YOLO AI Object Detection Drone

Ata Ulas Guler, Nathan Frinier, Dhruv Chaudhary, Mohsin Nadir

20.04.2025

GTA: Yuhang Zhu
Professor: Ryan Beasley

Design Document

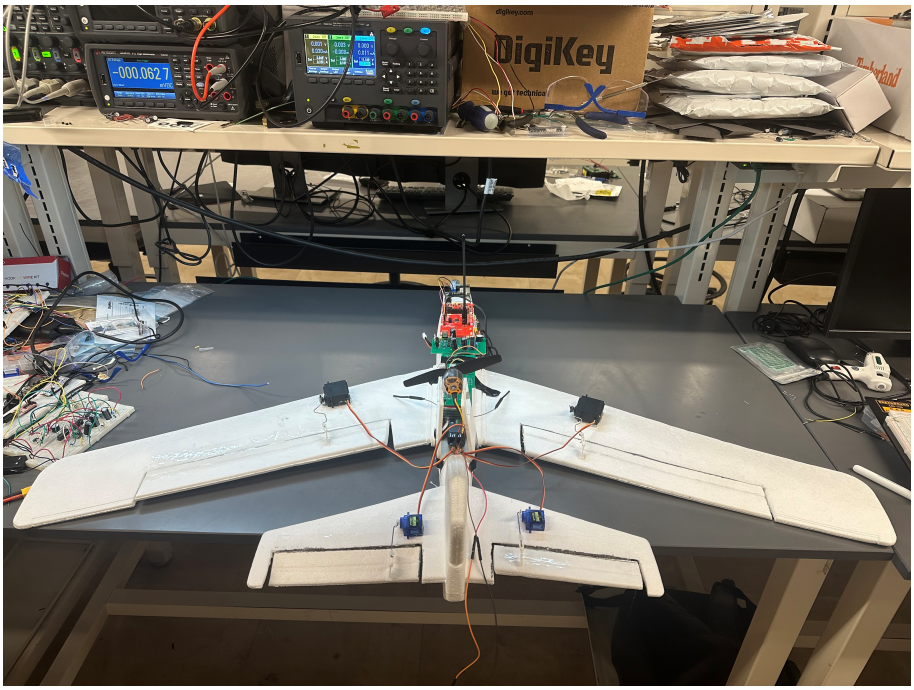


Figure 0.0.1: YOLO AI Drone

Contents

List of Figures

List of Tables

Revision Log

Date	Revision	Changes
02/03/2025	1	• Added Project Title Page • Added Team and Instructor Information • Added Executive Description and User Story • Added Initial System Sketch • Added Design Requirements • Added Requirement Influencing Factors • Added Glossary • Added Initial Block and Activity Diagrams
03/15/2025	2	• Added Subsystems Section • Added Subsystem Specifications and Interactions • Added Subsystem Algorithms and Theory of Operation • Added Parts Lists and Standards • Added PCB Schematics and Layouts • Updated Glossary, Tables, and Figures
04/20/2025	3	• Updated Software Implementation Code and UI • Added Subsystem Integration and Operation Explanation • Updated More Detailed Specifications for All Subsystems • Added Specification Measurements for All Subsystems • Moved Pseudo Code to Appendices • Updated Final Block Diagrams and Visuals • Finalized Glossary, Team Section, and Appendix Structure

Table 1: Table 0.1: Revision Log

Glossary

- **ESC**: Electronic Speed Controller; A circuit that generates a 3-phase wave to drive a brushless DC motor.
- **Aileron** : A flight control surface responsible for roll.
- **Elevator** : A flight control surface responsible for pitch.
- **BLDC**: Brushless DC.
- **LiPo**: Lithium polymer battery chemistry commonly used for drones and remote controlled aircraft.

1 Introduction

1.1 Executive Description

Many modern search and rescue drones face significant limitations: they're often prohibitively expensive, require specialized expertise to operate, and consume substantial power due to continuous video streaming. Additionally, these systems typically depend on constant human monitoring, making them impractical for large-scale or remote missions. Meanwhile, affordable commercial drones, while widely accessible, lack the advanced features necessary for autonomous search operations.

Our solution bridges this gap with a mid to low cost object detection drone that's powered by AI . This low cost sensor drone uses real-time YOLO AI technology to autonomously identify and pinpoint individuals in between dense forests and rubble. By automating detection and opting for long range image transfer, our system minimizes the need for constant oversight, conserves battery life, and significantly improves search efficiency. With this innovation, we aim to transform search and rescue operations into more accessible, cost-effective endeavors.

1.2 User Story

Alex is a search and rescue chief working with the American Red Cross to provide relief from natural disasters primarily for earthquakes. He often helps locate lost individuals in remote areas and wants to investigate ways in which he can improve search times utilizing technology. He has heard of drones used for deployment to dangerous areas; however, the current drones available for such missions are prohibitively expensive and require significant expertise to operate. Additionally, there is no way to utilize new technologies that increase efficiency and detection without buying a military grade drone.

To solve this problem, Alex envisions a solution: an affordable and easy-to-use mid to low cost drone that still uses advanced object detection features. This system would use YOLO AI for real-time video analysis and a long range communications setup to help locate people in need.

Alex also wants the system to reduce the need for constant supervision by automatically detecting and highlighting objects of interest. With this solution, Alex can streamline search and rescue missions straight to a laptop, making them more accessible, efficient, and resource-intensive, ultimately saving more lives.

2 Design Requirements

2.1 Requirements

1. The system must come as a full setup of drone, remote controller, ground station kit, and software download.
2. The system must have access to a video feed so it can do object detection using AI/ML algorithms.
3. The system must feature efficient power consumption.
4. The battery must be easily rechargeable to support frequent operational use.
5. All components must be lightweight enough to avoid significantly impacting the flight performance.
6. The object detection system must be responsive to enable real-time data processing.
7. Decrease the time it takes to search for and locate targets to increase efficiency.
8. Object detection by YOLO should be operable under different environmental conditions, such as night, fog, and debris.
9. The system should be able to adapt to diverse geographic locations: urban, remote, disaster areas.
10. Must follow international drone and RF regulations for massive deployment.
11. UI should support multilingual languages for rescue teams around the world.
12. Shall consider ethical considerations, including privacy and security in AI-driven searches.
13. Easily affordable and accessible by humanitarian organizations and first emergency responders.
14. AI shall complement, not replace human judgment and rescue workers.
15. Power-efficient hardware will allow operation over longer times while reducing impact on the environment.
16. Shall use only commercially available low-cost hardware to make it more affordable.

2.2 Factors influencing requirements

2.2.1 Public Health, Safety, and Welfare

1. Decrease the time it takes to search for and locate targets to increase efficiency.
2. Object detection by YOLO should be operable under different environmental conditions, such as night, fog, and debris.

2.2.2 Global Factors

1. The system should be able to adapt to diverse geographic locations: urban, remote, disaster areas.
2. Must follow international drone and RF regulations for massive deployment.

2.2.3 Cultural Factors

1. UI should support multilingual languages for rescue teams around the world.
2. Shall consider ethical considerations, including privacy and security in AI-driven searches.

2.2.4 Social Factors

1. Easily affordable and accessible by humanitarian organizations and first emergency responders.
2. AI shall complement, not replace human judgment and rescue workers.

2.2.5 Environmental Factors

1. Power-efficient hardware will allow operation over longer times while reducing impact on the environment.

2.2.6 Economical Factors

1. Shall use only commercially available low-cost hardware to make it more affordable.
2. Needs to balance performance and cost to avoid reliance on expensive, military-grade drones.

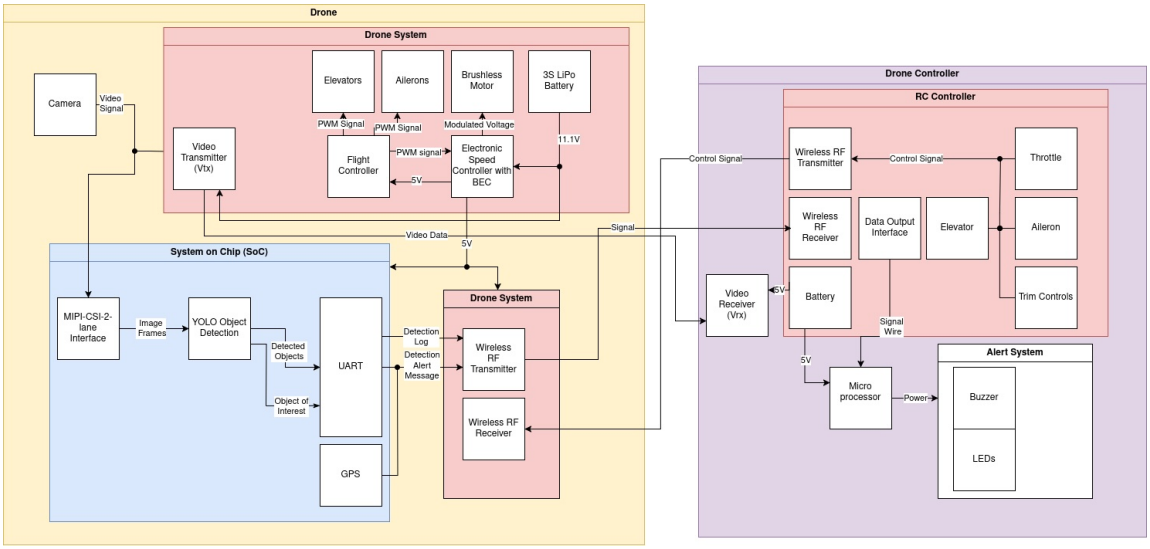


Figure 3.1.1: System Block Diagram

3 System Overview

3.1 System Block Diagram

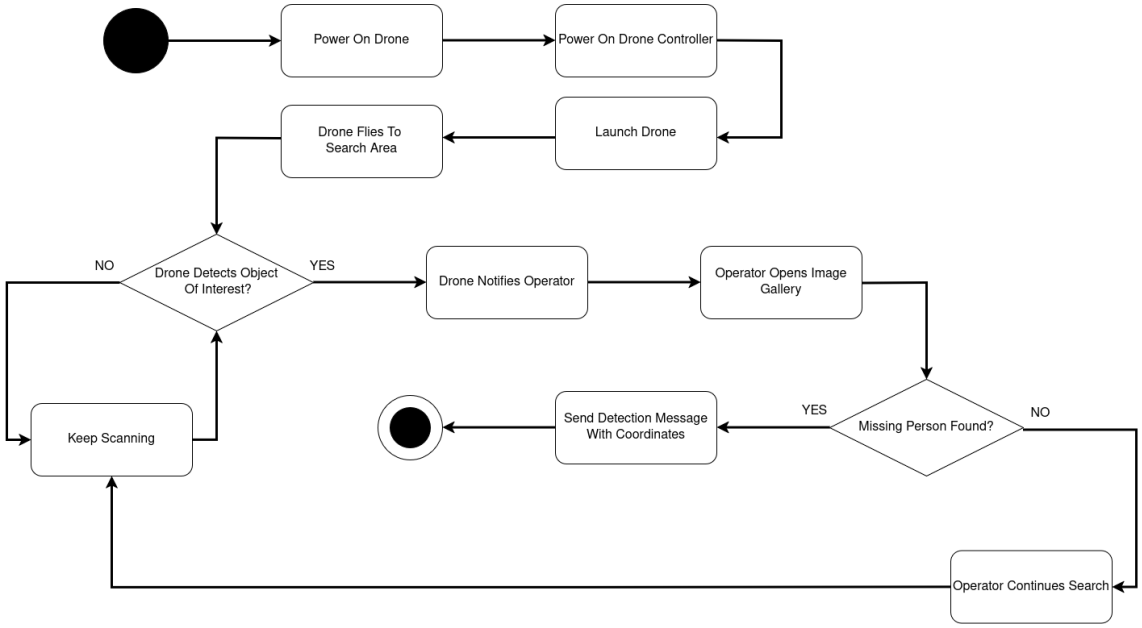


Figure 3.2.1: System Activity Diagram

3.2 System Activity Diagram

3.3 Integration Approach

The system design is split into four subsystems which will be integrated as follows:

Power Subsystem

The power subsystem will provide power to all other subsystems:

- Flight systems will require $3.3V$ to power the flight controller PCB as well as $5V$ to power the Atmega328p of the ESC.
- The signal communications subsystem will need $3.3V$ to power the RF transmitter onboard the drone.
- The software subsystem will require $5V$ to power the Luckfox Pico Ultra and its camera.

Software and Signal Communications Integration

The integration between the software and signal communications subsystems is as follows:

- The Luckfox SoC sends frames through its serial interface when the camera detects a person.
- These frames are passed to the image transmitter.
- The ground station antenna receives the transmitted image and forwards it to the User Interface.

Flight Systems Integration

The integration between the flight systems and power subsystems is structured as:

- Flight systems is integrated with the signal communications system as the flight controller can send barometer and GPS data through the transmitter.
- The flight controller receives power from the power subsystem.
- The ESC, which includes the Atmega328p, is also powered by the power subsystem.

3.4 System Photographs

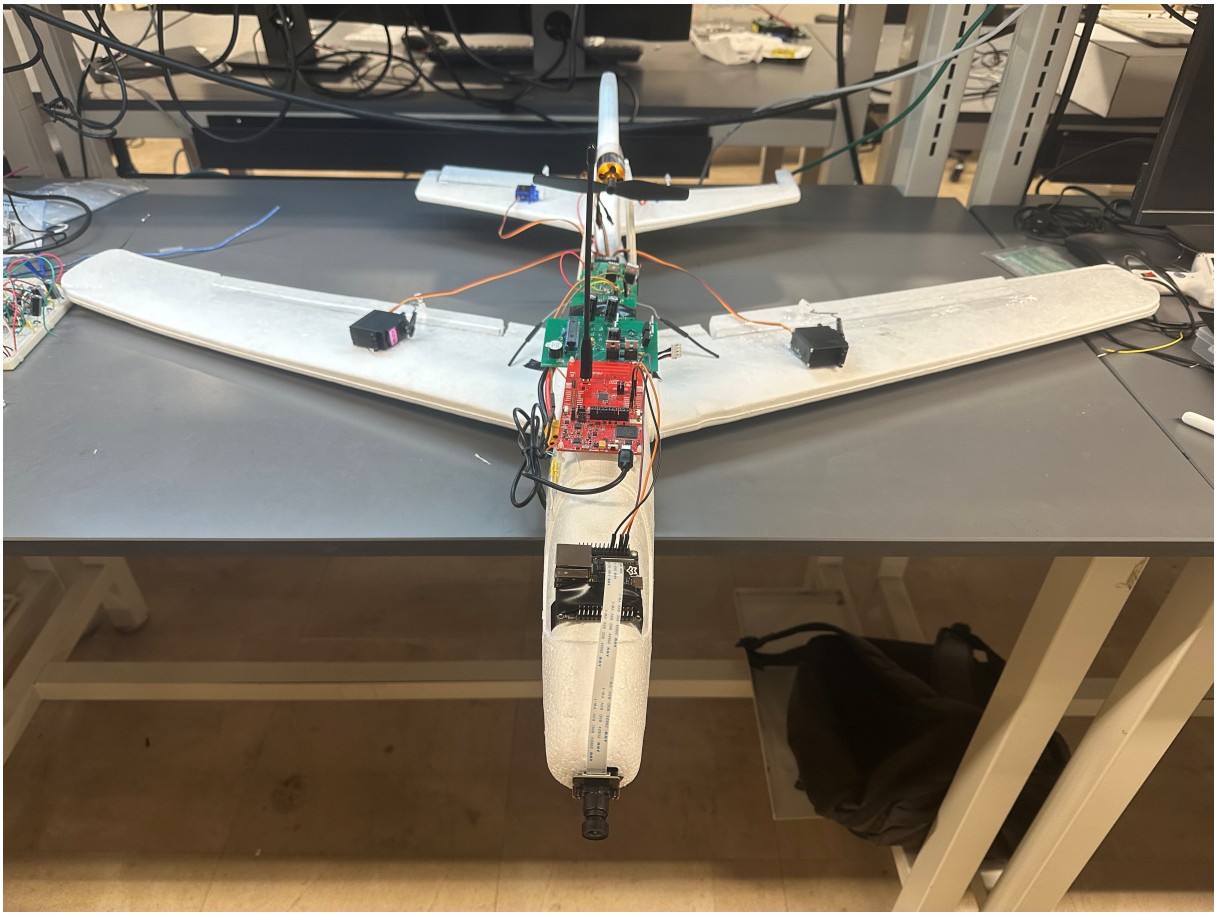


Figure 3.4.1: [Yolo System Front, Flight controller Mid, Power - Hardware In the Back, Front view]

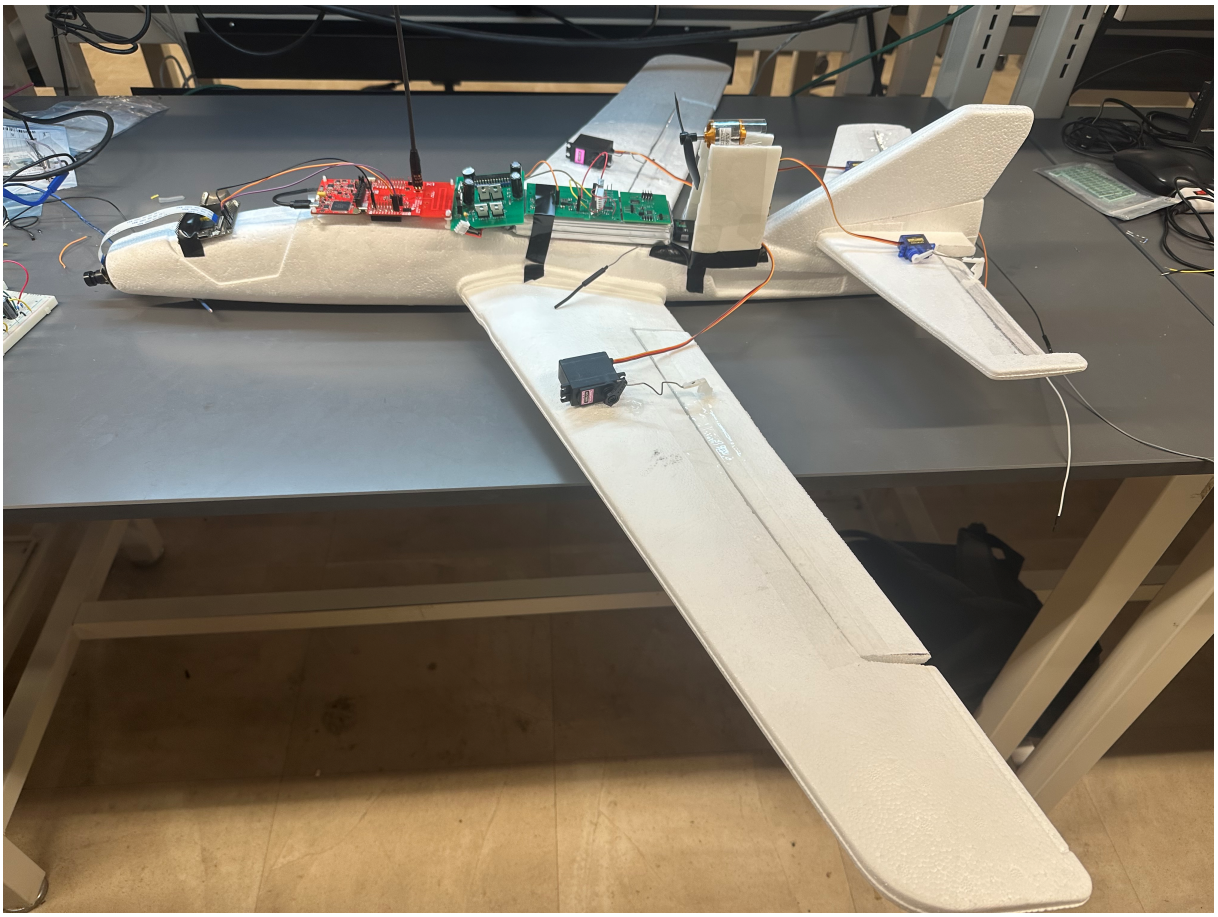


Figure 3.4.2: [Yolo System Front, Flight controller Mid, Power - Hardware In the Back, Side View]

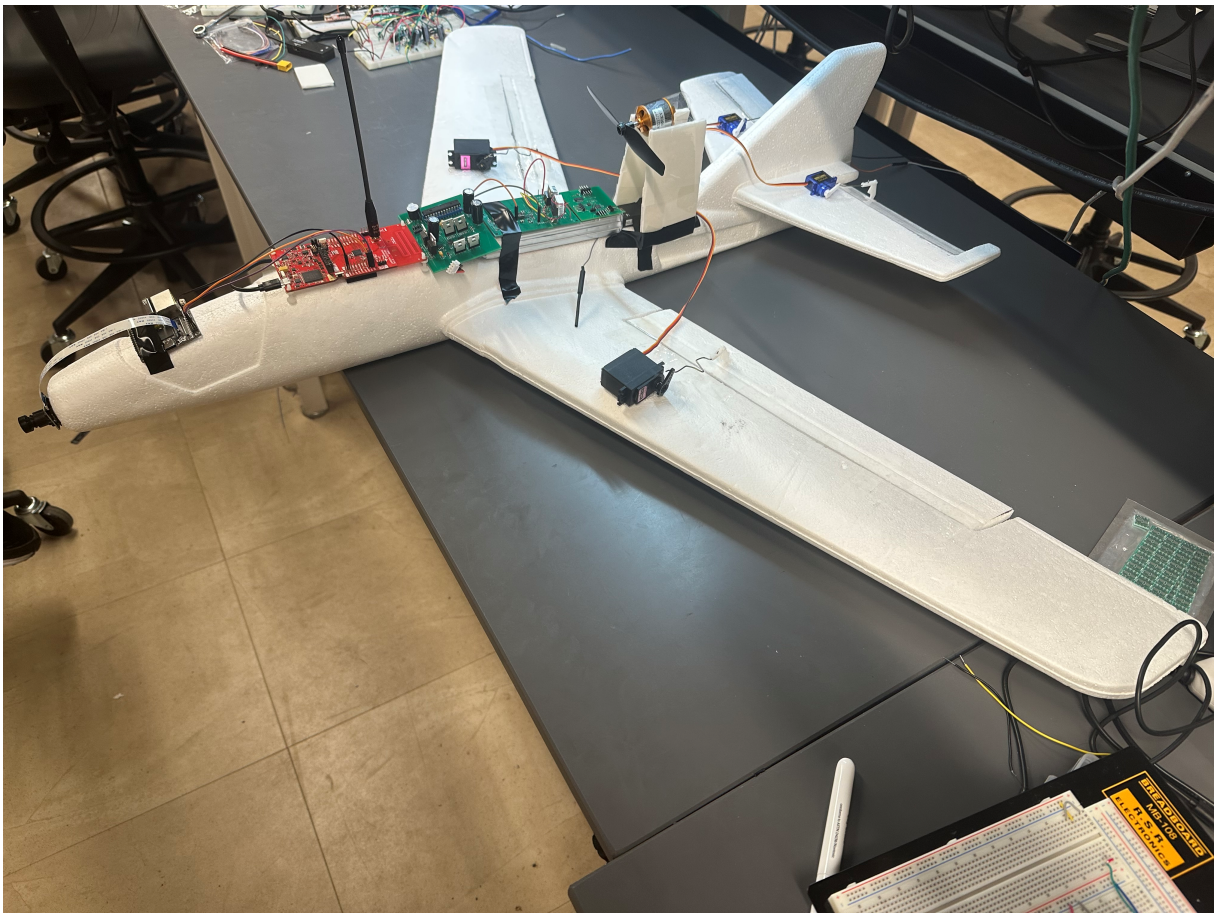


Figure 3.4.3: [Yolo System Front, Flight controller Mid, Power - Hardware In the Back, Diagonal View]

4 Subsystems

4.1 Subsystem 1: Flight Systems

4.1.1 Subsystem Diagrams

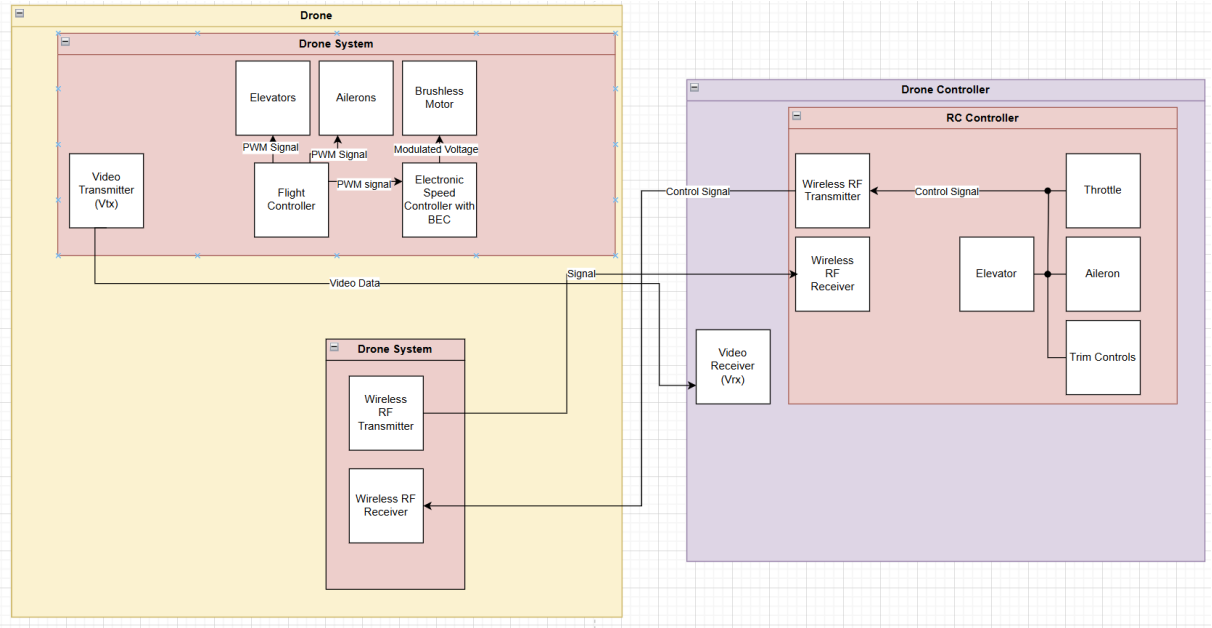


Figure 4.1.1: Subsystem Block Diagram

4.1.2 Specifications

1. The flight controller must be able to power at least 4 servos, 2 for ailerons and 2 for elevators.
2. The flight controller must integrate a PID control system algorithm that can respond to disturbances.
3. The ESC should be able to generate a PWM signal of at least 500Hz to ensure reasonably smooth motor operation.
4. The flight controller servos must have full range of motion from 0 degrees deflection to 180 degrees deflection to ensure that the flight surfaces can move freely.
5. The flight controller should react to disturbance after at most 2s.

4.1.3 Subsystem Interactions

This subsystem interacts with the Signal Communications subsystem as our system must be able to use RF communications to navigate the plane. The subsystem also interacts with the Hardware subsystem as the motor and servos on the plane require specific amounts of power to operate effectively. RF transceivers also require power to operate.

4.1.4 Core ECE Design Tasks

- **ECE 382**: Control Systems, required to program flight controller, which is a closed-loop feedback system.

4.1.5 Schematics

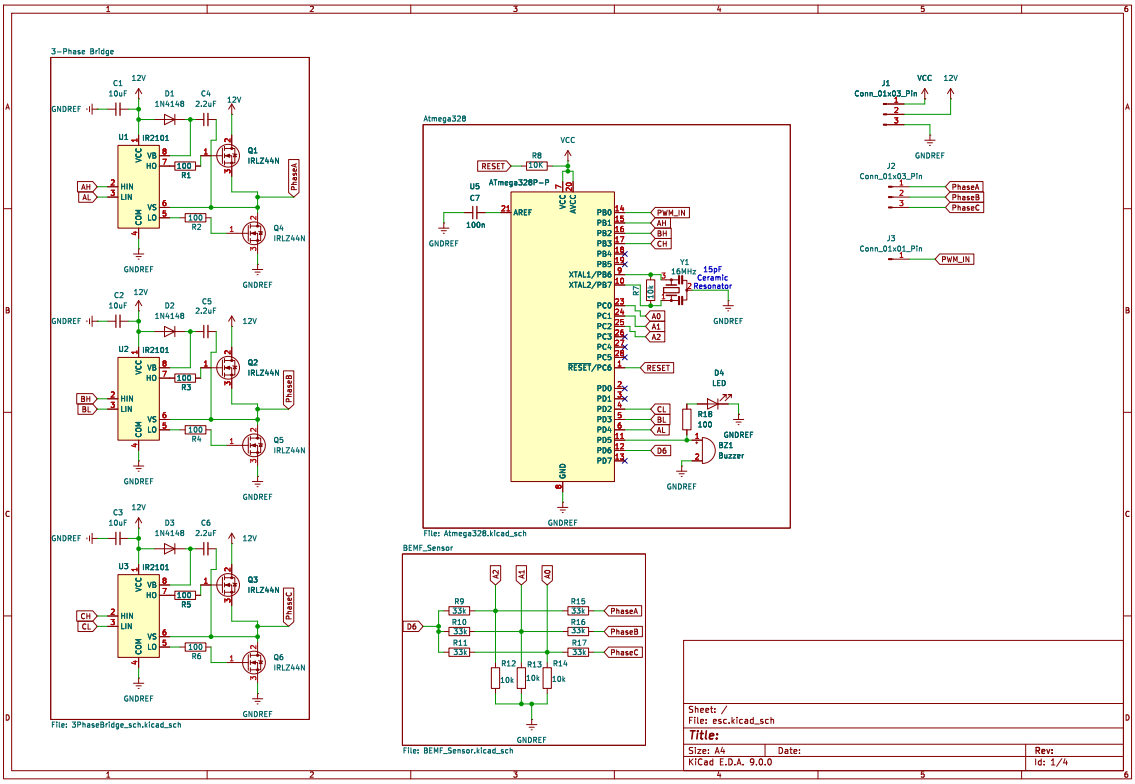


Figure 4.1.2: [BLDC ESC Schematic]

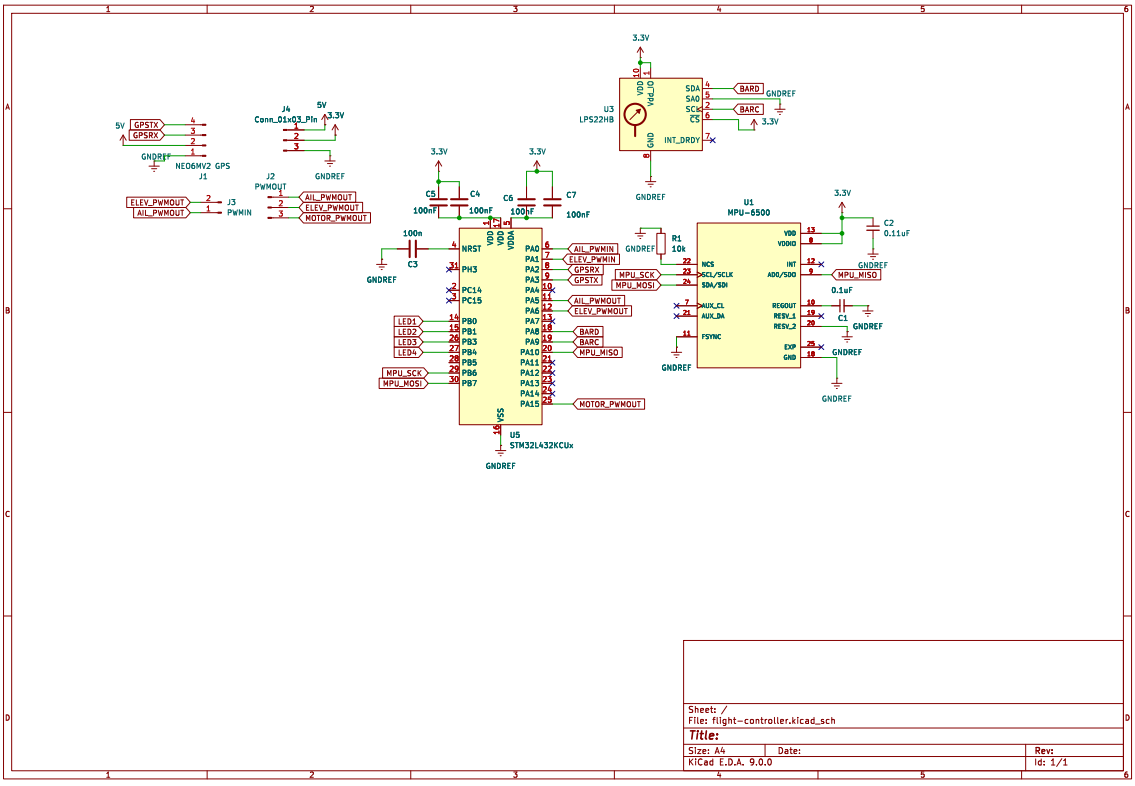


Figure 4.1.3: [Flight Controller Schematic]

4.1.6 Parts

- 2212-980KV Brushless Motor
- 5 blade propeller
- 3S LiPo Battery 11.1V
- MG90S Servos
- Custom-designed 10A ESC circuit
  - Atmega328 microprocessor
  - 3x 1N4148 switching diodes
  - 3x IR2101 gate drivers

  - 6x IRLZ44N power MOSFETs
- Custom-designed flight controller circuit
  - STM32L432KCUx microprocessor
  - LPS22HB barometer
  - MPU-6500 gyroscope and accelerometer
- Jumbo Foam Glider

4.1.7 Algorithm

ESC
The ESC is programmed to be able to read back-EMF values and determine when to switch and commutate the phases. Its behavior is defined by the following procedure:

1. Startup motor (soft start)
2. Back-EMF based commutation
3. Adjust PWM signal in real time using comparator
4. Timeout for safety
5. Phase sequence management

Flight Controller
The flight controller circuit is able to read the gyroscope and accelerometer data and correct for turbulence and other disturbances by sending balancing signals to maintain aircraft stability. Its behavior is defined in the following procedure:

1. IMU calibration
2. PID tuning
3. Fail safe mechanisms (detect timeout, maximum impulse threshold, etc.)
4. Correction algorithm to adjust servos

4.1.8 Theory of Operation

BLDC ESC

1. The ESC will take 3.3 volts to power the Atmega328 microprocessor and 11.1 volts to power the phase circuitry

2. The circuit will take a PWM input signal, and the microprocessor will generate shifted signals for each phase of the motor.

3. The gate drivers will process these signals and the power MOSFETs will provide amplification, creating a PWM pulse capable of powering the motor.

4. Each phase of the motor will be connected through a resistor matrix such that we have a virtual ground reference point and a comparator input that measures voltage at each phase due to back-EMF

5. The microprocessor will read backEMF values in order to keep motor operation smooth and determine when to commutate the signal

Flight Controller

1. The flight controller will take 5V to power the GPS and 3.3V to power the STM32L432KCUx as well as the rest of the circuitry

2. The microprocessor will sample accelerometer and gyroscope data and detect impulsive movements such as shaking from turbulence

3. The barometer and GPS will provide supplementary information to the microprocessor

4. The microprocessor will determine how much to adjust the ailerons and elevators in order to stabilize flight

4.1.9 Specifications Measurement

1. Specification 1: The ESC must have a PWM output frequency of at least 500Hz
   We measured the PWM frequency of each phase for a 980KV motor and recorded a result of 543.3Hz. We measured the PWM frequency of each phase for a 2205KV motor and recorded a frequency of 1.006kHz.

2. Specification 2: The Flight system will provide a 50Hz, 5V pk-pk (200mV tolerance but not to exceed 5V) with a positive duty cycle of 0 to 13%
   We measured 50Hz, 4.9V pk-pk, with a duty cycle of 6.97%.

3. Specification 3: Flight controller must have an accurate mapping (within 5 degrees of accuracy) of duty cycle corresponding to a degree deflection. At 0% duty cycle there should be 0 degrees deflection and at 13% duty cycle there should be 180 degrees deflection.
   We measured 0 degrees deflection at 0% duty cycle and 180 degrees deflection at 13% duty cycle.

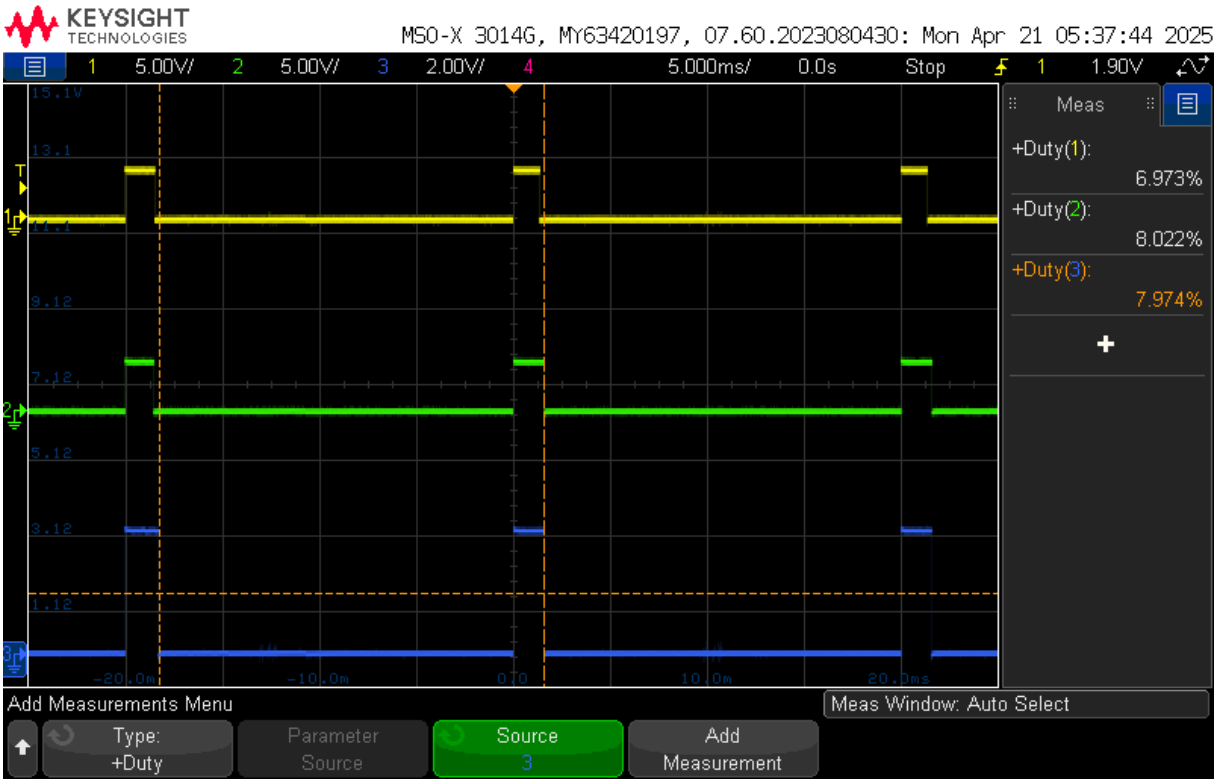


Figure 4.1.4: Flight Controller PWM signal output

4. Specification 3: closed system control loop PID method should have a time allotment of 1.5s between oscillation stabilization period
   Our algorithm has an allotment of 1.5s

4.1.10 Standards

- **I2C**: Inter-Integrated Circuit communication protocol for communicating with MPU-6500 gyroscope and accelerometer as well as LPS22HB barometer
- **FAA**: Compliance with Federal Aviation Administration Regulations
- **UAS**: Compliance with Small Unmanned Aircraft Systems Regulations

4.2 Subsystem 2: Signal Communications

4.2.1 Subsystem Diagrams

Block Diagram

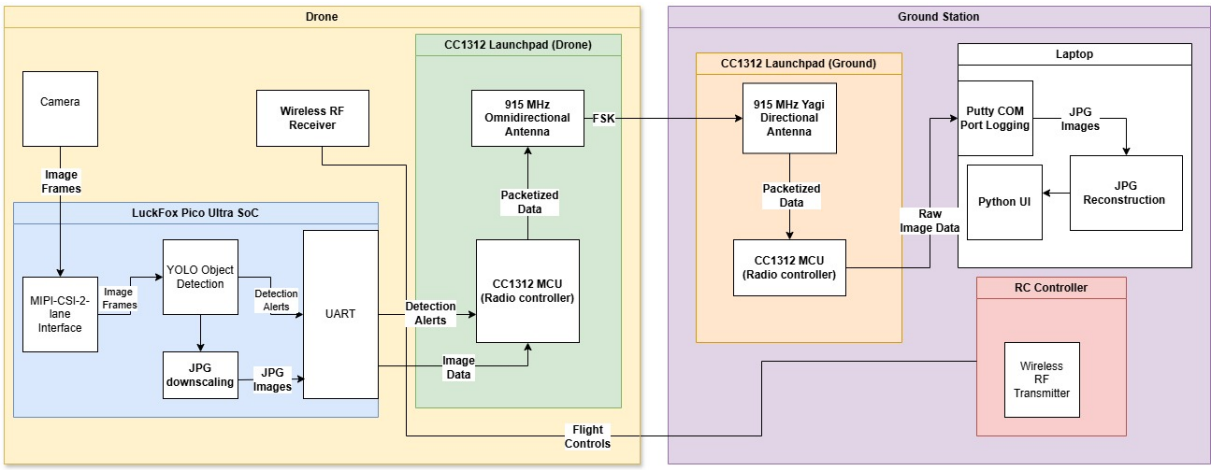


Figure 4.2.1: Subsystem Block Diagram

4.2.2 Specifications

1. Must use PPM or sbus protocol between the flight controller and the receiver utilizing the FS-ia6b receiver and FS-i6X transmitter (remote controller)
   Justification: This flight controller and receiver are already in our team's inventory so using these parts helps extend our budget to other places

2. Must be capable of transmitting Image data via a 915 MHz antenna. This must be at least 500Kbps to offer fast enough in-flight transfers and alerts based on object detection
   Justification: Sub GHz RF connections prioritize distance and ability to operate under sub optimal conditions in terms of terrain, making it the correct choice for this drone connection given it's search and rescue context.

3. Must provide a connection of up to 1 km range atleast
   Justification: Must have this long range connection to account for the drone being piloted into dangerous areas from safety.

4. Must use a custom packet protocol that ensures reliability in connection and no more than 10% packet loss.

Justification: The need for a custom protocol is to ensure a specifically tailored connection for speed. The packet loss must be minimal or countered by Cyclic Redundancy Checks (CRC) to ensure human readability and accuracy within the images.

5. Must use UART at 115200 baud rate to interact between other devices
   Justification: Must use a data transfer rate that can send images fast enough and easily interface with the launchpad boards.

4.2.3 Subsystem Interactions

Communications acts as a bridge for the data between most of our other subsystems. A big role of this subsystem is transferring data between the other systems in a safe and correct way. Communication and interaction between the in-air data and the on ground user interface.

1. The subsystem that interacts the most is Software and Processing. I access the luckfox and get images through UART. I also communicate the JPEGs given to my ground antenna with the software for the User Interface by saving them in a tracked folder.
2. The next subsystem flight systems utilizes the RF connection established with the remote controller (fs-i6x) and the receiver (fs-ia6b). This connection could possibly be used to modify flight control preferences.
3. The final subsystem is the power and hardware system. The Launchpad on the drone side will be powered through a spliced micro-usb cable that is provided with 5V and Ground.

4.2.4 Core ECE Design Tasks

- **ECE 301 Signals and Systems**: [Signal processing and understanding frequency domain]
- **ECE 2k7 Circuits Lab**: [Oscilloscope utilization]
- **ECE 362 Microprocessor Interfacing**: [PWM and PPM as well as UART in practice not just theoretical (301)]
- **ECE 463 Networking**: [Knowledge of data transfer protocols]

4.2.5 Schematics/Pseudo Code

The system has two main pieces of code operated on the Ground Station and the Drone. You open the Ground station code through Code Composer Studio (TI Software) and it sends `CMD_IMG_REQUEST (0x01)` to start the image transfer through the Putty terminal connected to the XDS110 Debug UART on the Launchpad CC1312.

The Drone Launchpad sets up from flash the second the Drone is powered on and listens for an image request from the ground station. Then it sends a single flash image to ensure it's connected and then starts automatically sending images from the Luckfox shell script I wrote that sends images through the antenna as soon as they are saved in the human photos directory within the Luckfox.
The Ground station Debug will print:

- **CMD_IMG_BEGIN (0x10)**: Announces the start of the image.
- **CMD_IMG_DATA (0x02)** in multiple packets: Each contains a small piece of the image.
- **CMD_IMG_END (0x11)**: Indicates the end of the image transfer.

These are the packets from the Drone Antenna. Then it will print a header for jpegs and the hex dump of the binary data and then a end jpg header. These headers allow the python script I wrote on the laptop to reconstruct the jpgs and save them into a folder for the python UI to read from and display them to the user.

In the Appendix there is the main DroneTx code as well as GroundStation code that runs on the Launchpads. There is also the jpg reconstruction script that extracts the images and saves them. There is also the shell script I wrote for the luckfox to activate the uart connection to my antenna.

4.2.6 Parts

- FlySky IA6B Receiver
- FS-i6x Transmitter
- 2 TI CC1312 Launchpads
- Directional Yagi 915MHz Antenna
- Omnidirectional 915MHz antenna

4.2.7 Algorithm

Uses a 2-GFSK Modulation at 1 Mbps 868 Hz RF connection with a custom packet protocol. designed to follow TCP-like reliable custom made connection for image transfers.

4.2.8 Theory of Operation

The Drone side antenna will be always on waiting for an initial image request from the ground station as well as detecting objects using YOLO. Once a person is detected by the YOLO script, the drone side will send an image from the camera on the drone from that moment to the ground station that is operated through a laptop. Then through the COM port that it is connected to, the UART connection will let the laptop reassemble the packets and display the images in the UI. (Pseudo Code section gives the high level overview of the Theory of operation, this is the main theory for the product at face value).

Main Hardware Setup:

For flight controls use the fs i6x RF transmitter as the remote controller on the ground that connects to the 2.4GHz iA6B RF receiver on the drone for streamlined flight controls and initial flight.

Use a camera with an MIPI-CSI2 connection to transmit images to the object detection system on the drone (luckfox) . Then connect to a Launchpad CC1312R microcontroller through UART to separate JPEG compressed images and alert messages into custom packets with headers and send them through a 915 MHz antenna using Frequency Shift Keying (FSK) to an existing ground microcontroller with its own Launchpad CC1312R microcontroller with an antenna to demodulate the signal. Then send the images through USB-Serial to a laptop with a basic user interface for viewing images.

4.2.9 Specifications Measurement

1. Must use PPM or sbus protocol between the flight controller and the receiver utilizing the FS-ia6b receiver and FS-i6X transmitter (remote controller)

   This is met and the remote controller as well as receiver are hooked up to the servos and control surfaces.

2. Must be capable of transmitting Image data via a 915 MHz antenna. This must be at least 500Kbps to offer fast enough in-flight transfers and alerts based on object detection

   The custom PHY settings used within the project and RFStudio config file are set for 868 MHz and 1 Mbps throughput, this setup works well with the 915 MHz antennas purchased.

3. Must provide a connection of up to 1 km range atleast

Figure 4.2.2: Fs-iA6B receiver connected to control surfaces being driven by the remote controller in demo video from FDR.

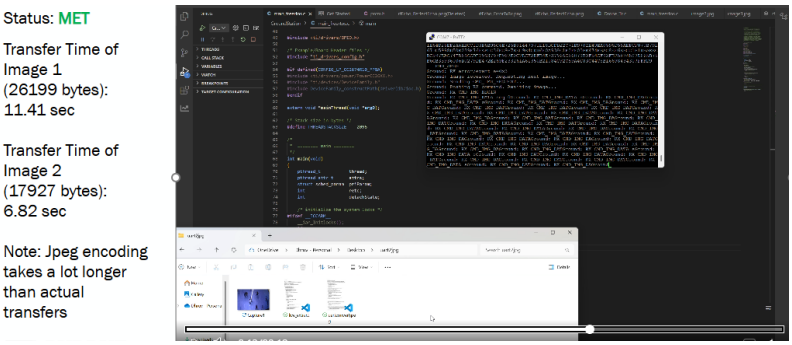


Figure 4.2.3: Speed measurements from video in FDR of image transfers.

Made multiple Connection tests including an 800 meter test, 1.4 km test, and a through multiple walls test. The antenna was able to transfer the image with no noticeable packet loss in all scenarios.

4. Must use a custom packet protocol that ensures reliability in connection and no more than 10% packet loss.

   Checking the pseudo code section shows the packet protocol used. The protocol is indeed custom because of the ability to both requests images with image requests as well as the ability to send image data packets.

5. Must use UART at 115200 baud rate to interact between other devices.

   Pictures show the multiple different UART Debug logs as well as the jpg hex dump

Figure 4.2.4: Map showing the area where distance tests were run. Red X marks where the ground station stayed and the blue circles are the two different locations of the distance tests.

```
=~=~=~=~=~=~=~=~=~=~=~=~= PuTTY log 2025.03.07 09:37:47 =~=~=~=~=~=~=~=~=~=~=~=~=
Ground: UART opened.
Ground: mainThread starting.
Ground: RF opened.
Ground: Frequency set.
Ground: Press Enter to send CMD_IMG_REQUEST...
Ground: UART opened.
Ground: mainThread starting.
Ground: RF opened.
Ground: Frequency set.
Ground: Press Enter to send CMD_IMG_REQUEST...
Ground: Sending CMD_IMG_REQUEST...
Ground: Posting RX command. Awaiting image...
Ground: RX CMD_IMG_BEGIN
Ground: RX CMD_IMG_DATA seq=0Ground: RX CMD_IMG_DATAGround: RX CMD_IMG_DATAGround: RX CMD
CMD_IMG_DATAGround: RX CMD_IMG_DATGround: RX CMD_IMG_DATAGround: RX CMD_IMG_DATAGround: R
CMD_IMG_DATGround: RX CMD_IMG_DATAGround: RX CMD_IMG_DATGround: RX CMD_IMG_DATAGround: RX
CMD_IMG_DATGround: RX CMD_IMG_DATGround: RX CMD_IMG_DATAGround: RX CMD_IMG_DATAGround: RX
```

Figure 4.2.5: This Putty Log shows the output of the ground station COM port as it requests an image and then starts receiving data. These logs show proof of the custom packet protocol and this interface helped log start and stop time for transmissions.

logs.

Figure: Ground Station Completed transfer Debug Output

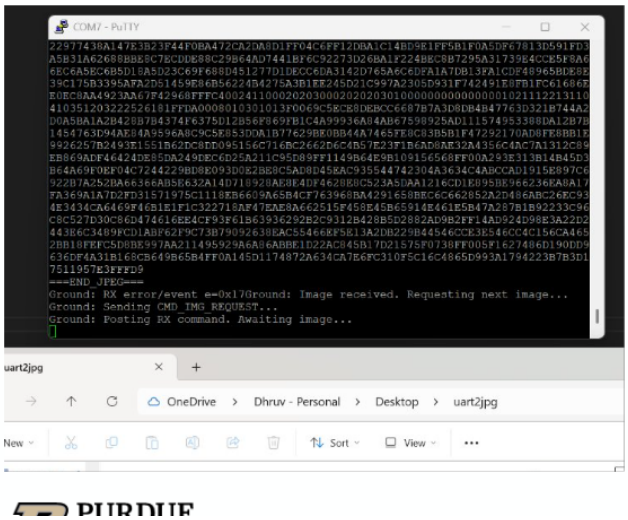


Figure: Ground station Initial Debug Output

```
Ground: RF opened.
Ground: Frequency set.
Ground: Press Enter to send CMD_IMG_REQUEST...
Ground: UART opened.
Ground: mainThread starting.
Ground: RF opened.
Ground: Frequency set.
Ground: Press Enter to send CMD_IMG_REQUEST...
Ground: Sending CMD_IMG_REQUEST...
Ground: Posting RX command. Awaiting image...
```

```
Drone: System start -> flash2 + UART2 slot
Drone: RF opened, freq set.
Drone: Posting RX command...
Drone: Debug UART opened on DIO2/DIO3 at 115200 baud
Drone: Data UART opened on DIO11/DIO12 for image RX

Drone: System start -> flash2 + UART2 slot
Drone: RF opened, freq set.
Drone: Posting RX command...
Drone: slot=0 size=40955 pkts=189
Drone: UART2 slot ready (26199 bytes)
Drone: UART2 slot ready (26199 bytes)
```

Figure: Drone Receive Debug Output

Figure 4.2.6: UART logs and debugs.

4.2.10 Standards

- **Safety level for human exposure**: 0-300GHz
- **FCC**: Uses specific FCC ID given by transceiver

4.3 Subsystem 3: Software and Processing

4.3.1 Subsystem Diagrams

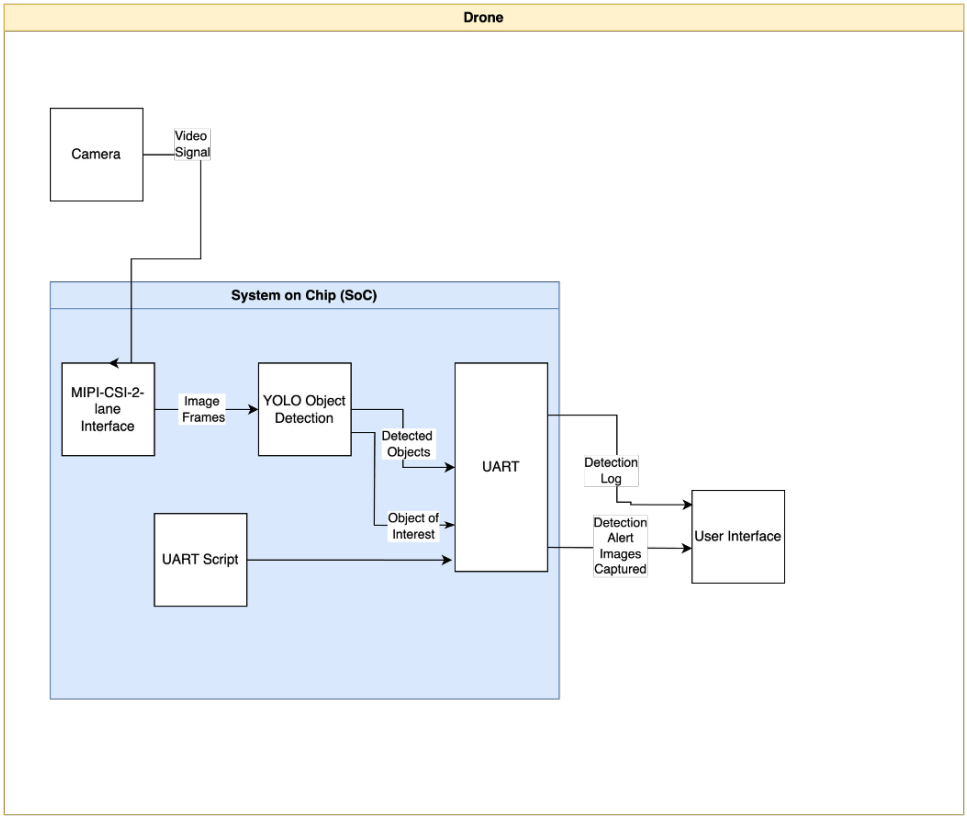


Figure 4.3.1: Subsystem Block Diagram

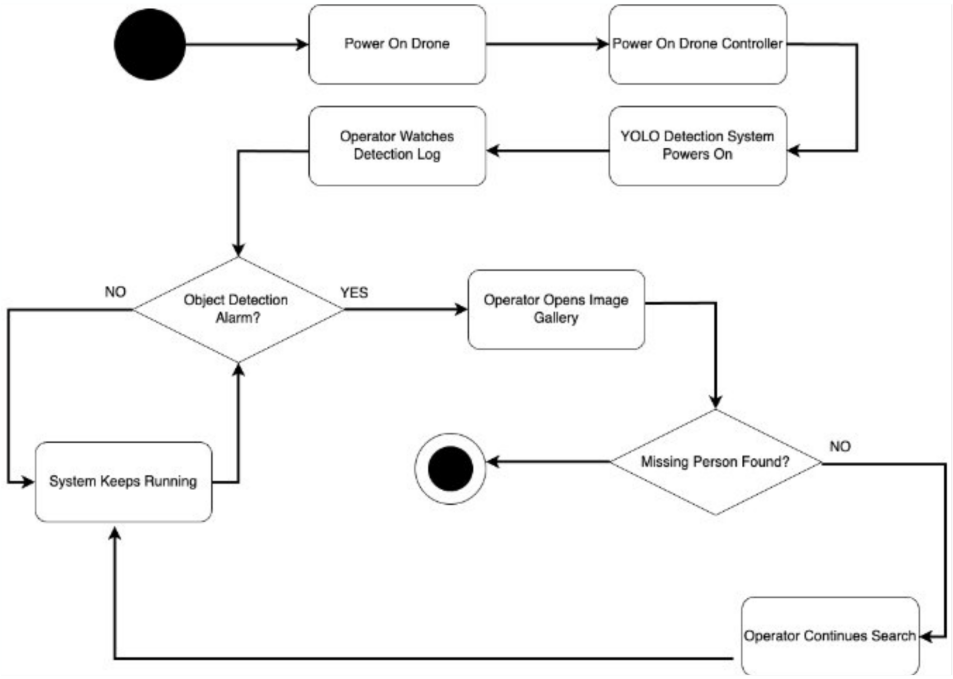


Figure 4.3.2: Subsystem Activity Diagram

4.3.2 Specifications

1. The Luckfox Pico Ultra, equipped with an RV1106 CPU and NPU, will capture video and execute the YOLO algorithm.
2. The YOLOv5 model will achieve real-time object detection at a resolution of 640×480 pixels (480p) with a minimum frame rate of 10 frames per second (FPS).
3. The system will run Buildroot Linux (armv7 architecture), deployed using Rockchip's SocToolkit.
4. The SC3336 3MP Camera Module will provide real-time video capture at a resolution of 2048×1536 pixels.
5. The user interface will be developed in Python which will support English and Turkish languages.
6. Data transfer will be conducted via UART, emulating a serial communication protocol.

4.3.3 Subsystem Interactions

1. Interacts with Communications: Sends staged images and detection logs from the Luckfox Pico Ultra to the antenna; the ground station UI receives both the transmitted image and log from the antenna.

2. Interacts with Flight Controls Supplies GPS-based waypoints to support semi-autonomous drone navigation.

3. Interacts with Hardware: Utilizes the SBC's processing capabilities and 5V power supply to run the YOLO model with GPU acceleration.

4.3.4 Core ECE Design Tasks

1. Embedded Linux Implementation, ECE 46100 – Software Engineering, enables modular development for UI, AI, and alert systems while ensuring efficient debugging and testing.

2. Real-time data transmission, ECE 46300, Introduction to Computer Communication Networks, optimizes video streaming and telemetry using socket programming for low-latency communication.

3. YOLO Object Detection Optimization, ECE 57000 – Artificial Intelligence, improves AI inference efficiency for real-time object detection and automated alerts.

4.3.5 User Interface Implementation

The user interface (UI) is implemented as a standalone desktop application using Python's Tkinter library. It provides a real-time image viewer and control dashboard for interacting with the system's detection pipeline.

The UI features the following components:

- **Main Canvas:** Displays the most recent image from the monitored folder, automatically updating when new images arrive.

- **Input Section:** Includes labeled entry fields for control parameters such as aileron, elevator, throttle, and GPS coordinates. Entries can be updated manually, and changes are logged in real time.

- **Button Panel:** Offers three main actions — capturing a screenshot of the UI, triggering a photo capture command, and toggling the interface language between English and Turkish.

- **Multilingual Support:** The UI can switch between English and Turkish labels and titles with a single click, enhancing accessibility for different users.

The UI continuously monitors a designated image directory:

- The system watches the `Photos` folder, automatically displaying the newest image when detected.
- Screenshots are saved to the `screenshots` directory with timestamped filenames.
- Logging is included to track control changes and system events.

Key technologies used include:

- **Tkinter** for the graphical user interface
- **Pillow (PIL)** for image processing and display
- **Threading** to enable non-blocking folder monitoring

This interface enables seamless interaction with the system by allowing users to monitor incoming data, adjust controls, and perform live captures, all within an integrated environment.

4.3.6 Parts

1. Luckfox Pico Ultra for video analysis and processing.
2. 3MP Camera Module for real-time video input.
3. USB Type-C to Type-A cable.
4. USB Type-C to USB Type-C cable.
5. Ethernet cable (for display purposes).

4.3.7 Algorithm

1. Capture real-time video/image feed from the camera.
2. Process frames with YOLO object detection (optimized for edge computing).
3. If an object is detected, log detected object and bounding box coordinates.
4. Send image frames to image transmitter.

4.3.8 Theory of Operation

The system setup and operation follow a structured, multi-step process to enable YOLOv5-based object detection on the Luckfox Pico Ultra platform:

1. **Flash Buildroot Linux on Luckfox Pico Ultra:** Install the Buildroot-based Linux operating system onto the Luckfox board, preparing the environment for cross-compilation and deployment.
2. **Obtain YOLOv5 and Luckfox-SDK sources:** Download the YOLOv5 repository and the Luckfox SDK to prepare the software components required for model compilation and integration.
3. **Modify YOLOv5 source code:** Update the YOLOv5 codebase to allow toggling bounding boxes on detected objects, enabling compatibility with downstream processing.
4. **Cross-compile YOLOv5 for Luckfox Pico Ultra:** Build the YOLOv5 binary and dependencies for the Luckfox target architecture using a cross-compilation toolchain.
5. **Deploy binary and libraries to Luckfox:** Transfer the compiled YOLOv5 binary and all necessary libraries to the Luckfox board for execution.
6. **Write wrapper script for YOLO inference:** Develop a shell wrapper that runs YOLOv5 inference and automatically saves images when a person is detected.
7. **Write data transmission script:** Create an additional script to transmit images and metadata through UART2 to an image transmission module.
8. **Configure inittab for autostart:** Use `inittab` to ensure all scripts automatically launch on system boot, enabling headless operation.

This pipeline enables automated object detection, image capture, and transmission directly from the embedded system without requiring manual intervention after startup.

4.3.9 Specifications Measurements

- **SPEC:** The Luckfox Pico Ultra, equipped with an RV1106 CPU and NPU, will capture video and execute the YOLOv5 algorithm.

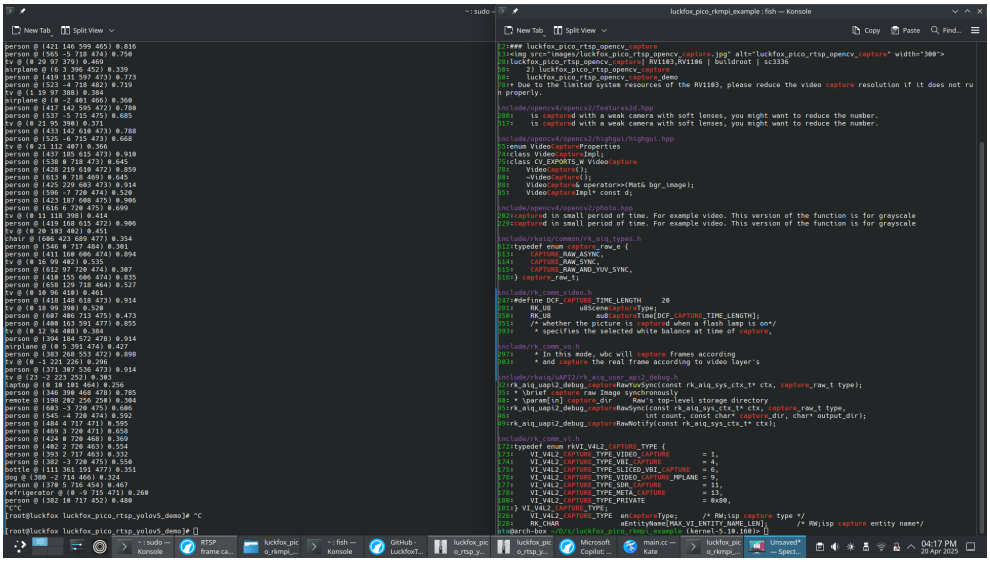


Figure 4.3.3: YOLOv5 terminal output displaying real-time detection of multiple objects such as "person," "tv," and "airplane" with associated confidence scores. Demonstrates successful inference and camera input on the Luckfox Pico Ultra.

- **SPEC:** The YOLOv5 model will achieve real-time object detection at a resolution of 640×480 pixels (480p) with a minimum frame rate of 10 frames per second (FPS).

Figure 4.3.4: Terminal output showing real-time YOLOv5 detections, confirming model inference at the expected frame rate and resolution.

Measurement: The camera was required to support a bitrate exceeding 25,000 kb/s for smooth real-time video feed. The following command was used to validate frame rate and pixel format:

```
ffplay -video_size 640x480 -pixel_format nv12 -framerate 10 -i
    video50.yuv
ffplay version n7.1 Copyright (c) 2003-2024 the FFmpeg developers
  built with gcc 14.2.1 (GCC) 20250207
  configuration: --prefix=/usr --disable-debug --disable-static --
      disable-stripping --enable-amf
  --enable-avisynth --enable-cuda-llvm --enable-lto --enable-
      fontconfig --enable-frei0r --enable-gmp
  --enable-gnutls --enable-gpl --enable-ladspa --enable-libaom --
      enable-libass --enable-libbluray
  --enable-libbs2b --enable-libdav1d --enable-libdrm --enable-
      libdvdnav --enable-libdvdread
  --enable-libfreetype --enable-libfribidi --enable-libglslang --
      enable-libgsm --enable-libharfbuzz
  --enable-libiec61883 --enable-libjack --enable-libjxl --enable-
      libmodplug --enable-libmp3lame
  --enable-libopencore_amrnb --enable-libopencore_amrwb --enable-
      libopenjpeg --enable-libopenmpt
  --enable-libopus --enable-libplacebo --enable-libpulse --enable-
      librav1e --enable-librsvg
  --enable-librubberband --enable-libsnappy --enable-libsoxr --
      enable-libspeex --enable-libsrt
  --enable-libssh --enable-libsvtav1 --enable-libtheora --enable-
      libv4l2 --enable-libvidstab
  --enable-libvmaf --enable-libvorbis --enable-libvpl --enable-
      libvpx --enable-libwebp
  --enable-libx264 --enable-libx265 --enable-libxcb --enable-
      libxml2 --enable-libxvid
  --enable-libzimg --enable-libzmq --enable-nvdec --enable-nvenc --
      enable-opencl --enable-opengl
  --enable-shared --enable-vapoursynth --enable-version3 --enable-
      vulkan
libavutil      59. 39.100 / 59. 39.100
libavcodec     61. 19.100 / 61. 19.100
libavformat    61.  7.100 / 61.  7.100
libavdevice    61.  3.100 / 61.  3.100
libavfilter    10.  4.100 / 10.  4.100
libswscale      8.  3.100 /  8.  3.100
libswresample   5.  3.100 /  5.  3.100
libpostproc    58.  3.100 / 58.  3.100
[rawvideo @ 0x7c7278000c80] Estimating duration from bitrate, this
    may be inaccurate
Input #0, rawvideo, from 'video50.yuv':
  Duration: 00:00:06.00, start: 0.000000, bitrate: 36864 kb/s
  Stream #0:0: Video: rawvideo (NV12 / 0x3231564E), nv12, 640x480,
      36864 kb/s, 10 tbr, 10 tbn
  14.01 M-V:  0.000 fd=   0 aq=    0KB vq=    0KB sq=    0B
```

In this output, our bitrate is recorded as 36864 kb/s.

- **SPEC:** The SC3336 Camera Module will provide real-time video capture at a resolution of 2048×1536 pixels.

  The camera output was successfully obtained using the SC3336 module, producing a real-time video feed through the system. This was validated through terminal logs and visual output. A demonstration video of the live camera feed can be found at: `https://youtu.be/VczKnOCv5yg`.

- **SPEC:** The user interface will be developed in Python which will support English and Turkish languages.

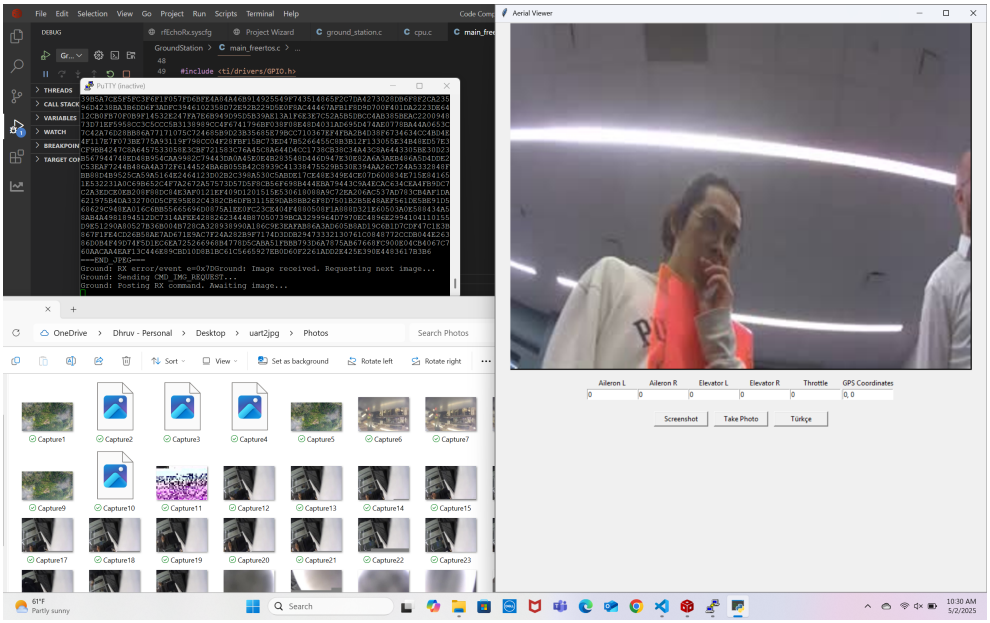


Figure 4.3.5: Subsystem Activity Diagram

- **SPEC:** Data transfer will be conducted via USB, emulating a serial communication protocol.

  The communication subsystem utilizes the UART available on the Luckfox Pico Ultra. UART transfer connection was configured to emulate a serial protocol, allowing structured data exchange between the embedded system and external modules.

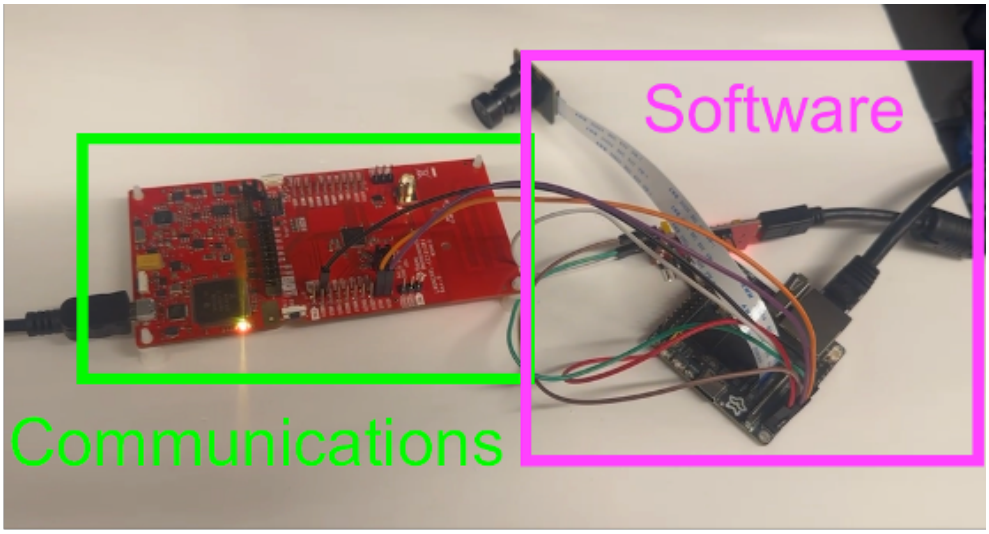


Figure 4.3.6: UART - Luckfox Connection

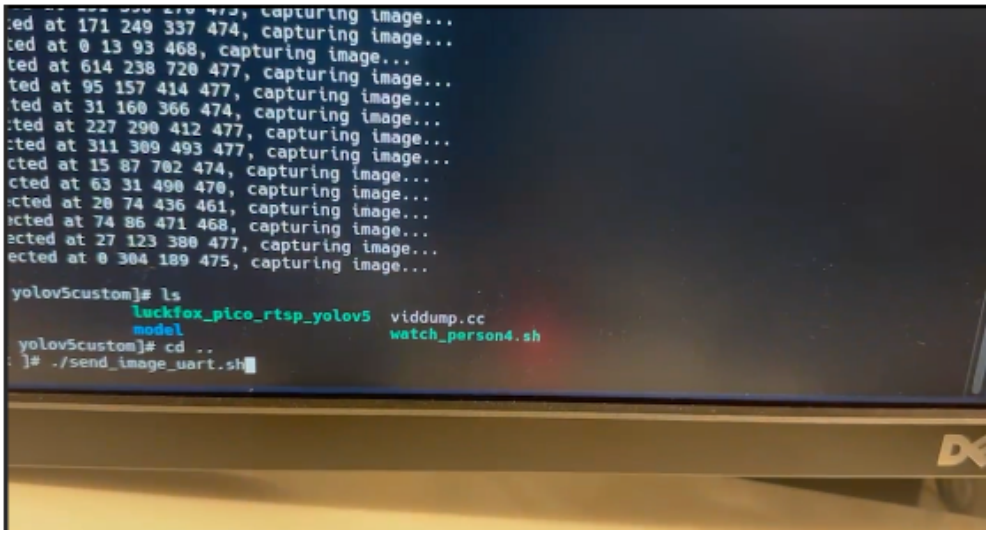


Figure 4.3.7: Script on Luckfox

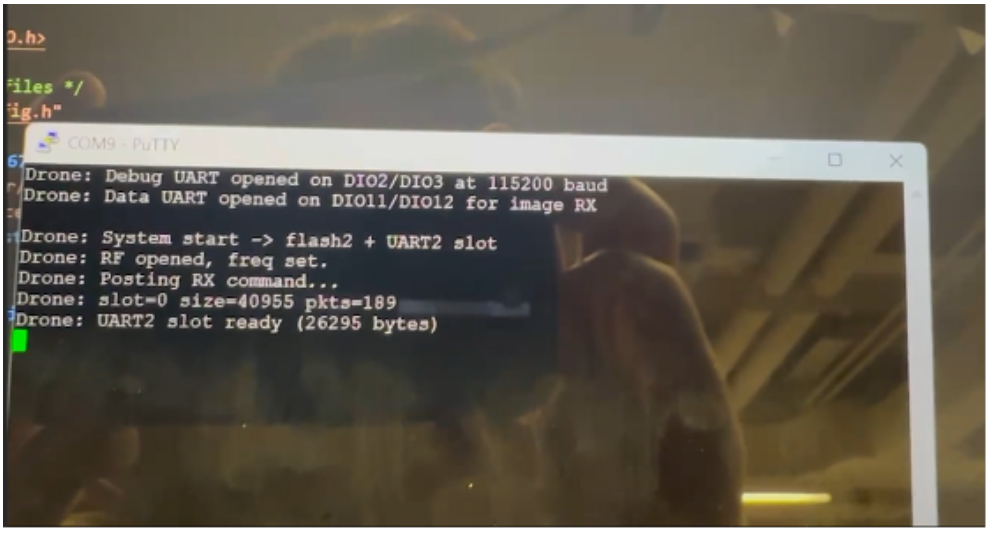


Figure 4.3.8: Image transfer succesful

4.3.10 Standards

1. POSIX Compliance: Ensures compatibility with embedded Linux systems.
2. OpenCV & TensorFlow Standards: Used for real-time image processing.
3. IEEE 802.11 (Wi-Fi Standard): Enables communication with operator UI.

4.4 Subsystem 4: Hardware

4.4.1 Subsystem Diagrams

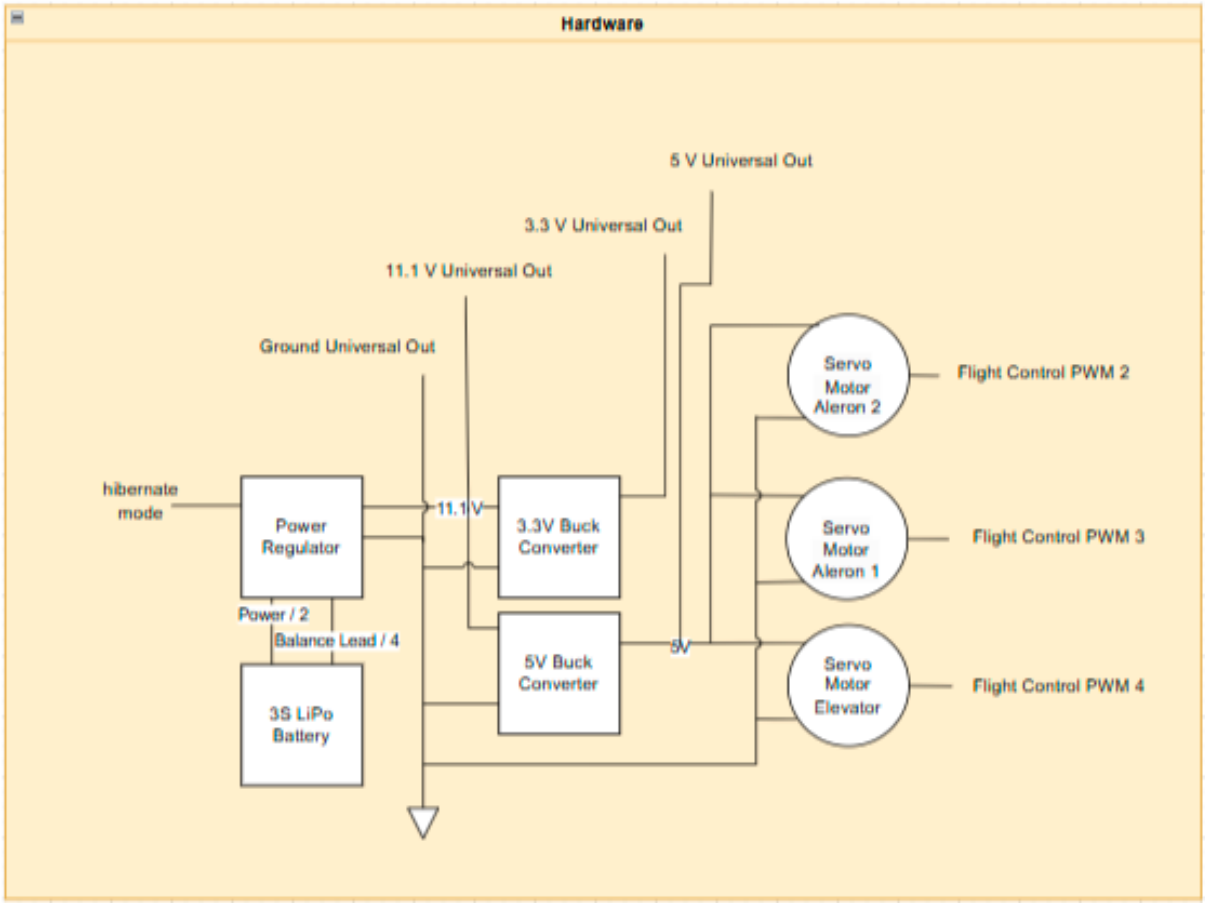


Figure 4.4.1: Hardware Block Diagram

4.4.2 Specifications

1. The system will provide 12 to 10 volts of on-demand power up to 10 amps.
2. Voltage step-downs will be done to 3.3 and 5 volts using buck converters to maximize current output.
3. The buck converters will provide power with a ripple less than 2%.
4. The power regulator will provide cell balancing and keep the cells within 20 millivolts of one another.

5. The servo motors will be driven at five volts and have an working error range of 4%.
6. The LiPo battery will be recharged from the main power pins of the bms and the bms will balance the cells to within 20 millivolts of the average cell voltage.

4.4.3 Subsystem Interactions

This subsystem interacts with all other subsystems as it provides universal power and ground for all subsystems at 11.1, 5 and 3.3 volts. Additionally, this subsystem interacts directly with the Flight Systems subsystem to enable servo motor control (PWM) for elevators and ailerons, and the electronic speed controller control (PWM).

4.4.4 Core ECE Design Tasks

1. Electronic Speed Controller (20002, 36200, 33700) General circuit design with transistors.
2. Buck Converter 3.3 V (20002, 20001, 20008, 20007) General circuit design.
3. Buck Converter 5 V (20002, 20001, 20008, 20007) General circuit design.
4. Power regulator for safe charging (20001, 33700) In-depth circuit design.
5. PCB design (27000, 36200, 33700) In-depth circuit design.

4.4.5 Schematics

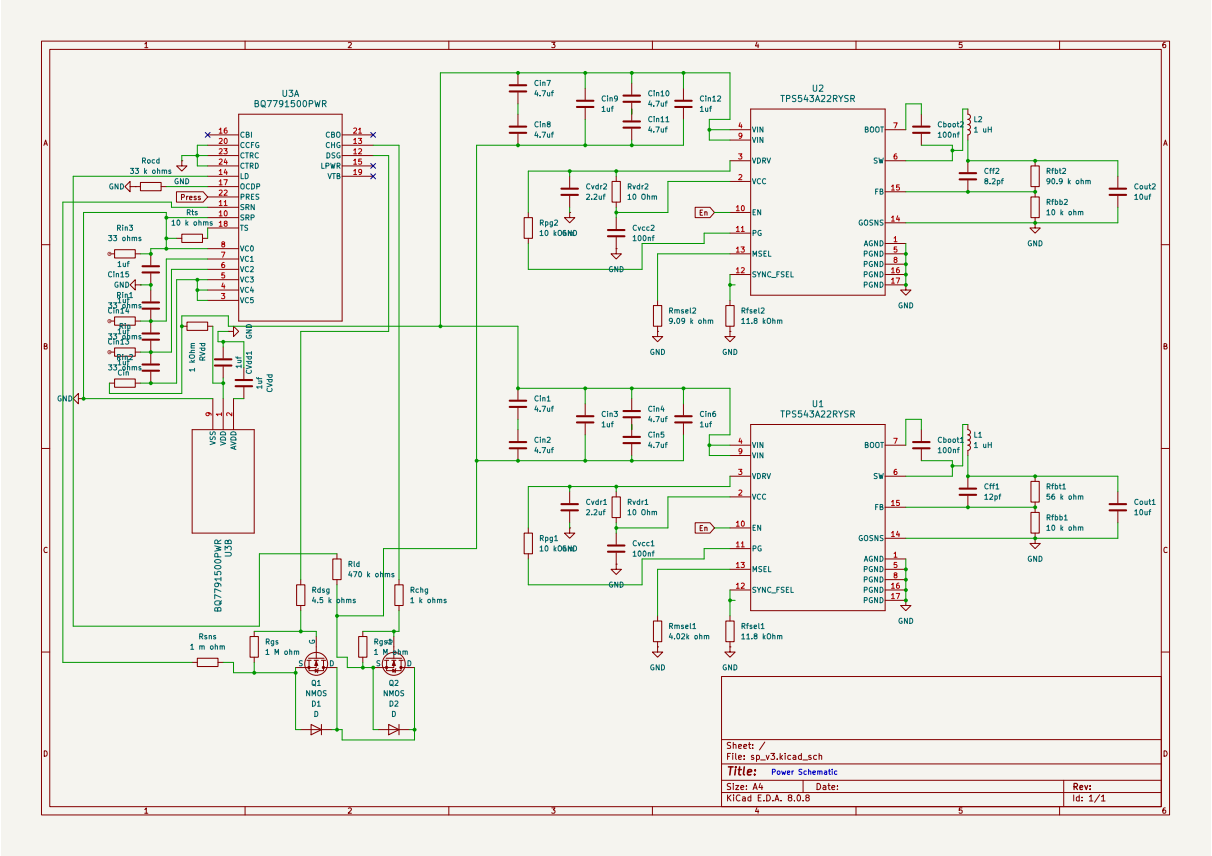


Figure 4.4.2: [Power Schematic]

4.4.6 Parts

1. 3 Servos
2. 3s LiPo
3. 2 Buck converters TPS543A22
4. Power Regulator BQ7791500

4.4.7 Algorithm

A fine-tuned electrical speed controller circuit diagram will need to be calculated to ensure that proper power delivery is delivered depending on the duty cycle of the PWM wave.

A fine-tuned Buck converter circuit diagram will need to be calculated to ensure that the proper voltage and current are delivered.

4.4.8 Theory of Operation

1. The Power Regulator will internally handle cell balancing, voltage protection, and current protection through the BQ7791500PWR chip for both charging and discharging.
2. The hibernate line will be powered or to prevent the system from going into hibernation mode and low when hibernation mode is required.
3. The power regulator will nominally provide 11.1 volts dc to the buck converters when hibernate mode is off. The power regulator will provide 12 to 10 volts in a range of 12 to 10 volts as the battery discharges.
4. The buck converters will provide 3.3 and 5 volts, respectively, at across the out capacitors through the use of the bias circuit of the TPS543A22 chip.
5. The buck converters will provide power when the enable pin is pulled low or left floating.
6. The buck converters will provide power with a very small ripple due to the 1 MHz operating frequency.

4.4.9 Specifications Measurement

1. The system will provide 12 to 10 volts of on-demand power up to 10 amps.
   Measurement: The 3s LiPo battery was measured with a benchtop DMM showing a nominal voltage of 11.1 volts, a maximum voltage of 11.9 volts, and a "dead voltage" of 10.3 volts, at which the battery could no longer power the subsystems. Using a handheld RC watt meter, the maximum discharge current under load was approximately 4 amps. As no additional load demands were present, these results sufficiently meet this specification. The battery performed 0.8% under the expected maximum voltage and 3% above the expected minimum voltage, handling all current demands adequately.
2. Voltage step-downs will be done to 3.3 and 5 volts using buck converters to maximize current output.
   Measurement: The 3.3V step-down was measured on a Breadboard and the pcb, while the 5V step-down was measured on the PCB. The TPS543A22 converter chip on the PCB for the 3.3 is incorrectly biased with the voltage divider causes the average voltage to be read at 3.9 volts. The 5 volt the buck converter works properly. The measurements for the pcbs is displayed in figure 4.4.7 and 4.4.8.

3. The buck converters will have an efficiency greater than 85% and provide power with a ripple less than 2 %.
   Measurement: Efficiency measured by benchtop DMMs was recorded at 82%, slightly below specification. Ripple measurements, conducted repeatedly as solder connections improved, demonstrated gradual reduction. The ripple waveform from the 5V and 3.3 V step-down is presented below in figure 4.4.5.
4. The power regulator will provide cell balancing and keep the cells within 20 millivolts of one another.
   Measurement: Cell voltage balancing was intermittently verified using a benchtop DMM. The largest observed voltage difference was 63 millivolts, averaging around 40 millivolts, exceeding the specified 20 millivolt threshold. All successful testing of this part of the subsystem was done on the bread board.
5. The servo motors will be driven at five volts and have a working error range of 4 %.
   Measurement: Servo motor supply, derived from the 5V step-down output, was measured at an average of 5.5 volts, deviating from the specified 5 volts. This results in an error margin of approximately 10%, far exceeding the specified 4%.
6. The LiPo battery will be recharged from the main power pins of the BMS and the BMS will balance the cells to within 20 millivolts of the average cell voltage.
   Measurement: Intermittent voltage checks with a benchtop DMM showed the largest recorded difference was 63 millivolts, averaging roughly 40 millivolts. This indicates the balancing performance did not meet the specified 20 millivolts criterion. All successful testing of this part of the subsystem was done on the bread board

V load	Units	I load	Units2
11.05	Volts	0	Amps
10.805	Volts	0.05	Amps
10.43	Volts	0.1	Amps
10.4	Volts	0.15	Amps
10.485	Volts	0.2	Amps
10.5	Volts	0.25	Amps
10.38	Volts	0.3	Amps
10.337	Volts	0.35	Amps
10.31	Volts	0.4	Amps
10.35	Volts	0.45	Amps
10.32	Volts	0.5	Amps
10.33	Volts	0.55	Amps
10.32	Volts	0.6	Amps
10.28	Volts	0.65	Amps
10.25	Volts	0.7	Amps
10.329	Volts	0.75	Amps
10.27	Volts	0.8	Amps
10.24	Volts	0.85	Amps
10.25	Volts	0.9	Amps
10.22	Volts	0.95	Amps
10.22	Volts	1	Amps

Figure 4.4.3: 3s LiPo Battery Voltage Measurement

Figure 4.4.4: 5 Volt Load Voltage Measurement

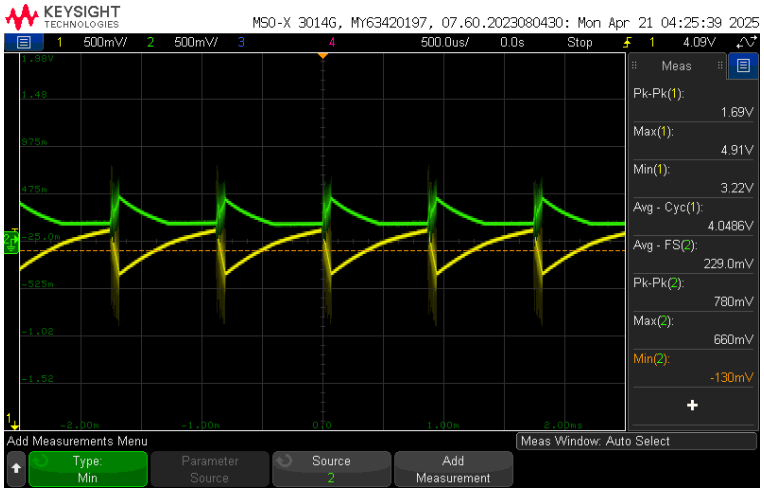


Figure 4.4.5: Breadboard 3.3V Load Test from 11.1V Source at 1 Amp

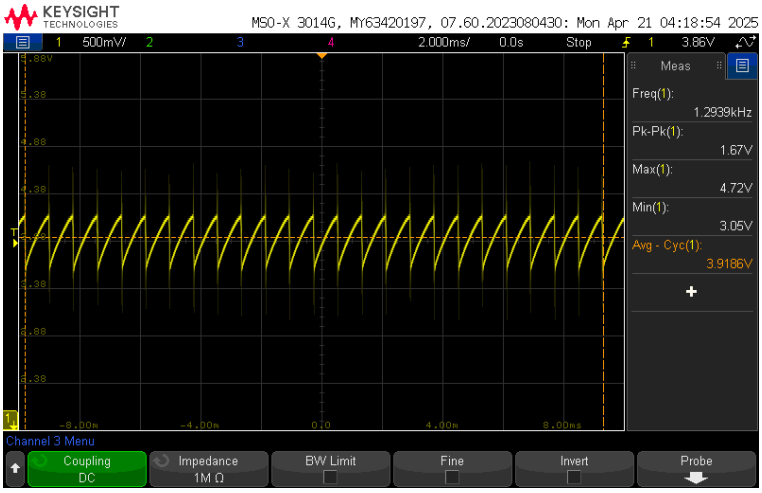


Figure 4.4.6: Ripple Measurement of the 5V Step Down

Figure 4.4.7: Ripple Measurement of the 5V Step Down and 3.3 volt step down with BMS

Figure 4.4.8: Ripple Measurement of the 5V Step Down and 3.3 Volt step down

4.4.10 Standards

- We will be using a 3S LiPo Battery for the power that will run the drone. As such the charging balance load charging standard will need to be followed to prevent over charging and damage or explosion of the LiPo battery pack.

5 PCB Design

5.1 PCB Schematics

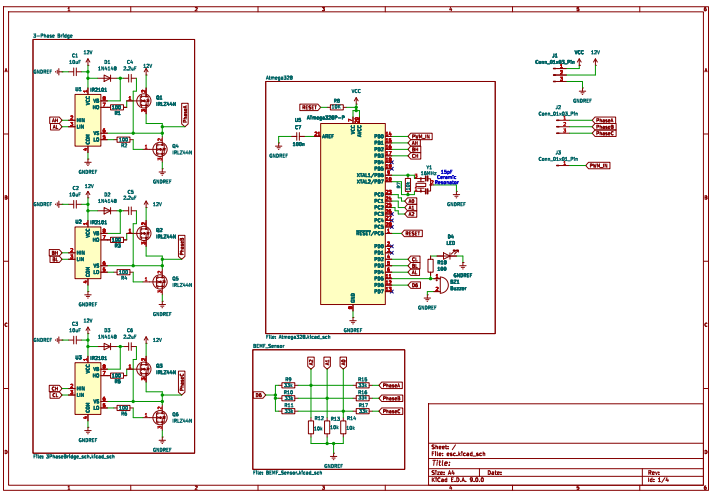


Figure 5.1.1: PCB Flight Systems ESC Schematic

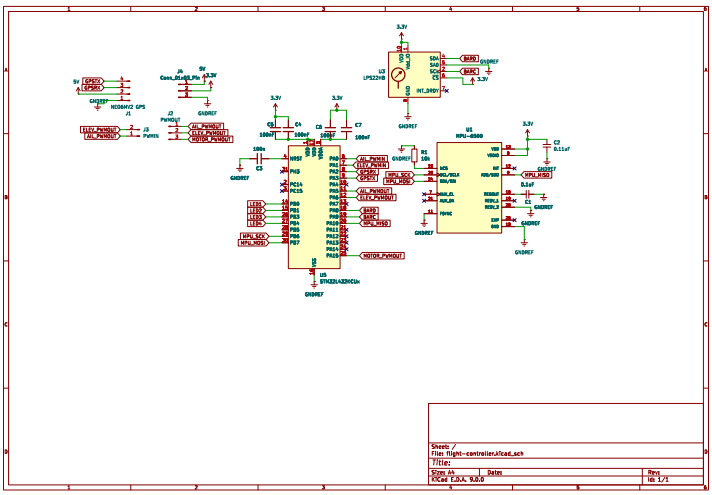


Figure 5.1.2: PCB Flight Systems Flight Controller Schematic

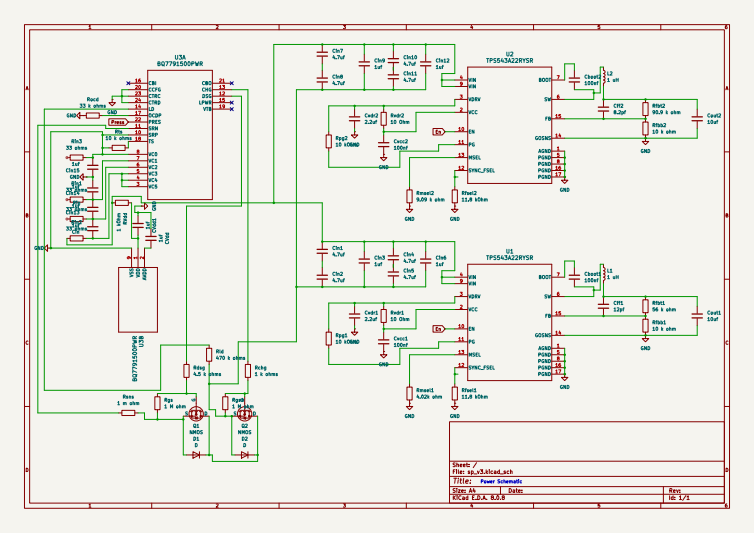


Figure 5.1.3: PCB Power Schematic

5.2 PCB Layout

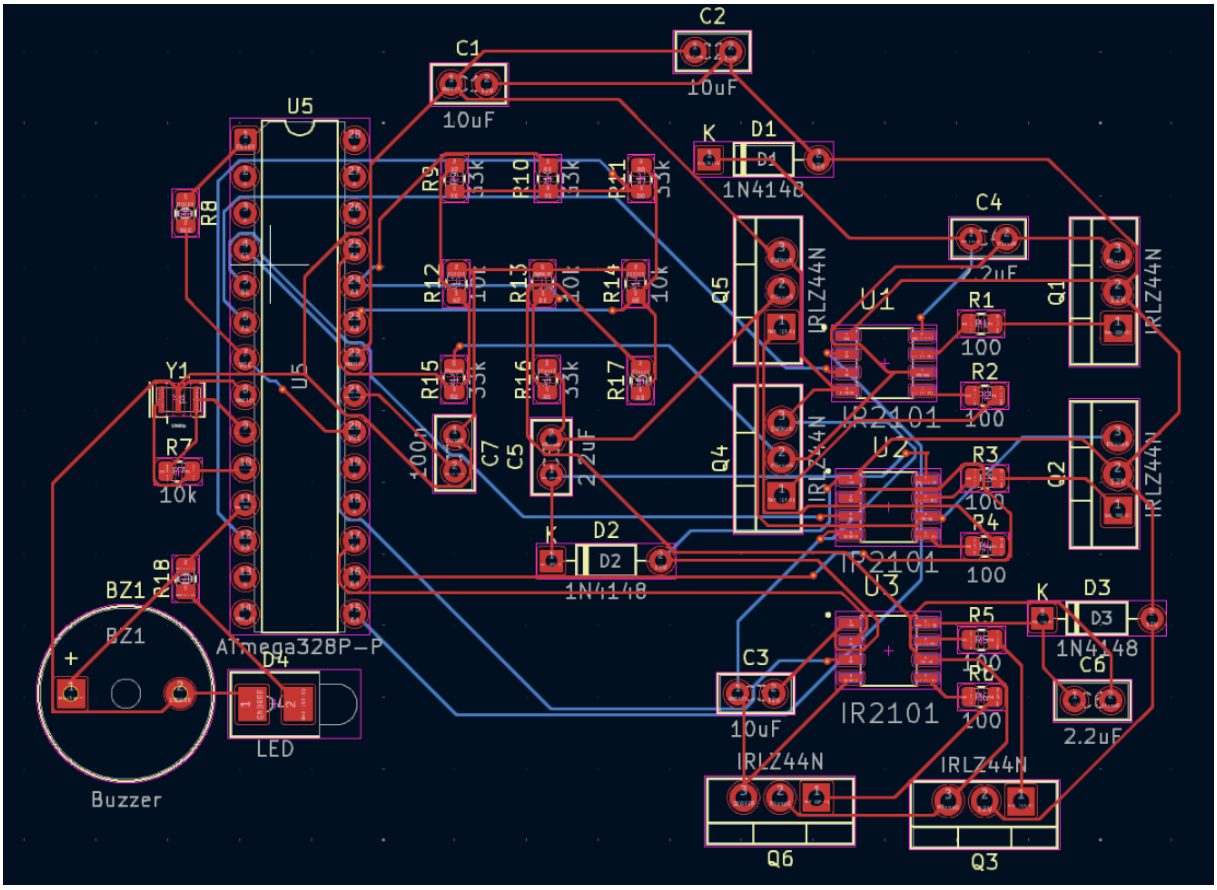


Figure 5.2.1: PCB Flight Systems ESC Layout

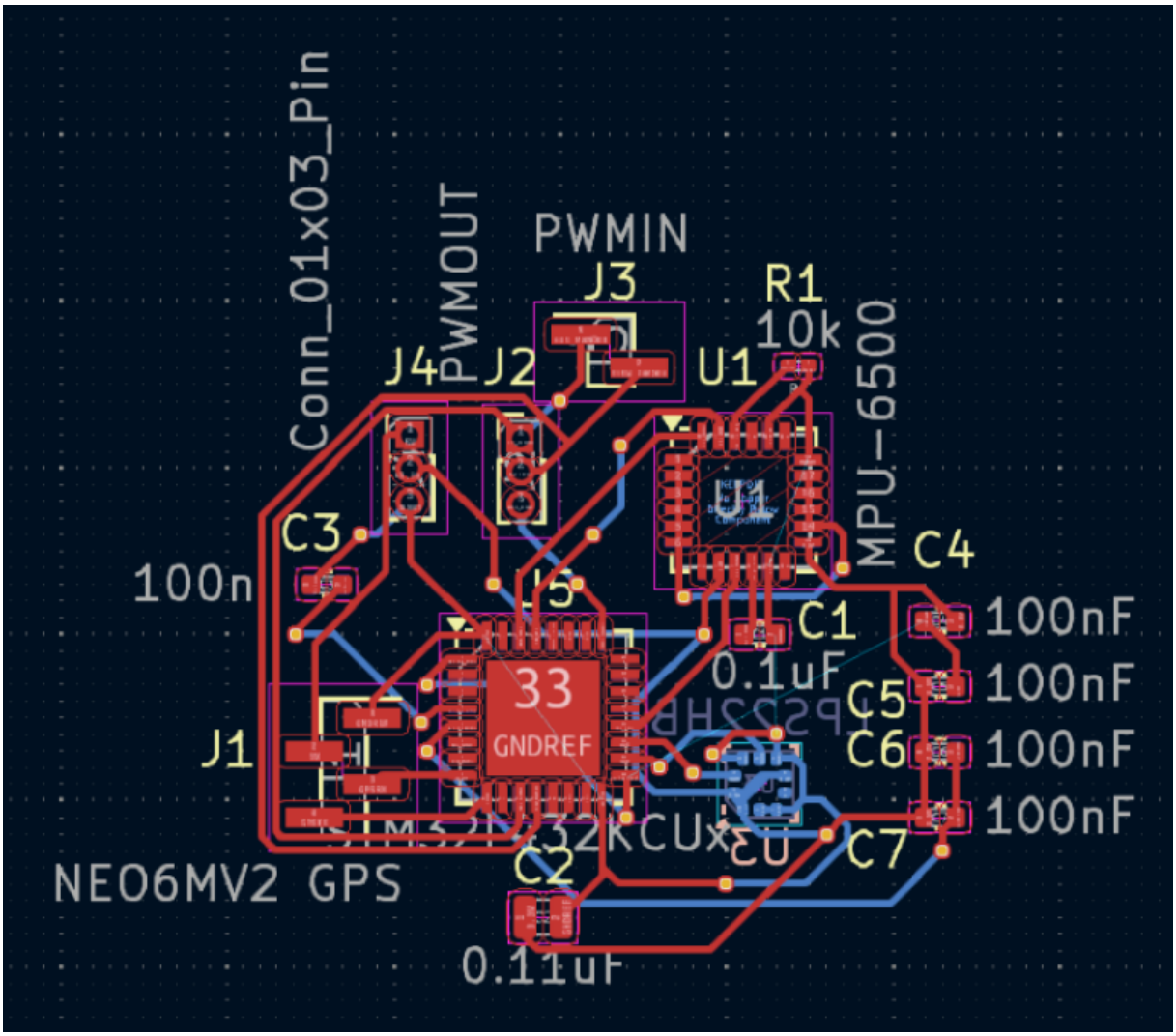


Figure 5.2.2: PCB Flight Systems Flight Controller Layout

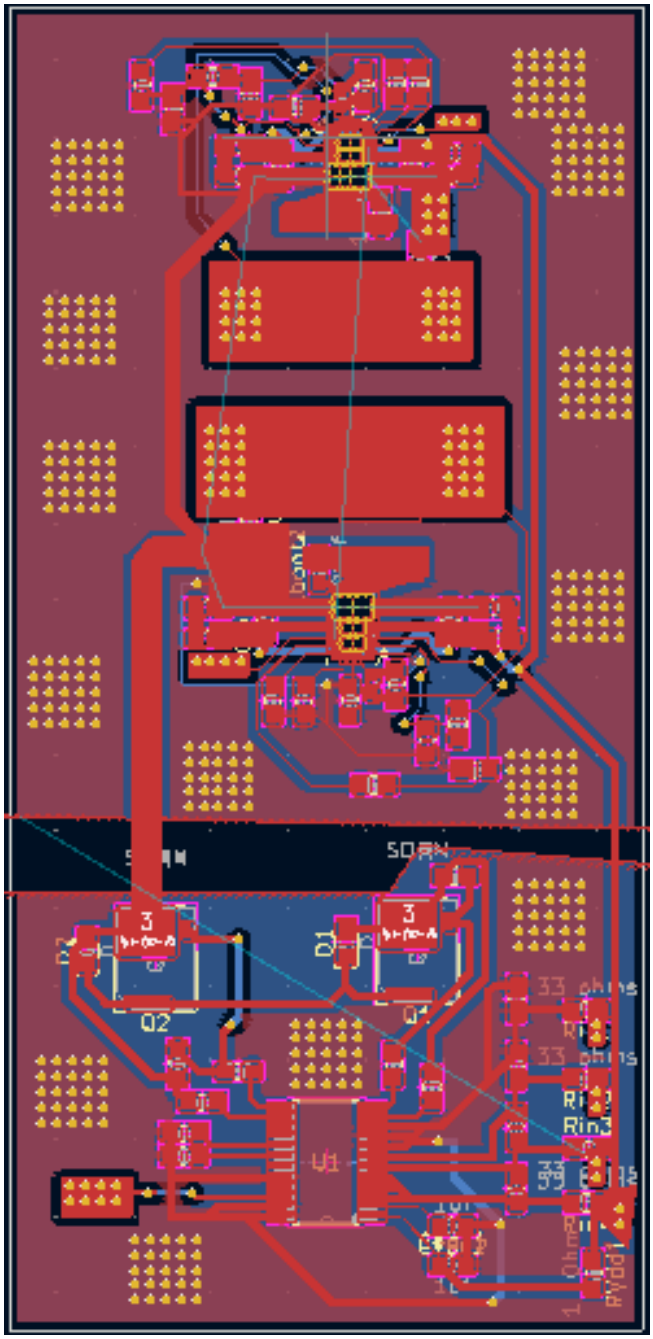


Figure 5.2.3: PCB Power Layout

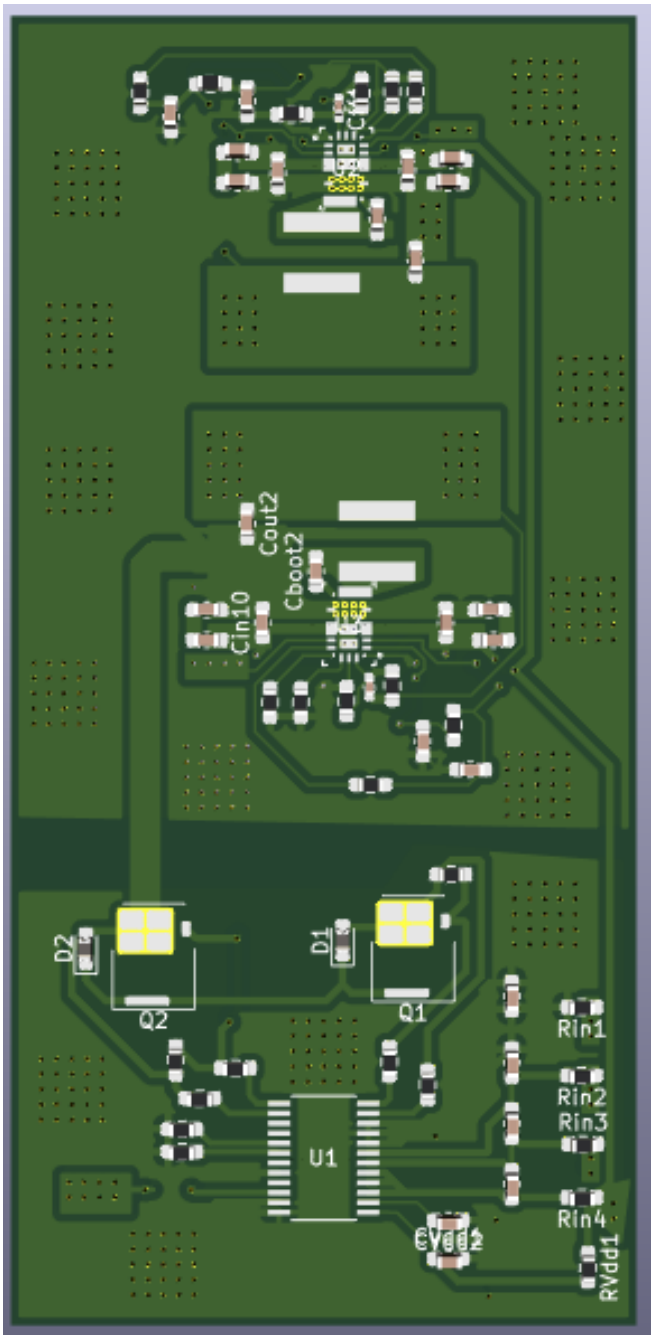


Figure 5.2.4: PCB Power Render Front

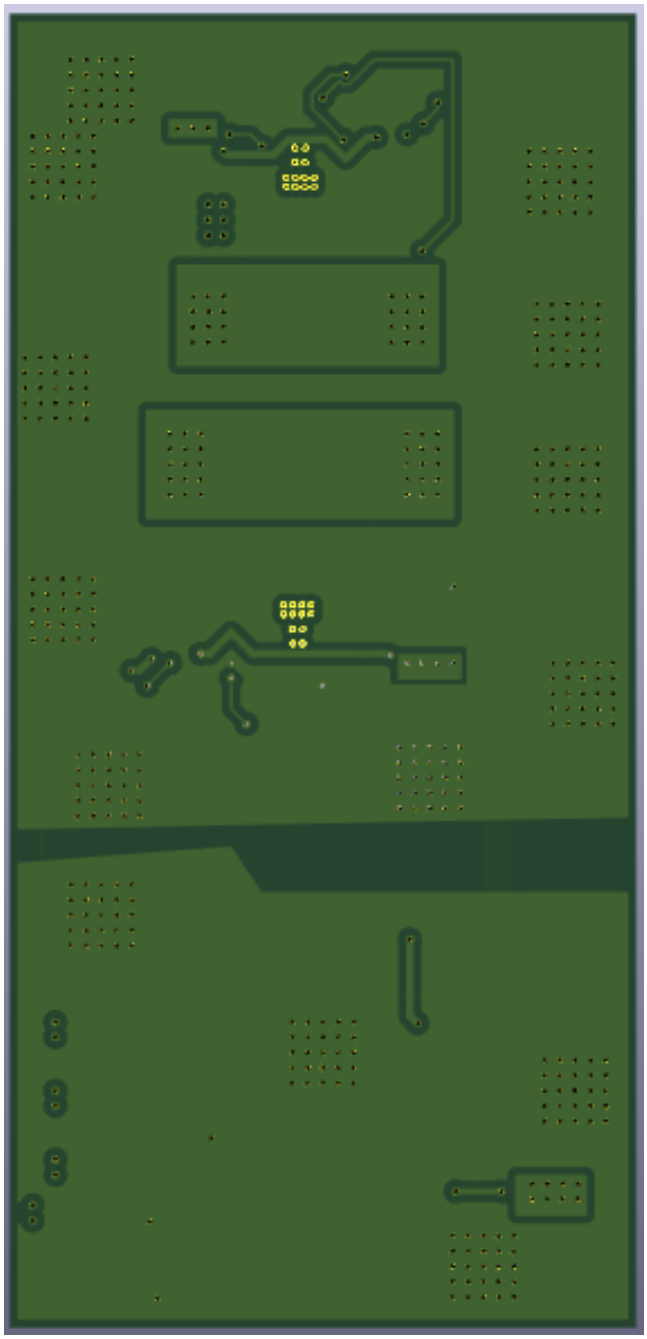


Figure 5.2.5: PCB Power Render Back

6 Final Status of Requirements

1. Requirement 1: The system must come as a full setup of drone, remote controller, ground station kit, and software download.
   Met: The system comes with a drone, a ground station, a remote control, and UI software.
2. Requirement 2: The system must have access to a video feed so it can do object detection using AI/ML algorithms.
   Met: The system uses the SC3336 3MP camera module to capture a live feed and utilizes YOLO V5 to detect objects.
3. Requirement 3: The system must feature efficient power consumption.
   Partially Met: The system was designed for high efficiency; however, in practice, due to a variety of poor solder connections and design compromises, the system is less efficient than hoped.
4. Requirement 4: The battery must be easily rechargeable to support frequent operational use.
   Met: To charge the battery, simply connect a 12-volt lead to the power pin and a ground lead to the ground pin.
5. Requirement 5: All components must be lightweight enough to avoid significantly impacting the flight performance.
   Partially Met: The servo motors are heavier than ideal for flight, but the rest of the components are optimized for minimal weight.
6. Requirement 6: The object detection system must be responsive to enable real-time data processing.
   Met: YOLO V5 provides real-time detection capabilities.
7. Requirement 7: Decrease the time it takes to search for and locate targets to increase efficiency.
   Partially Met: The system has been designed, but it is not yet capable of being deployed operationally.
8. Requirement 8: Object detection by YOLO should be operable under different environmental conditions, such as night, fog, and debris.
   Partially Met: Theoretically, the system should function under these conditions, but it remains untested.
9. Requirement 9: The system should be able to adapt to diverse geographic locations: urban, remote, disaster areas.
   Not Met: International adaptability is not met due to the absence of a required transponder.
10. Requirement 10: Must follow international drone and RF regulations for massive deployment.

Partially Met: The system operates point-to-point and has no sale or data collection mechanisms in place.

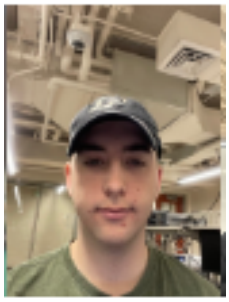
11. Requirement 11: UI should support multilingual languages for rescue teams around the world.
    Met: The UI supports Turkish and English languages.
12. Requirement 12: Shall consider ethical considerations, including privacy and security in AI-driven searches.
    Met: The system costs less than $500, meeting affordability criteria.
13. Requirement 13: Easily affordable and accessible by humanitarian organizations and first emergency responders.
    Met: The system only forwards data to the user without making autonomous decisions.
14. Requirement 14: AI shall complement, not replace human judgment and rescue workers.
    Partially Met: The current system does not achieve the ideal flight duration but can be scaled by employing a larger battery management system.
15. Requirement 15: Power-efficient hardware will allow operation over longer times while reducing impact on the environment.
    Met: Met by definition of the requirement.
16. Requirement 16: Shall use only commercially available low-cost hardware to make it more affordable.
    Not Met: The drone remains in a prototype state and is not yet finalized for widespread use.

7 Team Structure

7.1 Team Member 1

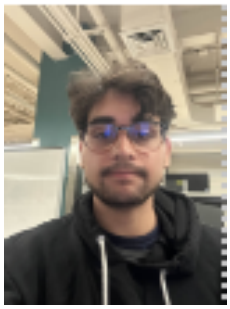
Ata Ulas Guler
Major: Computer Engineering
Contact: gulera@purdue.edu
Team Role: Software and Processing
Bio: I am a Computer Engineering (ECE) major at Purdue from Turkey.

7.2 Team Member 2



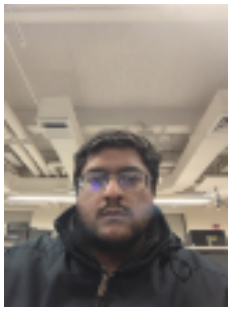
Nathan Frinier
Major: Computer Engineer
Contact: nfrinier@purdue.edu
Team Role: Hardware
Bio: I am from LA California.

7.3 Team Member 3



Dhruv Chaudhary
Major: Computer Engineering
Contact: dchaudh@purdue.edu
Team Role: Communications
Bio: I'm From Washington state.

7.4 Team Member 4



Mohsin Nadir
Major: Electrical Engineering
Contact: mnadir@purdue.edu
Team Role: Flight Controls
Bio: I like to fly RC planes.

8 Biblography

References

[1] Luckfox, "Luckfox Pico Quick Start." [Online]. Available: `https://wiki.luckfox.com/Luckfox-Pico/Luckfox-Pico-RV1106/Luckfox-Pico-Ultra-W/Luckfox-Pico-quick-start/`.

References

[1] "Data Platform – Open Power System data," Apr. 15, 2020. https://data.open-power-system-data.org/household_data/

[2] Author,"Title," *Journal*,volume,number, page range, month year, DOI.

[3] Author. "Page."Website. URL(accessed month day,year)

9 Appendices

9.1 Appendix A: Signal Communications Antenna Peudo Code

```
/*
 *  ======== groundStation.c ========
 *
 *  Sends CMD_IMG_REQUEST (0x01) to the drone, then listens for:
 *   - CMD_IMG_BEGIN (0x10)   -> Start accumulating JPEG bytes
 *   - CMD_IMG_DATA  (0x02)   -> Append chunk to our buffer
 *   - CMD_IMG_END   (0x11)   -> Stop; we have the entire JPEG
 *
 *  Once CMD_IMG_END is received, we print the entire collected image
    in
 *  hexadecimal, wrapped between:
 *    ===BEGIN_JPEG===
 *    (hex bytes...)
 *    ===END_JPEG===
 *
 *  we parse that from the serial log to reconstruct the image.
 */

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

/* TI Drivers */
#include <ti/drivers/rf/RF.h>
#include <ti/drivers/GPIO.h>
#include <ti/drivers/UART2.h>
#include "ti_drivers_config.h"

/* FreeRTOS */
#include <FreeRTOS.h>
#include <task.h>

/* RF Queue API */
#include "RFQueue.h"

/* Radio config (custom PHY) */
#include <ti_radio_config.h>

/* Command codes */
#define CMD_IMG_REQUEST  0x01
#define CMD_IMG_DATA     0x02
#define CMD_IMG_BEGIN    0x10
#define CMD_IMG_END      0x11
```

```
/*
 *  For demonstration, up to 32 KB
 */
#define MAX_IMAGE_SIZE   32768

static uint8_t  imageBuffer[MAX_IMAGE_SIZE];
static uint32_t imageIndex     = 0;
static uint32_t totalBytes     = 0;
static uint8_t  inJpegTransfer = 0;   // Are we within BEGIN..END?
static volatile uint8_t  imageReceived = 0;   // Flag indicating a full
     image was received

/* Each data packet can have up to 217 bytes of image data. */
#define PAYLOAD_LENGTH   220  /* Use 220 for throughput; must match
    drone. Adjust for testing (e.g., 30). */

/* Global variables */
static RF_Object  rfObject;
static RF_Handle  rfHandle;
static uint8_t    txPacket[PAYLOAD_LENGTH];

/* RX queue definitions */
#define NUM_DATA_ENTRIES   2
#define NUM_APPENDED_BYTES 2

#if defined(__TI_COMPILER_VERSION__)
#pragma DATA_ALIGN(rxDataEntryBuffer, 4)
static uint8_t rxDataEntryBuffer[RF_QUEUE_DATA_ENTRY_BUFFER_SIZE(
    NUM_DATA_ENTRIES,
                                                                 PAYLOAD_LENGTH
                                                                     ,
                                                                 NUM_APPENDED_BYTES
                                                                     )
                                                                     ];
#elif defined(__IAR_SYSTEMS_ICC__)
#pragma data_alignment=4
static uint8_t rxDataEntryBuffer[RF_QUEUE_DATA_ENTRY_BUFFER_SIZE(
    NUM_DATA_ENTRIES,
                                                                 PAYLOAD_LENGTH
                                                                     ,
                                                                 NUM_APPENDED_BYTES
                                                                     )
                                                                     ];
#elif defined(__GNUC__)
static uint8_t rxDataEntryBuffer[RF_QUEUE_DATA_ENTRY_BUFFER_SIZE(
    NUM_DATA_ENTRIES,
                                                                 PAYLOAD_LENGTH
                                                                     ,
                                                                 NUM_APPENDED_BYTES
                                                                     )]
```

```
                                                                    __attribute__
                                                                        ((
                                                                        aligned
                                                                        (4)
                                                                        ))
                                                                        ;
#else
#error This compiler is not supported
#endif

static dataQueue_t        dataQueue;
static rfc_propRxOutput_t rxStatistics;

/* UART for debug printing */
static UART2_Handle uart2;

static void debugPrint(const char *fmt, ...)
{
    char buffer[128];
    size_t bytesWritten;
    va_list args;

    va_start(args, fmt);
    vsnprintf(buffer, sizeof(buffer), fmt, args);
    va_end(args);

    UART2_write(uart2, buffer, strlen(buffer), &bytesWritten);
}

/*
 *  ======== imgRxCallback ========
 *  Processes RX packets for CMD_IMG_BEGIN, CMD_IMG_DATA, CMD_IMG_END.
 */
static void imgRxCallback(RF_Handle h, RF_CmdHandle ch, RF_EventMask e)
{
    if (e & RF_EventRxEntryDone)
    {
        rfc_dataEntryGeneral_t *currentDataEntry = RFQueue_getDataEntry
            ();
        if (!currentDataEntry)
            return;

        uint8_t packetLen = *((uint8_t *)&currentDataEntry->data);
        uint8_t *pktData  = (uint8_t *)(&currentDataEntry->data + 1);

        if (packetLen > 0 && packetLen <= PAYLOAD_LENGTH)
        {
            uint8_t cmd = pktData[0];

            switch(cmd)
```

```
{
case CMD_IMG_BEGIN:
    debugPrint("Ground: RX CMD_IMG_BEGIN\r\n");
    inJpegTransfer = 1;
    imageIndex     = 0;
    totalBytes     = 0;
    GPIO_toggle(CONFIG_GPIO_RLED); // LED feedback
    break;

case CMD_IMG_DATA:
    if (inJpegTransfer)
    {
        uint8_t seq       = pktData[1];
        /* totalPkts = pktData[2];  // for info, if needed
           */
        uint8_t dataLen   = packetLen - 3;  // minus header

        // Append data to imageBuffer if we have space
        if ((imageIndex + dataLen) <= MAX_IMAGE_SIZE)
        {
            memcpy(&imageBuffer[imageIndex], &pktData[3],
               dataLen);
            imageIndex  += dataLen;
            totalBytes  += dataLen;
            GPIO_toggle(CONFIG_GPIO_RLED);

            debugPrint("Ground: RX CMD_IMG_DATA seq=%u,
               dataLen=%u\r\n",
                       seq, dataLen);
        }
        else
        {
            debugPrint("Ground: Buffer overflow, dropping
               data!\r\n");
        }
    }
    else
    {
        debugPrint("Ground: CMD_IMG_DATA but not
           inJpegTransfer\r\n");
    }
    break;

case CMD_IMG_END:
    if (inJpegTransfer)
    {
        inJpegTransfer = 0;
        GPIO_toggle(CONFIG_GPIO_GLED);

        //debugPrint("Ground: Total bytes received = %u\r\n
```

```
                        ", totalBytes);

                    /*
                     *  Now we print the entire JPEG in hex, bracketed
                         by
                     *  "===BEGIN_JPEG===" and "===END_JPEG==="
                     */
                    debugPrint("===BEGIN_JPEG===\r\n");
                    for (uint32_t i = 0; i < totalBytes; i++)
                    {
                        // Print without spaces/newlines for easier
                            parsing
                        debugPrint("%02X", imageBuffer[i]);
                    }
                    debugPrint("\r\n===END_JPEG===\r\n");

                    imageReceived = 1;
                }
                else
                {
                    debugPrint("Ground: CMD_IMG_END but not
                        inJpegTransfer\r\n");
                }
                break;

            default:
                debugPrint("Ground: Unknown cmd=0x%02X\r\n", cmd);
                break;
            }
        }
        else
        {
            debugPrint("Ground: Invalid packetLen=%u\r\n", packetLen);
        }

        RFQueue_nextEntry();
    }
    else if (e & RF_EventLastCmdDone)
    {
        debugPrint("Ground: RX EventLastCmdDone\r\n");
    }
    else
    {
        debugPrint("Ground: RX error/event e=0x%X\r\n", e);
        GPIO_write(CONFIG_GPIO_GLED, CONFIG_GPIO_LED_ON);
    }
}

/*
 *  ======== mainThread ========
```

```
 *  Sends CMD_IMG_REQUEST, then enters RX mode to collect the JPEG.
 */
void *mainThread(void *arg0)
{
    RF_Params rfParams;
    RF_Params_init(&rfParams);

    /* Open UART2 for debug prints */
    UART2_Params uart2Params;
    UART2_Params_init(&uart2Params);
    uart2 = UART2_open(CONFIG_UART2_0, &uart2Params);
    if (!uart2)
    {
        while(1); // failed
    }
    debugPrint("Ground: UART opened.\r\n");

    /* Setup LEDs */
    GPIO_setConfig(CONFIG_GPIO_RLED, GPIO_CFG_OUT_STD |
        GPIO_CFG_OUT_LOW);
    GPIO_setConfig(CONFIG_GPIO_GLED, GPIO_CFG_OUT_STD |
        GPIO_CFG_OUT_LOW);
    GPIO_write(CONFIG_GPIO_RLED, CONFIG_GPIO_LED_OFF);
    GPIO_write(CONFIG_GPIO_GLED, CONFIG_GPIO_LED_OFF);

    debugPrint("Ground: mainThread starting.\r\n");

    /* Configure RX data queue */
    if (RFQueue_defineQueue(&dataQueue,
                            rxDataEntryBuffer,
                            sizeof(rxDataEntryBuffer),
                            NUM_DATA_ENTRIES,
                            PAYLOAD_LENGTH + NUM_APPENDED_BYTES))
    {
        debugPrint("Ground: RFQueue_defineQueue failed.\r\n");
        GPIO_write(CONFIG_GPIO_RLED, CONFIG_GPIO_LED_ON);
        GPIO_write(CONFIG_GPIO_GLED, CONFIG_GPIO_LED_ON);
        while(1);
    }

    /* Open RF driver (custom PHY commands) */
    rfHandle = RF_open(&rfObject,
                       &RF_prop_2gfsk1mbps868_0,
                       (RF_RadioSetup*)&
                           RF_cmdPropRadioDivSetup_2gfsk1mbps868_0,
                       &rfParams);

    if (!rfHandle)
    {
        debugPrint("Ground: RF_open failed.\r\n");
```

```
    while(1);
}
debugPrint("Ground: RF opened.\r\n");

/* Set frequency */
RF_runCmd(rfHandle,
          (RF_Op*)&RF_cmdFs_2gfsk1mbps868_0,
          RF_PriorityNormal,
          NULL,
          0);
debugPrint("Ground: Frequency set.\r\n");

/* Optional: Wait for user input to proceed */
debugPrint("Ground: Press Enter to send CMD_IMG_REQUEST...\r\n");
{
    char c;
    size_t br;
    UART2_read(uart2, &c, 1, &br);
}

/* --- Send CMD_IMG_REQUEST --- */
txPacket[0] = CMD_IMG_REQUEST;
RF_cmdPropTx_2gfsk1mbps868_0.pPkt   = txPacket;
RF_cmdPropTx_2gfsk1mbps868_0.pktLen = 1;
RF_cmdPropTx_2gfsk1mbps868_0.startTrigger.triggerType = TRIG_NOW;

debugPrint("Ground: Sending CMD_IMG_REQUEST...\r\n");
RF_runCmd(rfHandle,
          (RF_Op*)&RF_cmdPropTx_2gfsk1mbps868_0,
          RF_PriorityNormal,
          NULL,
          0);

/* --- Switch to RX mode to receive the image (BEGIN, DATA, END)
    --- */
RF_cmdPropRx_2gfsk1mbps868_0.pQueue                    = &dataQueue
    ;
RF_cmdPropRx_2gfsk1mbps868_0.rxConf.bAutoFlushIgnored = 1;
RF_cmdPropRx_2gfsk1mbps868_0.rxConf.bAutoFlushCrcErr  = 1;
RF_cmdPropRx_2gfsk1mbps868_0.maxPktLen                =
    PAYLOAD_LENGTH;
RF_cmdPropRx_2gfsk1mbps868_0.startTrigger.triggerType  = TRIG_NOW;
RF_cmdPropRx_2gfsk1mbps868_0.pktConf.bRepeatOk        = 1;
RF_cmdPropRx_2gfsk1mbps868_0.pktConf.bRepeatNok       = 1;
RF_cmdPropRx_2gfsk1mbps868_0.pOutput = (uint8_t *)&rxStatistics;

debugPrint("Ground: Posting RX command. Awaiting image...\r\n");
RF_postCmd(rfHandle,
           (RF_Op*)&RF_cmdPropRx_2gfsk1mbps868_0,
           RF_PriorityNormal,
```

```
                imgRxCallback,
                (RF_EventRxEntryDone | RF_EventLastCmdDone));

    /*
     *  The callback handles received image packets.
     *  After each image is printed, the main loop sends another
         request after a 1-second pause.
     */
    while (1)
    {
        if (imageReceived)
        {
            imageReceived = 0;
            RF_flushCmd(rfHandle, RF_CMDHANDLE_FLUSH_ALL, 0);
            debugPrint("Ground: Image received. Requesting next image
                ...\r\n");
            vTaskDelay(pdMS_TO_TICKS(1000));
            txPacket[0] = CMD_IMG_REQUEST;
            debugPrint("Ground: Sending CMD_IMG_REQUEST...\r\n");
            RF_runCmd(rfHandle,
                      (RF_Op*)&RF_cmdPropTx_2gfsk1mbps868_0,
                      RF_PriorityNormal,
                      NULL,
                      0);
            debugPrint("Ground: Posting RX command. Awaiting image...\r
                \n");
            RF_postCmd(rfHandle,
                       (RF_Op*)&RF_cmdPropRx_2gfsk1mbps868_0,
                       RF_PriorityNormal,
                       imgRxCallback,
                       (RF_EventRxEntryDone | RF_EventLastCmdDone));
        }
        vTaskDelay(pdMS_TO_TICKS(500));
    }
}

/*****************************************************************************

 *  Drone_Tx.c
 *
 *  - Defines DeviceFamily so TI-Drivers headers resolve without
    SysConfig.
 *  - Opens UART2 (CONFIG_UART2_0) on DIO2 (RX) / DIO3 (TX) for debug
    prints (via USB-CDC).
 *  - Optionally opens a second UART2 (CONFIG_UART2_1) on header pins
    for external
 *    "Luckfox" JPEG capture (detects EOI 0xFF 0xD9).
 *  - Cycles through one Flash image (image3_jpg) plus the UART slot on
     each
 *    CMD_IMG_REQUEST   or  , if SEND_FLASH_ALWAYS==0, sends flash only
```

```
     once then
 *    only UART images thereafter.
 ******************************************************************************/


#define DeviceFamily_CC13X2_CC26X2
#include <ti/devices/DeviceFamily.h>

/* Toggle behavior: 1 = flash+UART cycle each time; 0 = flash once,
    then only UART */
#define SEND_FLASH_ALWAYS  0

/* Standard C headers */
#include <stdint.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdarg.h>
#include <stdbool.h>      // for flashSent flag

/* TI Drivers */
#include <ti/drivers/GPIO.h>
#include <ti/drivers/rf/RF.h>
#include <ti/drivers/UART2.h>
#include "ti_drivers_config.h"

/* FreeRTOS */
#include <FreeRTOS.h>
#include <task.h>
#include <semphr.h>

/* RF Queue & PHY */
#include "RFQueue.h"
#include <ti_radio_config.h>

/* Flash-resident JPEG */
#include "image3.h"   // extern const unsigned char image3_jpg[];
    extern const unsigned int image3_jpg_len;

/* Command codes */
#define CMD_IMG_REQUEST   0x01
#define CMD_IMG_BEGIN     0x10
#define CMD_IMG_DATA      0x02
#define CMD_IMG_END       0x11

/* RF payload sizing */
#define PAYLOAD_LENGTH    220
#define DATA_CHUNK_SIZE   (PAYLOAD_LENGTH - 3)

/* Only one flash image */
```

```
#define NUM_FLASH_IMAGES  1

/* RFQueue RX buffering */
#define NUM_DATA_ENTRIES   2
#define NUM_APPENDED_BYTES 2
#if defined(__TI_COMPILER_VERSION__)
#pragma DATA_ALIGN(rxDataEntryBuffer,4)
static uint8_t rxDataEntryBuffer[
    RF_QUEUE_DATA_ENTRY_BUFFER_SIZE(NUM_DATA_ENTRIES,
                                    PAYLOAD_LENGTH,
                                    NUM_APPENDED_BYTES)
];
#else
static uint8_t rxDataEntryBuffer[
    RF_QUEUE_DATA_ENTRY_BUFFER_SIZE(NUM_DATA_ENTRIES,
                                    PAYLOAD_LENGTH,
                                    NUM_APPENDED_BYTES)
] __attribute__((aligned(4)));
#endif
static dataQueue_t        dataQueue;
static rfc_propRxOutput_t rxStatistics;

/* Flash image table */
static const unsigned char *flashImages[NUM_FLASH_IMAGES] = {
    image3_jpg
};
static unsigned int flashImageLens[NUM_FLASH_IMAGES];

/* UART2 JPEG buffer (34 KB) */
#define UART2_BUFFER_SIZE 34000
static uint8_t   uart2StaticBuf[UART2_BUFFER_SIZE];
static uint8_t  *uart2Buf     = uart2StaticBuf;
static uint32_t  uart2Len     = 0;
static volatile uint8_t uart2Ready = 0;

/* Current TX image pointers */
static const unsigned char *imageData = NULL;
static uint32_t            imageSize  = 0;
static uint16_t            totalPackets = 0;

/* RF driver objects */
static RF_Object rfObject;
static RF_Handle rfHandle;
static uint8_t   txPacket[PAYLOAD_LENGTH];

/* RTOS sync and UART2 handles */
static SemaphoreHandle_t requestSemaphore = NULL;
static SemaphoreHandle_t txDoneSemaphore  = NULL;
static SemaphoreHandle_t uart2Mutex       = NULL;
static UART2_Handle      uart2Debug       = NULL;  // CONFIG_UART2_0
```

```
static UART2_Handle      uart2Rx          = NULL;  // CONFIG_UART2_1

/* Prevent double-handling of a request */
static volatile uint8_t requestReceived = 0;

/* Track whether we've sent flash once (for SEND_FLASH_ALWAYS==0) */
#if (SEND_FLASH_ALWAYS == 0)
static bool flashSent = false;
#endif

/*============================================================================

 * debugPrint(fmt, ...)     thread-safe printf over UART2 (debug UART)
 *============================================================================*/

static void debugPrint(const char *fmt, ...)
{
    char buf[128];
    size_t written;
    va_list ap;
    if (xSemaphoreTake(uart2Mutex, pdMS_TO_TICKS(50)) == pdTRUE) {
        va_start(ap, fmt);
        vsnprintf(buf, sizeof(buf), fmt, ap);
        va_end(ap);
        UART2_write(uart2Debug, buf, strlen(buf), &written);
        xSemaphoreGive(uart2Mutex);
    }
}

/*============================================================================

 * requestRxCallback     RF callback, fires on CMD_IMG_REQUEST
    reception
 *============================================================================*/

static void requestRxCallback(RF_Handle h, RF_CmdHandle ch,
   RF_EventMask e)
{
    BaseType_t woken = pdFALSE;
    if (requestReceived) return;
    if (e & RF_EventRxEntryDone) {
        rfc_dataEntryGeneral_t *entry = RFQueue_getDataEntry();
        if (entry) {
            uint8_t len = *((uint8_t *)&entry->data);
            uint8_t *pkt = (uint8_t*)(&entry->data + 1);
            if (len && len <= PAYLOAD_LENGTH && pkt[0] ==
                CMD_IMG_REQUEST) {
                requestReceived = 1;
                GPIO_toggle(CONFIG_GPIO_RLED);
                xSemaphoreGiveFromISR(requestSemaphore, &woken);
```

```
                portYIELD_FROM_ISR(woken);
            }
            RFQueue_nextEntry();
        }
    }
}

/*============================================================================

 * UART2ReceiveTask     captures a JPEG via UART2_1, detects 0xFF 0xD9
    EOI
 *============================================================================*/

static void UART2ReceiveTask(void *arg)
{
    size_t br;
    uint8_t c, prev = 0;
    uint32_t idx = 0;
    debugPrint("Drone: UART2ReceiveTask started\r\n");
    for (;;) {
        UART2_readTimeout(uart2Rx, &c, 1, &br, 10);
        if (br == 1) {
            uart2Buf[idx++] = c;
            if (prev == 0xFF && c == 0xD9) {
                uart2Len   = idx;
                uart2Ready = 1;
                debugPrint("Drone: UART2 slot ready (%lu bytes)\r\n",
                           (unsigned long)uart2Len);
                idx = prev = 0;
                continue;
            }
            prev = c;
            if (idx >= UART2_BUFFER_SIZE) { idx = prev = 0; }
        }
    }
}

/*============================================================================

 * imageTransmitTask     sends  B E G I N DATAEND  packets over RF
 *============================================================================*/

static void imageTransmitTask(void *param)
{
    debugPrint("Drone: TX slot size=%lu bytes, pkts=%u\r\n",
               (unsigned long)imageSize, totalPackets);

    /* CMD_IMG_BEGIN */
    memset(txPacket, 0, PAYLOAD_LENGTH);
    txPacket[0] = CMD_IMG_BEGIN;
```

```
    RF_cmdPropTx_2gfsk1mbps868_0.pPkt   = txPacket;
    RF_cmdPropTx_2gfsk1mbps868_0.pktLen = 1;
    RF_runCmd(rfHandle, (RF_Op*)&RF_cmdPropTx_2gfsk1mbps868_0,
              RF_PriorityNormal, NULL, RF_EventLastCmdDone);
    vTaskDelay(pdMS_TO_TICKS(5));

    /* CMD_IMG_DATA */
    for (uint16_t seq = 0; seq < totalPackets; seq++) {
        uint32_t off = seq * DATA_CHUNK_SIZE;
        uint32_t sz  = (off + DATA_CHUNK_SIZE > imageSize)
                       ? (imageSize - off)
                       : DATA_CHUNK_SIZE;
        memset(txPacket, 0, PAYLOAD_LENGTH);
        txPacket[0] = CMD_IMG_DATA;
        txPacket[1] = (uint8_t)seq;
        txPacket[2] = (uint8_t)totalPackets;
        memcpy(&txPacket[3], &imageData[off], sz);

        RF_cmdPropTx_2gfsk1mbps868_0.pPkt   = txPacket;
        RF_cmdPropTx_2gfsk1mbps868_0.pktLen = 3 + sz;
        RF_runCmd(rfHandle, (RF_Op*)&RF_cmdPropTx_2gfsk1mbps868_0,
                  RF_PriorityNormal, NULL, RF_EventLastCmdDone);
        GPIO_toggle(CONFIG_GPIO_RLED);
        vTaskDelay(pdMS_TO_TICKS(5));
    }

    /* CMD_IMG_END */
    memset(txPacket, 0, PAYLOAD_LENGTH);
    txPacket[0] = CMD_IMG_END;
    RF_cmdPropTx_2gfsk1mbps868_0.pPkt   = txPacket;
    RF_cmdPropTx_2gfsk1mbps868_0.pktLen = 1;
    RF_runCmd(rfHandle, (RF_Op*)&RF_cmdPropTx_2gfsk1mbps868_0,
              RF_PriorityNormal, NULL, RF_EventLastCmdDone);

    GPIO_write(CONFIG_GPIO_GLED, CONFIG_GPIO_LED_ON);
    xSemaphoreGive(txDoneSemaphore);
    vTaskDelete(NULL);
}

/*============================================================================

 * mainThread()     initializes UARTs, RF, then serves image requests
 *============================================================================*/

void *mainThread(void *arg0)
{
    RF_Params rfParams;
    RF_Params_init(&rfParams);
    rfParams.nInactivityTimeout = 0;
```

```
    /* Create semaphores & mutex */
    uart2Mutex       = xSemaphoreCreateMutex();
    requestSemaphore = xSemaphoreCreateBinary();
    txDoneSemaphore  = xSemaphoreCreateBinary();
    if (!uart2Mutex || !requestSemaphore || !txDoneSemaphore) {
        while (1);
    }

    /* Open debug UART */
    UART2_Params uparams;
    UART2_Params_init(&uparams);
    uparams.baudRate  = 115200;
    uparams.readMode  = UART2_Mode_BLOCKING;
    uparams.writeMode = UART2_Mode_BLOCKING;
    uart2Debug = UART2_open(CONFIG_UART2_0, &uparams);
    if (!uart2Debug) {
        while (1);
    }
    debugPrint("Drone: Debug UART opened\r\n");

    /* Open image-data UART if present */
#ifdef CONFIG_UART2_1
    uart2Rx = UART2_open(CONFIG_UART2_1, &uparams);
    if (uart2Rx) {
        debugPrint("Drone: Data UART opened for image RX\r\n");
        xTaskCreate(UART2ReceiveTask, "UART2 Rx", 1024, NULL, 1, NULL);
    } else {
        debugPrint("Warning: UART2_1 open failed\r\n");
    }
#else
    uart2Rx = NULL;
#endif

    /* LEDs off */
    GPIO_setConfig(CONFIG_GPIO_RLED, GPIO_CFG_OUT_STD |
       GPIO_CFG_OUT_LOW);
    GPIO_setConfig(CONFIG_GPIO_GLED, GPIO_CFG_OUT_STD |
       GPIO_CFG_OUT_LOW);
    GPIO_write(CONFIG_GPIO_RLED, CONFIG_GPIO_LED_OFF);
    GPIO_write(CONFIG_GPIO_GLED, CONFIG_GPIO_LED_OFF);

    debugPrint("Drone: System start     flash+UART slot mode=%d\r\n",
       SEND_FLASH_ALWAYS);

    /* RFQueue define */
    if (RFQueue_defineQueue(&dataQueue,
                            rxDataEntryBuffer,
                            sizeof(rxDataEntryBuffer),
                            NUM_DATA_ENTRIES,
                            PAYLOAD_LENGTH + NUM_APPENDED_BYTES)) {
```

```
        debugPrint("Drone: RFQueue define failed\r\n");
        while (1);
    }

    /* Open RF & set frequency */
    rfHandle = RF_open(&rfObject,
                       &RF_prop_2gfsk1mbps868_0,
                       (RF_RadioSetup*)&
                           RF_cmdPropRadioDivSetup_2gfsk1mbps868_0,
                       &rfParams);
    if (!rfHandle) {
        debugPrint("Drone: RF_open failed\r\n");
        while (1);
    }
    RF_runCmd(rfHandle, (RF_Op*)&RF_cmdFs_2gfsk1mbps868_0,
              RF_PriorityNormal, NULL, 0);
    debugPrint("Drone: RF opened, freq set.\r\n");

    /* Configure RX command */
    RF_cmdPropRx_2gfsk1mbps868_0.pQueue                = &dataQueue;
    RF_cmdPropRx_2gfsk1mbps868_0.pOutput               = (uint8_t*)&
        rxStatistics;
    RF_cmdPropRx_2gfsk1mbps868_0.rxConf.bAutoFlushCrcErr  = 1;
    RF_cmdPropRx_2gfsk1mbps868_0.rxConf.bAutoFlushIgnored = 1;
    RF_cmdPropRx_2gfsk1mbps868_0.pktConf.bRepeatOk        = 1;
    RF_cmdPropRx_2gfsk1mbps868_0.pktConf.bRepeatNok       = 1;
    RF_cmdPropRx_2gfsk1mbps868_0.maxPktLen               =
        PAYLOAD_LENGTH;
    RF_cmdPropRx_2gfsk1mbps868_0.endTrigger.triggerType  = TRIG_NEVER;

    /* Initialize flash length */
    flashImageLens[0] = image3_jpg_len;

    /* Main loop */
    int cycleIdx = -1;
    int totalSlots = NUM_FLASH_IMAGES + (uart2Rx ? 1 : 0);

    for (;;) {
        requestReceived = 0;
        debugPrint("Drone: Posting RX command...\r\n");
        RF_postCmd(rfHandle,
                   (RF_Op*)&RF_cmdPropRx_2gfsk1mbps868_0,
                   RF_PriorityNormal,
                   requestRxCallback,
                   RF_EventRxEntryDone | RF_EventLastCmdDone);

        /* Decide which slot to send */
#if (SEND_FLASH_ALWAYS == 1)
        /* Cycle flash + UART as before */
        do {
```

```
            cycleIdx = (cycleIdx + 1) % totalSlots;
        } while (cycleIdx == NUM_FLASH_IMAGES && !uart2Ready);
#else
        /* Flash once, then only UART thereafter */
        if (!flashSent) {
            cycleIdx   = 0;          // flash slot
            flashSent = true;
        } else {
            cycleIdx = NUM_FLASH_IMAGES;  // UART slot only
        }
#endif

        /* Choose data pointer */
        if (cycleIdx < NUM_FLASH_IMAGES) {
            imageData = flashImages[cycleIdx];
            imageSize = flashImageLens[cycleIdx];
        } else {
            imageData  = uart2Buf;
            imageSize  = uart2Len;
            uart2Ready = 0;  // clear ready flag
        }
        totalPackets = (uint16_t)((imageSize + DATA_CHUNK_SIZE - 1) /
            DATA_CHUNK_SIZE);

        debugPrint("Drone: slot=%d size=%lu pkts=%u\r\n",
                   cycleIdx,
                   (unsigned long)imageSize,
                   totalPackets);

        /* Wait for request */
        xSemaphoreTake(requestSemaphore, portMAX_DELAY);
        debugPrint("Drone: Received CMD_IMG_REQUEST\r\n");

        /* Transmit */
        RF_flushCmd(rfHandle, RF_CMDHANDLE_FLUSH_ALL, 0);
        debugPrint("Drone: sending slot %d\r\n", cycleIdx);
        xTaskCreate(imageTransmitTask, "ImgTxTask", 2048, NULL, 2, NULL
            );

        /* Wait done */
        xSemaphoreTake(txDoneSemaphore, portMAX_DELAY);
        debugPrint("Drone: slot %d done.\r\n", cycleIdx);
    }
}
```

Appendix B: JPG Reconstruction Python Code

```
import os
import time
```

```
#                    Configuration
LOG_FILE    = r"C:\Users\dhruv\OneDrive\Desktop\Putty␣Logging\uartlog"
OUT_DIR     = r"C:\Users\dhruv\OneDrive\Desktop\uart2jpg\Photos"
BEGIN_MARK  = b"===BEGIN_JPEG==="
END_MARK    = b"===END_JPEG==="
SLEEP_TIME  = 0.1   # seconds to wait when no new data
#



def tail_and_extract(log_path, out_dir):
    # Ensure output directory exists
    os.makedirs(out_dir, exist_ok=True)

    # Open log file for reading in binary mode
    with open(log_path, "rb") as f:
        # Seek to end of file     we only want new data
        f.seek(0, os.SEEK_END)

        buffer = b""      # holds any partial data
        img_count = 0     # how many images  w e ve  saved

        while True:
            chunk = f.read(4096)
            if not chunk:
                # No new data? sleep a bit
                time.sleep(SLEEP_TIME)
                continue

            buffer += chunk

            # Process all complete JPEGs in the buffer
            while True:
                start = buffer.find(BEGIN_MARK)
                if start < 0:
                    # no begin marker yet
                    break

                end = buffer.find(END_MARK, start + len(BEGIN_MARK))
                if end < 0:
                    # begin found but no end yet
                    break

                # We have a full hex dump between start+len and end
                hexdata = buffer[start + len(BEGIN_MARK) : end]
                # strip any whitespace or separators
                hexdata = hexdata.strip(b"\r\n␣:")

                try:
```

```
                    img_bytes = bytes.fromhex(hexdata.decode("ascii"))
                except Exception as e:
                    print(f"[!] Error decoding hex: {e!r}")
                else:
                    # Save to file
                    img_count += 1
                    out_path = os.path.join(out_dir, f"Capture{
                        img_count}.jpg")
                    with open(out_path, "wb") as imgf:
                        imgf.write(img_bytes)
                    print(f"[+] Saved image #{img_count}    {out_path}
                        ")

                # Remove processed portion from buffer
                buffer = buffer[end + len(END_MARK):]

            # loop back to read more

if __name__ == "__main__":
    print("      Starting live JPEG extractor")
    tail_and_extract(LOG_FILE, OUT_DIR)
```

Appendix C: UI Python Code

```
import os, time, threading
from tkinter import *
from PIL import Image, ImageTk, ImageGrab

IMAGE_FOLDER = "Photos"
SCREENSHOT_FOLDER = "screenshots"
os.makedirs(SCREENSHOT_FOLDER, exist_ok=True)

LANGUAGES = {
    "EN": {"title": "Aerial Viewer", "aileron_l": "Aileron L",
    "aileron_r": "Aileron R", "elevator_l": "Elevator L",
    "elevator_r": "Elevator R", "throttle": "Throttle",
    "gps": "GPS", "capture": "Screenshot",
    "take_photo": "Take Photo", "lang": "T  rk  e"},
    "TR": {"title": "Havadan G  r  nt  leyici", "aileron_l": "Aileron S
        ",
    "aileron_r": "Aileron D", "elevator_l": "  rtifa  S",
    "elevator_r": "  rtifa  D", "throttle": "Gaz",
    "gps": "GPS", "capture": "Ekran G  r .", "take_photo": "Foto raf",
    "lang": "English"}
}
current_lang = "EN"

class AerialApp:
    def __init__(self, root):
```

```
        self.root = root
        self.root.title(LANGUAGES[current_lang]["title"])
        self.canvas = Canvas(root, width=800, height=600, bg="black")
        self.canvas.pack()
        ctrl_frame = Frame(root); ctrl_frame.pack(pady=5)
        self.entries = {}
        for i, key in enumerate(
          ["aileron_l", "aileron_r", "elevator_l", "elevator_r", "
              throttle", "gps"]):
            Label(ctrl_frame, text="").grid(row=0, column=i)
            e = Entry(ctrl_frame, width=10); e.grid(row=1, column=i)
            e.insert(0, "0" if key != "gps" else "0,0")
            e.bind("<FocusOut>", lambda e, k=key: self.update_input(k))
            e.bind("<Return>", lambda e, k=key: self.update_input(k))
            self.entries[key] = e
        btn_frame = Frame(root); btn_frame.pack(pady=5)
        self.cap_btn = Button(btn_frame, width=10, command=self.
            capture_screen)
        self.cap_btn.grid(row=0, column=0)
        self.photo_btn = Button(btn_frame, width=10, command=self.
            take_photo)
        self.photo_btn.grid(row=0, column=1)
        self.lang_btn = Button(btn_frame, width=10, command=self.
            toggle_lang)
        self.lang_btn.grid(row=0, column=2)
        self.update_lang()
        threading.Thread(target=self.watch_folder, daemon=True).start()
    def update_input(self, key): print(f"{key}: {self.entries[key].get
        ()}")
    def update_lang(self):
        lang = LANGUAGES[current_lang]; self.root.title(lang["title"])
        for i, key in enumerate(["aileron_l","aileron_r","elevator_l",
        "elevator_r","throttle","gps"]):
            self.entries[key].master.grid_slaves(row=0, column=i)[0].
                config(text=lang[key])
        self.cap_btn.config(text=lang["capture"])
        self.photo_btn.config(text=lang["take_photo"])
        self.lang_btn.config(text=lang["lang"])
    def toggle_lang(self):
        global current_lang; current_lang = "TR" if current_lang=="EN"
            else "EN"
        self.update_lang()
    def capture_screen(self):
        ts = time.strftime("%Y%m%d_%H%M%S")
        f = os.path.join(SCREENSHOT_FOLDER, f"screenshot_{ts}.png")
        x, y = self.root.winfo_rootx(), self.root.winfo_rooty()
        w, h = x+self.root.winfo_width(), y+self.root.winfo_height()
        ImageGrab.grab().crop((x, y, w, h)).save(f)
    def take_photo(self): print("Take Photo (stub)")
    def watch_folder(self):
```

```
        last_img = None
        while True:
            try:
                files = sorted([f for f in os.listdir(IMAGE_FOLDER)
                if f.lower().endswith(('.png','.jpg'))],
                key=lambda f: os.path.getmtime(os.path.join(
                    IMAGE_FOLDER,f)))
                if files and files[-1]!=last_img:
                    last_img = files[-1]
                    img = Image.open(os.path.join(IMAGE_FOLDER,last_img
                        )).resize((800,600))
                    self.tk_img = ImageTk.PhotoImage(img)
                    self.canvas.create_image(0,0,anchor=NW,image=self.
                        tk_img)
            except Exception as e: print(e)
            time.sleep(2)
if __name__ == "__main__":
    root=Tk(); app=AerialApp(root); root.mainloop()
```

Appendix D: Detection Shell Script

```
#!/bin/sh
echo "class,x1,y1,x2,y2,conf" > boxes.csv
IMG_DIR="/pictures"
mkdir -p "$IMG_DIR"

last_detect_time=0

# Temp file for output
tmpfile=$(mktemp)

# Run detection
./luckfox_pico_rtsp_yolov5 > "$tmpfile" 2>&1 &

# Process detection output
tail -F "$tmpfile" | while IFS= read -r line; do
 echo "$line" | grep -q '@ (' || continue

 # Extract data
 cls=$(echo "$line" | awk '{print $1}')
 coords=$(echo "$line" | awk -F'[()]' '{print $2}')
 conf=$(echo "$line" | awk '{print $NF}')

 echo "$cls,$coords,$conf" >> boxes.csv

 if [ "$cls" = "person" ]; then
   now=$(date +'%Y-%m-%d_%H-%M-%S')
   if [ $(($(date +%s) - last_detect_time)) -ge 2 ]; then
     last_detect_time=$(date +%s)
```

```
      echo ">>> Person detected at $coords, capturing image..." >&2

      # Format coords for filename
      coords_fmt=$(echo "$coords" | tr ' ' '_')

      # Create full image path
      filename="$IMG_DIR/person_${coords_fmt}_${now}.jpg"

      # Capture frame
      ffmpeg -y -rtsp_transport udp -i rtsp://localhost:554/live/0 -
          frames:v 1 -q:v 2 "$filename" < /dev/null &>/dev/null &
    fi
  fi
done
```