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

Detailed Design Specification
CSE 4317: Senior Design II
Spring 2023

Top Drone: Mavericks
Raytheon Drone Project

Jaedyn Brown
Robert Carr
Javier Lopez
Ja'lun Morris
Pearl Iyayi

Revision History

Revision	Date	Author(s)	Description
0.1	2.20.2023	JB, RC, JM, JL, PI	document creation
0.2	2.24.2023	JB, RC, JM, JL, PI	draft completed

CONTENTS

List of Figures

1 INTRODUCTION

This document's purpose is to provide a detailed design specification for the UAV. This document builds upon the System Requirements Specification and the Architecture Design Specification. As described in the statement of work provided by the sponsor, the UAV must autonomously identify and immobilize opposing team UGVs within a given time. Section 4 provides a system overview of the UAV Sections 3 through 7 will describe in detail in system layer of the UAV.

At the core of the UAV is the computer layer subsystem. It is responsible for communicating between all other layers in the UAV's system. Other layers include the flight, weapons, power, and navigation layers.

- The flight system contains the flight controller, motors, and propellers - it will allow the drone to generate lift and fly stably.
- The navigation system will contain components such as GPS, RTK, and telemetry radios that will send and receive information for steering the drone.
- The weapons system will contain the A.C.E. Combat System that is required by the statement of work. The weapons will be used to fire at hostile vehicles. It also contains a camera that will be used for object detection and a gimbal for the camera.
- The power system will consist of the battery, electronic speed controllers, a power distribution board, connectors/cables, and power modules required for components. It's job is simply to provide power to each system.

The rest of this document will explain in greater detail each system and its subsystems, hardware components, operating systems, and software dependencies.

2 System Overview

The UAV drone system architecture can be divided into five high-level layers: Navigation, Weapons, Flight, Computer, and Power as seen in Figure 1. The Computer Layer pertains to the Raspberry Pi that is mounted to the drone. The Computer Layer shares relationships with the rest of the layers as all data is communicated to and from the computer. The Navigation Layer is responsible for the drone's ability to perform its autonomous movements. Its relationship with the Computer Layer is bidirectional. The Weapons Layer is responsible for the drone's ability to locate targets and fire upon them with a laser. This layer has a bidirectional relationship with the Computer Layer. The Flight Layer focuses on the physical components that are responsible for the drone taking flight. The Flight Layer has a bidirectional relationship with the Computer Layer. The Power Layer is responsible for providing power to the drone. This layer has a unidirectional relationship pointing toward the Computer Layer.

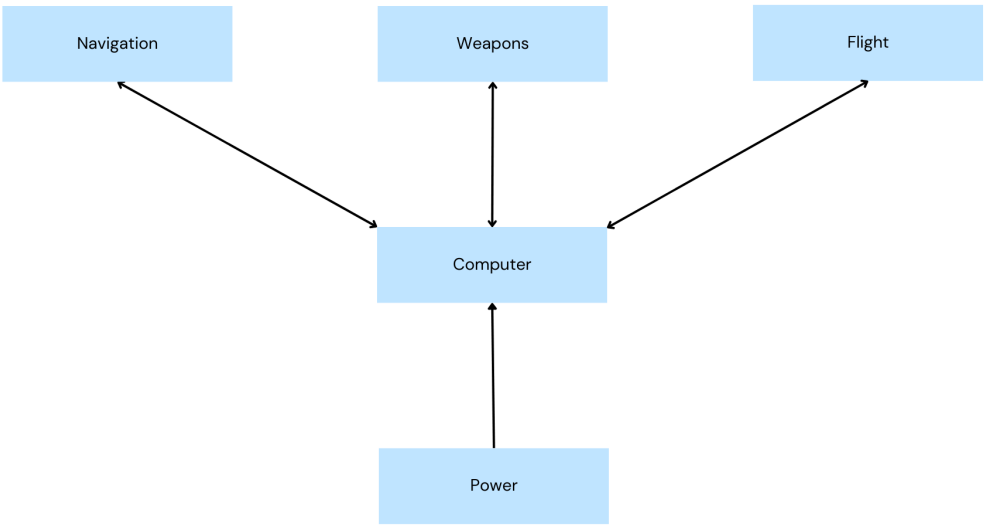

Figure 1: UAV Drone System Overview

3 Computer Layer Subsystems

Undoubtedly, the computer layer and its associated subsystems represent a mission-critical aspect of the drone's operational framework. The drone's ability to execute its intended functions with the utmost effectiveness is wholly dependent on the presence and performance of these highly intricate components. In their absence, the drone's operational capacity would be severely limited, rendering it virtually ineffective in carrying out its designated tasks.

3.1 Computer Layer Hardware

Some of the common hardware dependencies associated with the Raspberry Pi 4 include:

- Power supply
  - The Raspberry Pi 4 requires a power supply capable of delivering at least 5 volts and 3 amps to operate reliably.
- MicroSD card
  - The Raspberry Pi 4 uses a MicroSD card for storage of the operating system, software, and data.
- USB peripherals
  - Various USB peripherals can be used with the Raspberry Pi, including a keyboard and mouse.
- Display
  - A display, such as an HDMI monitor or TV, will be needed to connect to the Raspberry Pi and view the output.
- Camera
  - To use computer vision, a Pi Camera will need to be attached.
- Flight controller
  - For drone use, a flight controller that is compatible with the Raspberry Pi 4 will be required. This flight controller must also be able to communicate using the MAVLink protocol.

3.2 Computer Layer Operating System

The Raspberry Pi 4 runs the Raspberry Pi OS (32-bit), which is based on the Debian version 11 (Bullseye) operating system. Specifically, the kernel version utilized by this system is 5.15, and the release date for this version is September 22nd, 2022.

3.3 Computer Layer Software Dependencies

The specific software dependencies for the Raspberry Pi 4 will depend on the particular application it is being used for. For the current drone, the software dependencies are:

- Python

3.4 DroneKit Subsystem

DroneKit for Python is a subsystem, specifically a software development kit, that provides a Python API for building drone applications. The purpose of this subsystem is to simplify the development of the drone by providing a high-level, Python-based interface that abstracts the complexities of working with drone hardware and software.

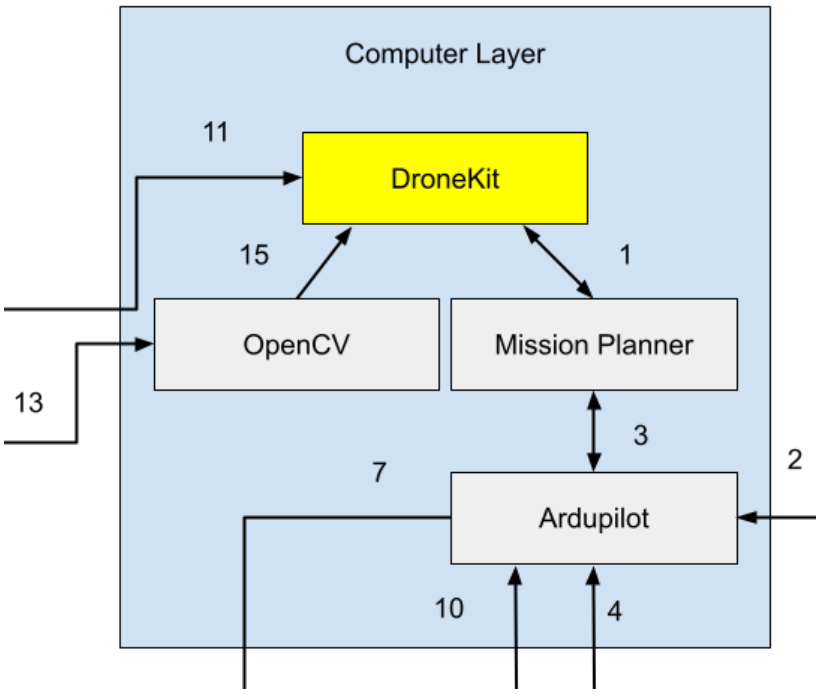

Figure 2: DroneKit Subsystem diagram

3.4.1 DroneKit Hardware

As a software library, DroneKit does not require any specific hardware dependencies. Even though it does not have any hardware dependencies, DroneKit requires proper hardware configuration to function properly.

3.4.2 DroneKit Operating System

DroneKit can be installed and used on any operating system that supports Python and the software dependencies listed below in section 3.4.3.

3.4.3 DroneKit Software Dependencies

- Python 2.7 or Python 3.4 and above
  - This is the primary programming language that DroneKit is built on, and as such, it is a prerequisite for DroneKit to operate correctly.
- MAVProxy
  - This is a command-line tool that provides communication and control between the drone and ground control station. DroneKit relies on MAVProxy to establish communication with the drone.

- Pymavlink
  - This is a Python implementation of the MAVLink protocol, which is a lightweight messaging protocol designed for communication between unmanned systems and ground control stations. DroneKit relies on Pymavlink to communicate with the drone using the MAVLink protocol.
- PySerial
  - This is a Python module for accessing the serial port, which is commonly used for communication between the drone and the ground control station. DroneKit relies on PySerialfor serial communication with the drone.

It is worth noting that the specific software dependencies required for DroneKit may vary depending on the version of DroneKit being used, as well as the specific use case and operating environment.

3.4.4 DroneKit Programming Language

DroneKit is a software development kit that provides a Python API for building drone applications.

3.4.5 DroneKit Data Structures

One of the key features of DroneKit is its rich set of classes and data structures that make it easy to interact with and control UAVs. Here are just a few of the notable classes and data structures:

- The 'Vehicle' class
  - This class represents a connected UAV and provides a wide range of methods and properties that can be used to interact with the UAV. The Vehicle class includes methods for arming and disarming the UAV, setting its mode, and controlling its movement.
- The 'LocationGlobal' class
  - This class represents a global coordinate location and includes methods for calculating the distance and bearing between two locations, as well as methods for converting between different coordinate systems.

DroneKit does not specify a particular structure for the packets that are used to transmit the data. Instead, the structure of the packets will depend on the specific communication protocol that is being used to transmit the data. However, DroneKit does provide a range of tools and libraries that can be used to parse and manipulate the data once it has been received by the computer.

3.4.6 DroneKit Data Processing

- Path planning algorithms
- Sensor fusion algorithms
- Control algorithms
- Data fusion algorithms

3.5 Mission Planner Subsystem

Mission Planner is a subsystem that provides a graphical user interface for configuring and controlling unmanned aerial vehicles and other autonomous vehicles. The purpose of this subsystem is to provide an easy-to-use interface and simplify the process of planning and executing missions for UAVs.

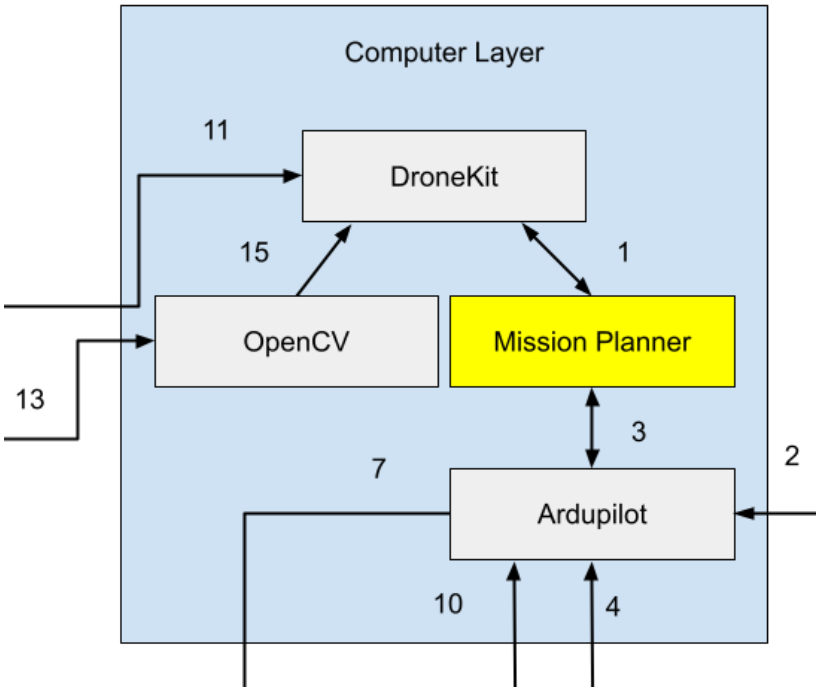

Figure 3: Mission Planner Subsystem diagram

3.5.1 Mission Planner Hardware

Mission Planner is a software application that runs on a Windows PC, and it does not require any specific hardware components. However, to use Mission Planner to control and communicate with unmanned aerial vehicles, it is necessary to have certain hardware components such as a flight controller and a telemetry radio.

The flight controller is a small, microcontroller-based device that is installed on the UAV and is responsible for controlling its flight and other operations. The flight controller must support MAVLink or ArduPilot communication protocols.

Telemetry radios are a wireless communication device that is used to transmit telemetry data and other information between the UAV and the ground station (the Windows PC running Mission Planner). Usually, telemetry radios will be based on the 3DR radio and the SiK firmware.

3.5.2 Mission Planner Operating System

Mission Planner is a Windows-based software application, which means it is designed to run on a Windows operating system. Specifically, Mission Planner requires a Windows PC running Windows 7 or later, with at least 2 GB of RAM and a compatible graphics card.

3.5.3 Mission Planner Software Dependencies

There are a number of software dependencies that Mission Planner relies on. Some of these include:

- .NET Framework
  - Mission Planner is built using the Microsoft .NET Framework. This means the .NET Framework must be installed on the Windows PC in order to run Mission Planner. The version of .NET Framework required by Mission Planner varies depending on the specific version of Mission Planner being used.
- Microsoft Visual C++ Redistributable
  - Mission Planner also relies on the Microsoft Visual C++ Redistributable Package in order to function properly. This package includes a set of libraries and runtime components that are required by many Windows applications, including Mission Planner.
- MAVLink
  - Mission Planner uses the MAVLink communication protocol to communicate with unmanned aerial vehicles and other autonomous vehicles. MAVLink is an open-source, lightweight protocol that is designed to be platform-independent.
- ArduPilot
  - Many of the UAVs that are controlled using Mission Planner rely on the ArduPilot autopilot software. Mission Planner includes support for ArduPilot and is designed to work seamlessly with ArduPilot-powered UAVs.

3.5.4 Mission Planner Programming Language

Mission Planner supports several programming languages that can be used to customize its functionality and create custom missions. Here are some of the programming languages that can be used with Mission Planner:

- C#
  - Mission Planner is mostly written in C#, so it is the primary language used for customization.
- Python
  - Python scripting is also supported in Mission Planner.
- JavaScript
  - With JavaScript, custom web pages can be created that integrate with Mission Planner.
- Lua
  - Mission Planner supports Lua scripting for advanced users who want to create custom functions and scripts.

3.5.5 Mission Planner Data Structures

Mission Planner uses various data structures and classes to organize and process data related to autonomous vehicle missions. Here are some examples of the classes and data structures used in Mission Planner:

- MAVLink
  - MAVLink messages are sent as packets that consist of a header, a payload, and a checksum. The header contains information about the message type, the system that sent the message, and the intended recipient. The payload contains the actual data being transmitted.
- Mission
  - A mission is a set of commands that the autonomous vehicle should execute in a specific order. A mission is represented as a list of waypoints, where each waypoint contains information about the location, altitude, and other parameters.
- Parameters
  - Parameters are variables that can be adjusted to fine-tune the behavior of the autonomous vehicle. The maximum altitude, speed, and acceleration can be set using parameters. The parameter values are stored in a data structure that can be sent to the vehicle as part of a MAVLink message.
- Telemetry
  - Telemetry data includes information about the vehicle's position, altitude, velocity, and sensor readings.
- Vehicle
  - A vehicle is an instance of an autonomous vehicle that is connected to Mission Planner. Mission Planner provides a data structure to store information about the vehicle, such as its type, ID, and current status. This data is used to communicate with the vehicle over MAVLink and to display information about the vehicle in the user interface on the ground station.

3.5.6 Mission Planner Data Processing

Here are some examples of algorithms and processing strategies used in Mission Planner:

- Path planning algorithms
- Sensor fusion algorithms
- PID control

3.6 ARDUPILOT SUBSYSTEM

Ardupilot is an open-source autopilot software suite designed to control unmanned aerial vehicles. It is not a piece of hardware, but rather a software package that can be installed on a variety of different hardware platforms, including the popular Pixhawk and Cube flight controllers. The primary purpose of Ardupilot in this project is to provide the development team with a comprehensive set of features and tools for controlling the unmanned aerial vehicle. This includes autonomous flight and mission planning.

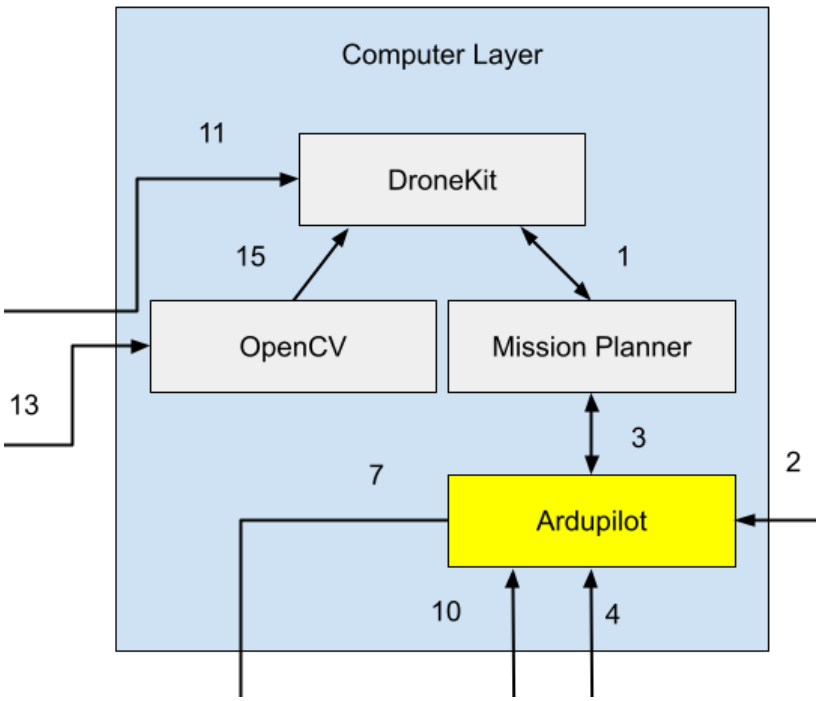

Figure 4: Ardupilot Subsystem diagram

3.6.1

sectionArdupilot Hardware The hardware components required to run Ardupilot will depend on the specific configuration and use case of the UAV. However, some common components include:

- Flight controller
  - This is the primary component that runs the Ardupilot software and communicates with various sensors and other peripherals to control the UAV's behavior. The flight controller typically includes a microcontroller, sensors, and other hardware components.
- GPS module
  - Ardupilot relies on GPS data to perform autonomous flight and other functions. A GPS module is typically connected to the flight controller to provide this data.
- Radio transceiver

  - A radio transceiver is used to communicate with the UAV during flight. The transceiver typically includes a transmitter and receiver and can be used to send and receive telemetry data, commands, and other information.

- Sensors
  - Various sensors may be used to provide data on the UAV's orientation, altitude, speed, and other parameters. Examples of sensors commonly used with Ardupilot include accelerometers, gyroscopes, barometers, and magnetometers.

- Power supply
  - An adequate power supply is essential for the safe and reliable operation of the UAV. This may include batteries, power distribution boards, and voltage regulators.

3.6.2 Ardupilot Operating System

Ardupilot is written in C++ and can be compiled and run on several different operating systems, including:

- Linux
  - Arudpilot is primarily designed to run on Linux-based operating systems, such as Ubuntu, Debian, and Raspbian. These operating systems provide a stable and reliable platform for running the Ardupilot software and are commonly used in embedded systems and other similar applications.

- Windows
  - Ardupilot can also be compiled and run on Windows-based operating systems, such as Windows 10. However, Windows is generally not recommended for running Ardupilot as it is less stable and reliable than Linux.

- macOS
  - Ardupilot can be compiled and run on Mac OS X, but support for this operating system is limited, and may not be as stable or reliable as running Ardupilot on Linux.

3.6.3 Ardupilot Software Dependencies

Ardupilot is a complex software suite that relies on a number of different software dependencies to operate. Some of the key software dependencies include:

- C++ libraries
  - Ardupilot itself is written in C++, and it relies on a number of libraries for core functionality. These libraries include Boost, Eigen, and others.

- GCS software
  - Ground Control Station (GCS) software is used to communicate with the Ardupilot software and to monitor and control the UAV during flight. Popular GCS software includes Mission Planner, QGroundControl, and MAVProxy.

- MAVLink
  - This is a lightweight communication protocol that is used to send data between the Ardupilot software and other components of the UAV, such as sensors and radios.
- ROS
  - The Robot Operating System (ROS) is a popular framework for developing robotic software. Ardupilot can be integrated with ROS to provide additional capabilities, such as advanced navigation and mapping.
- Python
  - Python is a popular programming language that is used extensively in the Ardupilot ecosystem. Many tools and scripts for working with Ardupilot are written in Python, and the language is also used for developing custom plugins and extensions.

3.6.4 Ardupilot Programming Language

Ardupilot is written in the C++ programming language and can be compiled for a variety of platforms, including Linux, Windows, and embedded systems. In addition to C++, Ardupilot uses several other programming languages for different purposes, including:

- Python
  - Ardupilot's scripting language is Python. It allows users to write custom scripts that can interact with the autopilot's APIs to control various functions of the vehicle.
- MAVLink
  - Ardupilot's communication protocol is MAVLink, which is a lightweight messaging protocol that allows for easy communication between different components of the system. MAVLink is implemented in C, but there are also MAVLink libraries available for Python, Java, C++, and other languages.
- Lua
  - Ardupilot includes a Lua scripting engine that allows users to write Lua scripts to automate certain tasks or implement custom behavior.
- JavaScript
  - Ardupilot also includes a web-based ground station called Mission Planner that is written in JavaScript and runs on a web browser. Mission Planner allows users to interact with their vehicle and perform various tasks, such as mission planning and telemetry monitoring.

3.6.5 Ardupilot Data Structures

Ardupilot has several classes and data structures that are worth discussing. Some of them are:

- MAVLink messages

  - MAVLink is a protocol used for communication between different components of the Ardupilot system. MAVLink messages are structured data packets that are used to transfer information between the vehicle and ground station or other components. These messages contain information such as sensor readings, vehicle status, and control commands. MAVLink messages are defined using XML and are automatically generated into C++ and other programming languages by MAVLink code generators.

- Parameters
  - Ardupilot has a parameter system that allows users to adjust various settings and parameters that affect the behavior of the vehicle. These parameters are stored in a structured data format and can be read or written to by the user of other components of the system. Parameters can be set via a ground station or over the MAVLink protocol.

- Senor data
  - Ardupilot supports a wide range of sensors, including GPS, accelerometers, gyroscopes, and magnetometers. Sensor data is typically stored in structured data formats, such as arrays or structs, and is read and processed by the autopilot software.

- Control structures
  - Ardupilot uses various control structures to regulate the behavior of the vehicle. For example, the autopilot might use a PID controller to regulate the pitch, roll, and yaw of the vehicle. These control structures are implemented as classes or data structures in the autopilot software.

- Waypoints
  - Ardupilot supports mission planning and waypoint navigation. Waypoints are structured data packets that specify a location and other information, such as altitude and speed. The autopilot uses these waypoints to navigate the vehicle to a particular location or along a specific path.

3.6.6 Ardupilot Data Processing

Ardupilot uses several algorithms and processing strategies to control the behavior of unmanned vehicles. Some of the notable algorithms and strategies used in Ardupilot are:

- Path planning algorithms
- Geofencing algorithms
- Command handling
- Sensor fusion
- PID control
- Kalman filter

3.7 OpenCV Subsystem

OpenCV (Open Source Computer Vision) is a free and open-source library of computer vision and machine learning algorithms. It is designed to be used by developers and researchers working on computer vision applications such as image and video processing, object detection and tracking, and machine learning. OpenCV will be a powerful tool to make performing complex tasks such as object detection, tracking, and recognition of ArUco markers.

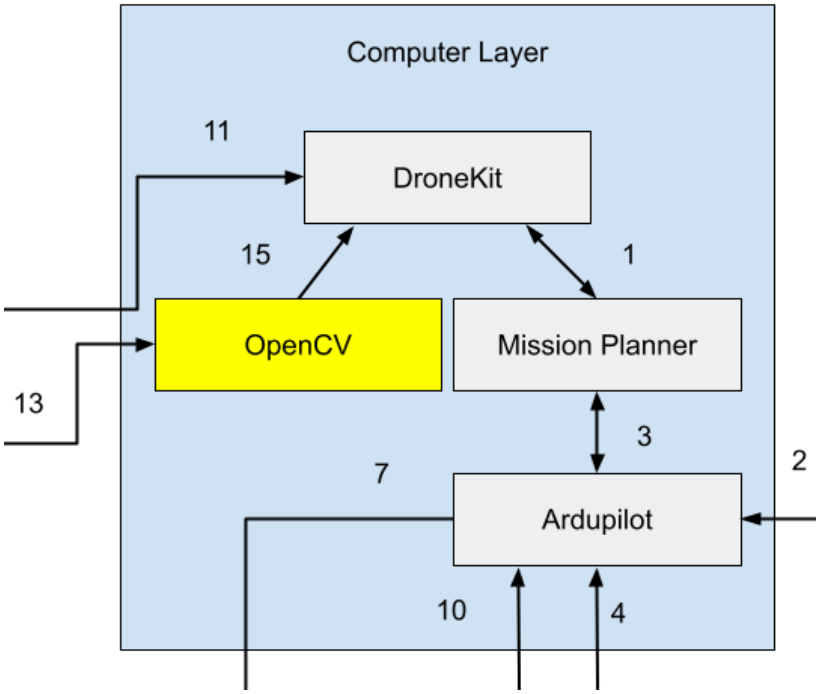

Figure 5: OpenCV Subsystem diagram

3.7.1 OpenCV Hardware

OpenCv is a software library and does not require any specific hardware components. However, OpenCV can be used in conjunction with a wide variety of hardware components, including cameras, microcontrollers, and specialized image-processing hardware.

3.7.2 OpenCV Operating System

OpenCV is a cross-platform software library that can be used on a wide range of operating systems. This includes Windows, Linux, macOS, and Android operating systems. The library is written in C++ and provides APIs for various programming languages including Python, Java, and MATLAB.

3.7.3 OpenCV Software Dependencies

OpenCV has several software dependencies that are required for proper installation and usage. Some of the important dependencies include:

- NumPy
  - NumPy is a Python library for numerical computing. OpenCV relies heavily on NumPy for image and array processing.

- Matplotlib
  - Matplotlib is a plotting library for Python. OpenCV can use Matplotlib to display images and other data.
- PySerial
  - Pyserial is a Python library for scientific computing. OpenCV can use PySerial to communicate with microcontrollers or other hardware devices.
- Pandas
  - Pandas is a Python library for data analysis. Pandas can be used to manipulate and analyze data from image processing tasks.
- SciPy
  - SciPy is a python library for scientific computing. SciPy can be used for image processing tasks such as image filtering and segmentation.
- PyQT or Tkinter
  - These are Python GUI frameworks that can be used to build user interfaces for OpenCV applications.

3.7.4 OpenCV Programming Language

OpenCV Python is a Python library and therefore primarily uses the Python programming language. The library provides a Python interface to the OpenCV C++ library, allowing developers to use OpenCV functionality within Python programs.

In addition to Python, OpenCV also supports several other programming languages including C++, Java, MATLAB, and Octave. These other language bindings may be useful in certain situations, but Python is the most commonly used language for OpenCV development due to its ease of use and versatility.

3.7.5 OpenCV Data Structures

OpenCV provides several classes and data structures that are commonly used in computer vision and image processing tasks. Some of the important ones include:

- KMeans
  - The KMeans class is used for clustering data points into groups. It is commonly used for color quantization and image segmentation.
- Mat
  - The Mat class represents an n-dimensional array of data. It is used extensively in OpenCV for representing images and other data.
- Point
  - The Point class represents a point in 2D space, with x and y coordinates. It is used for specifying pixel coordinates in image processing tasks.

- Rect
  - The Rect class represents a rectangular region of interest in an image. It is commonly used for cropping and resizing images.
- CascadeClassifier
  - The CascadeClassifier class is used for object detection in images and videos. It uses Haar cascades to identify objects such as faces, eyes, and cars.
- VideoCapture
  - The VideoCapture class is used for capturing video from a camera or a file. It provides methods for accessing frames from a video stream.
- FeatureDetector and DescriptorExtractor
  - These classes are used for feature detection and extraction in images. They are used in tasks such as image matching, object recognition, and tracking.

3.7.6 OpenCV Data Processing

Here are some notable algorithms and processing strategies in OpenCV:

- Image filtering
- Edge detection
- Feature detection
- Object detection
- Image segmentation
- Optical flow
- Deep learning

4 Flight Subsystems

In this section, the flight layer is a critical aspect of the drone. The flight layer controls and manages how the drone flies and reacts to different situations. Inside of the flight layer there are different types of hardware components that will help the drone take flight.

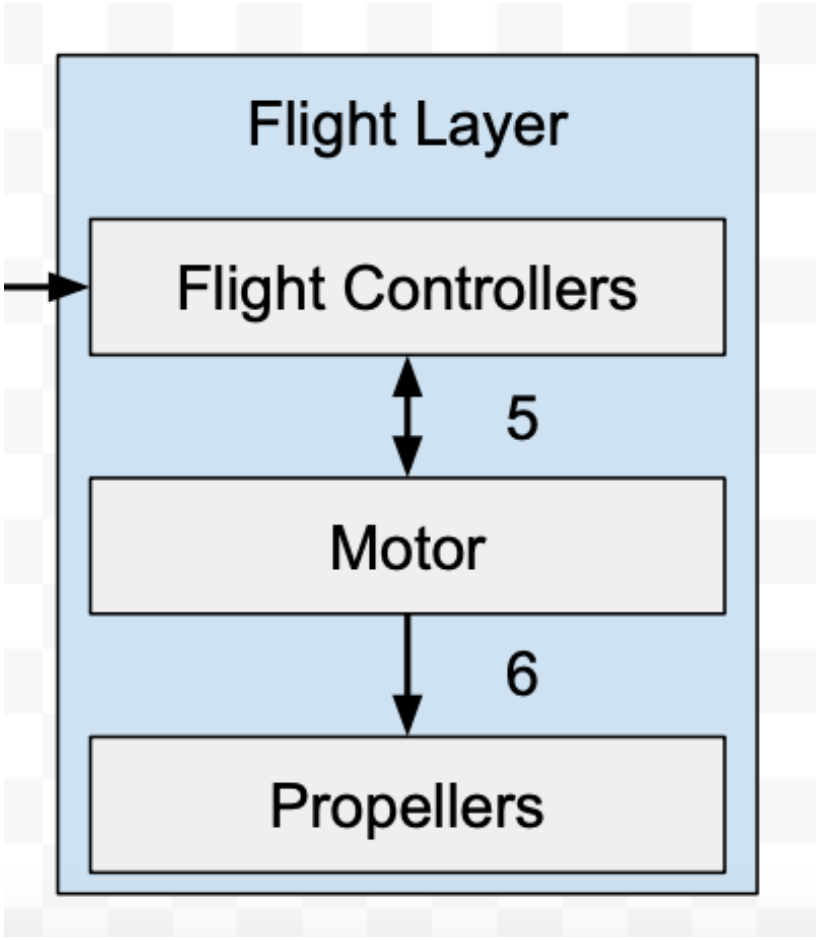

Figure 6: Flight Subsystem diagram

4.1 Flight Layer Hardware

The Drone flight layer is designed to contain four propellers, four motors, and a flight controller. With the flight controller the drone will be able to move horizontal and vertical on command.

4.2 Flight Layer Operating System

The flight aspect will be using Raspberry Pi OS as out operating system

4.3 Flight Layer Software Dependencies

The flight aspect will be using Raspberry Pi OS as out operating system

4.4 FLIGHT CONTROLLER SUBSYSTEM

4.4.1 Flight Controller HardWare

- Micro controller

- Sensors
- Onboard Data Storage
- Radio Transmitter and Receiver
- Barometer
- Electronic Speed Controllers

4.4.2 Flight Controller Operating System

The Flight controller uses a Raspberry Pi OS.

4.4.3 Flight Controller Software Dependencies

The Flight controller subsystem will be relying on the Ardupilot to be controlled.

4.4.4 Flight Controller Programming Language

The Flight controller subsystem will be using a python script when flying autonomous.

4.4.5 Flight Controller Programming Language

The Flight controller subsystem will be using a python script when flying autonomous.

4.4.6 Flight Controller Data Structures

- Attitude
- Velocity
- Configuration
- Telemetry
- Control Commands

4.4.7 Flight Controller Data Structures

The Flight controller relies on the sensors to communicate with the user or the software to send and receive its data

4.4.8 Flight Controller Data Processing

The Flight controller relies on the sensors to communicate with the user or the software to send and receive its data.

4.5 Propellers subsystem

The propellers subsystem is an important part of the drone.The propellers are required to generate thrust to lift the drone off the ground, while also keeping it in the air.

4.5.1 Propellers HardWare

This layer does not have any hardware.

4.5.2 Propellers Operating System

This layer does not have an operating system.

4.5.3 Propellers Software Dependencies

This layer does not have any software dependencies.

4.5.4 Propellers Programming language

This layer does not have any programming language.

4.5.5 Propellers Data Structures

While this layer does not have any data structures, the software that controls the propellers uses data to make sure the propellers function correctly.

4.5.6 Propellers Data Processing

This layer does not have any Data Processing, propellers rely on the motion of air to generate thrust.

4.6 Motors subsystem

There are four motors.The purpose of the motor subsystem is to generate enough power so the propellers can lift, take off, hoover, and move in various types of directions.

4.6.1 Motors HardWare

- Electronic Speed Controller
- Stator and Rotor
- Coil and Winding

4.6.2 Motor Operating System

This layer does not have an operating system.

4.6.3 Motor Software Dependencies

This layer does not have any software dependencies.

4.6.4 Motor Programming language

This layer does not have any programming language.

4.6.5 Motor Data Structures

This layer does not have any data structures.

4.6.6 Motor Data Processing

This layer does not have data processing.

5 Weapon Subsystems

The Weapons layer is responsible for containing the A.C.E combat system. The weapon's main object is to locate its target and fire upon the target using its laser. Once the target is hit it would send a signal back to the computer layer.

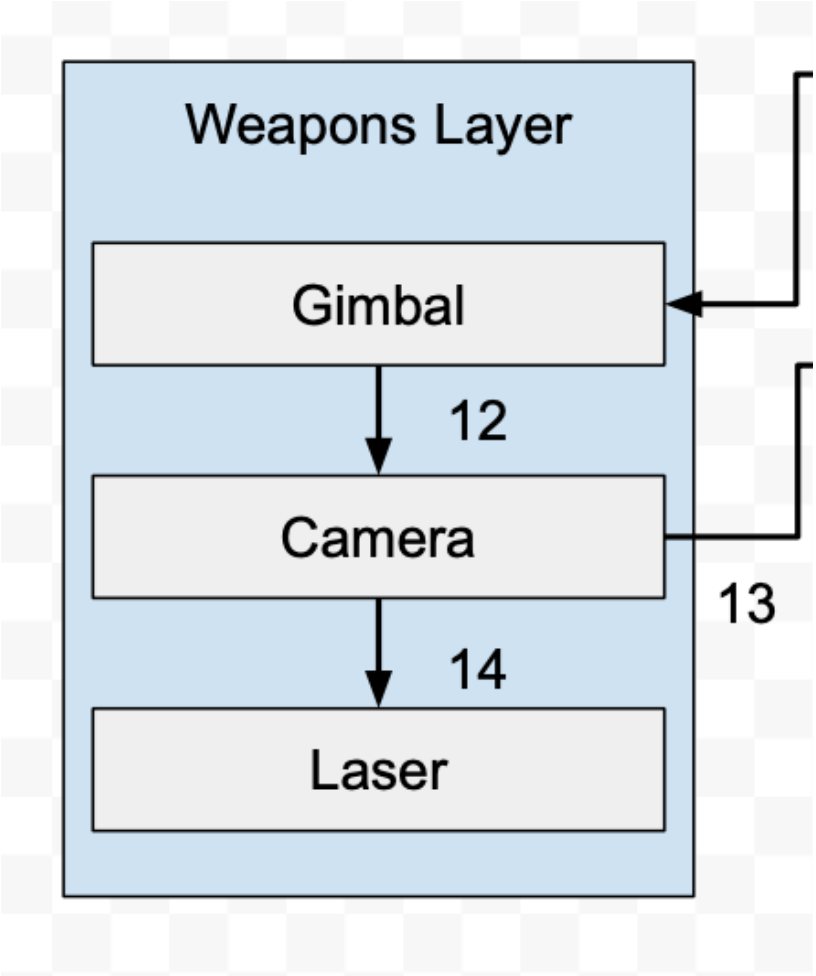

Figure 7: Weapons Subsystem diagram

5.1 Weapon Layer Hardware

- 1- IR Laser Gun board
- 1- IR Receiver Array board
- 2- 5 LED 5VDC light strips (Random, Package will include either Red, Blue, Green or White
- 1- 30x30mm ACE Combat receiver interface w/extensions (PWM Input)

5.2 Weapon Layer Operating System

The weapons layer will be using Raspberry Pi OS as out operating system

5.3 Weapon Layer Software Dependencies

- Input API
- Middleware

5.4 GIMBAL Subsystem

The gimbal subsystem stabilizes and rotates the camera along multiple axes. When mounted to the drone the gimbal would rotate, while in air to locate arUco markers.

5.4.1 Gimbal HardWare

- Mounting hardware
- Frame
- Frame
- Sensors

5.4.2 Gimbal Operating System

The Gimbal subsystem does not have an operating system.

5.4.3 Gimbal Software Dependencies

The Gimbal subsystem does not have any software Dependencies.

5.4.4 Gimbal Programming language

The Gimbal subsystem does not have any programming language.

5.4.5 Gimbal Data Structures

The Gimbal subsystem would not use data structures.

5.4.6 Gimbal Data Processing

The Gimbal subsystem does not have data processing

5.5 LASER Subsystem

The laser subsystem responsibility is to shoot a led light at a target once the camera recognizes it. Once the laser is shot, if the target is hit a buzzer sound will go off. A time stamp will be recorded every time the laser is shot.

5.5.1 Laser HardWare

- Enclosure
- Power Supply
- Optics
- Laser Diode

5.5.2 Laser Operating System

The Laser subsystem does not have an operating system.

5.5.3 Laser Software Dependencies

The software dependencies for the laser subsystem would be communication from the camera that an arUco marker has been detected.

5.5.4 Laser Programming language

The laser subsystem would have a python script to shoot once the camera spots its target.

5.5.5 Laser Data Structures

The laser subsystem would not use data structures.

5.5.6 Laser Data Processing

The laser subsystem would send out a timestamp every time the laser is shot. In addition to this when the laser hits its target it will be using a twitter api, to post a tweet to confirm the target has been hit.

5.6 CAMERA Subsystem

The Camera subsystem responsibility is to locate ArUco markers and sending the data to OpenVc to process vision recognition

5.6.1 Laser HardWare

- Lens
- Power Source
- Transmitter
- Camera sensor

5.6.2 Camera Operating System

The Camera subsystem will be using the raspberry pi Linux operating system.

5.6.3 Camera Software Dependencies

One of the Camera software dependencies would be the python library.

5.6.4 Camera Programming language

The camera subsystem would have a python script to capture images of arUco markers .

5.6.5 Camera Data Structures

The camera subsystem would not use data structures.

5.6.6 Camera Data Processing

The camera subsystem will send the data to openVc to proceed with vision recognition.This step will help determine if it is an enemy.

6 Power Subsystems

The power layer is responsible for providing and distributing power to every component on the UAV, including the Raspberry Pi, Cube Orange, GPS, ESCs, and motors. The power layer must deliver enough voltage and current to each component for the duration of the flight.

6.1 Power Layer Hardware

- Battery
- Power Distribution Board
- Power Modules
- Connectors/cables

6.2 Power Layer Operating System

This layer does not contain an Operating System

6.3 Power Layer Software Dependencies

This layer does not contain Software Dependencies

6.4 Battery Subsystem

The battery provides power to the integrated power distribution board (PDB) on the UAV frame. From the PDB, wires are soldered to attach the motors, ESCs, Pi-Connect Lite, which powers the Raspberry Pi, and the Power Brick Mini module that regulates power for Cube Orange. The battery used is a 3 cell, 2200mAh, lithium polymer battery that outputs 11.1V.

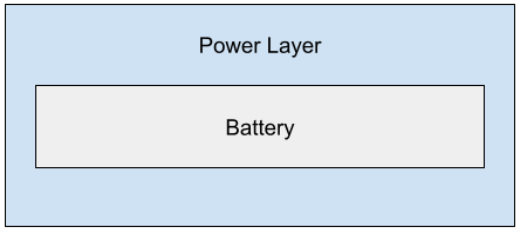

Figure 8: Power Subsystem diagram

7 Navigation Subsystems

The Navigation subsystem is responsible for controlling the drone's movements and ensuring that it can reach its destination safely and efficiently. The subsystem consists of several components, including sensors and hardware for tracking.

7.1 Navigation Layer Hardware

- 1-Here3 With iStand GPS

7.2 Navigation Layer Operating System

N/A

7.3 Navigation Layer Software Dependencies

N/a

7.4 Subsystem GPS

The GPS will be used to identify the location of the drone. It receives signals from Ardopilot and sends signals to the RTK unit. This communication will assist with stabilizing the coordinates and flying in autopilot mode.

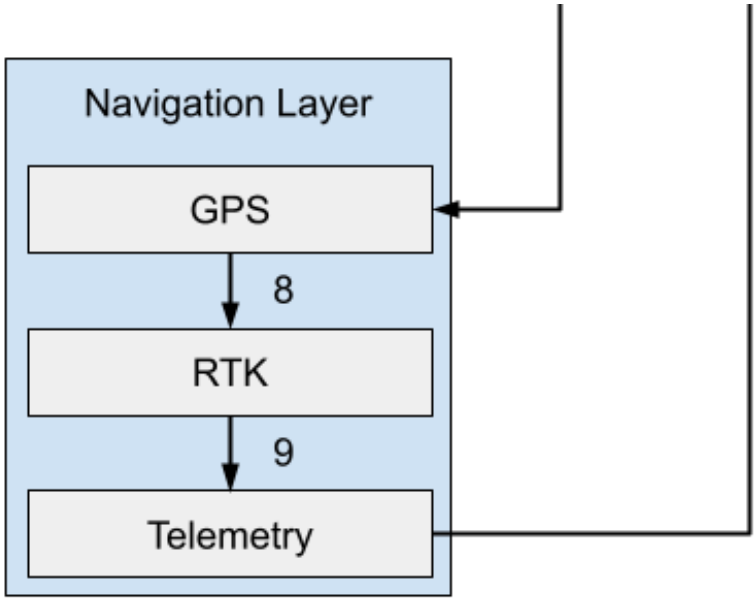

Figure 9: GPS Subsystem diagram

7.4.1 Subsystem Hardware

- 1-HX4-06206

7.4.2 Subsystem Operating System

N/A

7.4.3 Subsystem Software Dependencies

GPS library: The GPS library is a software component that provides an interface between the GPS receiver and the drone's operating system. The library is responsible for parsing the GPS data received from the receiver and providing the location and timing information to other components of the drone's software stack.

7.4.4 Subsystem Data Processing

- GPS signal reception: The GPS receiver on the drone receives signals from GPS satellites in orbit around the earth. These signals contain information about the satellite's location and time, as well as a unique identifier for each satellite.
- GPS signal processing: The GPS receiver processes the signals it receives to determine the drone's location and velocity. This involves calculating the time it takes for the signal to travel from the satellite to the drone, as well as correcting for any errors or interference in the signal.
- GPS data fusion: The GPS data is combined with other sensor data from the drone, such as accelerometer and gyroscope readings, to provide a more accurate estimate of the drone's position and velocity. This process is known as sensor fusion, and it helps to reduce the effects of noise and errors in the individual sensor readings.
- GPS waypoint navigation: Once the drone's position and velocity are known, the drone's autopilot system can use this information to navigate to specific waypoints or follow pre-defined flight paths. This involves calculating the heading and altitude adjustments needed to steer the drone towards the desired location.

7.5 Subsystem RTK

The RTK or Real-Time Kinematic is a corrective surveying technique used by many Global Navigation Satellite Systems (GNSS)/Global Positioning Systems (GPS) to significantly improve the accuracy of their receivers. Compared to conventional satellite navigation systems (which have a positional accuracy of about two to four meters). The addition of RTK to the system increases the accuracy by a hundred times.

7.5.1 Subsystem Hardware

N/A

7.5.2 Subsystem Operating System

N/A

7.5.3 Subsystem Software Dependencies

N/A

7.5.4 Subsystem Data Processing

The GPS receiver on the drone uses the measurements from the base station to calculate a more accurate position estimate. This involves comparing the phase and timing of the GPS signals received by the drone with the signals received by the base station. Once the drone's position is known with high accuracy, the drone's autopilot system can use this information to navigate to specific waypoints or follow pre-defined flight paths. This involves calculating the heading and altitude adjustments needed to steer the drone towards the desired location.

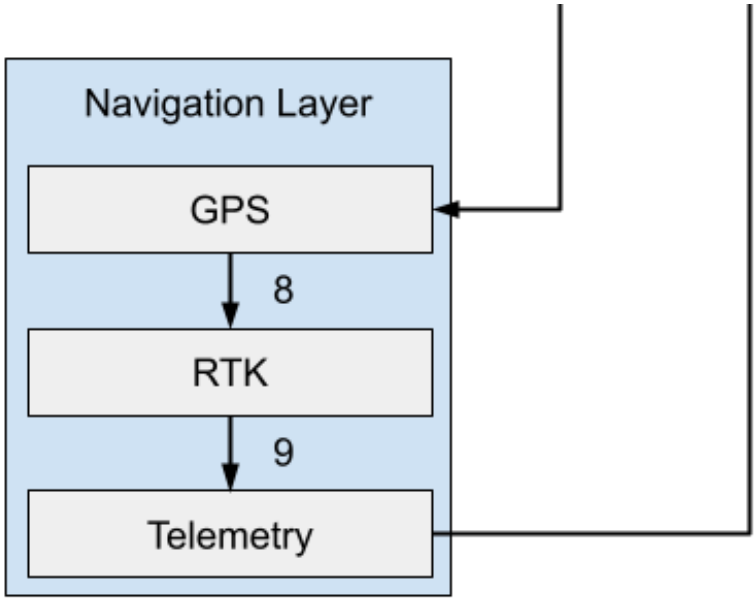

Figure 10: RTK Subsystem diagram

7.6 Subsystem Telemetry

Telemetry data provides the ability to track UAV status in real-time, allowing pilots to monitor position, and altitude to ensure smooth and efficient flight.

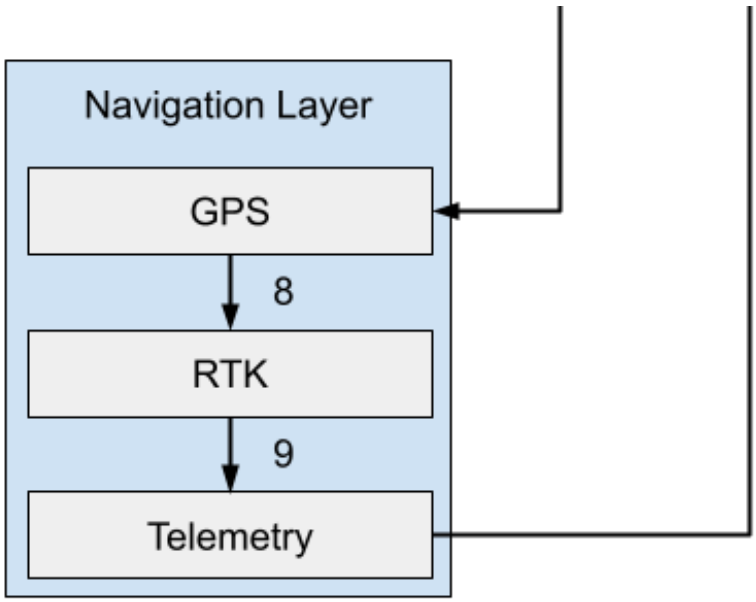

Figure 11: Telemetry Subsystem diagram

7.6.1 Subsystem Hardware

N/A

7.6.2 Subsystem Operating System

N/A

7.6.3 Subsystem Software Dependencies

N/A

7.6.4 Subsystem Data Processing

The data received by the ground control station is processed and analyzed in real-time to provide information about the drone's status and performance. This includes information on the drone's location, altitude, speed, heading, battery level, and sensor readings. Telemetry data processing can also be used to detect potential problems with the drone's systems, such as low battery levels, sensor malfunctions, or GPS signal loss. This allows the operator to take corrective actions, such as landing the drone or adjusting its flight path, before a serious issue occurs.

8 Appendix