# Description

The FlyPi Drone project is the final iteration of the Drone Project that incorporates Camera, GPS and IMU capabilities of previous projects.

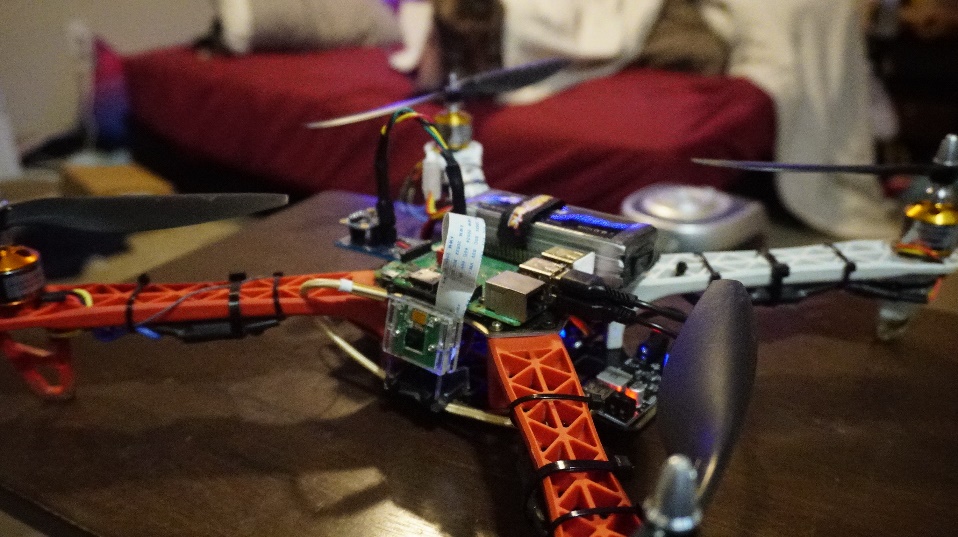


Figure – FlyPi Drone

# Requirements

The following requirements describe the functionality set forth by the FlyPi Drone project. This project includes three separate entities that operate in tandem, the Arduino Microcontroller (MCU), the Raspberry Flight Data Computer (RPi) and the Ground Station Computer (Host). The User referenced in the requirements refer to the person(s) operating the Drone and the Ground Station Computer:

* The MCU shall interface with a BNO055 Absolute Orientation Sensor.
* The MCU shall interface with the RPi using the I2C Protocol.
* The MCU shall transmit Yaw, Pitch and Roll data to the RPi.
* The RPi shall receive in the MCU Yaw, Pitch and Roll data and process the data for transmission to the Host.
* The RPi shall interface with an onboard camera and process the footage for transmission to the Host.
* The RPi shall interface with an onboard GPS module and process latitude and longitude data for transmission to the Host.
* The RPi shall transmit all data to the Host via UDP.
* The Host shall receive the RPi transmitted data through UDP.
* The Host shall display the received Yaw, Pitch and Roll data to the User.
* The Host shall display the received Footage to the User.
* The Host shall display the received GPS 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 onboard Flight Data Computer. A generic GPS module was connected directly to the RPi. A small camera with CSI (Camera Serial Interface) replaced the older USB camera due to greater performance capability.

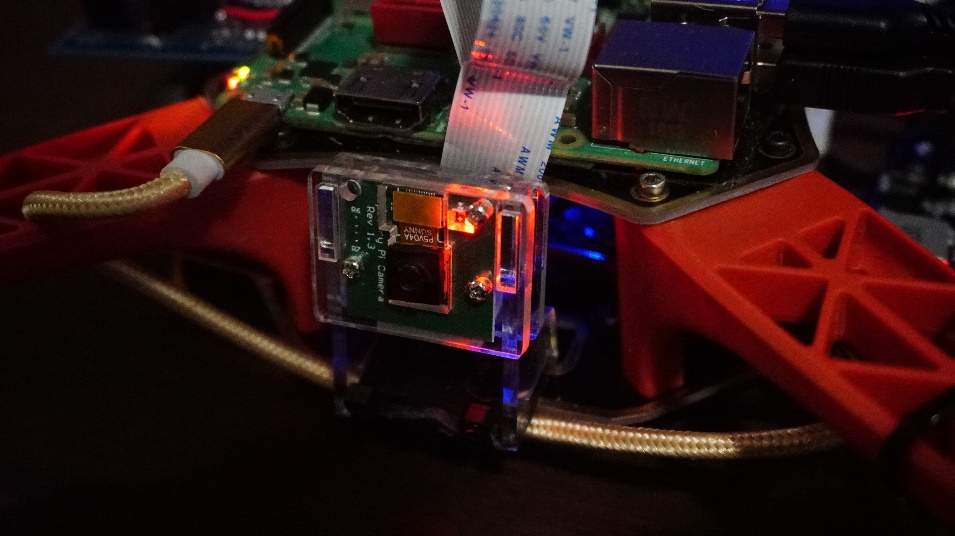


Figure – Drone Camera

A second revision MCU IMU support board was created to support the I2C interface between the MCU and RPi. In addition, the placement of the IMU support board had to be mounted to the right side of the board to avoid collision with the new camera. To compensate for this, the BNO055 chip was rotated 90 degrees counterclockwise from the original design. Female pin headers were added for access to 3.3V, ground, SCL, SDA lines. A specialty cable was created for connecting the IMU support board to the RPi.

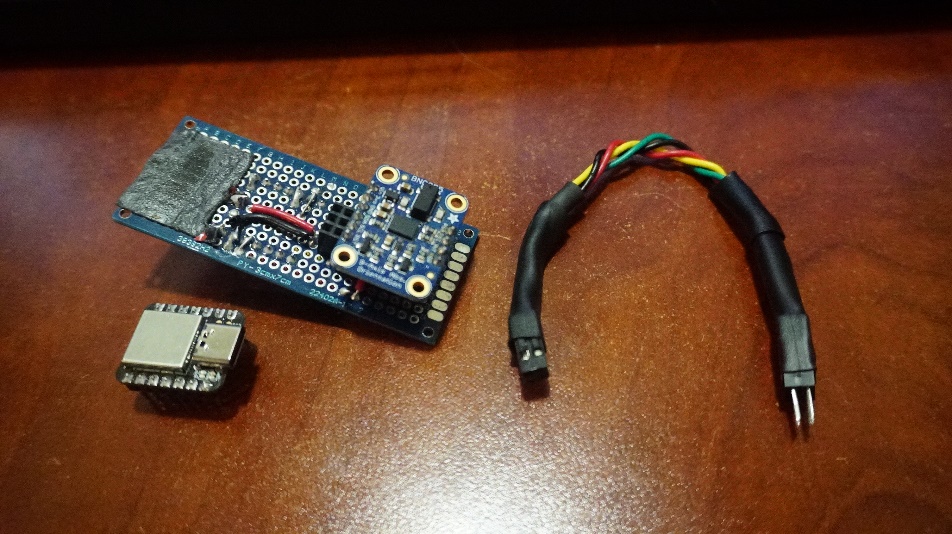


Figure – IMU Support Board with Connection Cable (MCU uninstalled)

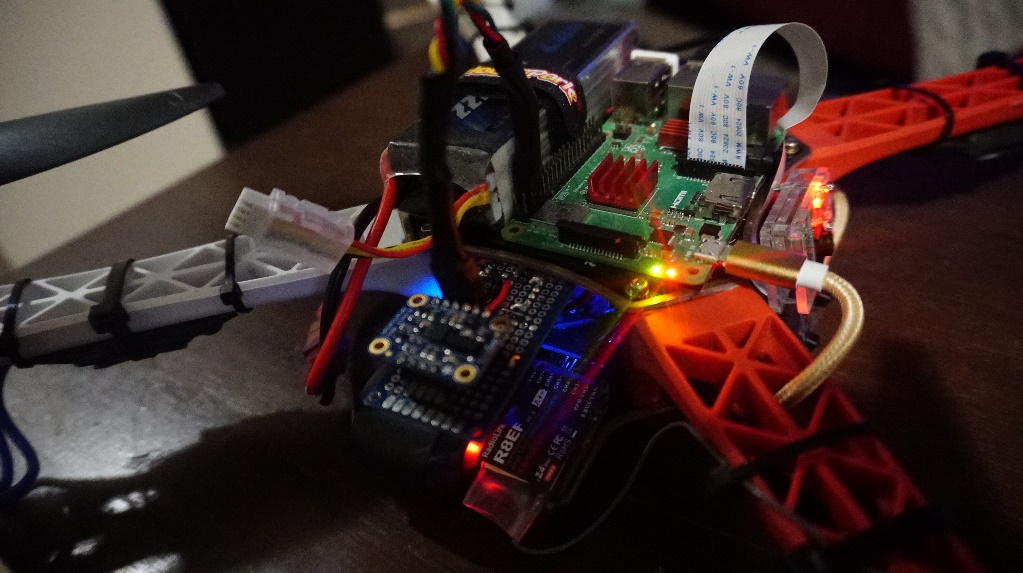


Figure – IMU Support Board Mounting Position

The following diagram describes the schematic of the IMU support board.

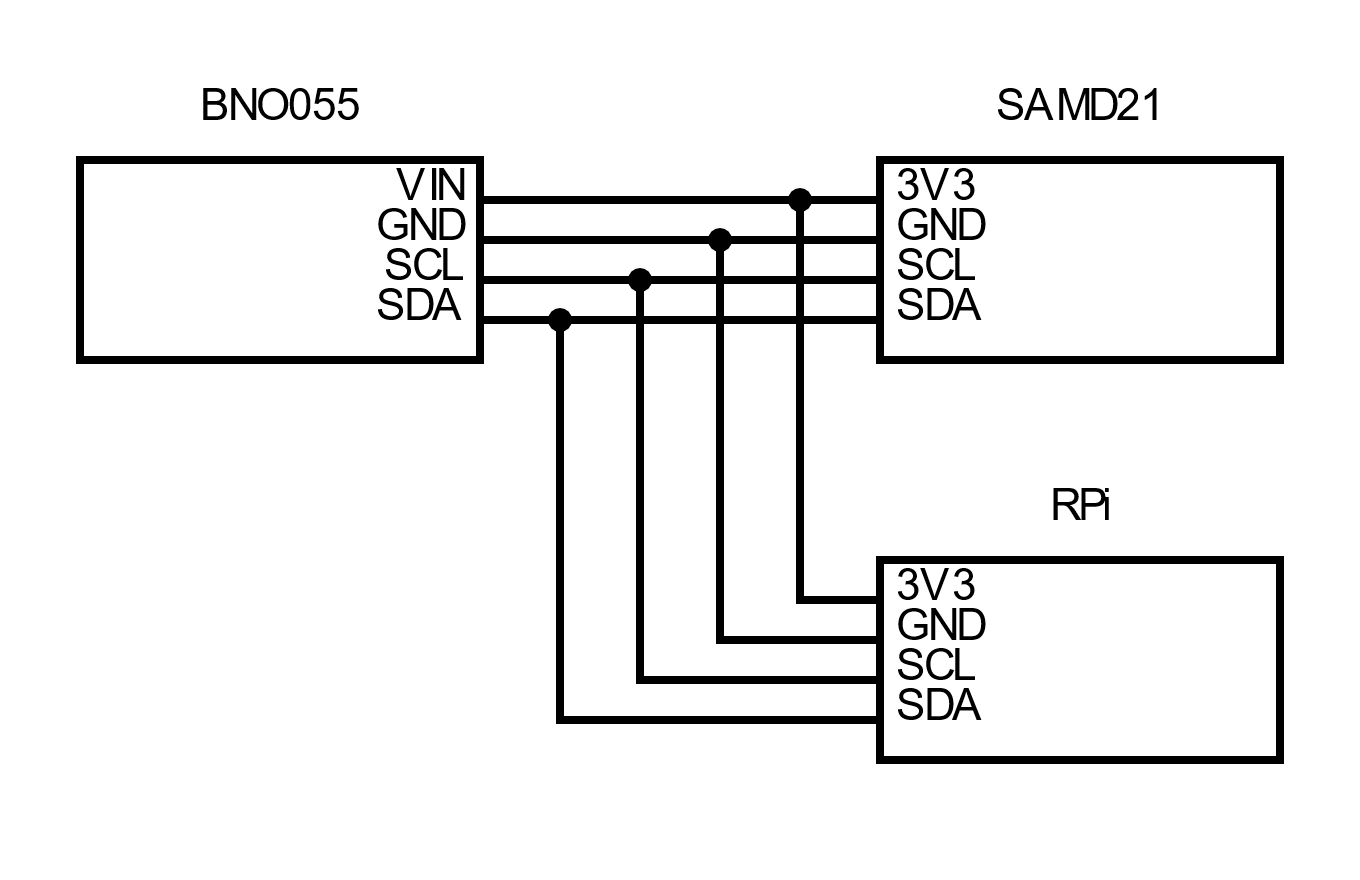


Figure – IMU Support Board Schematic

## 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.
* I2CThread: transmits the global buffer via I2C as defined by the TX\_RATE.

The I2CThread 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

#### IMU Processing

The RPi (Raspberry Pi) software interfaces with the MCU via I2C. The I2C Address 0x8 was assigned to the MCU. 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. The processing performed is just a straightforward pass-thru with no calculations performed.

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 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/flypi-imu.d

[Install]

WantedBy=multi-user.target-

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

#### Camera Processing

The RPi (Raspberry Pi) software interfaces with the Camera via the Camera Serial Interface (CSI). The OpenCV library is used to handle Camera API calls and image processing operations.

The software then captured footage from the Camera. The rate of capture is constrained by the rate that OpenCV can capture from the Camera.

The RPi loads the processed data into a UDP payload buffer to be transmitted on the Network. Every frame captured is transmitted in a single UDP packet. To fit within the size limitations of a UDP packet, each raw frame is encoded to JPEG format. Special considerations must be taken to balance the quality of the transmitted image and the frame rate; the amount of data transmitted per packet has a negative impact on the rate of transmission. Measurements of the UDP transmit rate were monitored via Wireshark. Images of the generated JPEGs were sampled for quality.

The RPi software then transmits the loaded payload to UDP Multicast IP and port 239.100.100.250:8251. The payload will be received on the Host system software.

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

[Unit]

Description=Drone Camera Service

After=multi-user.target

[Service]

Type=simple

ExecStart=/home/pi/flypi.d/flypi-cam.d

[Install]

WantedBy=multi-user.target

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

#### GPS Processing

The RPi (Raspberry Pi) software interfaces with the GPS Module via USB Serial. The GPS serial device appears as “/dev/ttyAMC0”.

The software then processes the data received. The RPi parses through the serial input based on the NMEA protocol. Only the Latitude and Longitude data is processed, all other values are ignored.

The RPi loads the processed data into a UDP payload buffer to be transmitted on the Network. The lat/long string is directly loaded into the payload.

The RPi software then transmits the loaded payload to UDP Multicast IP and port 239.100.100.250:8252. The payload will be received on the Host system software.

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

[Unit]

Description=Drone GPS Service

After=multi-user.target

[Service]

Type=simple

ExecStart=/home/pi/flypi.d/flypi-gps.d

[Install]

WantedBy=multi-user.target

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

### Host

The Host software executes on a Windows OS. The Host software is written in Python 3 and uses the socket, PySimpleGUI and OpenCV libraries. The Host listens for UDP Multicast messages sent to 239.100.100.250. There are three distinct packets transmitted on different ports. The port number of each packet is described in the following table:

Table – UDP Packet Port Mapping Table

|  |  |
| --- | --- |
| Packet | Port |
| IMU | 8250 |
| CAMERA | 8251 |
| GPS | 8252 |

The packets are parsed and displayed on a GUI window appropriate to the input data. If invalid data is transmitted, the value(s) are dashed out “--“ or the display is blacked out with a red X (INOP).

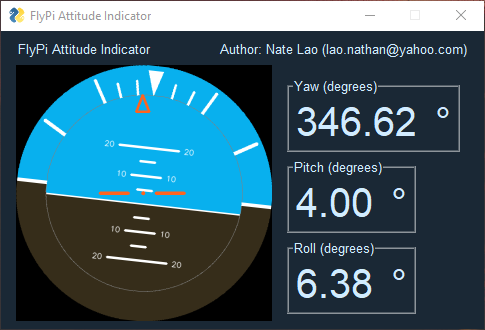


Figure – Attitude Indicator GUI



Figure – GPS Monitor GUI

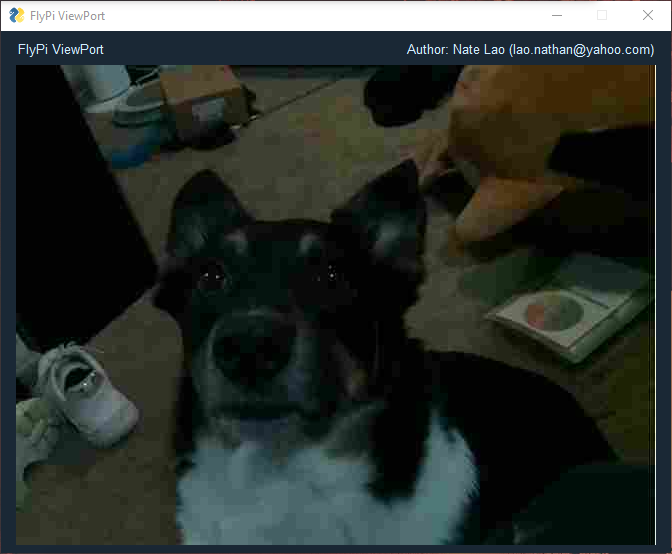


Figure – Camera Viewport GUI

# Implementation

The source code of the FlyPi Drone 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>

// ----- SYSTEM LIBRARIES ----- //

#include <Wire.h>

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

#include "include/imu\_def.hpp"

#include "include/imu\_sensor.hpp"

const uint8\_t SLAVE\_I2C\_ADDRESS = 0x8; /\*\* @note I2C Address for Arduino\*/

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 intialize 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);

  }

}

UBaseType\_t PRIORITY\_I2C\_TASK = tskIDLE\_PRIORITY + 3;

TaskHandle\_t Handle\_I2CTask;

static void I2CThread(void \*pvParameters)

{

  while (1)

  {

    Wire.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);

  // Setup I2C

  Wire.begin(SLAVE\_I2C\_ADDRESS);

  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);

  xTaskCreate(I2CThread,    "I2C Transmission",    256, NULL, PRIORITY\_I2C\_TASK,    &Handle\_I2CTask);

  // 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);

}

namespace GPS

{

    /// @todo to be defined

    void process();

}

namespace CAM

{

    /// @todo to be defined

    void process();

}

#endif

### protocol.hpp

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

/// @file protocol.hpp

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

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

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

#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 <unistd.h>

#include <wiringPiI2C.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 Defines the I2C Address for the Arduino IMU Controller.

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

const int SLAVE\_I2C\_ADDRESS = 0x8;

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

/// @brief Main Driver for FlyPi IMU process

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

int main()

{

    ssize\_t retval = 0;

    // ----- INPUT OBJECTS ----- //

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

    int i2c\_fd = wiringPiI2CSetup(SLAVE\_ADDRESS);

    IMU::PAYLOAD imu\_payload;

    // ----- OUTPUT OBJECTS ----- //

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

    PROTOCOL::IMU\_DATA imu\_data;

    // ----- CONTROL LOOP ----- //

    while (true)

    {

        // ----- INPUT ----- //

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

        /// @note so far, syncing has not been an issue and can be validated in process()

        /// @note kinda risky...

        retval = read(i2c\_fd, &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);

}

### cam.cpp

#include <opencv2/core.hpp>

#include <opencv2/videoio.hpp>

#include <opencv2/highgui.hpp>

#include <stdio.h>

#include "protocol.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 = 8251;

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

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

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

const int PROCESS\_PORT = 8001;

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

/// @brief Defines the compression parameters for each frame.

/// @note Special care has to be done to ensure that the output image sent

/// through UDP does not exceed the UDP size limit. This has to be done by a

/// case by case basis and can verified by calling buffer.size().

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

const char \*IMAGE\_ENCODING = ".jpg";

const int IMAGE\_PARAM = cv::IMWRITE\_JPEG\_QUALITY;

const int IMAGE\_QUALITY = 15;

int main(int, char \*\*)

{

    // ----- OUTPUT OBJECTS ----- //

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

    //--- INITIALIZE VIDEOCAPTURE ----- //

    cv::VideoCapture cap;

    cv::Mat frame;

    std::vector<uchar> buffer;

    int deviceID = 0;        // 0 = open default camera

    int apiID = cv::CAP\_ANY; // 0 = autodetect default API

    cap.open(deviceID, apiID); // open selected camera using selected API

    // Validate Camera open

    if (!cap.isOpened())

    {

        printf("ERROR! Unable to open camera\n");

        return -1;

    }

    else

    {

        printf("Starting camera transmission -> %s:%d\n", TARGET\_IP, TARGET\_PORT);

    }

    // Setup Image Compression

    std::vector<int> COMPRESSION\_PARAMS;

    COMPRESSION\_PARAMS.push\_back(IMAGE\_PARAM);

    COMPRESSION\_PARAMS.push\_back(IMAGE\_QUALITY);

    // ----- CONTROL LOOP ----- //

    while (true)

    {

        // ----- READ RAW CAMERA FRAME ----- //

        cap.read(frame);

        // ----- ENCODE FOOTAGE ----- //

        cv::imencode(".jpg", frame, buffer, COMPRESSION\_PARAMS);

        // ----- OUTPUT ----- //

        udp->transmit(buffer.data(), buffer.size());

    }

    if (udp > 0) delete udp;

    return 0;

}

### gps.cpp

// Based on https://www.electronicwings.com/raspberry-pi/gps-module-interfacing-with-raspberry-pi

#include <stdio.h>

#include <string.h>

#include <errno.h>

#include <wiringPi.h>

#include <wiringSerial.h>

#include "protocol.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 = 8252;

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

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

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

const int PROCESS\_PORT = 8002;

int main()

{

    // ----- OUTPUT OBJECTS ----- //

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

    int serial\_port;

    char dat, buff[100], GGA\_code[3];

    unsigned char IsitGGAstring = 0;

    unsigned char GGA\_index = 0;

    unsigned char is\_GGA\_received\_completely = 0;

    if ((serial\_port = serialOpen("/dev/ttyACM0", 9600)) < 0) /\* open serial port \*/

    {

        fprintf(stderr, "Unable to open serial device: %s\n", strerror(errno));

        return 1;

    }

    if (wiringPiSetup() == -1) /\* initializes wiringPi setup \*/

    {

        fprintf(stdout, "Unable to start wiringPi: %s\n", strerror(errno));

        return 1;

    }

    // Ensure clearout of transmit buffer

    memset(buff, 0, sizeof(buff));

    while (1)

    {

        if (serialDataAvail(serial\_port)) /\* check for any data available on serial port \*/

        {

            dat = serialGetchar(serial\_port); /\* receive character serially \*/

            if (dat == '$')

            {

                IsitGGAstring = 0;

                GGA\_index = 0;

            }

            else if (IsitGGAstring == 1)

            {

                buff[GGA\_index++] = dat;

                if (dat == '\r')

                    is\_GGA\_received\_completely = 1;

            }

            else if (GGA\_code[0] == 'G' && GGA\_code[1] == 'G' && GGA\_code[2] == 'A')

            {

                IsitGGAstring = 1;

                GGA\_code[0] = 0;

                GGA\_code[0] = 0;

                GGA\_code[0] = 0;

            }

            else

            {

                GGA\_code[0] = GGA\_code[1];

                GGA\_code[1] = GGA\_code[2];

                GGA\_code[2] = dat;

            }

        }

        if (is\_GGA\_received\_completely == 1)

        {

            printf("GGA: %s", buff);

            udp->transmit(buff, sizeof(buff));

            is\_GGA\_received\_completely = 0;

        }

    }

    return 0;

}

## Host

### 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 cv2

import attitude\_indicator\_lib as lib

import attitude\_indicator\_gui as gui

sg.theme('Dark Blue')

HYSTERESIS = 5

FONT\_SIZE = ('Any 30')

RAW\_DATA\_COLUMN = sg.Column([

    [sg.Push()],

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

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

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

LAYOUT = [

    [sg.Text('FlyPi Attitude Indicator'), sg.Push(), sg.Text('             Author: Nate Lao (lao.nathan@yahoo.com)')],

    [sg.Image(filename='',key='--viewport--'),RAW\_DATA\_COLUMN]

    ]

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

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

    # Setup Receiver

    receiver = lib.ETHERNET\_ADAPTER\_IMU()

    ########## Event Loop ##########

    persistance = 0

    while True:

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

        event, values = window.read(timeout=0)

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

            break

        # Poll Rx

        payload = receiver.recv()

        # Apply Hysteresis on Input

        if payload != None:

            cache = payload

            persistance = HYSTERESIS

        elif persistance > 0:

            payload = cache

            persistance -= 1

        # Render/Update Values

        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 "--")

            # Update Instrumentation

            img = gui.attitude\_indicator\_instrument(pitch, roll, 50)

            imgbytes = cv2.imencode('.png', img)[1].tobytes() # this is faster, shorter and needs less includes

            window['--viewport--'].update(data=imgbytes)

        else:

            # Invalidate GUI

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

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

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

            # INOP Instrumentation

            img = gui.attitude\_indicator\_inop(50)

            imgbytes = cv2.imencode('.png', img)[1].tobytes() # this is faster, shorter and needs less includes

            window['--viewport--'].update(data=imgbytes)

    # Loop Terminated - Kill Window

    window.close()

else:

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

### attitude\_indicator\_lib.py

# Attitude Indicator Monitor Libraries

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

# Designed for Windows OS

import socket

import struct

import time

# TODO create superclass

class ETHERNET\_ADAPTER\_IMU:

    # 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

    # For Windows, use ipconfig and enter the host's current IP

    INTERFACE = "192.168.4.35"

    PAYLOAD\_SIZE = 16

    PAYLOAD\_FORM = '!fff1B3b'

    # TODO parameterize arguments

    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(('',ETHERNET\_ADAPTER\_IMU.PORT))

        group = socket.inet\_aton(ETHERNET\_ADAPTER\_IMU.BROADCAST)

        iface = socket.inet\_aton(ETHERNET\_ADAPTER\_IMU.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(ETHERNET\_ADAPTER\_IMU.PAYLOAD\_SIZE)

            success = True

        except socket.timeout:

            success = False

        if success:

            # Unpack payload

            payload\_fields = struct.unpack(ETHERNET\_ADAPTER\_IMU.PAYLOAD\_FORM, payload)

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

            return yaw, pitch, roll, validity

        else:

            return None

    def send(self, payload):

        return self.sock.sendto(payload, (ETHERNET\_ADAPTER\_IMU.BROADCAST, ETHERNET\_ADAPTER\_IMU.PORT))

import sys

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

    error\_msg = lambda : print("ERROR: Invalid arguments expect --recv or --send")

    if len(sys.argv) == 2:

        receiver = ETHERNET\_ADAPTER\_IMU()

        if (sys.argv[1].strip() == "--recv"):

            while True:

                print(receiver.recv())

                time.sleep(0.1)

        elif (sys.argv[1].strip() == "--send"):

            yaw = 0.0

            pitch = 0.0

            roll = 0.0

            yaw\_inc = True

            pitch\_inc = True

            roll\_inc = True

            while True:

                payload = struct.pack(ETHERNET\_ADAPTER\_IMU.PAYLOAD\_FORM,yaw,pitch,roll,0xff,0,0,0)

                receiver.send(payload)

                yaw += 0.1 if yaw\_inc else -0.1

                pitch += 0.5 if pitch\_inc else -0.5

                roll += 0.2 if roll\_inc else -0.2

                if yaw < -45.0: yaw\_inc = True

                elif yaw > 45.0: yaw\_inc = False

                if roll < -45.0: roll\_inc = True

                elif roll > 45.0: roll\_inc = False

                if pitch < -45.0: pitch\_inc = True

                elif pitch > 45.0: pitch\_inc = False

                print(f"sending yaw={yaw} pitch={pitch} roll={roll}")

                time.sleep(0.001)

        else:

            error\_msg()

    else:

        error\_msg()

### attitude\_indicator\_gui.py

#!/usr/bin/env python

import cv2

import numpy as np

INST\_DIMENSION = 512

INST\_SKY\_COLOR = (238,176,8)

INST\_GROUND\_COLOR = (26,45,54)

INST\_MARKER\_COLOR = (255,255,255)

INST\_INDICATOR\_COLOR = (30,100,255)

def attitude\_indicator\_inop(scale):

    output = np.zeros((INST\_DIMENSION,INST\_DIMENSION,3), np.uint8)

    cv2.line(output, (0,0),  (INST\_DIMENSION,INST\_DIMENSION),(0,0,255),2)

    cv2.line(output, (0,INST\_DIMENSION),  (INST\_DIMENSION,0),(0,0,255),2)

    if scale < 100 and scale > 0:

        scale\_dim = int(INST\_DIMENSION \* scale / 100.0)

        scale\_dim = (scale\_dim, scale\_dim)

        output = cv2.resize(output, scale\_dim, interpolation=cv2.INTER\_AREA)

    return output

def attitude\_indicator\_instrument(pitch: float, roll: float, scale:int = 100):

    output = np.zeros((INST\_DIMENSION,INST\_DIMENSION,3), np.uint8)

    output = \_\_render\_pitch(output, pitch)

    output = \_\_render\_roll(output, roll)

    if scale < 100 and scale > 0:

        scale\_dim = int(INST\_DIMENSION \* scale / 100.0)

        scale\_dim = (scale\_dim, scale\_dim)

        output = cv2.resize(output, scale\_dim, interpolation=cv2.INTER\_AREA)

    return output

# Inner/Outer Circle Instrument Mask

\_\_inner\_circle\_mask = np.zeros((INST\_DIMENSION,INST\_DIMENSION), np.uint8)

cv2.circle(\_\_inner\_circle\_mask,\

        (INST\_DIMENSION // 2, INST\_DIMENSION // 2), \

        7 \* (INST\_DIMENSION // 18), 255, -1) # Diameter is 7/9 -> convert to radians

\_\_outer\_circle\_mask = np.zeros((INST\_DIMENSION,INST\_DIMENSION), np.uint8)

cv2.circle(\_\_outer\_circle\_mask,\

        (INST\_DIMENSION // 2, INST\_DIMENSION // 2), \

        INST\_DIMENSION // 2, 255, -1)

cv2.circle(\_\_outer\_circle\_mask,\

        (INST\_DIMENSION // 2, INST\_DIMENSION // 2), \

        7 \* (INST\_DIMENSION // 18), 0, -1) # Diameter is 7/9 -> convert to radians

# Displayed pitch is directly proportional to the raw pitch, whereas 90 degrees is maximum

\_\_pitch\_displacement = lambda pitch: int(INST\_DIMENSION \* (pitch / 90.0))

# Prerendered Roll Indicators

\_\_roll\_x\_coor = lambda radius, roll: (INST\_DIMENSION // 2) + int((radius) \* np.sin(roll \* np.pi / 180.0))

\_\_roll\_y\_coor = lambda radius, roll: (INST\_DIMENSION // 2) - int((radius) \* np.cos(roll \* np.pi / 180.0))

\_\_center\_roll\_ptr = np.array([

    [INST\_DIMENSION // 2 - 3,   (INST\_DIMENSION // 9) + 4 ],    # Top Point

    [INST\_DIMENSION // 2 - 18,  (INST\_DIMENSION // 9) - 45],    # Left Corner

    [INST\_DIMENSION // 2 + 12,  (INST\_DIMENSION // 9) - 45]     # Right Corner

    ],np.int32)

\_\_r10\_roll\_displacement = (\_\_roll\_x\_coor((INST\_DIMENSION / 2) - 30, 10.0),\_\_roll\_y\_coor((INST\_DIMENSION / 2) - 30, 10.0))

\_\_r20\_roll\_displacement = (\_\_roll\_x\_coor((INST\_DIMENSION / 2) - 30, 20.0),\_\_roll\_y\_coor((INST\_DIMENSION / 2) - 30, 20.0))

\_\_r30\_roll\_displacement = (\_\_roll\_x\_coor((INST\_DIMENSION / 2) - 10, 30.0),\_\_roll\_y\_coor((INST\_DIMENSION / 2) - 10, 30.0))

\_\_r60\_roll\_displacement = (\_\_roll\_x\_coor((INST\_DIMENSION / 2) - 10, 60.0),\_\_roll\_y\_coor((INST\_DIMENSION / 2) - 10, 60.0))

\_\_l10\_roll\_displacement = (\_\_roll\_x\_coor((INST\_DIMENSION / 2) - 30, -10.0),\_\_roll\_y\_coor((INST\_DIMENSION / 2) - 30, -10.0))

\_\_l20\_roll\_displacement = (\_\_roll\_x\_coor((INST\_DIMENSION / 2) - 30, -20.0),\_\_roll\_y\_coor((INST\_DIMENSION / 2) - 30, -20.0))

\_\_l30\_roll\_displacement = (\_\_roll\_x\_coor((INST\_DIMENSION / 2) - 10, -30.0),\_\_roll\_y\_coor((INST\_DIMENSION / 2) - 10, -30.0))

\_\_l60\_roll\_displacement = (\_\_roll\_x\_coor((INST\_DIMENSION / 2) - 10, -60.0),\_\_roll\_y\_coor((INST\_DIMENSION / 2) - 10, -60.0))

def \_\_render\_pitch(src: np.ndarray, pitch: float) -> np.ndarray:

    output = src

    pitch\_pos = INST\_DIMENSION // 2 + \_\_pitch\_displacement(pitch)

    # TODO this is kinda inefficient since redrawing all these shapes occur at every frame.

    # For now, this is good enough

    # Outer circle

    outer = np.zeros((INST\_DIMENSION,INST\_DIMENSION,3), np.uint8)

    cv2.rectangle(outer,(0,0),(INST\_DIMENSION,INST\_DIMENSION),INST\_SKY\_COLOR,-1)

    cv2.rectangle(outer,(0,INST\_DIMENSION // 2),(INST\_DIMENSION,INST\_DIMENSION),INST\_GROUND\_COLOR,-1)

    cv2.circle(outer,(INST\_DIMENSION // 2, INST\_DIMENSION // 2), 7 \* (INST\_DIMENSION // 18) + 1, (128,128,128), -1)

    cv2.line(outer, (0,INST\_DIMENSION//2), (INST\_DIMENSION,INST\_DIMENSION //2),INST\_MARKER\_COLOR,6)

    cv2.line(outer, (INST\_DIMENSION//2, INST\_DIMENSION//2), \_\_r10\_roll\_displacement,INST\_MARKER\_COLOR,3)

    cv2.line(outer, (INST\_DIMENSION//2, INST\_DIMENSION//2), \_\_r20\_roll\_displacement,INST\_MARKER\_COLOR,3)

    cv2.line(outer, (INST\_DIMENSION//2, INST\_DIMENSION//2), \_\_r30\_roll\_displacement,INST\_MARKER\_COLOR,5)

    cv2.line(outer, (INST\_DIMENSION//2, INST\_DIMENSION//2), \_\_r60\_roll\_displacement,INST\_MARKER\_COLOR,5)

    cv2.line(outer, (INST\_DIMENSION//2, INST\_DIMENSION//2), \_\_l10\_roll\_displacement,INST\_MARKER\_COLOR,3)

    cv2.line(outer, (INST\_DIMENSION//2, INST\_DIMENSION//2), \_\_l20\_roll\_displacement,INST\_MARKER\_COLOR,3)

    cv2.line(outer, (INST\_DIMENSION//2, INST\_DIMENSION//2), \_\_l30\_roll\_displacement,INST\_MARKER\_COLOR,5)

    cv2.line(outer, (INST\_DIMENSION//2, INST\_DIMENSION//2), \_\_l60\_roll\_displacement,INST\_MARKER\_COLOR,5)

    # Roll Pointer

    output = cv2.fillPoly(output, [\_\_center\_roll\_ptr], INST\_INDICATOR\_COLOR)

    outer = cv2.bitwise\_and(outer, outer, mask=\_\_outer\_circle\_mask)

    # Inner circle

    inner = np.zeros((INST\_DIMENSION,INST\_DIMENSION,3), np.uint8)

    cv2.rectangle(inner,(0,0),(INST\_DIMENSION,INST\_DIMENSION),INST\_SKY\_COLOR,-1)

    cv2.rectangle(inner,(0,pitch\_pos),(INST\_DIMENSION,INST\_DIMENSION),INST\_GROUND\_COLOR,-1)

    cv2.line(inner, (0,pitch\_pos), (INST\_DIMENSION,pitch\_pos),INST\_MARKER\_COLOR,2)

    cv2.line(inner, (INST\_DIMENSION // 2 - 60,pitch\_pos + \_\_pitch\_displacement(-20.0)), (INST\_DIMENSION // 2 + 60,pitch\_pos + \_\_pitch\_displacement(-20.0)),INST\_MARKER\_COLOR,3)

    cv2.putText(inner, '20', (INST\_DIMENSION // 2 - 90,pitch\_pos + \_\_pitch\_displacement(-20.0) + 5),cv2.FONT\_HERSHEY\_SIMPLEX,0.5,INST\_MARKER\_COLOR,1,cv2.LINE\_AA)

    cv2.putText(inner, '20', (INST\_DIMENSION // 2 + 70,pitch\_pos + \_\_pitch\_displacement(-20.0) + 5),cv2.FONT\_HERSHEY\_SIMPLEX,0.5,INST\_MARKER\_COLOR,1,cv2.LINE\_AA)

    cv2.line(inner, (INST\_DIMENSION // 2 - 15,pitch\_pos + \_\_pitch\_displacement(-15.0)), (INST\_DIMENSION // 2 + 15,pitch\_pos + \_\_pitch\_displacement(-15.0)),INST\_MARKER\_COLOR,3)

    cv2.line(inner, (INST\_DIMENSION // 2 - 30,pitch\_pos + \_\_pitch\_displacement(-10.0)), (INST\_DIMENSION // 2 + 30,pitch\_pos + \_\_pitch\_displacement(-10.0)),INST\_MARKER\_COLOR,3)

    cv2.putText(inner, '10', (INST\_DIMENSION // 2 - 60,pitch\_pos + \_\_pitch\_displacement(-10.0) + 5),cv2.FONT\_HERSHEY\_SIMPLEX,0.5,INST\_MARKER\_COLOR,1,cv2.LINE\_AA)

    cv2.putText(inner, '10', (INST\_DIMENSION // 2 + 40,pitch\_pos + \_\_pitch\_displacement(-10.0) + 5),cv2.FONT\_HERSHEY\_SIMPLEX,0.5,INST\_MARKER\_COLOR,1,cv2.LINE\_AA)

    cv2.line(inner, (INST\_DIMENSION // 2 - 15,pitch\_pos + \_\_pitch\_displacement(-5.0)), (INST\_DIMENSION // 2 + 15,pitch\_pos + \_\_pitch\_displacement(-5.0)),INST\_MARKER\_COLOR,3)

    cv2.line(inner, (INST\_DIMENSION // 2 - 15,pitch\_pos + \_\_pitch\_displacement(5.0)),  (INST\_DIMENSION // 2 + 15,pitch\_pos + \_\_pitch\_displacement(5.0)),INST\_MARKER\_COLOR,3)

    cv2.line(inner, (INST\_DIMENSION // 2 - 30,pitch\_pos + \_\_pitch\_displacement(10.0)),  (INST\_DIMENSION // 2 + 30,pitch\_pos + \_\_pitch\_displacement(10.0)),INST\_MARKER\_COLOR,3)

    cv2.putText(inner, '10', (INST\_DIMENSION // 2 - 60,pitch\_pos + \_\_pitch\_displacement(10.0) + 5),cv2.FONT\_HERSHEY\_SIMPLEX,0.5,INST\_MARKER\_COLOR,1,cv2.LINE\_AA)

    cv2.putText(inner, '10', (INST\_DIMENSION // 2 + 40,pitch\_pos + \_\_pitch\_displacement(10.0) + 5),cv2.FONT\_HERSHEY\_SIMPLEX,0.5,INST\_MARKER\_COLOR,1,cv2.LINE\_AA)

    cv2.line(inner, (INST\_DIMENSION // 2 - 15,pitch\_pos + \_\_pitch\_displacement(15.0)),  (INST\_DIMENSION // 2 + 15,pitch\_pos + \_\_pitch\_displacement(15.0)),INST\_MARKER\_COLOR,3)

    cv2.line(inner, (INST\_DIMENSION // 2 - 60,pitch\_pos + \_\_pitch\_displacement(20.0)),  (INST\_DIMENSION // 2 + 60,pitch\_pos + \_\_pitch\_displacement(20.0)),INST\_MARKER\_COLOR,3)

    cv2.putText(inner, '20', (INST\_DIMENSION // 2 - 90,pitch\_pos + \_\_pitch\_displacement(20.0) + 5),cv2.FONT\_HERSHEY\_SIMPLEX,0.5,INST\_MARKER\_COLOR,1,cv2.LINE\_AA)

    cv2.putText(inner, '20', (INST\_DIMENSION // 2 + 70,pitch\_pos + \_\_pitch\_displacement(20.0) + 5),cv2.FONT\_HERSHEY\_SIMPLEX,0.5,INST\_MARKER\_COLOR,1,cv2.LINE\_AA)

    inner = cv2.bitwise\_and(inner, inner, mask=\_\_inner\_circle\_mask)

    output = cv2.add(output, outer)

    output = cv2.add(output, inner)

    return output

\_\_indicator\_ptr = np.array([

    [INST\_DIMENSION // 2 - 3,   (INST\_DIMENSION // 9) + 4 ],    # Top Point

    [INST\_DIMENSION // 2 - 16,  (INST\_DIMENSION // 9) + 37],    # Left Corner

    [INST\_DIMENSION // 2 + 10,  (INST\_DIMENSION // 9) + 37]     # Right Corner

    ],np.int32)

\_\_indicator\_ptr = \_\_indicator\_ptr.reshape((-1,1,2))

def \_\_render\_roll(src: np.ndarray, roll: float) -> np.ndarray:

    output = src

    # Displayed Roll is inverse of Raw Roll

    M = cv2.getRotationMatrix2D((INST\_DIMENSION/2, INST\_DIMENSION/2),(-1.0) \* roll,1)

    output = cv2.warpAffine(output,M,(INST\_DIMENSION,INST\_DIMENSION))

    # Pitch Indicator

    cv2.line(output, \

        ((INST\_DIMENSION // 9) \* 3, INST\_DIMENSION // 2), \

        ((INST\_DIMENSION // 9) \* 4, INST\_DIMENSION // 2), \

        INST\_INDICATOR\_COLOR, 5)

    cv2.line(output, \

        ((INST\_DIMENSION // 9) \* 5, INST\_DIMENSION // 2), \

        ((INST\_DIMENSION // 9) \* 6, INST\_DIMENSION // 2), \

        INST\_INDICATOR\_COLOR, 5)

    cv2.circle(output, \

        (INST\_DIMENSION // 2 - 2, INST\_DIMENSION // 2),

        4, INST\_INDICATOR\_COLOR, -1)

    # Roll Pointer

    output = cv2.polylines(output, [\_\_indicator\_ptr], True, INST\_INDICATOR\_COLOR, 4)

    return output

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

    pitch = -45.0

    pitch\_inc = True

    roll = 0.0

    roll\_inc = True

    while True:

        cv2.imshow("AI", attitude\_indicator\_instrument(pitch, roll))

        cv2.waitKey(1)

        pitch += 0.5 if pitch\_inc else -0.5

        if pitch < -45.0: pitch\_inc = True

        elif pitch > 45.0: pitch\_inc = False

        roll += 0.2 if roll\_inc else -0.2

        if roll < -45.0: roll\_inc = True

        elif roll > 45.0: roll\_inc = False

    cv2.destroyAllWindows()

### viewport.py

# Attitude Indicator

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

# Designed for Windows OS

# GUI Driver for AI measurements

import PySimpleGUI as sg

import numpy as np

import cv2

import viewport\_lib as lib

sg.theme('Dark Blue')

HYSTERESIS = 10

FONT\_SIZE = ('Any 30')

LAYOUT = [

    [sg.Text('FlyPi ViewPort'), sg.Push(), sg.Text('             Author: Nate Lao (lao.nathan@yahoo.com)')],

    [sg.Image(filename='',key='--viewport--')]

    ]

# RED X IF NO TRANSMISSION RECEIVED

INOP\_SCREEN = np.zeros((480,640,3), np.uint8)

cv2.line(INOP\_SCREEN, (0,0),  (640,480),(0,0,255),2)

cv2.line(INOP\_SCREEN, (0,480),  (640,0),(0,0,255),2)

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

    window = sg.Window('FlyPi ViewPort', LAYOUT, finalize=True)

    # Setup Receiver

    receiver = lib.ETHERNET\_ADAPTER\_CAM()

    persistance = 0

    while True:

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

        event, values = window.read(timeout=0)

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

            break

        # Poll Rx

        payload = receiver.recv()

        # Apply Hysteresis on Input

        if payload != None:

            cache = payload

            persistance = HYSTERESIS

        elif persistance > 0:

            payload = cache

            persistance -= 1

        # Show Live Footage

        if payload != None:

            # NOTE Unfortunately, PySimpleGUI does not support JPG,

            # the footage has to be converted to PNG.

            conversion = np.frombuffer(payload, np.uint8)

            img = cv2.imdecode(conversion, cv2.IMREAD\_UNCHANGED)

            imgbytes = cv2.imencode('.png', img)[1].tobytes() # this is faster, shorter and needs less includes

            window['--viewport--'].update(data=imgbytes)

        else:

            imgbytes = cv2.imencode('.png', INOP\_SCREEN)[1].tobytes() # this is faster, shorter and needs less includes

            window['--viewport--'].update(data=imgbytes)

    # Loop Terminated - Kill Window

    window.close()

else:

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

### viewport\_lib.py

# Attitude Indicator Monitor Libraries

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

# Designed for Windows OS

import socket

import time

# TODO create superclass

class ETHERNET\_ADAPTER\_CAM:

    # The Multicast IP and Port of the Rx Payload

    BROADCAST = "239.100.100.250"

    PORT = 8251

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

    # For Windows, use ipconfig and enter the host's current IP

    INTERFACE = "192.168.4.35"

    PAYLOAD\_SIZE = 50000

    # TODO parameterize arguments

    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(('',ETHERNET\_ADAPTER\_CAM.PORT))

        group = socket.inet\_aton(ETHERNET\_ADAPTER\_CAM.BROADCAST)

        iface = socket.inet\_aton(ETHERNET\_ADAPTER\_CAM.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(ETHERNET\_ADAPTER\_CAM.PAYLOAD\_SIZE)

            success = True

        except socket.timeout:

            success = False

        if success:

            return payload

        else:

            return None

import sys

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

    error\_msg = lambda : print("ERROR: Invalid arguments expect --recv or --send")

    if len(sys.argv) == 2:

        receiver = ETHERNET\_ADAPTER\_CAM()

        if (sys.argv[1].strip() == "--recv"):

            while True:

                print(len(receiver.recv()))

                time.sleep(0.1)

        else:

            error\_msg()

    else:

        error\_msg()

### gps\_monitor.py

# Attitude Indicator

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

# Designed for Windows OS

# GUI Driver for AI measurements

import PySimpleGUI as sg

import gps\_monitor\_lib as lib

sg.theme('Dark Blue')

FONT\_SIZE = ('Any 30')

LAYOUT = [

    [sg.Text('FlyPi GPS Monitor'), sg.Push(), sg.Text('             Author: Nate Lao (lao.nathan@yahoo.com)')],

    [sg.Text('GPS Data',font=FONT\_SIZE), sg.Push(), sg.Text('N/A',key='--data--',font=FONT\_SIZE)]

    ]

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

    window = sg.Window('FlyPi GPS Monitor', LAYOUT, finalize=True)

    # Setup Receiver

    receiver = lib.ETHERNET\_ADAPTER\_GPS()

    persistance = 0

    while True:

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

        event, values = window.read(timeout=0)

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

            break

        # Poll Rx

        payload = receiver.recv()

        # Display GPS Data

        if payload != None:

            window['--data--'].update(payload)

    # Loop Terminated - Kill Window

    window.close()

else:

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

### gps\_monitor\_lib.py

# Attitude Indicator Monitor Libraries

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

# Designed for Windows OS

import socket

import struct

import time

# TODO create superclass

class ETHERNET\_ADAPTER\_GPS:

    # The Multicast IP and Port of the Rx Payload

    BROADCAST = "239.100.100.250"

    PORT = 8252

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

    # For Windows, use ipconfig and enter the host's current IP

    INTERFACE = "192.168.4.35"

    PAYLOAD\_SIZE = 100

    # TODO parameterize arguments

    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(('',ETHERNET\_ADAPTER\_GPS.PORT))

        group = socket.inet\_aton(ETHERNET\_ADAPTER\_GPS.BROADCAST)

        iface = socket.inet\_aton(ETHERNET\_ADAPTER\_GPS.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(ETHERNET\_ADAPTER\_GPS.PAYLOAD\_SIZE)

            success = True

        except socket.timeout:

            success = False

        if success:

            # NOTE this is just a text payload, cleanup and return as string

            return payload.decode('utf-8').replace('\n','').replace('\r','')

        else:

            return None

    def send(self, payload):

        return self.sock.sendto(payload, (ETHERNET\_ADAPTER\_GPS.BROADCAST, ETHERNET\_ADAPTER\_GPS.PORT))

import sys

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

    error\_msg = lambda : print("ERROR: Invalid arguments expect --recv or --send")

    if len(sys.argv) == 2:

        receiver = ETHERNET\_ADAPTER\_GPS()

        if (sys.argv[1].strip() == "--recv"):

            while True:

                print(receiver.recv())

                time.sleep(0.1)

        else:

            error\_msg()

    else:

        error\_msg()

# 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 (RESOLVED)
  + ~~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~~
  + The final project used OpenCV and a custom broadcast protocol. Performance is significantly better.
* Poor Camera Mounting (RESOLVED)
  + ~~The camera will shift or break off at hard landings. Need to find a more secure way to lock position.~~
  + The final project used another camera with a better mount.
* No lat/long (RESOLVED)
  + ~~The xgps GUI did not feature lat/long coordinates. 3D fix was confirmed and zulu time was determined. Possible bug?~~
  + The final project used a custom operation for parsing lat/long that actually works.

# 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.
* OpenCV
  + Do not even try to compile OpenCV baremetal on the Raspberry Pi. Just find a precompiled library for the respective 32 or 64-bit OS.
* 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 FlyPi Drone can be found here:

<https://youtu.be/w4sMzEoYiLk>