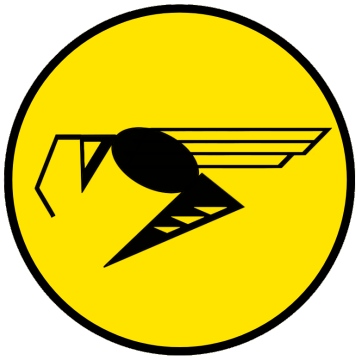
Department of Computer Science & Engineering
The University of Texas at Arlington

Detailed Design Specification
CSE 4317: Senior Design II
Spring 2025



Verminator
Wasp Drone

Isaac Medrano
Luis Narez
Tyler Nguyen
Nishchal Shrestha
Rajesh Yaksho

Revision History

Revision	Date	Author(s)	Description
0.1	2.03.2025	TN	document creation
0.2	2.10.2025	TN, IM, RY, LN, NS	complete draft
0.3	2.10.2025	TN, IM, RY, LN, NS	release candidate 1
1.0	2.10.2025	TN, IM, RY, LN, NS	official release
2.0	4.30.2025	TN	Revision 2 focuses on systems that were designed with greater detail.

CONTENTS

List of Figures

List of Tables

1 Introduction

This project, the Verminator Wasp Drone, is a low-cost, open-source, and modular drone developed for agricultural, commercial, and home applications, primarily focused on aerial insecticide spraying. Mantis is designed to be easily customizable and allows for adjustments based on specific user needs, while the open-source platform provides accessibility and flexibility for further development and integration with 3rd party or custom components.

The drone aims to address the high cost of commercial drones, making aerial spraying technology accessible to homeowners, small businesses, and farmers. By providing a more affordable option, the drone allows users to spray hard-to-reach areas efficiently. Beyond insecticide spraying, the drone's versatility extends to applications including fertilizer and pesticide distribution, as well as imaging and surveillance for agricultural and property management purposes. This adaptability ensures that the Verminator Wasp Drone can meet a wide range of needs while maintaining cost-effectiveness and ease of use.

The Architectural Design Specification covers all of the drone's systems at a high-level. This document, the Detailed Design Specification, will cover the systems that were designed and implemented by the team in detail. Other systems that the team did not design ourselves will be discussed at a high level.

The Appendix only contains information referenced in this document. The project's repository contains CAD files, source code, and other documents for this project [3].

2 System Overview

The system architecture consists of six layers: avionics, power and propulsion, controls, camera, sprayer, and the camera application. [Figure 1] is the complete architectural design diagram from the Architectural Design Specification. It is here as a recap of the overall design of the system.

2.1 Avionics System Layer Description

The avionics layer, which includes the flight controller, telemetry subsystem, and GPS module, serves as the brain of the drone. It processes user commands from the ground station or remote controller, relaying them to the relevant components to execute actions.

2.2 Power and Propulsion System Layer Description

The propulsion system layer ensures that the drone has power and generates sufficient lift to carry its payload. The propulsion system is composed of the lithium polymer (LiPo) batteries, electronic speed controllers (ESC), the power distribution board, and the brushless motors.

2.3 Controls System Layer Description

The control system layer is responsible for both the drone's flight control and sending commands to the attached payload. The drone can be controlled through two subsystems: The Shell Subsystem allows the drone to be programmatically controlled using a command-line interface at the ground station. The RC subsystem allows the drone to be controlled using an RC remote and transmitter. The RC subsystem gives the user direct control of the drone.

2.4 Camera System Layer Description

The Camera System Layer for the Verminator Wasp Drone is designed to provide real-time visual monitoring and transmission capabilities. This layer consists of several subsystems that are each tasked with various parts of the image transmission pipeline. The FPV Camera Subsystem is used to capture live video. The Video Transmission Subsystem is used to send video over a radio connection. The Video

Capture Subsystem allows the live video feed to be presented to a host computer as a standard USB Video device.

2.5 Camera Application Layer Description

The Camera Application layer provides visual feedback for the user. Through an Android application, the user can view a live feed from the camera in order to help piloting outside line-of-sight.

2.6 Sprayer System Layer Description

The Sprayer Layer is built around a PWM-controlled switch that activates a liquid pump. The Verminator Wasp Drone is modular in the sense that the flight controller has auxiliary outputs to control any other kind of payload device.

2.7 System Architecture Diagram

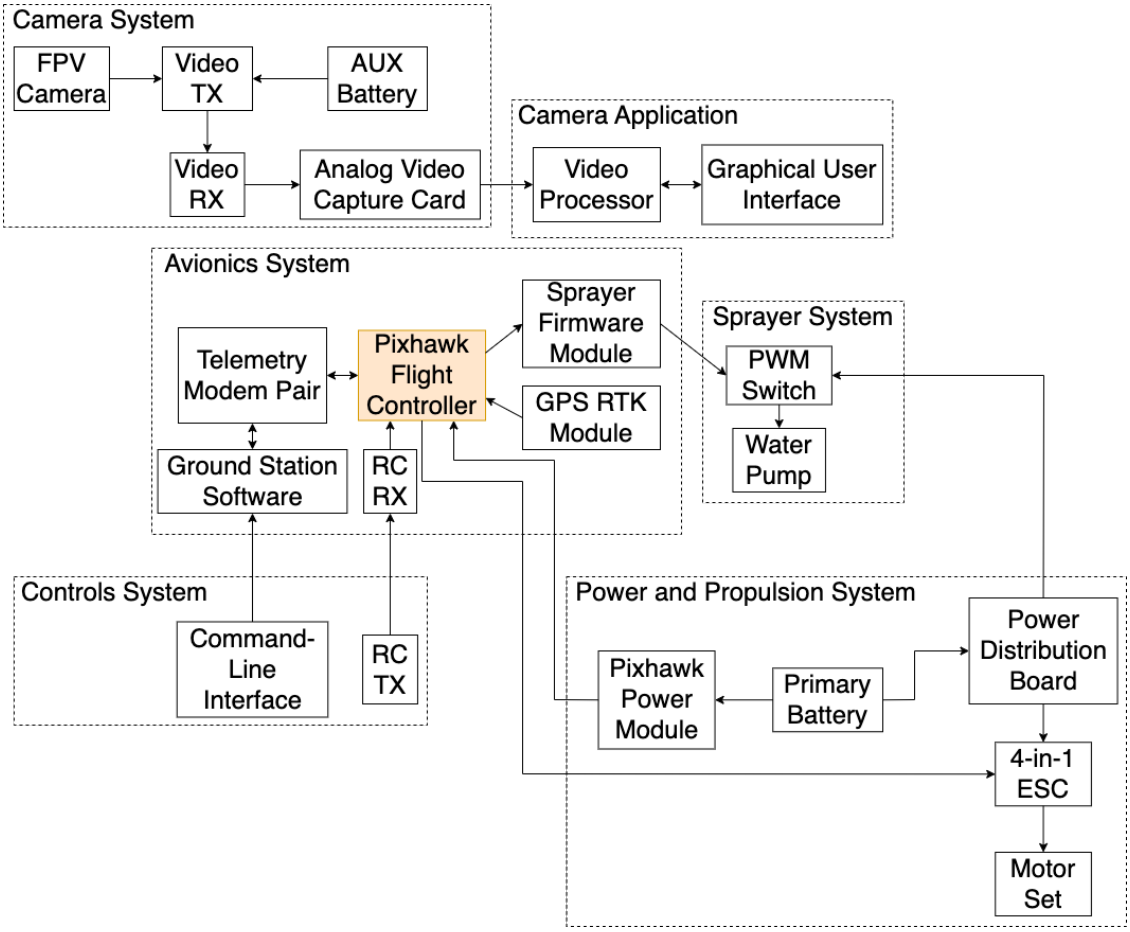


Figure 1: Overall system architecture showing data and power connections.

3 Avionics System Layer Subsystems

The avionics layer consists of the flight controller subsystem, the telemetry subsystem, the sprayer firmware module, and the GPS module. The flight controller interacts with all components of the layer, and the rest of the drone, using the PX4 firmware which is compatible with Pixhawk standard flight controllers.

In the Avionics Layer, only the Sprayer Firmware Module subsystems was designed and implemented in-house for this project. The other subsystems are comprised of off-the-shelf parts and communicate using industry-standard open-source protocols. As such, only the Sprayer Firmware Module Subsystem will be discussed in detail.

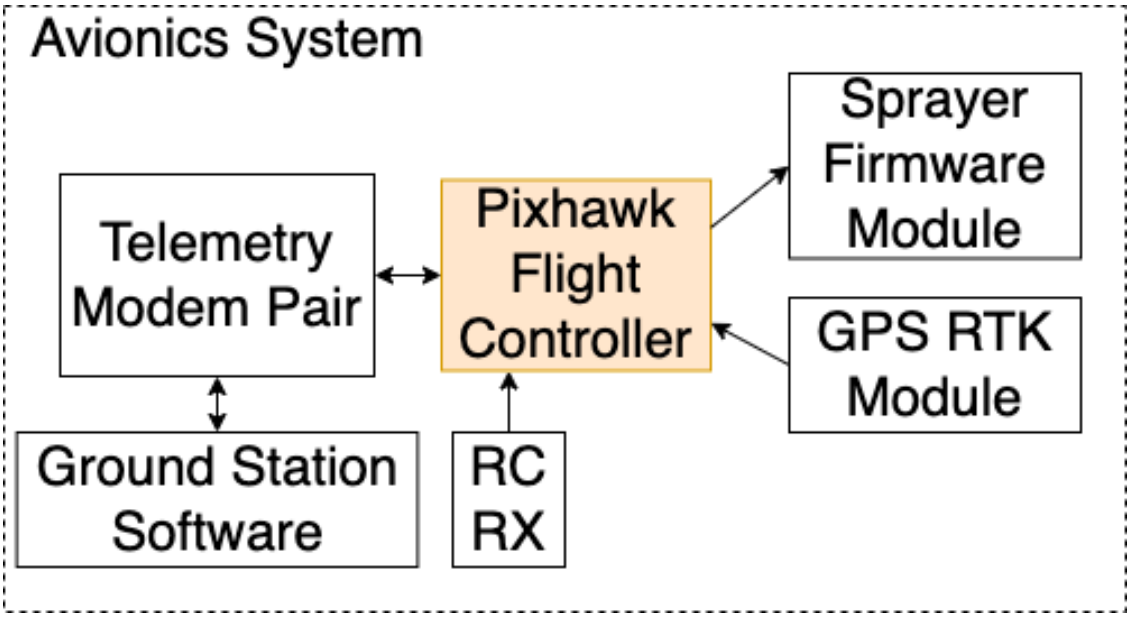


Figure 2: Avionics system architecture

3.1 Layer Operating System

The Cube Orange+ is fully compatible with the open-source PX4 firmware. In this product, the flight controller is running the PX4 Autopilot Firmware.

3.2 Layer Software Dependencies

The PX4 Development Team specifies several dependencies to develop for PX4 Autopilot. These include the following Python 3 packages:

- pyserial
- empty
- toml
- numpy
- pandas
- jinja2

- pyyaml
- pyros-genmsg
- packaging
- kconfiglib
- future
- jsonschema

And the following C and C++ development tools:

- make
- cmake
- gcc
- g++

3.3 Layer Data Structures

The PX4 firmware and hardware uses the industry-standard MAVLink protocol to communicate between hardware components over wire and wireless networks. The structure of the MAVLink protocol packet is detailed below:

```
uint8_t magic;              ///< protocol magic marker
uint8_t len;                ///< Length of payload
uint8_t incompat_flags;     ///< flags that must be understood
uint8_t compat_flags;       ///< flags that can be ignored if not understood
uint8_t seq;                ///< Sequence of packet
uint8_t sysid;              ///< ID of message sender system/aircraft
uint8_t compid;             ///< ID of the message sender component
uint8_t msgid 0:7;          ///< first 8 bits of the ID of the message
uint8_t msgid 8:15;         ///< middle 8 bits of the ID of the message
uint8_t msgid 16:23;        ///< last 8 bits of the ID of the message
uint8_t payload[max 255];   ///< A maximum of 255 payload bytes
uint16_t checksum;          ///< CRC-16/MCRF4XX
uint8_t signature[13];      ///< Signature which allows ensuring that
                            ///< the link is tamper-proof (optional)
```

The PX4 firmware and compatible hardware uses another industry-standard communication protocol called DroneCAN. While MAVLink is used for networking purposes, DroneCAN is a low-level protocol designed to allow peripheral components, such as motors, to interface with the flight controller. Implementation details for DroneCAN are readily available, however, it is outside the scope of this document [4].

3.4 Layer Data Processing

MAVLink messages propagate over several serial connections including UART and USB and a network through TCP and UDP. In the PX4 firmware, MAVLink packets are processed by the MAVLink module.

After a MAVLink packet is received and processed, it is categorized into several message types: Commands, Setpoints, Sensor Data, Status, and Parameters. Message transmission and processing works in

a similar fashion to message passing interface and snooping protocols. Messages are posted on the bus where other modules are listening. Then the intended destination will receive the message and process them.

Inter-process communication is handled uORB (micro Object Request Broker) which is included in the PX4 firmware. After received data is parsed, it is published as a uORB topic. Control loops (attitude, position, navigation) subscribe to relevant topics, process them, and publish actuation commands to target controllers.

Like MAVLink, DroneCAN utilizes a single bus architecture with a snooping-like policy. Messages are broadcasted on the bus where individual components can listen on the bus and receive relevant messages. The difference between DroneCAN and MAVLink is that DroneCAN is on the CAN bus and MAVLink is on a serial bus.

In this layer, MAVLink is used for telemetry information between the drone and the ground station while DroneCAN is used as the communication protocol between drone components.

3.5 Programming Languages

The PX4 firmware is written in C and C++. The custom modules used for the Verminator Wasp Drone are programmed in C. Makefiles are used to compile the firmware.

3.6 Sprayer Firmware Module Subsystem

3.6.1 Subsystem Hardware

The sprayer firmware runs as a module on the CubePilot Cube Orange+. The sprayer firmware module can be triggered using the built-in PX4 shell, or through the RC components of the Controls System.

3.6.2 Subsystem Operating System

The firmware module is built as a component of the PX4 Autopilot Firmware.

3.6.3 Subsystem Software Dependencies

The dependencies for this module are provided by the PX4 Development Team. These dependencies can be accessed using the following header files:

- uORB/uORB.h
- uORB/topics/actuator_outputs.h

Other dependencies are required to integrate the module into the firmware as a whole. To do so, CMake is required.

3.6.4 Subsystem Programming Languages

This module is programmed entirely in C.

3.6.5 Subsystem Data Processing

The module will be called as a command with the name of the module and user input. To start the sprayer, sprayer start will be given as a call in the drone's shell. The following code snippet describes the algorithm that will run with the start call:

```
struct actuator_outputs_s actuators = {};
memset(&actuators, 0, sizeof(actuators));
int actuator_pub = orb_advertise(
    ORB_ID(actuator_outputs_1), &actuators);
if (actuator_pub == nullptr) {
```

```
        PX4_INFO("Error: Failed to advertise actuator_outputs_1");
        return 1;
    }

    if (strcpy(argv[1], "start") == 0) {
        actuators.output[0] = 1.0f;
        orb_publish(ORB_ID(actuator_outputs_1), actuator_pub, &actuators);

    }
```

The sprayer can be stopped from the command line using the sprayer stop call. The following code snippet describes the algorithm that will run with stop call:

```
    struct actuator_outputs_s actuators = {};
    memset(&actuators, 0, sizeof(actuators));
    int actuator_pub = orb_advertise(
        ORB_ID(actuator_outputs_1), &actuators);
    if (actuator_pub == nullptr) {
        PX4_INFO("Error: Failed to advertise actuator_outputs_1");
        return 1;
    }

    if (strcpy(argv[1], "stop") == 0) {
        actuators.output[0] = 0.0f;
        orb_publish(ORB_ID(actuator_outputs_1), actuator_pub, &actuators);

    }
```

In both cases, the steps are similar: subscribe to the actuator controls for the particular aux output that the pump switch is connected to, set a high PWM output on start, set a low PWM output on stop.

3.7 Flight Controller Subsystem

The flight controller serves as the "brain" of the drone. All inputs go into the flight controller and all commands used to execute actions originate from the flight controller.

3.7.1 Flight Controller Subsystem Hardware

The flight controller for this drone is the CubePilot Cube Orange+ with the ADS-B IN Carrier Board. The Cube Orange is a standard Ardupilot flight controller that is compatible with the Pixhawk standard and ecosystem of third-party products.

3.8 Telemetry Subsystem

The telemetry subsystem ensures that the drone is aware of its location and is capable of receiving and transmitting vital commands and information between itself and the ground station.

3.8.1 Telemetry Subsystem Hardware

The telemetry subsystem consists of a pair of telemetry receivers and transmitters and a separate computer as the ground station. For the Verminator Wasp Drone, the telemetry radios are a set of RF Design RFD 900x-US. One radio is connected to the ground station using USB while the other is connected to the flight controller. The telemetry modem is mounted using custom 3D printed hardware [Appendix 9.2].

3.8.2 Subsystem Data Processing

The ground station software and the telemetry radios communicate with the flight controller using the MAVLink protocol.

3.9 GPS Subsystem

3.9.1 Subsystem Hardware

The GPS module is the CubePilot Here3 which is capable of centimeter-level accuracy.

3.9.2 Subsystem Data Processing

The flight controller and the GPS module communicate using the DroneCAN protocol.

4 Controls Layer Subsystems

The Controls Layer serves as the intermediary between the drone and its operator. It consists of two independent subsystems that are grouped based on their role in the system: the RC Subsystem and the Command-Line Interface Subsystem. Both subsystems can be used to relay commands to the drone's onboard systems. This communication not only handles the drone's overall behavior but also manages specific functions, such as activating the sprayer outlined in the Sprayer Layer. The Controls Layer also works in close coordination with the Avionics Layer to ensure that all commands are executed in real-time, for the RC Subsystem, or programmatically for the Shell subsystem.

The Controls System layer is made up of off-the-shelf components, so a high-level description is provided in this section.

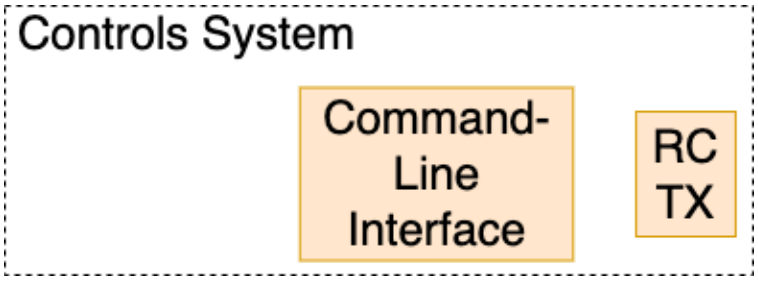


Figure 3: Controls Layer

4.1 RC Subsystem

The RC subsystem allows the user to control the drone using an RC remote. The RC subsystem directly controls the drone's hardware using the RC output ports on the receiver and the RC inputs on the flight controller. Commands are sent through the RC transmitter and are picked up by the RC receiver as part of the Flight Controller System.

4.1.1 RC Subsystem Hardware

The RC Subsystem consists of the following hardware:

- FrSky Taranis Q X7 RC Controller
- 2.4 GHz Radio Transmitter
- 2S (7.2V) 2600mAh 18.72Wh LiPo Battery

4.1.2 RC Subsystem Data Processing

The RC controller and transmitter sends commands to the drone using a 2.4 GHz radio connection to the RC receiver mounted on the drone. There is a guided handshake setup to calibrate the transmitter and receiver. There is also the option to select a frequency and channel manually.

4.2 Shell Subsystem

The Shell Subsystem is a built-in system in the PX4 Autopilot Firmware. The Shell provides a command-line interface for the user to send commands to the drone's flight controller. Enabling the command-line feature as part of the Shell Subsystem allows the user to have programmatic control over all of the drone's functions. This feature is more useful for automated missions configured using PX4's autopilot system.

4.2.1 Shell Subsystem Hardware

The Shell Subsystem is runs on Pixhawk-compliant flight controllers that are loaded with the PX4 Autopilot firmware. The flight controller is part of the Avionics System, so refer to Avionics System Layer section of this document.

4.2.2 Shell Subsystem Software Dependencies

The Shell Subsystem is accessed through the ground station software that is part of the Avionics System Layer. The Avionics System Layer uses QGroundControl as its ground station software. In this product, QGroundControl is required to use the Shell Subsystem. Alternative products such as Mission Planner will also have access to similar command-line interfaces, but that will require new firmware such as ArduPilot.

5 Power and Propulsion Layer Subsystems

The Propulsion System Layer is responsible for providing the required thrust and control to maneuver the drone during the flight. This layer is responsible for making sure that the drone remains stable while carrying its payload, such as the camera system. The propulsion system consists of four motors, four electronic speed controllers, and a lithium-polymer battery. Combining all of these components generates power to ensure smooth flight operations.

The Power and Propulsion System is largely made up of off-the-shelf components, so little software implementation will be discussed here. Instead, hardware implementation will be discussed in detail.

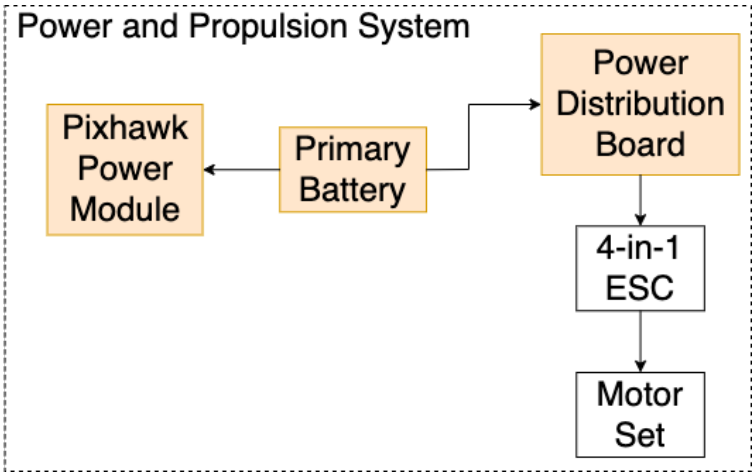


Figure 4: Propulsion consists of motor and power subsystems.

5.1 Propulsion Subsystem

The motor subsystem provides the necessary thrust to lift the drone and payload.

5.1.1 Propulsion Subsystem Hardware

The Propulsion Subsystem is composed of 4 brushless motors and a 4-in-1 electronic speed controller (ESC). The following are the particular components used in this drone:

- Readytosky 2212 920KV Brushless Motors (x4)
- Diatone Mamba 4-in-1 40A Electronic Speed Controller (ESC)
- Minimum 16 AWG Wires for ESC to Motor Connection
- 3.5mm Bullet Connectors for ESC to Motor Connection
- Minimum 12 AWG Wires for Battery to ESC
- XT60 Connectors for Battery to ESC connection

5.1.2 Propulsion Subsystem Hardware Configuration

The Propulsion Subsystem requires specific hardware configuration for correct implementation. Firstly, the ESC will need to receive power from the Primary Battery. To do so, an XT60 connector needs to be soldered onto at least 12 AWG wires. The 12 AWG wires needs ot be soldered onto the battery leads of the ESC.

Power will also need to be sent to the motors themselves. Each motor is attached independently to the ESC. At least 16 AWG wires with 3.5mm female bullet connectors will be soldered to the power leads of the ESC. Refer to [appendix reference here] for the proper setup used in this product.

To receive input from the flight controller, PWM leads will need to be used. The flight controller's actuator PWM outputs are in the form of GPIO-style male leads. Only one power and ground wires are needed to connect from the ESC to the flight controller. The remaining 4 signal wires from the ESC will be connected to their corresponding actuator outputs on the flight controller. In our case, 22 AWG wires are soldered onto the leads on the ESC. The opposing ends of those wires are crimped with 2.54mm single female DuPont connectors. The crimped end of the wire will connect to the signal output of the flight controller. Refer to Appendix 9.8 for more details on wiring the ESC and flight controller.

5.1.3 Propulsion Subsystem Operating System

The ESC has its own firmware that interacts with the flight controller's PX4 Autopilot Firmware. This interaction occurs using the DShot protocol.

5.1.4 Subsystem Software Dependencies

The Propulsion Subsystem interacts with the flight controller through DShot and PWM. Both methods are widely supported on firmware options such as PX4, Ardupilot, and Betaflight. In this product, we will be using PX4.

5.2 Power Subsystem

The power subsystem provides the electric power necessary for the drone to function.

5.2.1 Power Subsystem Hardware

The Power Subsystem consists of the following hardware components:

- 3s (11.1V) 50C 5200mAh LiPo Battery
- 8 Output Power Distribution Board
- Pixhawk Power Module
- 14 AWG Wire for power to payload components
- XT60 Connectors for all electric connections

The 3S LiPo battery provide serves as the Primary Battery for the drone. The Primary Battery is connected to a Pixhawk Power Module which leads to the flight controller and the power distribution board which leads to the Propulsion Subsystem and may be used to power payload components.

5.2.2 Power Subsystem Hardware Configuration

The hardware implementation for the Power Subsystem is relatively simple compared to the Propulsion Subsystem.

A minimum of 12 AWG wires are used to connect the Primary Battery to the Power Distribution Board (PDB) using XT60 connectors. This requires The wire to be soldered to the battery leads on the PDB and an XT60 connector is soldered on the opposing ends of those wires. Then, the power module can be connected to the PDB which gives power to all of the payload components and motors on the drone.

6 Camera System Layer

The Camera System Layer is designed to provide visual monitoring and transmission capabilities that are important to locate the target and for drone navigation. This layer is responsible for ensuring that the operator receives uninterrupted video feedback to track the wasp nest and monitor its position. The two main subsystems are the FPV Camera subsystem and Video Transmitter subsystem, which work together to capture and transmit live video data.

The Camera System Layer is made up of 3 subsystems which are listed below. This system is largely made up of off-the-shelf components, so this section will mainly focus on the hardware implementation.

- FPV Camera Subsystem
- Video Transmission Subsystem
- Video Capture Subsystem

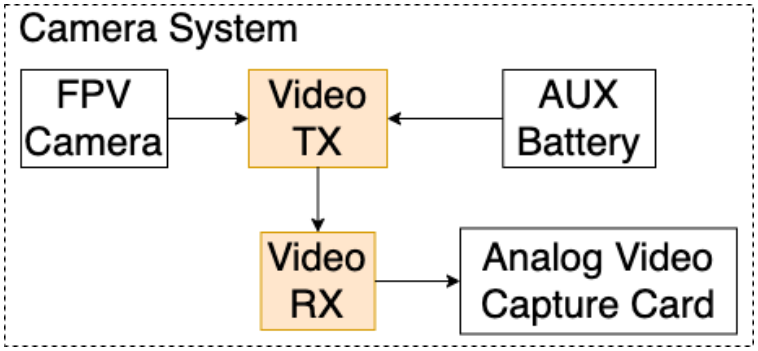


Figure 5: Camera system highlighting FPV camera

6.1 FPV Camera Subsystem

The FPV Camera Subsystem allows the user to have a first-person view of the drone. This allows for the drone to be maneuvered in confined spaces or outside line-of-sight.

6.1.1 FPV Camera Subsystem Hardware

The FPV Camera Subsystem has these hardware components:

- RunCam Night Eagle 3 v2
- RunCam On-Screen Display (OSD)

6.1.2 FPV Camera Subsystem Hardware Configuration

The FPV camera produces frames in the analog NTSC format. A cable from the FPV camera will connect to the video transmitter using a JST connector. A particular wire in connector is for composite video. Composite video will be used as the medium to transmit to the user.

6.2 Video Transmission Subsystem

The Video Transmission System is used to send video from the drone to the user wirelessly. Data is sent over 5.8 GHz radio as NTSC format video.

6.2.1 Video Transmission Subsystem Hardware

The Video Transmission Subsystem is made up of off-the-shelf hardware components. The following are the parts:

- AKK 5.8 GHz Analog Video Transmitter (VTX) [Appendix 9.5]
- RC832S 5.8 GHz Analog Video Receiver (VRX)

6.2.2 Video Transmission Subsystem Hardware Configuration

The connection between the FPV Camera Subsystem and Video Transmission Subsystem required some custom work. The connectors included with the VTX and the FPV camera are not compatible due to differing pinouts and connector standards. To alleviate this, the connectors were cut off and the wires from the FPV camera were spliced with their corresponding wires on the VTX by soldering. Refer to Appendix 9.7 for more information regarding electrical wiring. The VTX is mounted to the drone with custom 3D printed hardware [Appendix 9.5].

6.3 Video Capture Subsystem

The Video Capture Subsystem is used to retrieve video frames from the Video Transmission Subsystem and present them to a user-space camera application on a Linux device. The Video Capture Subsystem appears as a USB Video Device for the Camera Application System.

6.3.1 Video Capture Subsystem Hardware

The following are the hardware chosen for the Video Capture Subsystem:

- Analog Video Capture Card with Digital USB Output
- USB 2.0 OTG Cable

6.3.2 Video Capture Subsystem Hardware Configuration

On some systems, particularly Android devices, issues arise over which device is the host in software. Typically, Android smartphones are not designed to be a host, so a USB 2.0 OTG cable is necessary for the Android device to become a host. In our case, the video capture card was plug and play with Linux systems.

Regardless of the device type, setup is relatively simple. The video capture card will connect to the USB 2.0 OTG cable (if necessary), then will be connected to an available USB port on the host computer. The opposite end of the video capture card will have several RCA connectors. The connector of interest for this product is composite video (labeled yellow). The composite video male connector from the VRX will connect to the composite video female connector on the capture card.

6.3.3 Video Capture Subsystem Data Processing

The video capture card is equipped with hardware components to convert analog video to digital video.

7 Sprayer System Layer

The Sprayer System is the hardware that will be used to spray insecticides from a safe distance. The Sprayer System Layer is a system made up of only electrical hardware components. For simplicity, this system is not divided into subsystems. Because this system is assembled from various off-shelf electronic components, this section will primarily focus on the hardware aspect of the implementation.

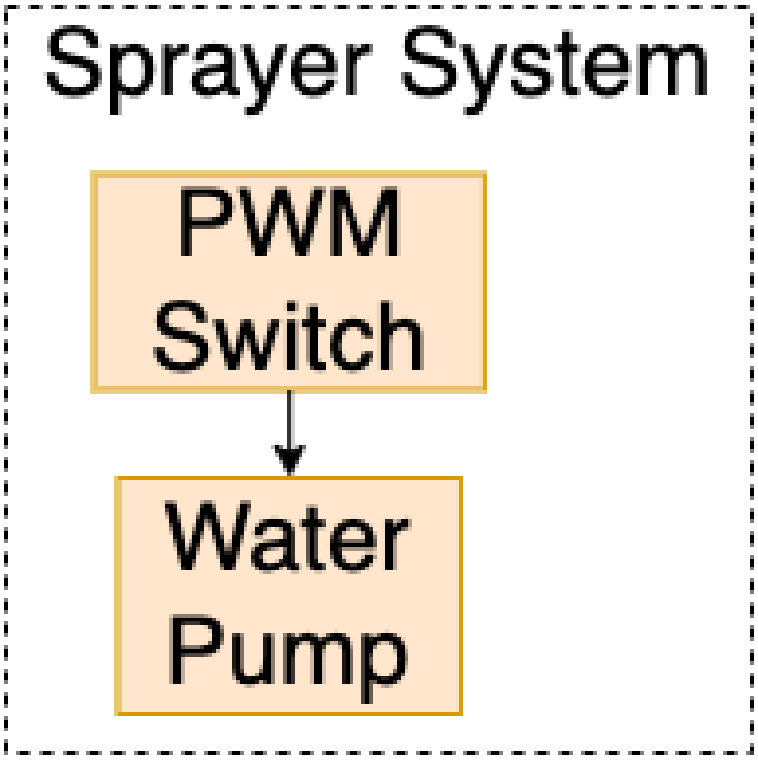


Figure 6: Sprayer system highlighting brushed ESC

7.1 Sprayer System Hardware

The Sprayer System includes the following hardware components:

- 12V PWM-Controlled Switch
- 12V Water Pump
- HDPE Bottle 300 mL
- Custom 3D Printed Bottle Mount [Appendix 9.3]
- 1/4" ID 3/8" OD Silicone Tubing
- 1/4" Male NPT Cone Pattern Spray Nozzle Brass
- 1/4" Female NPT to Barbed Fitting Adapter Brass
- Custom 3D Printed Nozzle Mount [Appendix 9.1]

7.2 Sprayer System Hardware Configuration

The hardware for the Sprayer System requires custom configuration for the drone. The PWM-controlled switch allows a circuit to be completed upon either a high PWM signal or a low PWM signal through normally open and normally closed gates. Because the drone's feature is the distribution of insecticide on-demand, the normally closed lead will be used. A circuit will be created by soldering at least an 18 AWG wire to the positive lead on the water pump. The opposite end of the wire is connected to the normally closed, labeled NC, on the PWM switch. A hot wire from the power distribution board (PDB) will be connected to the common power lead on the PWM switch, labeled COM. The ground wire from the water pump will be soldered to the ground lead on the pump, then connected to the PDB through an XT60 connector.

To establish control of the PWM switch and pump, a PWM wire is connected from the PWM switch to an AUX actuator output on the flight controller. The connector used is a 2.54 mm DuPont female which is connected to GPIO-style pins on the flight controller. Refer to Appendix 9.8 for the PWM pinouts on the flight controller.

The liquid pump is mounted to a subframe on the drone. Both components are custom 3D printed hardware [Appendix 9.4].

7.3 Sprayer System Data Processing

The Sprayer System's only data processing (as part of this specific system) is through the use of PWM signals to toggle the PWM-controlled switch. The PWM signal is sent from the flight controller to the Sprayer System. The particular component is the custom Sprayer Firmware Module which sets high and low PWM signals for the pump.

8 Camera Application System Layer

The Camera Application System is a software system designed to provide a real-time first-person view for the user to pilot the drone. The Camera Application System consists of two subsystems:

- Video Processor
- Graphical User Interface

The customer-focused name for this system is RJCamera.

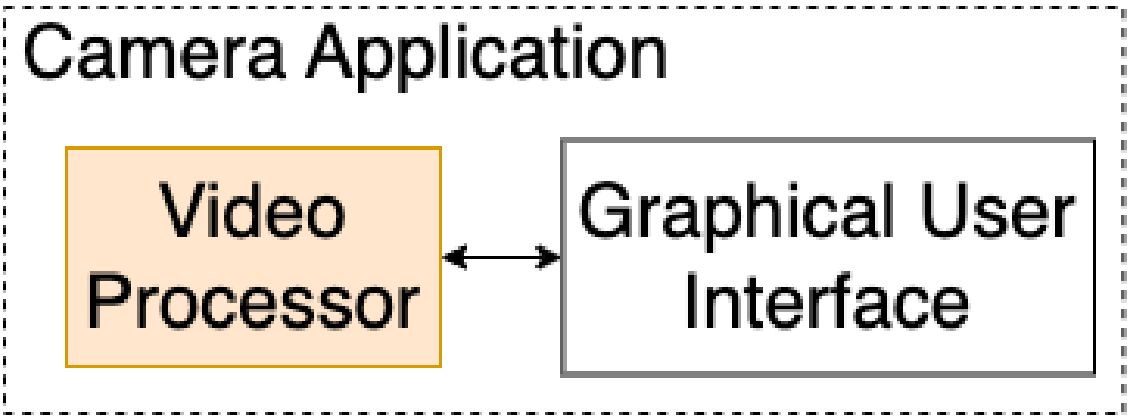


Figure 7: Avionics system architecture

8.1 Camera Application System Hardware

The Camera Application System requires the user provide a mobile or desktop computer. The following are requirements for this computer:

- x86-64 or ARM64 Multi-core Processor
- 1 GB System Memory
- Graphics Processor with OpenGL support for hardware acceleration

The Camera Application System is compatible on a wide-range of hardware, but the Verminator Team recommends the above specifications for optimal performance. The Camera Application System has been validated on the following hardware platforms:

- AMD Ryzen 7 5800x, 32GB RAM, Nvidia RTX 4070 Super
- Intel Core i7 9700, 16GB Ram, iGPU
- Intel Core i7 7700, 16GB Ram, iGPU
- Apple M1, 16GB Ram, iGPU
- Raspberry Pi 4 ARM Cortex 32-bit, 4GB Ram, iGPU

8.2 Camera Application System Operating System

Camera Application System is designed and developed on Debian-based and Fedora-based Linux distributions. The Camera Application System can be built from source to operate on Windows and MacOS systems, but the Verminator Team does not officially support Windows and MacOS operating systems.

The Camera Application System has been validated on the following operating systems:

- Fedora 42
- Fedora 41
- Ubuntu 22.04 LTS
- Ubuntu 24.04 LTS
- Debian 12 Stable
- Raspberry Pi OS 6.6

Targeting and validating for common Linux distributions and platforms allow the Camera Application System to be compatible on a wide-range of systems. Moreover, these platforms are open-source which follows the open-source requirement for this project. Users looking to run this application on non-validated systems will need to build from source.

8.3 Camera Application System Software Dependencies

The following are software dependencies to develop the application:

- Qt6 Base
- Qt6 Multimedia
- OpenCV C++
- OpenGL Mesa
- GNU C Compiler
- GNU C++ Compiler
- Make

The QT specific dependencies can be installed using the install script, named configure.sh, in the rjc repository. That shell script installs the following packages for Debian-based systems:

- qt6-base-dev
- qt6-multimedia-dev
- libopencv-dev
- libgl1-mesa-dev

There is no shell script to configure Fedora-based systems for development. Refer to the below list of packages to install using the DNF package manager:

- qt6-qtbase-devel
- qt6-qtmultimedia-devel
- opencv-devel
- mesa-libGLU-devel

8.4 Camera Application System Programming Languages

The Camera Application System and all of its components are programmed in C++ with C++17 Standard Library, Qt6, and OpenCV.

8.5 Video Processor Subsystem

The Video Processor is a software component that is in charge of managing any USB Video devices.

8.5.1 Video Processor Subsystem Data Structures

The Video Processor is implemented as a C++ class. This class can be instantiated as an object when accessing a USB Video device.

The following is a C++ code snippet describing the class and its functions and variables:

```
class VideoProcessor : public QObject{
    Q_OBJECT

public:
    explicit VideoProcessor(const QCameraDevice &cameraDevice,
        QObject *parent = nullptr);
    void stop();

signals:
    void frameReady(const QImage &frame);

private:
    QCamera *camera;
    QMediaCaptureSession videoCap;
    QVideoSink *videoSink;

    cv::Mat emblemOverlay;
    cv::Mat crosshairOverlay;

    void handleFrame(const QVideoFrame &frame);
    cv::Mat qimageToMat(const QImage &image);
    void addOverlay(cv::Mat &base, const cv::Mat &overlay);
};
```

The VideoProcessor class inherits from QObject, so that it can be run on a separate thread in order to keep the GUI subsystem alive and responsive. USB video devices are accessed as QCamera and QCameraDevices. Image processing on each frame from a USB Video device is handled using OpenCV and requires the conversion of a QFrame into a OpenCV matrix. QMediaCaptureSession and QVideoSink are Qt Multimedia objects that allow the USB video device to be used for image capture.

8.5.2 Video Processor Subsystem Data Processing

The Video Processor Subsystem is designed to allow a user to modify the source code to insert any image processing desired. For this project, OpenCV is used for alpha compositioning of frames and overlay images. Alpha composition is an image blending algorithm that allows a foreground and background image to be superimposed based on the alpha channel in an image's file format. In this case, the file format is PNG.

To prepare an image for blending, images need to be converted into an appropriate color channel format. For OpenCV operations, the color channel format is RGBA. Images can be read, converted, and resized like so:

```
overlay = cv::imread("path/to/image.png", cv::IMREAD_UNCHANGED);
cv::cvtColor(overlay, overlay, cv::COLOR_BGRA2RGBA);
cv::resize(overlay, overlay, cv::Size(64, 64));
```

To prepare a frame for blending, the frame, in QImage object, needs to be converted into an OpenCV matrix. The following code snippet shows how this is done. The image variable is of type const QImage&.

```
cv::Mat(image.height(), image.width(), CV_8UC4,
    const_cast<uchar*>(image.bits()), image.bytesPerLine()).clone();
```

After images are converted to the required color format and desired size, and QFrames are converted into an OpenCV matrix, blending can proceed using the alpha composition algorithm. The following C++ code snippet specifies how blending is to be done:

```
for (int y = 0; y < overlay.rows; ++y) {
    for (int x = 0; x < overlay.cols; ++x) {
        const cv::Vec4b &pixel = overlay.at<cv::Vec4b>(y, x);
        uchar alpha = pixel[3];
        if (alpha > 0) {
            for (int c = 0; c < 3; ++c) {
                baseFrame.at<cv::Vec4b>(y, x)[c] =
                    (alpha * pixel[c] + (255 - alpha)
                    * baseFrame.at<cv::Vec4b>(y, x)[c]) / 255;
            }
        }
    }
}
```

The alpha composition algorithm is to be called for every frame before the frame is sent to the GUI for display.

8.6 Graphical User Interface Subsystem

The Graphical User Interface Subsystem, will now be referred to as GUI Subsystem, provides a system that the user can interact with to control the Camera Application System.

8.6.1 Graphical User Interface Subsystem Data Structures

The GUI Subsystem is to be implemented as a class inheriting from the QMainWindow class found in Qt6 Base. Moreover, in order to run the GUI on a separate thread, specifying QObject is required.

```
class MainWindow : public QMainWindow {
    Q_OBJECT

public:
    MainWindow();
    void setStatus(QString statusmsg);

private:
```

```
    // Private variables
    QMenuBar *mainMenu;
    QMenu *fileMenu;
    QMenu *cameraMenu;
    QLabel *videoDisplay;
    VideoProcessor *vidProc;

    // Private functions
    void init();
    void createMenuActions();
    void updateCameraMenu();
    void startCamera(const QCameraDevice &camDev);
};
```

The MainWindow class is instantiated in the main function and is used to control the user interface. Additional private variables and functions are used for internal processing of the system. A pointer to a VideoProcessor object is used to give the user interface control of any USB video deivce.

8.6.2 Graphical User Interface Subsystem Data Processing

Data processing in this subsystem are primarily responsible for setting up a USB video device for video capture and obtaining a list of USB devices.

A list of USB video devices can be obtained using Qt6 Multimedia library. The particular function is QMediaDevices::videoInputs() which returns a list of USB video devices. For the user interface, the list of devices is used to create menu buttons, so that the user can select a USB device that they want to use video capture on.

The function startCamera() is used to prepare a USB video device for video capture. To do so, a connection is established between a QLabel object (used for video display) and the Video Processor object. From there, frames outputted by the Video Processor object (using emitted signals) is displayed to the user.

9 Appendix A

9.1 Sprayer Nozzle Mounts

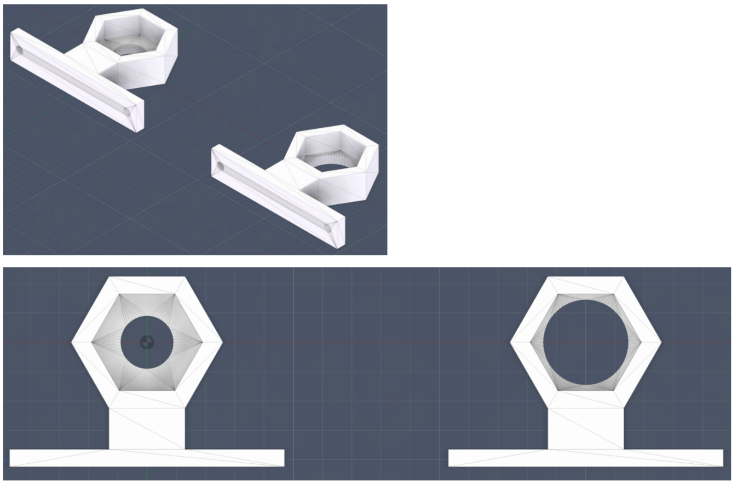


Figure 8: Sprayer nozzle mount

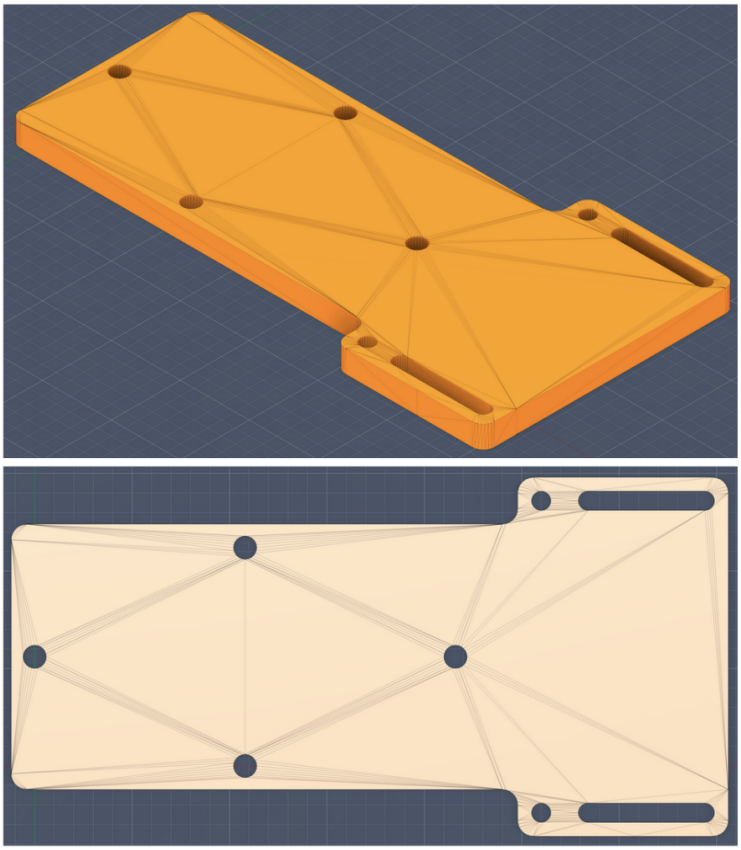


Figure 9: Sprayer nozzle subframe mount

9.2 Telemetry Modem Mounts

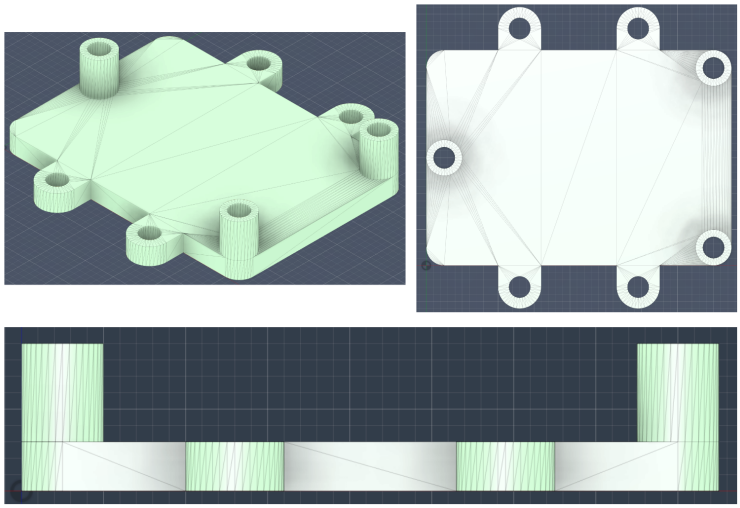


Figure 10: Telemetry modem mount

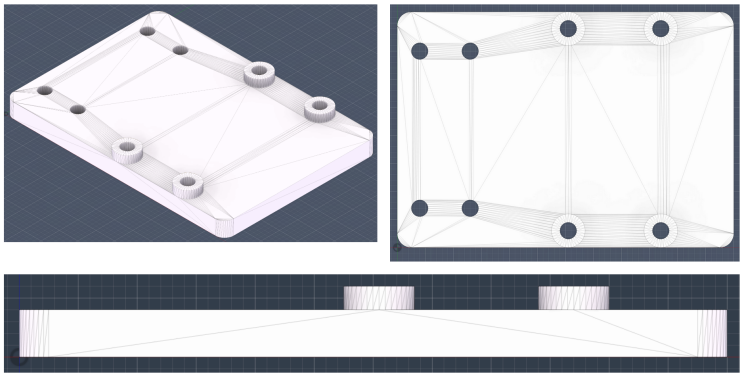


Figure 11: Telemetry modem subframe mount

9.3 Liquid Insecticide Tank Mounts

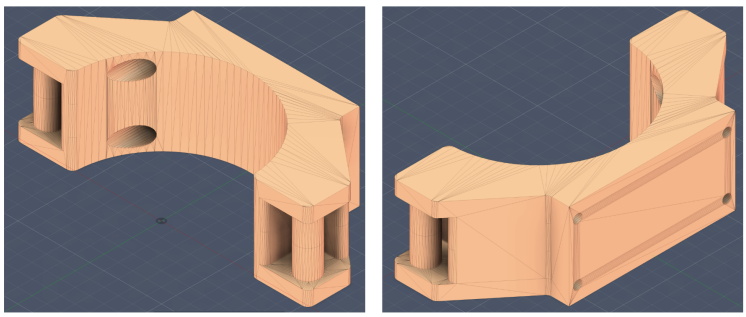


Figure 12: Mount for insecticide tank

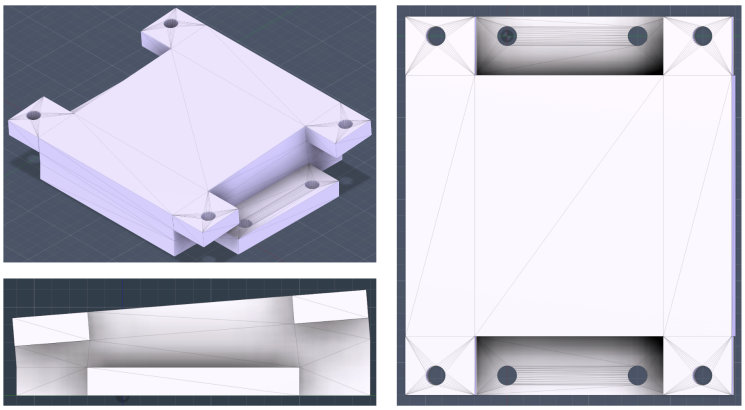


Figure 13: Mount to place insecticide tank at an angle to help dispend liquid.

9.4 Liquid Pump Mount

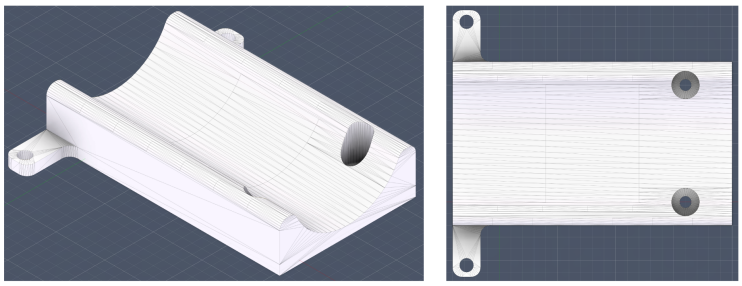


Figure 14: Liquid tank mount

9.5 Video Transmitter Mount

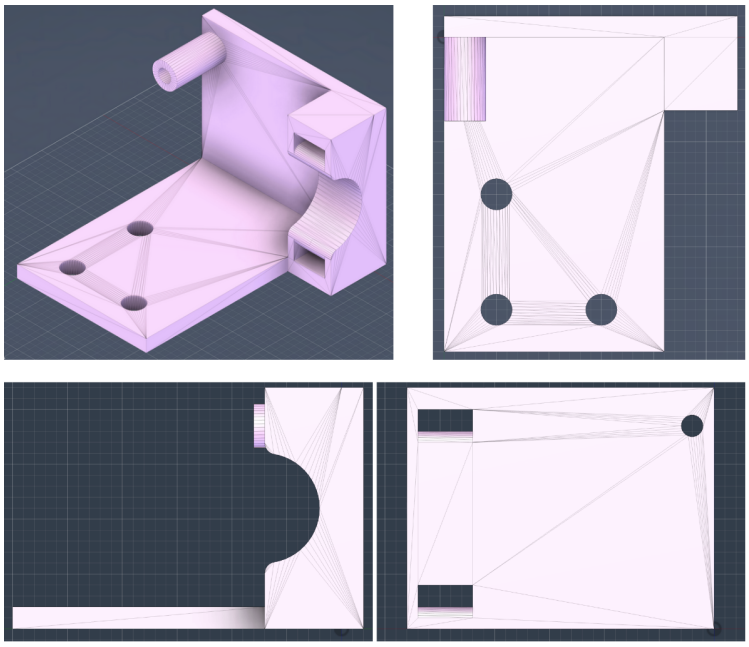


Figure 15: Mount for VTX

9.6 FPV Camera Mount

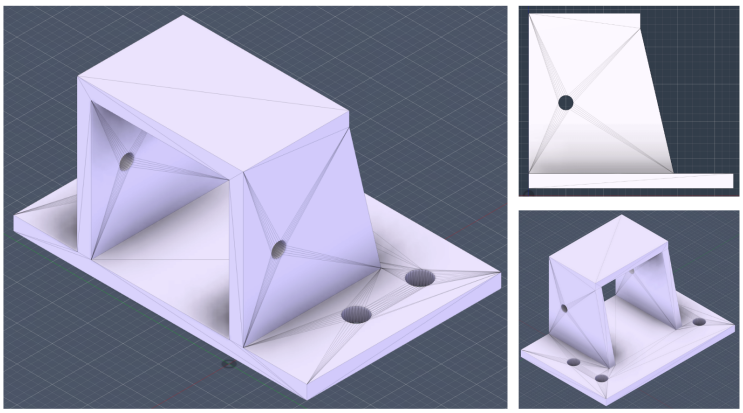


Figure 16: Mount for the RunCam Night Eagle 3 v2.

9.7 Wiring Diagram for Camera, On-Screen Display, and Video Transmitter

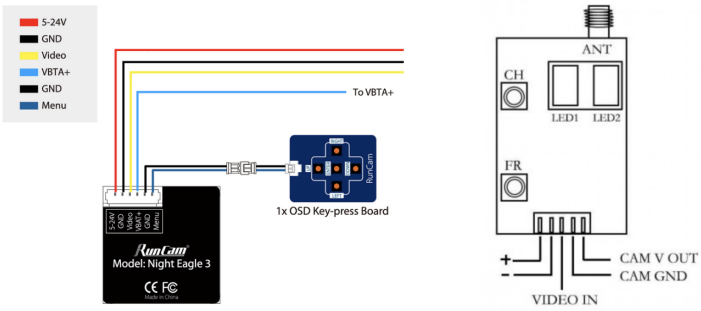


Figure 17: Wiring diagram for FPV camera and video transmitter [1, 5].

9.8 Wiring Diagram for Flight Controller and ESC PWM and Power

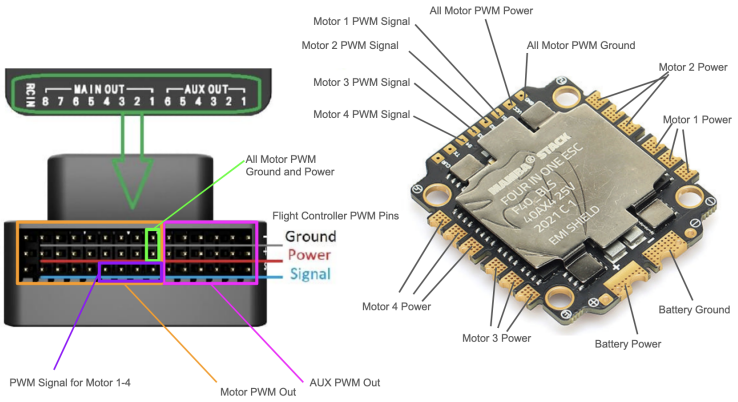


Figure 18: Wiring diagram for flight controller and ESC [2, 6].

REFERENCES

[1] AKK. *AKK K31 600mW manual*.

[2] Diatone. Mamba f40$_b$*lsdshot*6004*in*1*esc*40*a*6*s*.

[3] Tyler Nguyen et al. vermwasp-drone, 2025.

[4] Pavel Kirienko and UAVCAN. Dronecan specification.

[5] RunCam. *Night Eagle 3 v2 Manual*.

[6] PX4 Development Team. *CubePilot Cube Orange+ Flight Controller*.