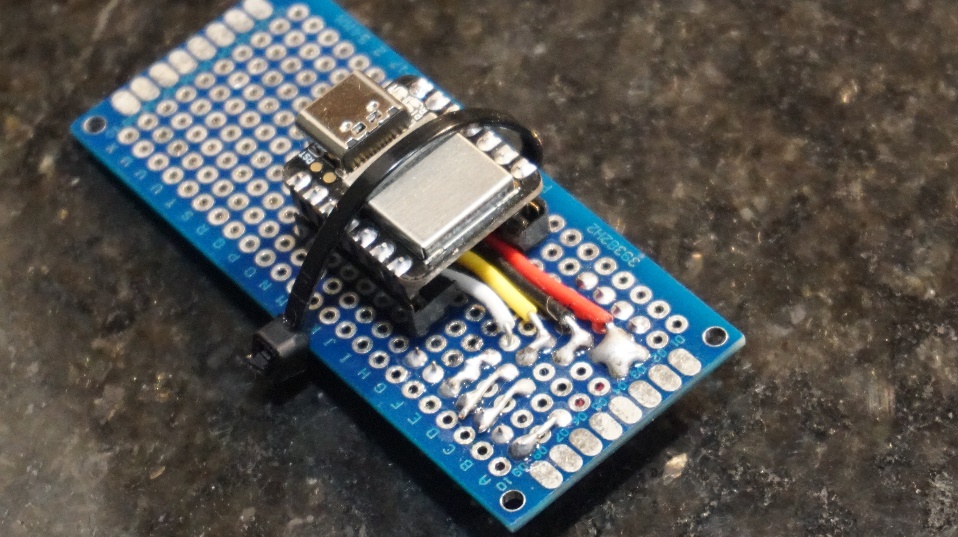


Figure – IMU Support Board

# Requirements

The following requirements describes the functionality set forth by the Drone IMU project:

* The MCU shall be implemented in a FreeRTOS environment.
* The MCU shall interface with a BNO055 Absolute Orientation Sensor.
* The MCU shall interface with the RPi via USB Serial.
* The MCU shall transmit Yaw, Pitch and Roll data to the RPi.
* The RPi shall read in the MCU Yaw, Pitch and Roll data and process the data for transmission to a Host.
* The RPi shall transmit to the Host via WiFi.
  + The RPi shall transmit to the Host via UDP Multicast.
* The Host shall receive the transmitted data from the RPi.
* The Host shall display the received Yaw, Pitch and Roll transmitted data to the User.

# Design

The following sections describe the hardware, software, and configuration design decisions for the IMU Drone project.

## Hardware

The Seeed Studio XIAO SAMD21, an Arduino-like microcontroller was used as the MCU for this project. The Raspberry Pi Model 3B+ was used as the data processing computer. In addition, the following materials were used:

* 1x BNO055 Absolute Orientation Sensor
* 1x 2’ USB-C cable
* 1x Perf Board
* An assortment of hookup cables.
* An assortment of pin headers.
* An assortment of chip sockets.
* Double sided mounting tape.

A picture containing text, electronics

Description automatically generated

Since the MCU is to interface with the BNO sensor onboard the Drone, it is highly recommended (if not, required) to solder electrical connections on a perf board. The MCU and BNO can be installed using chip sockets and securely zip-tied to the board. The wiring is simple, and a USB cable is used to connect the MCU to the RPi.

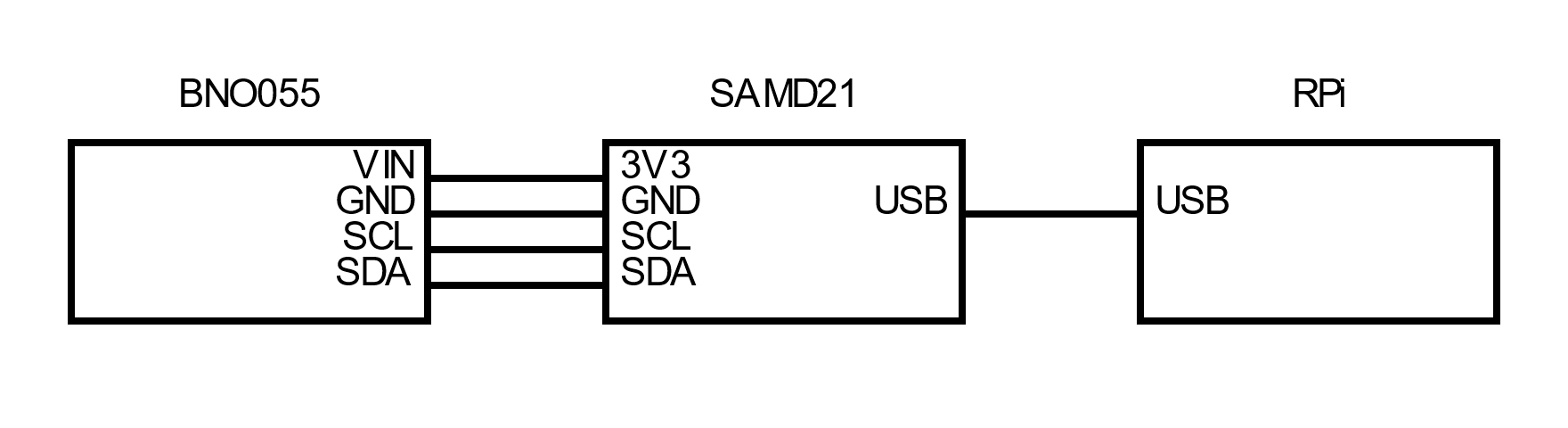


Figure – Schematic

The board was mounted at the front of the Drone, centered with the longitudinal axis of chassis.

## Software

The section is divided into the MCU (XIAO SAMD21), RPi and Host sections. Each component acts independently of software. Protocols are defined for communication between each component.

### MCU

The MCU software uses the Seeed\_Arduino\_FreeRTOS library for implementation of the RTOS. This library was specifically built to support the Seeed series microcontroller. Two task threads are defined in the software:

* SensorThread: polls the BNO sensor repeatedly and updates the global buffer.
* SerialThread: transmits the global buffer via serial as defined by the TX\_RATE.

The SerialThread is given higher priority than the SensorThread. This is to ensure that TX\_RATE is fulfilled and the SensorThread is given plenty of sampling events to occur. When loading to the global buffer occurs, the assignment of fields is surrounded by CRITICAL\_SECTION calls to prevent the SerialThread from interrupting mid-operation.

The payload that is transmitted from the MCU is defined as follows:

Diagram, table

Description automatically generated with medium confidence

A 32-bit sync word is used to ensure that the receiver can synchronize with the serial stream. If a failure occurs during the MCU operations, either no data is transmitted or garbage data is sent (that does not contain the sync word).

### RPi

The RPi (Raspberry Pi) software interfaces with the MCU via Serial USB. The MCU serial device appears as “/dev/ttyAMC1”. To ensure that the RPi receives alignment of data, a 32-bit sync word that is prefacing in the payload is scanned and the remaining payload is loaded.

The software then processes the data received. Note that both the MCU and RPi are little endian systems, when reading data no endian conversions are performed. At the time of writing, the processing performed is just a straightforward pass-thru with no calculations performed. Depending on the performance of this software, additional processing may be considered in the future (e.g. filtering, averaging, acceleration calculations, etc.…)

The RPi loads the processed data into a UDP payload buffer to be transmitted on the Network. Note that the RPi software will convert the buffer fields to big endian to conform to network standards. The structure of the buffer is as follows:

Table

Description automatically generated

The validity bit-field associates the validity of each input field with a specified bit. At the time of writing, the RPi software sets all fields as valid, unless no transmission from the MCU has been received. The validity field is defined as follows:

Table

Description automatically generated

The RPi software then transmits the loaded payload to UDP Multicast IP and port 239.100.100.250:8250. The payload will be received on the Host system software.

To execute the IMU operations on startup, the executable was registered as a flypi-imu daemon using systemctl. The /lib/systemd/system/flypi-imu.service file was created as follows:

[Unit]

Description=Drone IMU Service

After=multi-user.target

[Service]

Type=simple

ExecStart=/home/pi/flypi.d/imu

[Install]

WantedBy=multi-user.target

The flypi-imu daemon was then enabled to start on boot.

### Host

The Host software executes on the Windows OS. The Host software is written in Python 3 and uses the socket and PySimpleGUI libraries. The Host listens for UDP Multicast messages sent to 239.100.100.250:8250. The packets are parsed and displayed on GUI window. If invalid data is transmitted, the value(s) are dashed out “--“.

Graphical user interface, text, application, chat or text message

Description automatically generated

# Implementation

The source code of the Drone IMU is divided into the MCU, RPi and Host sections.

## MCU

### imu\_def.hpp

//-----------------------------------------------------------------------------

/// @file imu\_def.hpp

/// @author Nate Lao (nlao1@jh.edu)

/// @brief Defines the IMU serial payload

//-----------------------------------------------------------------------------

#ifndef \_\_IMU\_DEF\_HPP\_\_

#define \_\_IMU\_DEF\_HPP\_\_

#if RPI

#include <netinet/in.h>

#define float\_t float

#endif

namespace IMU

{

    //-----------------------------------------------------------------------------

    /// @brief Defines the serial payload buffer.

    /// @details The buffer contents are defined as follows:

    //

    /// Label            | Byte Index | Type     | Valid Values/Range  | Failure Values

    /// ---------------- | ---------- | -------- | ------------------- | ------------------

    /// Sync Word        | 0..3       | U32      | 0x444F5045          | 0x43524150

    /// Yaw (X-Rot)      | 4..7       | F32      | 0..360 (CW -> +)    | 0xFFFFFFFF

    /// Pitch (Y-Rot)    | 8..11      | F32      | -90..90 (UP -> +)   | 0xFFFFFFFF

    /// Roll (Z-Rot)     | 12..15     | F32      | -180..180 (R -> -)  | 0xFFFFFFFF

    //

    //-----------------------------------------------------------------------------

    typedef struct

    {

        uint32\_t sync\_word;

        float\_t yaw;

        float\_t pitch;

        float\_t roll;

    } PAYLOAD;

    const uint32\_t SYNC\_WORD = 0x444F5045;

    const uint32\_t FAIL\_WORD = 0x43524150;

    const PAYLOAD FAILURE\_PAYLOAD = {FAIL\_WORD, (float\_t)0xFFFFFFFF, (float\_t)0xFFFFFFFF, (float\_t)0xFFFFFFFF};

}

#endif

### imu\_sensor.hpp

//-----------------------------------------------------------------------------

/// @file imu\_sensor.hpp

/// @author Nate Lao (nlao1@jh.edu)

/// @brief Defines the IMU sensor class.

//-----------------------------------------------------------------------------

#ifndef \_\_IMU\_SENSOR\_HPP\_\_

#define \_\_IMU\_SENSOR\_HPP\_\_

#include <Wire.h>

#include <Adafruit\_Sensor.h>

#include <Adafruit\_BNO055.h>

#include <utility/imumaths.h>

#include "imu\_def.hpp"

namespace IMU

{

    class Sensor

    {

    public:

        Sensor(int32\_t sensorID, IMU::PAYLOAD\* bufferAddr);

        bool begin();

        bool frame();

    private:

        IMU::PAYLOAD\* buffer;

        Adafruit\_BNO055 sensor;

        sensors\_event\_t event;

    };

}

#endif

### imu\_sensor.cpp

//-----------------------------------------------------------------------------

/// @file imu\_sensor.cpp

/// @author Nate Lao (nlao1@jh.edu)

/// @brief Defines the IMU sensor initialization and reading operations.

//-----------------------------------------------------------------------------

#include <Seeed\_Arduino\_FreeRTOS.h>

#include <Wire.h>

#include <Adafruit\_Sensor.h>

#include <Adafruit\_BNO055.h>

#include <utility/imumaths.h>

#include "include/imu\_sensor.hpp"

//-----------------------------------------------------------------------------

/// @brief IMU Sensor

//-----------------------------------------------------------------------------

IMU::Sensor::Sensor(int32\_t sensorID, IMU::PAYLOAD \*bufferAddr)

{

    buffer = bufferAddr;

    sensor = Adafruit\_BNO055(sensorID);

}

//-----------------------------------------------------------------------------

/// @brief  Initializes the IMU Sensor. Must be called before operations

/// @return true if IMU Sensor intialization was successful.

//-----------------------------------------------------------------------------

bool IMU::Sensor::begin()

{

    bool retval = sensor.begin();

    if (retval)

        sensor.setExtCrystalUse(true);

    return retval;

}

//-----------------------------------------------------------------------------

/// @brief  Loads the io read buffer with sensor data.

//-----------------------------------------------------------------------------

bool IMU::Sensor::frame()

{

    bool retval = sensor.getEvent(&event);

    /// @todo investigate low-pass filtering

    /// @todo the retval for BNO sensor may always be true, investigate

    /// invalid values as failure operations may never occur (even in

    /// the event of an actual failure)

    if (retval)

    {

        /// @note Mismatched rotation measurements can be

        /// tolerated. The higher priority task should

        /// not be able to interrupt during singular assignments

        /// (though they might be atomic by nature...)

        // Sync Word

        taskENTER\_CRITICAL();

        buffer->sync\_word = IMU::SYNC\_WORD;

        taskEXIT\_CRITICAL();

        // Yaw Measurements

        taskENTER\_CRITICAL();

        buffer->yaw = event.orientation.x;

        taskEXIT\_CRITICAL();

        // Pitch Measurements

        taskENTER\_CRITICAL();

        buffer->pitch = event.orientation.y;

        taskEXIT\_CRITICAL();

        // Roll Measurements

        taskENTER\_CRITICAL();

        buffer->roll = event.orientation.z;

        taskEXIT\_CRITICAL();

    }

    else

    {

        taskENTER\_CRITICAL();

        memcpy(buffer, &IMU::FAILURE\_PAYLOAD, sizeof(IMU::PAYLOAD));

        taskEXIT\_CRITICAL();

    }

    return retval;

}

### imu.ino

//-----------------------------------------------------------------------------

/// @file imu.ino

/// @author Nate Lao (nlao1@jh.edu)

/// @brief Main RTOS driver for the IMU Sensor.

/// @details Designed for Seeed Studio Xiao microcontroller.

//-----------------------------------------------------------------------------

// ----- RTOS LIBRARY ----- //

#include <Seeed\_Arduino\_FreeRTOS.h>

// ----- LOCAL LIBRARIES ----- //

#include "include/imu\_def.hpp"

#include "include/imu\_sensor.hpp"

const int32\_t BNO\_SENSOR\_ID = 55; /\*\* @note Sensor ID, do not change \*/

const unsigned long TX\_RATE = 50; /\*\* @note Delay per transmission (smaller -> faster) \*/

IMU::PAYLOAD buffer;

IMU::Sensor \*sensor\_driver = 0;

// ----- RTOS THREADS ----- //

/// @note Prority Ranking (larger -> higher priority)

UBaseType\_t PRIORITY\_SENSOR\_TASK = tskIDLE\_PRIORITY + 1;

TaskHandle\_t Handle\_SensorTask;

static void SensorThread(void \*pvParameters)

{

  sensor\_driver = new IMU::Sensor(BNO\_SENSOR\_ID, &buffer);

  // Attempt to initialize the sensor

  // If failed, re-attempt indefinitely

  while (!sensor\_driver->begin())

  {

    delay(100);

  }

  // Sensor Intialization Success - Normal Operation

  while (1)

  {

    sensor\_driver->frame();

  }

}

/// @note Prority Ranking (larger -> higher priority)

UBaseType\_t PRIORITY\_SERIAL\_TASK = tskIDLE\_PRIORITY + 2;

TaskHandle\_t Handle\_SerialTask;

static void SerialThread(void \*pvParameters)

{

  while (1)

  {

    Serial.write((const char \*)&buffer, sizeof(buffer));

    delay(TX\_RATE);

  }

}

// ----- MAIN ----- //

void setup(void)

{

  // Intialize the output payload to FAILURE on startup

  memcpy(&buffer, &IMU::FAILURE\_PAYLOAD, sizeof(buffer));

  // Setup Serial

  Serial.begin(9600);

  vNopDelayMS(1000); // prevents USB driver crash on startup, do not omit this

  while (!Serial)

  {

  } // Wait for Serial terminal to open port before starting program

  // Spin up the threads

  xTaskCreate(SensorThread, "IMU Sensor Sampling", 256, NULL, PRIORITY\_SENSOR\_TASK, &Handle\_SensorTask);

  xTaskCreate(SerialThread, "Serial Transmission", 256, NULL, PRIORITY\_SERIAL\_TASK, &Handle\_SerialTask);

  // Start the RTOS, this function will never return and will schedule the tasks.

  vTaskStartScheduler();

}

void loop(void)

{

  // NOTHING

}

## RPi

### processing.hpp

//-----------------------------------------------------------------------------

/// @file processing.hpp

/// @author Nate Lao (nlao1@jh.edu)

/// @brief Defines Processing Functions for FlyPi operations.

//-----------------------------------------------------------------------------

#ifndef \_\_PROCESSING\_HPP\_\_

#define \_\_PROCESSING\_HPP\_\_

#include "imu\_def.hpp"

#include "protocol.hpp"

namespace IMU

{

    void process(const IMU::PAYLOAD\* input, PROTOCOL::IMU\_DATA\* output);

}

#endif

### protocol.hpp

//-----------------------------------------------------------------------------

/// @file protocol.hpp

/// @author Nate Lao (nlao1@jh.edu)

/// @brief Defines RPi-to-Host Protocol communications.

//-----------------------------------------------------------------------------

#ifndef \_\_PROTOCOL\_HPP\_\_

#define \_\_PROTOCOL\_HPP\_\_

#include <netinet/in.h>

namespace PROTOCOL

{

    class UDP

    {

    public:

        UDP(const char\* dest\_ip, const int dest\_port, const int process\_port);

        ~UDP();

        size\_t transmit(const void \*buf, size\_t len);

    private:

        int socket\_fd;

        struct sockaddr\_in dest\_addr;

    };

    typedef struct

    {

        float yaw;

        float pitch;

        float roll;

        unsigned char validity;

    } IMU\_DATA;

}

#endif

### serial.hpp

//-----------------------------------------------------------------------------

/// @file serial.hpp

/// @author Nate Lao (nlao1@jh.edu)

/// @brief Defines MCU-to-RPi Serial Interfaces.

//-----------------------------------------------------------------------------

#ifndef \_\_SERIAL\_HPP\_\_

#define \_\_SERIAL\_HPP\_\_

#include <stddef.h>

#include "imu\_def.hpp"

namespace SERIAL

{

    class USB

    {

    public:

        USB(const char\* device, int baud, uint32\_t sync\_word);

        ~USB();

        int read(void\* buf\_ptr, size\_t buf\_size);

    private:

        int fd;

        const static size\_t SYNC\_WORD\_SIZE = sizeof(uint32\_t);

        unsigned char sync\_word[SYNC\_WORD\_SIZE];

    };

    class SPI

    {

    public:

        /// @todo to be defined

        SPI();

        int read(void\* buf\_ptr, size\_t buf\_size);

    };

    class I2C

    {

    public:

        /// @todo to be defined

        I2C();

        int read(void\* buf\_ptr, size\_t buf\_size);

    };

}

#endif

### imu.cpp

//-----------------------------------------------------------------------------

/// @file imu.cpp

/// @author Nate Lao (nlao1@jh.edu)

/// @brief Main Driver for the FlyPi IMU process.

/// @details References:

/// https://stackoverflow.com/questions/49577244/serial-communication-between-rpi-and-arduino-using-c

/// https://www.dexterindustries.com/howto/run-a-program-on-your-raspberry-pi-at-startup/

//-----------------------------------------------------------------------------

#include <stdio.h>

#include <string.h>

#include <errno.h>

#include "imu\_def.hpp"

#include "serial.hpp"

#include "protocol.hpp"

#include "processing.hpp"

//-----------------------------------------------------------------------------

/// @brief Defines the destination IP for the FlyPi data.

/// @note IP 239.255.255.250:8250 works for some reason

/// see (http://www.iana.org/assignments/multicast-addresses/multicast-addresses.xhtml)

//-----------------------------------------------------------------------------

const char\* TARGET\_IP = "239.100.100.250";

//-----------------------------------------------------------------------------

/// @brief Defines the destination port for the FlyPi IMU data.

//-----------------------------------------------------------------------------

const int TARGET\_PORT = 8250;

//-----------------------------------------------------------------------------

/// @brief Defines the host port for the FlyPi IMU process.

//-----------------------------------------------------------------------------

const int PROCESS\_PORT = 8000;

//-----------------------------------------------------------------------------

/// @brief Main Driver for FlyPi IMU process

//-----------------------------------------------------------------------------

int main()

{

    int retval = 0;

    // ----- INPUT OBJECTS ----- //

    SERIAL::USB\* usb = new SERIAL::USB("/dev/ttyACM1",9600,IMU::SYNC\_WORD);

    IMU::PAYLOAD imu\_payload;

    // ----- OUTPUT OBJECTS ----- //

    PROTOCOL::UDP\* udp = new PROTOCOL::UDP(TARGET\_IP, TARGET\_PORT, PROCESS\_PORT);

    PROTOCOL::IMU\_DATA;

    while (1)

    {

        // ----- INPUT ----- //

        retval = usb->read(&imu\_payload,sizeof(imu\_payload));

        // ----- PROCESS ----- //

        if (retval > 0)

        {

            IMU::process(&imu\_payload, &imu\_data);

        }

        else

        {

            printf("IMU Thead: Serial Read Failure\n");

            imu\_data.validity = 0x0;

        }

        // ----- OUTPUT ----- //

        retval = udp->transmit(&imu\_data, sizeof(imu\_data));

#if DEBUG

        printf("IMU Thead: UDP transmission retval=%d\n",retval);

#endif

    }

    if (usb > 0) delete usb;

    if (udp > 0) delete udp;

}

//-----------------------------------------------------------------------------

/// @brief Host-to-Network conversion for 32-bit float values.

//-----------------------------------------------------------------------------

static float htonf32(float input)

{

    float output;

    unsigned long buffer;

    memcpy(&buffer, &input, 4);

    buffer = htonl(buffer);

    memcpy(&output, &buffer, 4);

    return output;

}

//-----------------------------------------------------------------------------

/// @brief Processes the given serial input to udp output

/// @note For now, this does a straight pass-thru, with no processing.

/// If any filtering is to be performed, it should be performed here.

//-----------------------------------------------------------------------------

void IMU::process(const IMU::PAYLOAD\* input, PROTOCOL::IMU\_DATA\* output)

{

    output->validity = 0x0;

    output->yaw = htonf32(input->yaw);

    output->validity |= 0x1 << 0;

    printf("YAW=%f ",input->yaw);

    output->pitch = htonf32(input->pitch);

    output->validity |= 0x1 << 1;

    printf("PITCH=%f ",input->pitch);

    output->roll = htonf32(input->roll);

    output->validity |= 0x1 << 2;

    printf("ROLL=%f\n",input->roll);

}

### serial\_usb.cpp

//-----------------------------------------------------------------------------

/// @file serial\_usb.cpp

/// @author Nate Lao (nlao1@jh.edu)

/// @brief Defines the USB interface class.

//-----------------------------------------------------------------------------

#include <stdio.h>

#include <string.h>

#include <wiringSerial.h>

#include "serial.hpp"

#include "imu\_def.hpp"

//-----------------------------------------------------------------------------

/// @brief Constructor

/// @details The sync\_word is used to verify input.

//-----------------------------------------------------------------------------

SERIAL::USB::USB(const char\* device, int baud, uint32\_t sync\_word)

{

    fd = serialOpen(device, baud);

    memcpy(this->sync\_word, &sync\_word, SYNC\_WORD\_SIZE);

#if DEBUG

    if (fd < 0) printf("SERIAL::USB initialization failed.\n");

#endif

}

//-----------------------------------------------------------------------------

/// @brief Destructor

//-----------------------------------------------------------------------------

SERIAL::USB::~USB()

{

    if (fd >= 0)

    {

        serialClose(fd);

    }

}

//-----------------------------------------------------------------------------

/// @brief Performs a USB read function. The stream is synced as per the defined

/// SYNC WORD.

//-----------------------------------------------------------------------------

int SERIAL::USB::read(void\* buf\_ptr, size\_t buf\_size)

{

    int retval = -1;

    if (fd >= 0)

    {

        // ----- SYNC READ HEAD ----- //

        /// @note This loop synchronizes the serial read head

        /// with the expected sync word.

        /// @note the Raspberry Pi and the Seeed Arduino slave are

        /// BOTH little endian. Sync will NOT work if endianess

        /// was mismatched...

        retval = 0;

        size\_t idx = 0;

        while ((idx < SYNC\_WORD\_SIZE) && (retval >= 0))

        {

            retval = serialGetchar(fd);

            if ((retval >= 0) && (((unsigned char) retval) == sync\_word[idx]))

                idx++;

        }

        if (idx == SYNC\_WORD\_SIZE)

        {

            memcpy(buf\_ptr, sync\_word, SYNC\_WORD\_SIZE);

            unsigned char\* buffer = (unsigned char\*) buf\_ptr;

            while (idx < buf\_size)

            {

                /// @note We are assuming that if the sync word got thru,

                /// then so did the rest... Pull in the rest of the data.

                /// @todo This could be more robust...

                /// @note The WiringPi library already does masking,

                /// casting from int to char should be OK.

                retval = serialGetchar(fd);

                buffer[idx] = (unsigned char) retval;

                idx++;

            }

            // Finally, set retval to the request size

            retval = (int) buf\_size;

        }

    }

    return retval;

}

### udp.cpp

//-----------------------------------------------------------------------------

/// @file udp.cpp

/// @author Nate Lao (nlao1@jh.edu)

/// @brief Defines the WiFi UDP Interface class.

//-----------------------------------------------------------------------------

#include <netinet/in.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include "protocol.hpp"

//-----------------------------------------------------------------------------

/// @brief Opens a UDP Multicast Socket for transmission to the given IP and port.

//-----------------------------------------------------------------------------

PROTOCOL::UDP::UDP(const char\* dest\_ip, const int dest\_port, const int process\_port)

{

    bool status = true;

    // ----- ALLOCATE SOCKET ----- //

    socket\_fd = socket(AF\_INET, SOCK\_DGRAM, 0);

    if (socket\_fd < 0)

    {

        printf("PROTOCOL::UDP socket allocation failed\n");

        status = false;

    }

#if DEBUG

    else printf("PROTOCOL::UDP socket 0x%x\n",socket\_fd);

#endif

    // ----- ENABLE BROADCAST ----- //

    if (status)

    {

        int enable = 1;

        if (setsockopt(socket\_fd, SOL\_SOCKET, SO\_BROADCAST, &enable, sizeof(enable)) != 0)

        {

            printf("PROTOCOL::UDP setsockopt failed\n");

            status = false;

        }

#if DEBUG

        else printf("PROTOCOL::UDP setsockopt success\n",socket\_fd);

#endif

    }

    // ----- BIND ----- //

    if (status)

    {

        struct sockaddr\_in addr;

        memset(&addr, 0, sizeof(addr));

        addr.sin\_family = AF\_INET;

        addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

        addr.sin\_port = htons(process\_port);

        if (bind(socket\_fd, (struct sockaddr\*)&addr, sizeof(addr)) != 0)

        {

            printf("PROTCOL::UDP bind failed\n");

            status = false;

        }

#if DEBUG

        else printf("PROTOCOL::UDP bind success\n",socket\_fd);

#endif

    }

    // ----- SET TARGET IP and PORT ----- //

    if (status)

    {

        memset(&dest\_addr, 0, sizeof(dest\_addr));

        dest\_addr.sin\_family = AF\_INET;

        dest\_addr.sin\_addr.s\_addr = inet\_addr(dest\_ip);

        dest\_addr.sin\_port = htons(dest\_port);

    }

}

//-----------------------------------------------------------------------------

/// @brief Destructor

//-----------------------------------------------------------------------------

PROTOCOL::UDP::~UDP()

{

    if (socket\_fd >= 0) close(socket\_fd);

}

//-----------------------------------------------------------------------------

/// @brief Transmit the given buffer on current io adapter.

//-----------------------------------------------------------------------------

size\_t PROTOCOL::UDP::transmit(const void \*buf, size\_t len)

{

    return sendto(socket\_fd, buf, len, 0, (struct sockaddr\*) &dest\_addr, sizeof(dest\_addr));

}

## Host

### attitude\_indicator\_lib.py

# Attitude Indicator Monitor Libraries

# Author: Nate Lao (nlao1@jh.edu)

# Designed for Windows OS

import socket

import struct

class RECEIVER:

    # The Multicast IP and Port of the Rx Payload

    BROADCAST = "239.100.100.250"

    PORT = 8250

    # NOTE: This is windows dependent for connecting to the correct interface

    INTERFACE = "192.168.4.35"

    PAYLOAD\_SIZE = 16

    PAYLOAD\_FORM = '!fff1b3b'

    TRAFFIC\_ID = 0x45535057

    def \_\_init\_\_(self):

        self.sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM, socket.IPPROTO\_UDP)

        self.sock.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

        self.sock.bind(('',RECEIVER.PORT))

        group = socket.inet\_aton(RECEIVER.BROADCAST)

        iface = socket.inet\_aton(RECEIVER.INTERFACE)

        self.sock.setsockopt(socket.IPPROTO\_IP, socket.IP\_ADD\_MEMBERSHIP, group+iface)

        self.sock.settimeout(0.1)

    def recv(self):

        try:

            payload = self.sock.recv(RECEIVER.PAYLOAD\_SIZE)

            success = True

        except socket.timeout:

            success = False

        if success:

            # Unpack payload

            payload\_fields = struct.unpack(RECEIVER.PAYLOAD\_FORM, payload)

            yaw, pitch, roll, validity, spare\_1, spare\_2, spare\_3 = payload\_fields

            return yaw, pitch, roll, validity

        else:

            return None

if \_\_name\_\_ == "\_\_main\_\_":

    receiver = RECEIVER()

    while True:

        print(receiver.recv())

### attitude\_indicator.py

# Attitude Indicator

# Author: Nate Lao (nlao1@jh.edu)

# Designed for Windows OS

# GUI Driver for AI measurements

import PySimpleGUI as sg

import attitude\_indicator\_lib as lib

sg.theme('Dark Blue')

FONT\_SIZE = ('Any 30')

LAYOUT = [

    [sg.Text('Attitude Indicator Raw Monitor',font=FONT\_SIZE)],

    [sg.Text('Yaw = ',font=FONT\_SIZE), sg.Text("N/A",key="--yaw--",font=FONT\_SIZE)],

    [sg.Text('Pitch = ',font=FONT\_SIZE), sg.Text("N/A",key="--pitch--",font=FONT\_SIZE)],

    [sg.Text('Roll = ',font=FONT\_SIZE), sg.Text("N/A",key="--roll--",font=FONT\_SIZE)],

    ]

if \_\_name\_\_ == "\_\_main\_\_":

    window = sg.Window('Attitude Indicator Raw Monitor', LAYOUT, finalize=True)

    # Setup Receiver

    receiver = lib.RECEIVER()

    ########## Event Loop ##########

    while True:

        # Check to see if the cancel button was clicked and exit loop if clicked

        event, values = window.read(timeout=10)

        if event == 'Cancel' or event == sg.WIN\_CLOSED:

            break

        # Poll Rx

        payload = receiver.recv()

        if payload != None:

            # Unpack payload

            yaw, pitch, roll, validity = payload

            # Update GUI

            window["--yaw--"].update(   "{:.2f}".format(yaw)    if (validity & 0x1 > 0) else "--")

            window["--pitch--"].update( "{:.2f}".format(pitch)  if (validity & 0x2 > 0) else "--")

            window["--roll--"].update(  "{:.2f}".format(roll)   if (validity & 0x4 > 0) else "--")

        else:

            window["--yaw--"].update(   "--")

            window["--pitch--"].update( "--")

            window["--roll--"].update(  "--")

    # Loop Terminated - Kill Window

    window.close()

else:

    print("This application cannot be imported, please execute directly.")

# Maintainance Squawks

The following list are known squawks encountered during Drone development. It should be noted that some issues have been resolved during development while others are still present and monitored.

* Bad Battery (RESOLVED):
  + ~~The battery delivered was apparently INOP. An attempt was made to charge the battery with the provided charger. The charger did not indicate a charge being performed. After letting charge for approximately 2 hours, the main output terminals indicated only approximately 3 volts DC, rather than the expected ~11 volts.~~
  + Extra batteries were ordered via Amazon. Specs are similar to the original battery (voltage is matches, but the milliampere hours (mAh) and charge constants (C) are allowed to vary).
* Yaw Potentiometer Broken (RESOLVED – WORKAROUND):
  + The Yaw Potentiometer in the KK Controller seems to broken. Rotating the pot does not have a stopping point. Further investigation is needed.
  + No noticeable impact from this issue. This is a non-grounding squawk.
* Inconsistent Motor (RESOLVED)
  + ~~The aft motors seem to be slow to start. If under no load, the aft motors spin at the same time as the front, however, having a propeller load impedes the motor start.~~
  + Extra motors were ordered via Amazon. Investigation over the impact of swapping motors is to be conducted in the future.
  + ~~There is a possibility that calibration is lacking or improperly configured. Future investigation will be conducted.~~
  + Replacing the bad motor allows for an actual takeoff. More practice is needed for proficiency.
* Motion Application Issues
  + Annoyingly, support for motion startup at boot (daemon mode) does not work.
  + The framerate of the Motion RPi library is 1 FPS on browser. Need a more robust method and investigation
* Poor Camera Mounting
  + The camera will shift or break off at hard landings. Need to find a more secure way to lock position.
* No lat/long
  + The xgps GUI did not feature lat/long coordinates. 3D fix was confirmed and zulu time was determined. Possible bug?

# Lessons Learned

* XIAO SAMD21 Bricked
  + The microcontroller can be un-bricked by loading a good application on the recovery bootloader (two quick reset on pads). Note that the COM interface will be different on recovery partition. The main partition can be restored and can be reprogrammed.
* UDP Multicast
  + There is a specific range of IPs that support UDP multicast. Reference <http://www.iana.org/assignments/multicast-addresses/multicast-addresses.xhtml>

# FAA Registration

The Drone was registered under FAA Part 107. The Drone is designed under Tail Number (Nickname) of N42069 (pronounced “November-four-twenty-sixty-nine”) and Registration Number of FA3NL7PRFE.

Graphical user interface, text

Description automatically generated

# DEMO

A demonstration of the Drone IMU can be found here:

<https://youtu.be/rViNShEdFUQ>