System Design for Mechatronics Engineering

Team # 34, ParkingLotHawk
Fady Zekry Hanna
Winnie Trandinh
Muhammad Ali
Muhammad Khan

January 19, 2023

1 Revision History

Date	Version	Notes
January 17, 2023	1.0	Initial Revision

2 Reference Material

No external references are referenced within the System Design document.

2.1 Abbreviations and Acronyms

symbol	description
CAD	Computer Aided Design
DMS	Degrees/Minutes/Seconds for GPS coordinates
EM	Electromagnetic
FR	Functional Requirements
GUI	Graphical User Interface
HMI	Human Machine Interface
NFR	Non Functional Requirements
SITL	Software in the Loop
VTX	Video Transmitter

Contents

List of Tables

List of Figures

3 Introduction

The System Design Document begins with defining the design goals of the ParkingLotHawk within the Purpose and Scope sections, An overview of the product and the interacting system is then provided to convey the high level system architecture of the design. The detailed design of the system components are then described within the User Interfaces, Design of Hardware, Design of Electrical Components, and Design of Communication Protocols. The System Design Document then concludes with a proposed Timeline for the development of the ParkingLotHawk.

Further specifications of the design are defined within the SRS and Hazard Analysis Document.

4 Purpose

The System Design Document is provided in conjecture with the Module Guide and Module Interface Specifications Documents to provide a detailed system and module design of the ParkingLotHawk. This allows for transparency and traceability in design decisions for future iterations of the product. The intended audience for these documents are the project manager, development team, and project team.

5 Scope

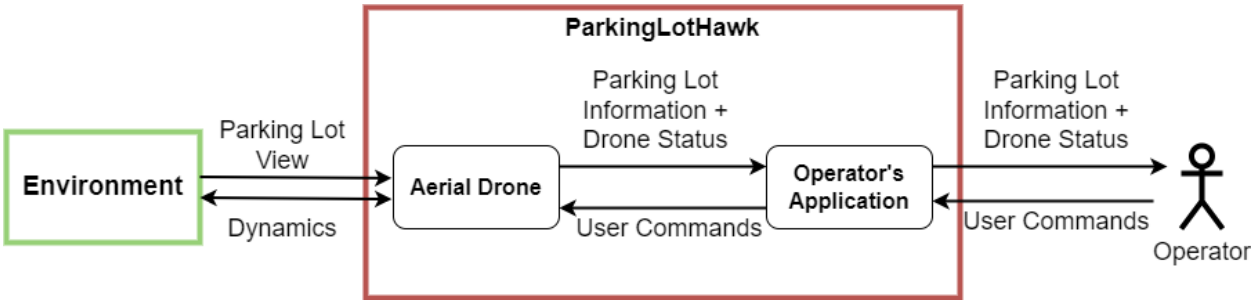
The product, ParkingLotHawk, is an Autonomous Aerial Drone used to gather live images and data within the confines of any given outdoor parking lot. The drone records the situation of each parking space periodically, and the information is sent to the user staying on the property in a timely manner. The user of such a product is the Parking Lot Operator (called Operator for short), who communicates with the Aerial Drone using an application running on their PC.

To limit the scope of the design, the following assumptions are made:

1. Operator does not fly the drone exceeding a specified amount of time, as specified by the SRS.
2. Birds will not interfere with the drone's operation.
3. Operator has access to a Windows computer with at least one functional USB port and Ethernet or Wifi capabilities.
4. Parking lot lines are visible to the naked eye.
5. Drone operates only under non-inclement weather, as defined in where Key terms, acronyms, and abbreviations are defined.

The system context diagram of the product can be found in System Context.

Figure 1: System Context



6 Project Overview

The product's operation can be split into two main states: Normal Behaviour and Undesired Event Handling, which are defined in more detail within State Implementation Requirements.

6.1 Normal Behaviour

The following normal states are listed as per the SRS. Within this behaviour, the product shall help operators understand the parking lots and requires fully functioning hardware, software components, and communication between them. During these flight states, a c_-currentView is also outputted along with an occupancy map to the operator.

Table 1: Normal Flight States

State	Transition Into	Behaviour	Purpose
Hover	After completion of Takeoff State.	The product shall fly and hover to height i_MaxHoverHeight. The drone shall keep the same lateral location it is currently at. The drone cannot transition to another flight state until it has reached a height of i_MaxHoverHeight.	This state is used for when the product is waiting for further operator commands. Hover height is selected to be i_MaxHoverHeight, so that the drone can see as much of the parking lot section as it can.
Autonomous Move	Operator asserts m_CompulsiveMove == true and requested location is within parking lot bounds.	If the Operator requests to move to a specific location within the parking lot, the product shall move to the m_DesiredUserLoc if it is within the parking lot, and hovers at that location with height i_DesiredHoverHeight. Note that this height does not need to be the same as the height specified in the Hover State.	This state is used for when the product needs to provide the operator the ability to move the drone to a specific location within the parking lot.
Compulsive Move	Operator sets m_CompulsiveMove == true and requested location is outside parking lot bounds.	If the Operator requests to move to a location, the product shall move to that location regardless of if it is within the parking lot. Once at that location, the product shall hover at a pre-defined operator-selected height. Note that this height does not need to be the same as the height specified in the Hover State.	This state is used for when the product needs to provide the operator the ability to move the drone to a specific location outside of the parking lot.

State	Transition Into	Behaviour	Purpose
Autonomous Explore	In Hover State and c_parkingLotDetected == true.	The product shall create its own path to explore and remain within the parking lot it currently detects.	This state is used for when the operator does not need to constantly instruct the drone to move.
Land	m_Land == true	The product shall first travel laterally to the initial launch location, and then lands vertically downward. Once physically landed, the drone enters the Idle state.	This state is used to explicitly designate a landing path and command.
Arm	In Idle state and m_Arm == true	The product shall arm the motors by spinning all motors at a slow speed.	This state is used to prime the motors and ensure that they are functional before liftoff.
Takeoff	In Arm state and m_Takeoff == true	During this state, the drone attempts to takeoff to i_MaxHoverHeight.	This state is used to explicitly designate a takeoff procedure.

There are also non-flight normal states, as listed in Table 2.

Table 2: Non-Flight States

State	Transition	Behaviour	Purpose
Idle	m_PowerOn == true and the previous state was Off.	The product is powered on and communicating with the operator application, but all motors are turned off.	This state is used to ensure that the operator can safely hold the drone and access the mechanical switch that controls m_PowerOn.
Configure	In Idle state and m_Configure == true.	During this state, the operator is able to define parameters of operation that are not configurable in other states. The Input Variables i_MinHoverHeight, i_MaxHoverHeight, and i_DesiredHoverHeight can be changed in this state. The product is powered on but motors are stationary.	This state is used to allow parameters that are unsafe to change during flight operation, to be safely changed through a special process. During this state the operator can safely hold the drone.
Off	m_PowerOn == false	The product shall be unpowered and stops communication with the operator application. All modules are powered off. No battery power is consumed. c_UserError is set to None, c_HealthStatus is set to Healthy, and c_Connected is set to false. All values in the matrices c_MotorSpeeds, c, c_OccupancyMap, and c_CurrentView are set to 0.	This state is used to explicitly state what it means for the drone to be off.

6.2 Undesired Event Handling

The following error states in Table 3 require no additional hardware features, and rely solely on existing functionality. Therefore, with the exception of the Malfunction state, all of the error states are guaranteed to perform correctly, even with the given error. These states are further defined within the SRS and Hazard Analysis documents.

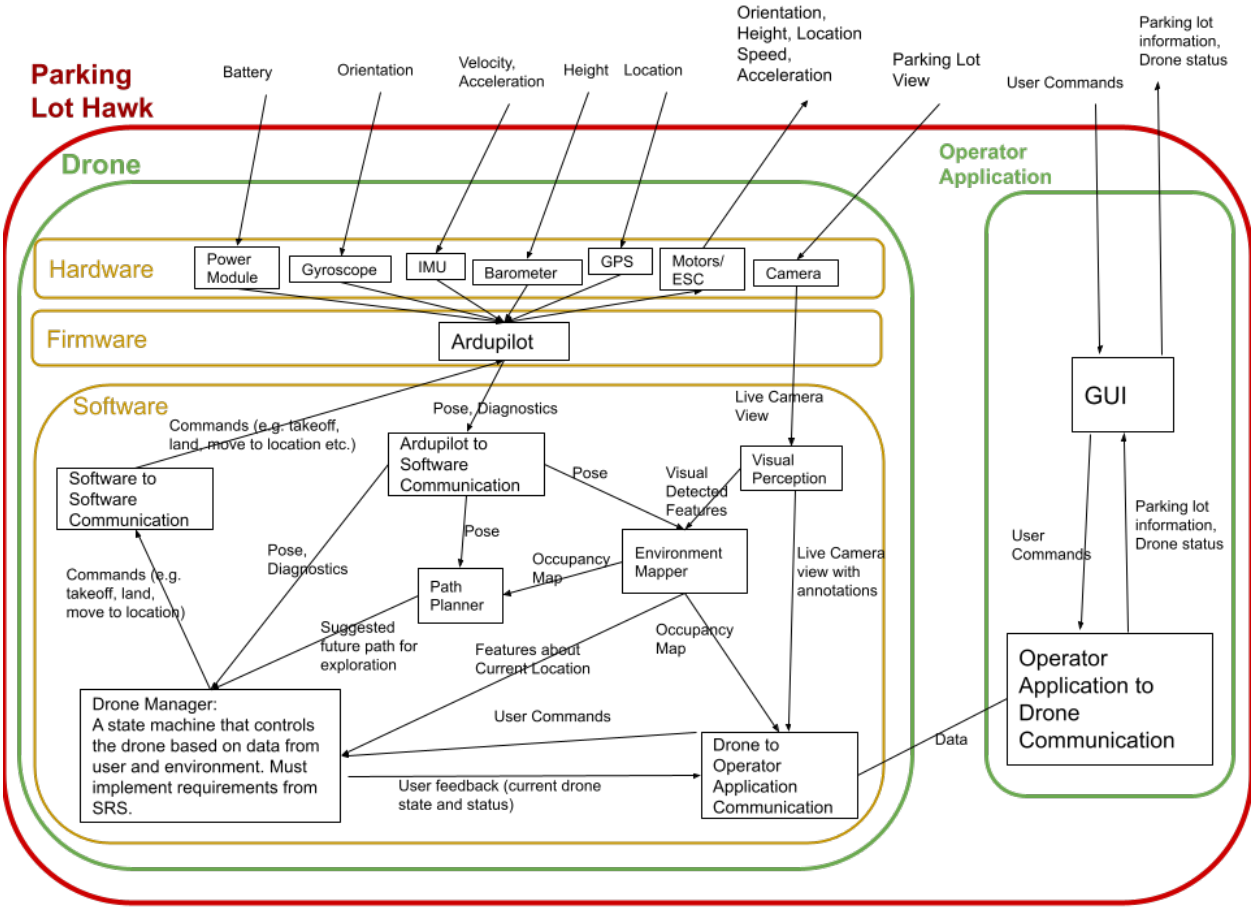
Table 3: Error States

State	Transition	Behaviour	Purpose
Desired Location Error	The operator specifies an invalid desired position.	Upon entry to this state, the c_UserError variable is set to Desired_Location_Out_Of_Bounds, and set to None upon exit. The drone proceeds to Hover at its current location. Upon exit of this state, the drone shall set c_UserError to None. Upon entry, the message "Desired Location out of parking lot bounds" shall be logged into c_Logs.	This state is used to indicate explicitly that the operator's request cannot be met.
No Parking Lot Detected Error	The product does not detect a valid parking lot.	Upon entry to this state, the c_UserError variable is set to No_Lot_Detected_State, and while upon exit c_UserError is set to None upon exit. The drone proceeds to Hover at its current location. Upon exit of this state, the drone shall set c_UserError to None. Upon entry, the message "No Parking Lot detected." shall be logged into c_Logs.	This state is used to indicate explicitly that the product cannot detect a parking lot.
Malfunction	An internal/external malfunction occurs in such a way that nominal performance is not possible.	Upon entry, the message "Major malfunction in drone detected, please inspect" shall be added to c_Logs. During this state the drone sets the c_HealthStatus to Unhealthy and sets c_HealthStatus to Healthy upon exit. It then tries to land the product at its launch location, which if is not possible the drone instead lands vertically on the land below. After landing, the drone enters the Off state.	This state is used to ensure that the product can handle large malfunctions that can occur during operation.
Communication Lost	The product loses connection or has a very weak connection to the Operator's Application.	During this state, the drone sets the c_HealthStatus to Unhealthy, and a message "Connection with drone lost." is sent to the Operator's Application. Upon exit, "Connection with drone established." is logged to c_Logs and c_HealthStatus to Healthy. It then tries to land at its launch location and turn off. After landing the drone enters the Off state.	This state is used to ensure that the product can handle communication loses that can occur during operation.

6.3 Component Diagram

A component diagram of the ParkingLotHawk showing the structural relations of the components can be found in Figure 2.

Figure 2: Component Diagram



6.4 Connection Between Requirements and Design

The traceability matrix between the requirements within the SRS and the design components are shown within the FR Traceability Table and NFR Traceability Table for the Functional and Non-Functional requirements respectively.

Table 4: FR Traceability Table

Functional Requirement	Relevant Components	Component Reference
GEN_001	Autonomous Explore State	Normal Flight States
GEN_002	Normal Flight States	Normal Flight States
GEN_003	Configure State	Non-Flight States
GEN_004	Configure State	Non-Flight States
GEN_005	Normal Flight States	Normal Flight States
GEN_006	Normal Flight States	Normal Flight States
STA_000	Idle State	Non-Flight States
STA_001	Hover State	Normal Flight States
STA_002	Autonomous Move State	Normal Flight States
STA_003	Autonomous Explore State	Normal Flight States
STA_004	Configure State	Non-Flight States
STA_005	Off State	Non-Flight States
STA_006	Land State	Normal Flight States
STA_007	Desired Location Error State	Error States
STA_008	No Parking Lot Detected Error State	Error States
STA_009	Malfunction State	Error States
STA_010	Communication Lost State	Error States
STA_011	Compulsive Move State	Normal Flight States
STA_012	Arm State	Normal Flight States
STA_013	Takeoff State	Normal Flight States
TRANS_001	Off State	Normal Flight States
TRANS_002	Off State, Idle State	Non-Flight States
TRANS_003	Idle State, Arm State	Non-Flight States, Normal Flight States
TRANS_004	Hover State, Autonomous Explore State	Normal Flight States
TRANS_005	Autonomous Move State	Normal Flight States
TRANS_006	Autonomous Move State, Desired Location Error State	Normal Flight States, Error States
TRANS_007	Autonomous Explore State	Normal Flight States
TRANS_008	Autonomous Explore State, No Parking Lot Detected Error State	Normal Flight States, Error States
TRANS_009	Land State	Normal Flight States
TRANS_010	Communication Lost State	Error States
TRANS_011	Communication Lost State, Hover State	Error States, Normal Flight States
TRANS_012	Compulsive Move State	Normal Flight States
TRANS_013	Arm State, Takeoff State	Normal Flight States
TRANS_014	Takeoff State, Hover State	Normal Flight States
TRANS_015	Configure State, Idle State	Non-Flight States

Table 5: NFR Traceability Table

Non-Functional Requirement	Relevant Components	Component Reference
PERF_001	Path Planner Module	Path Plan App
PERF_002	Land State	Normal Flight States
PERF_003	Autonomous Move State	Normal Flight States
PERF_004	MessageSocket Module	MIS of Message Socket
PERF_005	Hover State	Normal Flight States
PERF_006	Normal Flight States	Normal Flight States
PERF_007	All Flight States	Normal Flight States, Non-Flight States, Error States
PERF_008	Autunomous Move State	Normal Flight States
DES_001	Mechanical Hardware and Electrical Components	Design of Hardware, Design of Electrical Components
STD_001	Mechanical Hardware and Electrical Components	Design of Hardware, Design of Electrical Components
STD_002	Radio Receiver	Design of Electrical Components
SEC_001	UserInterface Module	MIS of User Interface
SEC_002	UserInterface Module	MIS of User Interface
MTNC_001	LiPo Battery	Design of Electrical Components
MTNC_002	Mechanical Hardware and Electrical Components	Design of Hardware, Design of Electrical Components
MTNC_003	Mechanical Hardware	Design of Hardware
SAFE_001	Flight States	Normal Flight States
SAFE_002	Configure State	Non-Flight States
SAFE_003	Flight States and Error States	Normal Flight States, Non-Flight States, Error States
SAFE_004	Flight States and Error States	Normal Flight States, Non-Flight States, Error States
SAFE_005	Mechanical Hardware	Design of Hardware

Non-Functional Requirement	Relevant Components	Component Reference
USE_001	UserInterface Module	MIS of User Interface
USE_002	UserInterface Module	MIS of User Interface
USE_003	Mechanical Hardware and Electrical Components	Design of Hardware, Design of Electrical Components
USE_004	UserInterface Module	MIS of User Interface
USE_005	UserInterface Module	MIS of User Interface
SR_002	UserInterface Module	MIS of User Interface
SR_003	UserInterface Module	MIS of User Interface
SR_004	Electrical Components	Design of Electrical Components
SR_005	Electrical Components	Design of Electrical Components
SR_006	UserInterface Module	MIS of User Interface
SR_007	UserInterface Module	MIS of User Interface
SR_008	Electrical Components	Design of Electrical Components
SR_009	UserInterface Module	MIS of User Interface
SR_010	UserInterface Module	MIS of User Interface
SR_011	Flight States	Normal Flight States
SR_012	UserInterface Module	MIS of User Interface
SR_013	UserInterface Module	MIS of User Interface

7 System Variables

The system variables within ParkingLotHawk are split into three main categories: Monitored Variables, Controlled Variables, and Constants Variables, and are defined in their respective sections.

7.1 Monitored Variables

Monitored Variables are variables that are continuously monitored by the product.

Table 6: Monitored Variables

Variable Name	Type	Unit	Description
m_Acceleration	Vector	m/s^2	three-dimensional vector containing acceleration relative to frame of the drone.
m_Gyroscope	Vector	Rad	three-dimensional vector containing orientation relative to frame of the drone.
m_Gps	Tuple	{DMS, m}	current GPS coordinates of the drone with height in the second tuple.
m_Barometer	Float	atm	Altitude detection using atmospheric pressure measurement
m_DesiredUserLoc	GPS Location	DMS	Desired location of the aerial drone set by user.
m_Arm	Boolean	-	Indicates if the operator desires the drone to arm.
m_Configure	Boolean	-	Indicates if the operator desires the drone to configure height parameters.
m_Takeoff	Boolean	-	Indicates if the operator desires the drone to takeoff.
m_AutonomousExplore	Boolean	-	Indicates if the operator desires the drone to autonomously explore the parking lot.
m_AutonomousMove	Boolean	-	Indicates if the operator desires the drone to go to a specific GPS location through the Autonomous Move State.
m_CompulsiveMove	Boolean	-	Indicates if the operator desires the drone to go to a specific GPS location through the Compulsive Move State.
m_PowerOn	Boolean	-	Indicates if the operator desires the drone to be On or Off.
m_BatteryCapacity	Float	sec	Estimated flight time in battery remaining.
m_Land	Boolean	-	Indicates if the operator desires the drone to land.
m_SaveOutput	Boolean	-	Opens a dialog to save the current images and maps files in a folder.
m_CamVision	Image	Array of Pixels	Latest image of the section of the parking lot currently visible to the drone.
m_Connected	Boolean	-	Indicates if the drone is connected and communicating with the Operator's Application.

7.2 Controlled Variables

Controlled Variables are variables that are outputted by the system. Some are visible to the operator on their application, while others help to indirectly accomplish functional requirements.

Table 7: Controlled Variables

Variable Name	Type	Unit	Description
c_CurrentView	Image	Array of Pixels	Live visual display of parking lot section the drone's currently sees, as well as any further annotations and text.
c_OccupancyMap	Image	Array of Pixels	Map of available parking spots based on the drone's previous paths.
c_CurrentLoc	Tuple	{DMS, m}	GPS coordinates are stored at the first index, height is stored in the second index. Estimated longitudinal coordinate, lateral coordinate and height of the drone.
c_PastLoc	Vector	DMS	Trace of the drone's location in the past 60 seconds (vector of GPS locations).
c_MotorSpeeds	Vector	rad/s^2	n-dimensional vector containing the motor speeds of however many motors the drone chooses to use (2 for helicopter, 4 for quadcopter, 6 for hexcopter, etc.). The vector contains speeds of each motor clockwise from front of the drone.
c_Connected	Boolean	-	Indicates if connection between the drone and the operator's application is established
c_ParkingLotDetected	Boolean	-	Indicated if a parking lot is detected in the c_CurrentView.
c_UserError	Enumeration	0 - None, 1 - Desired_Location_Out_Of_Bounds, 2 - No_Lot_Detected	Indicates if a command the user requested is not feasible.
c_HealthStatus	Enumeration	0 - Healthy, 1 - Unhealthy	Indicates if the drone's mechanical and electrical state allows it to safely fly. For example if there is mechanical damage, the value should be Unhealthy.
c_Logs	List of Strings	-	Contains a list of past log messages.

7.3 Constants Variables

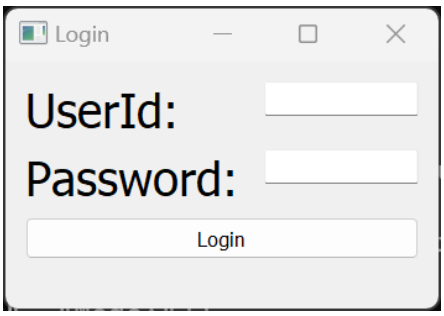
There are no constants required for the design. For constants in use for the Verification and Validation Plan, see Symbolic Constants.

8 User Interfaces

The user interface consists of two parts: Hardware Interface and Software Interface. The Hardware Interface will be limited in order to minimize damage to the product from improper handling, as well as decrease any technical experience required by the operator. The only Hardware Interface present is the disconnecting and reconnecting of the battery to the drone, and the mechanical On/Off switch.

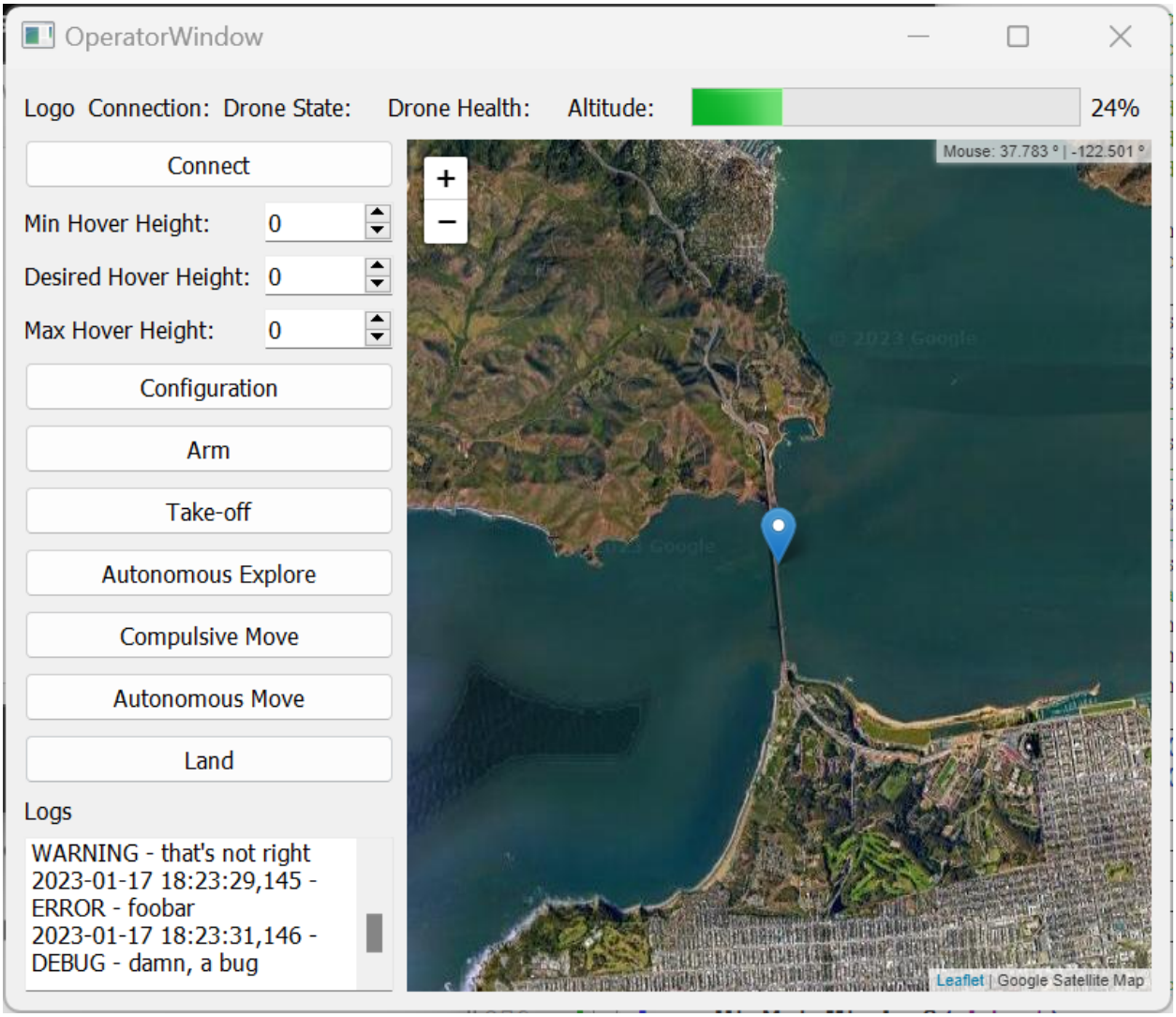
The Software User Interface for ParkingLotHawk is used to control the drone from the operator's console. It starts with a Login Window that requests for username and password. It displays an error message if the credentials do not match. Once the operator successfully logs in, the operator is presented with two windows as described below.

Figure 3: Login Window



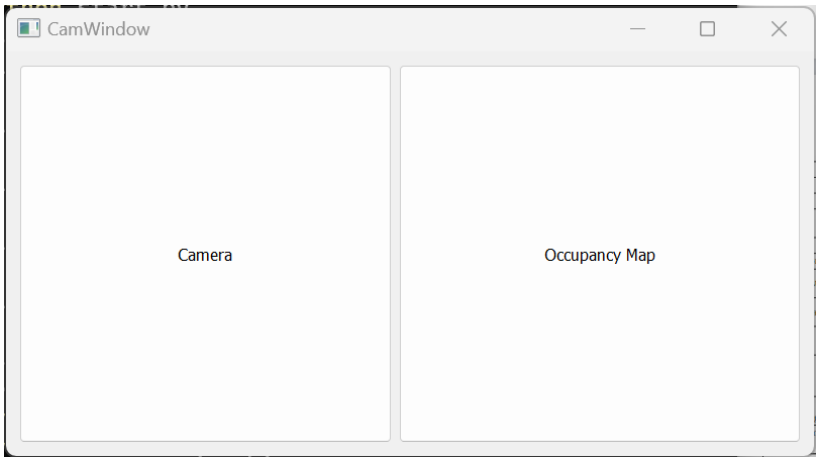
The first window is the User Interface where the operator can view the location of the drone on the map and various information related to the drone. The information on the drone is displayed at the top of the window. There is also a logs window which displays the program status or any changes that occur in the software at the bottom. The operator can control the configurations for the drone start-up from the left section of the window. The central map displayed in the window functions similarly to the maps within SITL programs that are used for drone simulation, which is to direct the drone's flight path from point A (drone position) to point B (drone destination) upon mouse click, and trace its movement as it moves.

Figure 4: User Interface



The second window, Camera and Occupancy Map, displays c_currentView and the occupancy map of the entire parking lot that is currently being explored by the drone. There exists a sliding divider in the center which displays the visual images from the camera on the left and the parking lot on the right, with the status of the parking spots constantly updated by the drone.

Figure 5: Camera and Occupancy Map



There are many objects that are used during the operation of the application, with the major ones being focused on in the User Interface Objects table.

Table 8: User Interface Objects

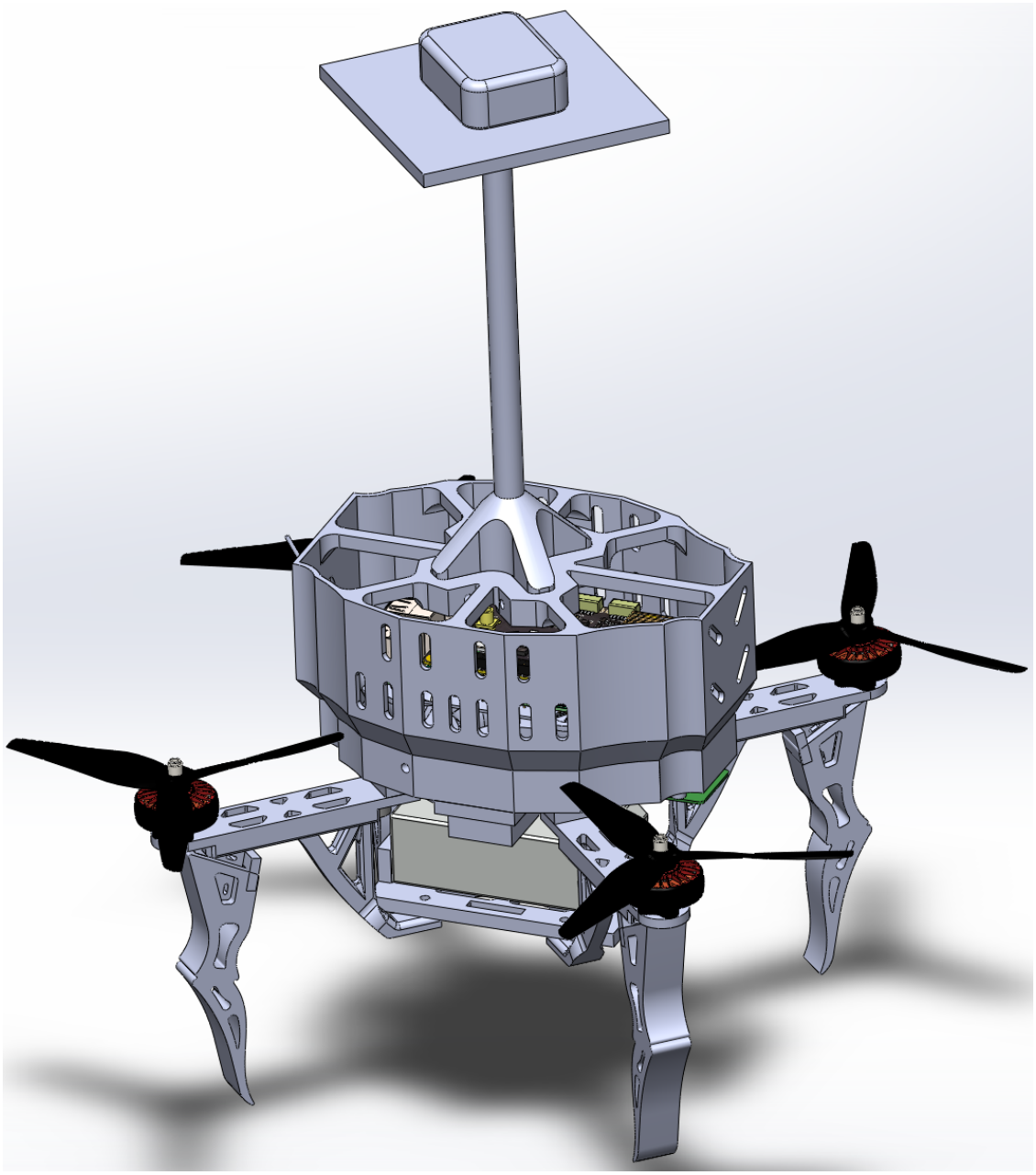
Name	Variable Name	Object Type	Description
Username	self.user_obj	Line Textbox	Allow user to input username.
Password	self.user_pwd	Line Textbox	Allow user to input password.
Login	button_login	Button	Enable user to login and access main application.
Logo	self.label_1	Label	Display Logo.
Connection	self.label_2	Label	Display Wifi connection Status.
Drone State	self.label_3	Label	Display the current state of the drone; whether it's still or in motion.
Drone Health	self.label_4	Label	Display any software issues or physical issues of the drone.
Altitude	self.label_5	Label	Display the Drone's altitude from the ground.
Battery	self.progressBar	Progress Bar	Display the drone's battery status through the progress bar.
Logs	self.textEdit	Textbox	Display the logs sent from the drone within a certain interval within the logs textbox.
Minimum Hover Height	self.spinBox_1	Spin Box	Adjust the minimum hover height for the drone using the spinbox.
Desired Hover Height	self.spinBox_2	Spin Box	Adjust the desired hover height for the drone using the spinbox.
Maximum Hover Height	self.spinBox_3	Spin Box	Adjust the maximum hover height for the drone using the spinbox.
Connect	self.pushButton_1	Button	Click to connect to the drone.
Configuration	self.pushButton_-2a	Button	Click to configure drone height.
Take-off	self.pushButton_-2b	Button	Click to Launch the drone.
Arm	self.pushButton_3	Button	Click to startup the drone.
Autonomous Explore	self.pushButton_4	Button	Click to automatically explore from the drone within the surroundings.
Compulsive Move	self.pushButton_5	Button	Click for allowing user to control the drone's movement.

Name	Variable Name	Object Type	Description
Autonomous Move	self.pushButton_6	Button	Click to have the drone move automatically.
Land	self.pushButton_7	Button	Click to land the drone.
Map	self.webView	Web Engine Widget	Displays an online map that should have the ability to interact with the movement path of the drone.
Camera	–	Input	Display the visuals from the camera mounted in the drone.
Occupancy Map	–	Input	Display the status of the parking spots within the parking lot through the drone.

9 Design of Hardware

The overall Computer Aided Design (CAD) of the product is shown within Figure 6.

Figure 6: CAD Overview



The overall component specifications are chosen in accordance to generally accepted specifications provided by Oscar Liang, and are as follows:

Table 9: General Design Specifications

Frame Size	210mm
Prop Size	5 inch
LiPo Battery	1000 - 1300mAh 3s/4s
Motor KV	2300KV - 2700KV
Motor Stator Size	2204 - 2206
Weight without Battery	250g - 450g

The components are then attached through M3 button head hex bolts and nuts to maintain a low bolt profile and weight. Particular consideration is also given to minimizing the vibrations and Electromagnetic (EM) interference that the sensors on the Navio2 are subjected to.

Each component is defined in further detail within Mechanical Hardware Components with the CAD shown within Mechanical Hardware.

Table 10: Mechanical Hardware Components

Component	Acquisition	Material	Fabrication Method	Design Explanation
Battery Compartment	Built	PLA	3D printed from CAD. 0.2mm layer height, 2 layer perimeter, 50% gyroid infill.	Separate compartment created below drone to isolate and protect battery against other components, and maintain a central center of balance. Battery also kept farthest away from Navio2 to minimize EM interference to sensors.
Frame Arm (x4)	Built	PLA	3D printed from CAD. 0.2mm layer height, 4 layer perimeter, 50% gyroid infill.	Removable arms to facilitate repairs upon damage or design changes.
Landing Leg (x4)	Built	PLA	3D printed from CAD. 0.2mm layer height, 4 layer perimeter, 50% gyroid infill.	Removable landing legs to facilitate repairs, and prevent damage to the drone's lower frame upon landing.
GPS Mast	Built	PLA	3D printed from CAD. 0.2mm layer height, 12 layer perimeter, 20% gyroid infill. Top part covered with foil tape to provide metallic plane.	To elevate the GPS 20cm above the electrical components and provide a metallic plane to boost GPS signals. Mast also mounted close to center of drone to maintain a central center of balance.
Enclosure	Built	PLA	3D printed from CAD. 0.3mm layer height, 4 layer perimeter, 20% gyroid infill.	Enclosure for electrical components and wiring. Prevents damage from the propellors, external forces, and the environment. Cutout slots available on sides to reduce weight and allow anchor points for zip ties of wiring.
Top Plate	Built	PLA	3D printed from CAD. 0.2mm layer height, 2 layer perimeter, 50% gyroid infill.	Main mounting plate for electronic components. Features slots and holes to attach electrical components. Attaches to a dampening plate that holds the Raspberry Pi and Navio2. X-Frame design chosen for simplicity in design and ease of control.
Dampening Plate	Built	PLA	3D printed from CAD. 0.2mm layer height, 2 layer perimeter, 50% gyroid infill.	Dampening plate to mount the Raspberry Pi and Navio2. Attaches to the Top Plate through four dampening balls, and minimizes vibrations to the Raspberry Pi and Navio2.
Dampening Balls (x4)	Bought	Silicone Rubber	–	Dampening balls to absorb vibrations from the frame.
Propeller (x4)	Bought	PC	–	5.1 inch diameter, 3.4 inch pitch, triblade design.
Aluminum Foil Tape	Bought	Aluminum	–	To provide a metallic ground plane for the GPS, and to provide EM protection on battery power wires.

10 Design of Electrical Components

The major electrical components are selected in accordance to Table 9, with the use of the Navio2 as the premade flight controller. The decision to use the Navio2 as opposed to creating a flight controller from scratch is further explored within Reflection. The Raspberry Pi 3B is then used to run the image processing and control logic of the drone. The overall schematic of the drone is shown in Figure 7, with the individual components defined in Table 11.

Figure 7: Electrical Schematic

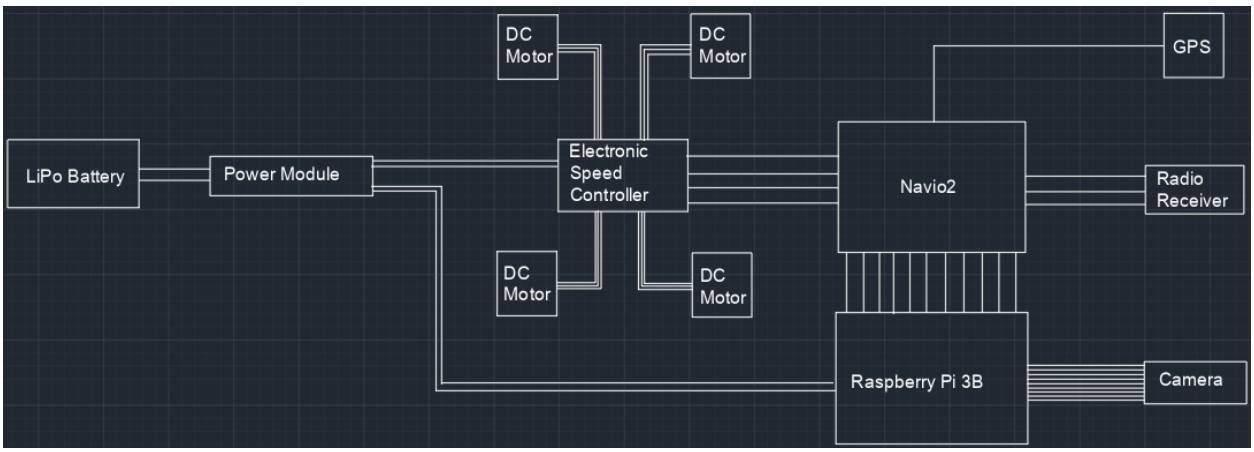


Table 11: Electrical Components

Component	Acquisition	Specifications	Design Explanation
Raspberry Pi 3B	Bought	1.2GHz Broadcom CPU, 1GB RAM, 40 Pin GPIO	Main controller for drone. Responsible for running software components.
Navio2	Bought	Redundant power supply with built in barometer, accelerometer, gyro, and compass.	Hat for Raspberry Pi, providing all the required sensors for operation. Also supported by Ardupilot software and runs interfaces with Debian, a real time Linux OS.
Radio Antenna and Receiver	Bought	2.4GHz Frequency	Redundant communication method and for development/testing. Will not be used by the operator.
Camera	Bought	5MP 1080p Camera	To acquire visual data. Infrared cameras not required due to operation of drone only during daytime periods or under well lit parking lot.
Electronic Speed Controller	Bought	4 in 1 45A ESC	To control the motors given PWM signals from the Navio2.
GPS	Bought	U-Blox NEO-M8N	To acquire localization of the drone.
Brushless DC Motor (x4)	Bought	iFlight Xing 2205 2300kV. For full specs, see Motor Specs.	To spin the propellors.
LiPo Battery	Bought	1500mAh 4S 120C LiPo Battery	To provide power to the drone.

11 Design of Communication Protocols

Communication is required between the different modules of the product, as well as to the Operator. These communication protocols are defined in Table 12.

Table 12: Communication Protocols

Component	Communication Between	Design Explanation
ROS Nodes	Interprocess communication on drone.	ROS allows the creation of a distribution of nodes, which communicate with one another through messages, called topics. The nodes can either subscribe or publish topics without any dependency on other nodes. Thus, each module can be developed as an independent node, allowing for encapsulation of each module.
MavROS	Ardupilot, the flight controller, and the Drone Software.	MavROS is a package within ROS that allows for reading and publishing MAVLink messages, which originate from Ardupilot. These messages contain the raw sensor data and other data provided by the flight controller. From the published topics from MavROS, other nodes within the Drone Software will then read the topics. Services are also offered by MavROS, allowing for sending commands to Ardupilot.
Local Area Network (LAN) Wifi	Drone Software and Operator's Application.	A LAN is created through a USB router connected to the Operator's PC. Both the Operator's PC and the drone shall connect to this LAN, allowing for communication between the devices though network sockets. Communication will be handled within the DroneSocket and OperatorSocket modules, and supports the usage of synchronous and asynchronous messages.
Graphical User Interface (GUI)	Operator's Application and Operator.	A GUI is used as the Human Machine Interface (HMI) due to the lower technical skill required, and to reduce training times required to operate the product.

12 Timeline

Individual timelines are provided to each major component of the product, and are defined within Mechanical Hardware Timeline, Electrical Components Timeline, Communication Protocols Timeline, and Modules Timeline. The modules mentioned within Modules Timeline are defined in further detail within the Module Interface Specifications document.

Table 13: Mechanical Hardware Timeline

Date	Components/Description	Test Method	Responsible By
October 10, 2022	Place order for all bought components. Includes dampening balls and propellers.	–	Ali, Fady, Winnie, Zaid
November 8, 2022	Design, fabricate, and test mounting plates and basic structure of the drone. Includes the Battery Compartment, Frame Arm, Top Plate, and Dampening Plate.	Test mounting of all electronic components.	Winnie
December 12, 2022	Design, fabricate, and test Landing Legs and mounts for external sensors such as the camera and GPS.	Test mounting of camera and GPS.	Winnie
December 14, 2022	All components.	Test flight.	Ali, Fady, Winnie, Zaid
January 9, 2023	Implement improvements to design from previous test flight. Includes the design and fabrication of the Enclosure, and revisions to other parts to improve structural integrity.	Reassemble drone with new components.	Winnie
January 23, 2023	All components.	Test flight.	Ali, Fady, Winnie, Zaid
January 30, 2023	Revise design if required from previous test flight.	Reassemble drone with new components.	Winnie

Table 14: Electrical Components Timeline

Date	Components/Description	Test Method	Responsible By
October 10, 2022	Place order for all bought components. Includes all electronic components.	–	Ali, Fady, Winnie, Zaid
October 24, 2022	Raspberry Pi and Navio2.	Verify sensor readings and communication with Raspberry Pi.	Ali, Fady
November 14, 2022	Connect battery, ESC, and motors to Raspberry Pi.	Test motor movement and perform ESC calibration.	Fady
December 14, 2022	All components.	Test flight.	Ali, Fady, Winnie, Zaid
January 9, 2023	Reassemble wiring to create more secure connections.	Verify motor movement and sensor readings.	Fady
January 23, 2023	All components.	Test flight.	Ali, Fady, Winnie, Zaid
January 30, 2023	Revise electrical design if required from previous test flight.	Verify motor movement and sensor readings.	Fady

Table 15: Communication Protocols Timeline

Date	Components/Description	Test Method	Responsible By
October 3, 2022	Establish LAN communication between PC and drone.	SSH into Raspberry Pi and verify bidirectional communication.	Ali, Winnie
October 7, 2022	Establish MavROS communication with the flight controller.	Verify sensor readings sent by the flight controller, and ability to send commands to the flight controller.	Ali, Fady, Winnie
October 15, 2022	Establish LAN communication between Windows PC and Software in the Loop (SITL) testing environment. The SITL testing environment is a series of applications that run on a Linux virtual machine.	Demonstrate that a sample message (like 'Hello World') between separate python applications running on the two environments.	Zaid
October 20, 2022	Write modules that present a usable interface for other modules to easily send and receive data. Create modules to simplify the usage of ROS services and topics offered by MavROS and for sending and receiving JSON messages between the python applications running on the drone and PC. Given that the SITL environment replicates the drone environment, communication modules designed for the drone will also work on the SITL environment.	Unit test the communication modules. For example, print the accelerometer values to the screen. Send and receive sample messages (like 'Hello World').	Zaid
October 25, 2022	Create modules to simplify the usage of ROS services and topics, i.e. complete DDC Topic Interface, DDC Service interface, and Algorithm Interface. Given that the SITL environment replicates the drone environment, communication modules designed for the drone will also work on the SITL.	Unit test the Topic Interface modules by printing sensor data to the screen. Unit test the Service modules in the SITL envirnment by calling the Arm service. The SITL environment should show the Copter armed.	Ali
December 1, 2022	Complete the Drone Camera and Operator Camera Services. In order to implement the modules one will have to learn how to read live camera images, annotate the images, and send live image from the drone's camera to the Windows PC.	Watch live camera output seen on the Windows PC. Also annotate some text, such as 'Hello World' in the bottom left hand corner to verify that the image annotation is visible on the Windows PC.	Ali

Table 16: Modules Timeline

Date	Modules/Description	Test Method	Responsible By
December 1, 2022	Complete the widgets for the interface modules, User Interface and Main User Interface. The widgets don't need to do anything, but the form should be complete.	The windows should appear visibly similar to the layout specified in the systems design.	Zaid
December 3, 2022	Complete the Main DDC Module, and the Main Algorithm Module. For the purposes of their development, create the Operation Manager and Algorithm Module, but leave the methods as empty.	In the SITL environment test that the two modules can be run as on the drone without crashing.	Ali
December 8, 2022	Complete the Operation Manager module, as all of the modules it needs to use should have been developed prior to December 1st. For the purposes of this component write the abstract Operation State interface, and implement the simplest state, the Idle state.	Run the Main DDC Module. For unit testing purposes print the current Operation state, it should be Idle and the process should not crash.	Ali

Date	Modules/Description	Test Method	Responsible By
December 20, 2022	Add operation state classes for the non-autonomous flight states: Configure, Arm, Takeoff, Hover, Compulsive Move, Land, and Communication Lost States. In order to implement the transitions of the states, the User Interface will need to be modified and integrated with the state machine, e.g. when the arm button is clicked, the User Interface needs to send the convey the button click command to the Message Socket, which conveys the message to the Operation Manager, which conveys the message to the Operation State.	In the SITL environment run the following test case: 1. Configure the height parameters to each be 10 meters. 2. Launch the drone from the user interface. 3. Do nothing and let the drone hover for 10 seconds. 4. From the user interface, make the drone move 50m forward. 5. Close the user interface (closes communication) and verify that drone attempts to land. 6. Open the user interface after closing, and verify that the drone hovers. 7. Use the user interface to land the drone.	Ali
December 28, 2022	Create the Vision App, Mapper App, and Path Plan App modules. Leave their algorithm related methods empty, and make them publish NULL values for now.	In the SITL environment, run the Main Algorithm Module. The program should not crash. Print the topics that they publish using ROS' command line interface, NULL values should be seen.	Ali

Date	Modules/Description	Test Method	Responsible By
December 31, 2022	Implement the autonomous operation states, that will use the topics published by the Vision App, Mapper App, and Path Plan App modules: Autonomous Move, Autonomous Explore, Desired Location Error, No Parking Lot Detected Error States. In order to implement the transitions of the states, the User Interface will need to be modified and integrated with the state machine similar to the previous component. The Operations States module is now complete.	In the SITL environment, run the Main Algorithm Module and the Main DDC Module as concurrent processes. For unit testing purposes print the current Operation State on the Main DDC Module. The processes should not crash. Using the user interface, make the drone enter the autonomous explore state. Then make the drone move 50 meters forward using the Autonomous Move state. Finally land the drone.	Ali
January 14, 2023	Research and implement the algorithms used by the Vision App, Mapper App, and Path Plan App modules.	Profile the accuracy of the Vision App algorithms on parking lot image. Also overlay the ground of the SITL environment with a 10 by 10 meter parking lot. Using the user interface and SITL, make the drone enter the autonomous explore state. It should explore the parking lot in about 1 minute. If this test works, the No Parking Lot Detected Error state automatically works as well. Also using the user interface, command to drone use the Autonomous Move state and move to a location outside the parking lot. Once the drone reaches near the edge of the parking lot, it should enter the Desired Location Error state, complaining the location is outside of the parking lot.	Fady, Winnie

Date	Modules/Description	Test Method	Responsible By
February 6, 2022	Integration testing of the non-autonomous operation states on the physical drone.	On the physical drone, run the following test case: 1. Configure the height parameters to each be 10 meters. 2. Launch the drone from the user interface. 3. Do nothing and let the drone hover for 10 seconds. 4. From the user interface, make the drone move 50m forward. 5. Close the user interface (closes communication) and verify that drone attempts to land. 6. Open the user interface after closing, and verify that the drone hovers. 7. Use the user interface to land the drone.	Ali, Fady, Zaid, Winnie
March 15, 2023	Integration testing the autonomous operation states on the physical drone.	Run the Main Algorithm Module and the Main DDC Module as concurrent processes on the drone. Using the user interface, make the drone enter the autonomous explore state. Wait 1 minute. It should explore some section of the parking lot. If this test works, the No Parking Lot Detected Error state automatically works as well. Also using the user interface, command to drone use the Autonomous Move state and move to a location outside the parking lot. Once the drone reaches near the edge of the parking lot, it should enter the Desired Location Error state, complaining the location is outside of the parking lot.	Ali, Zaid, Fady, Winnie

A Mechanical Hardware

Figure 8: Exploded CAD

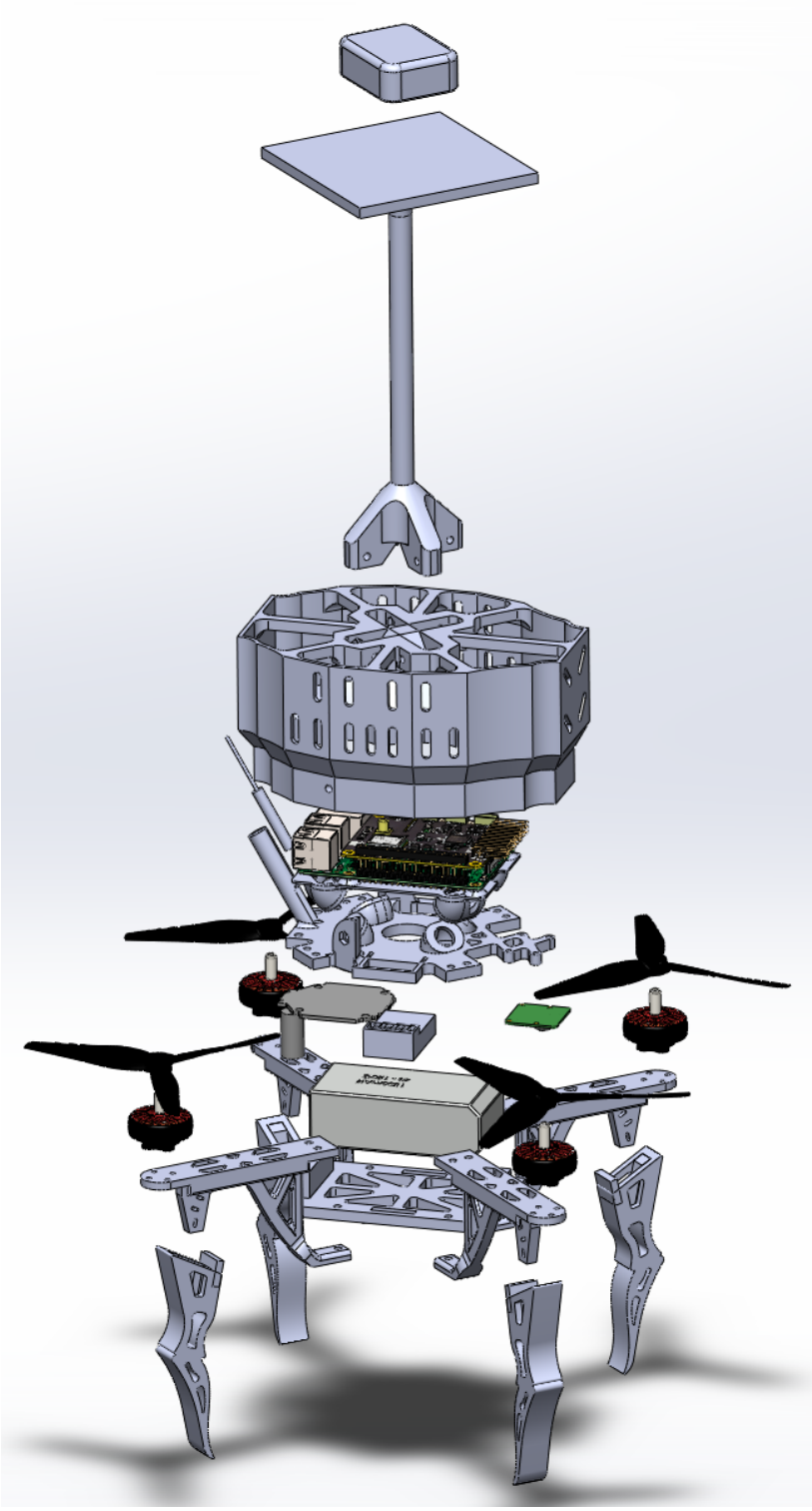


Figure 9: Battery Compartment

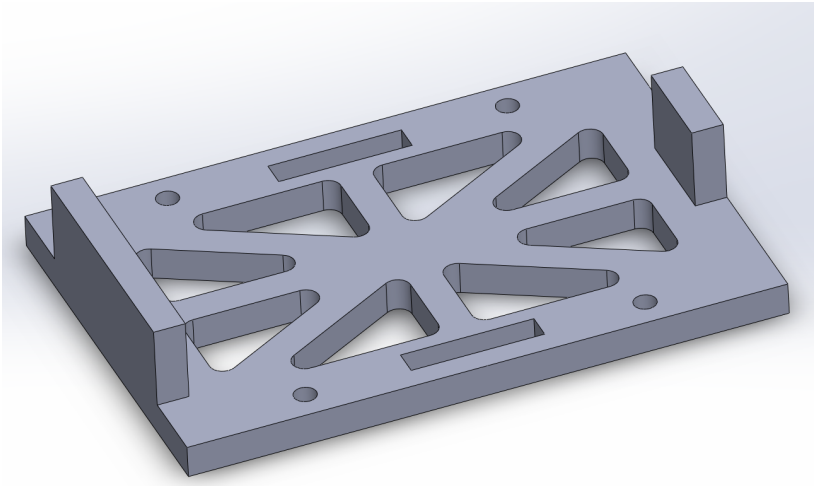


Figure 10: Frame Arm

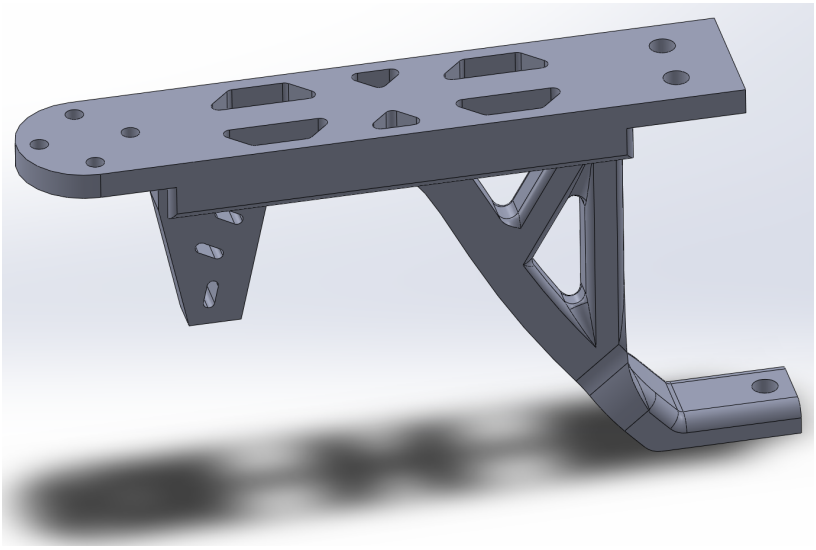


Figure 11: Landing Leg

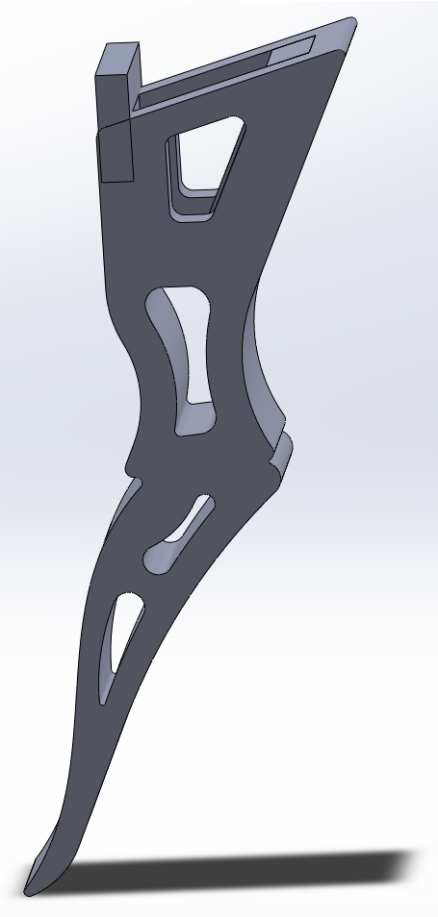


Figure 12: GPS Mast

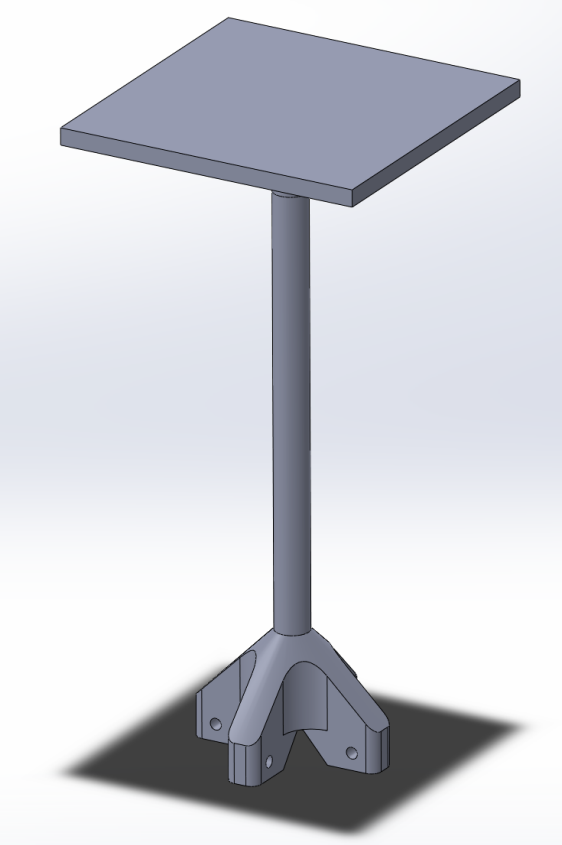


Figure 13: Enclosure

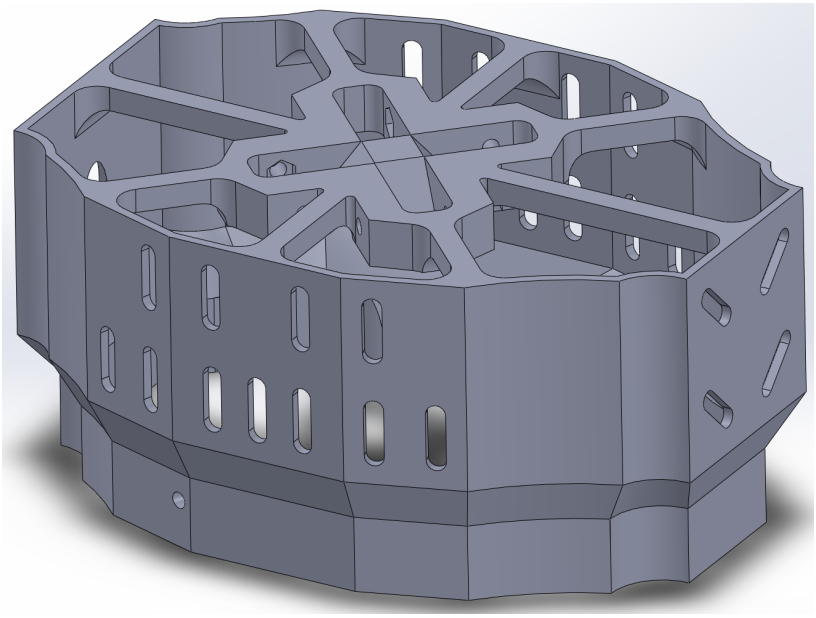


Figure 14: Top Plate

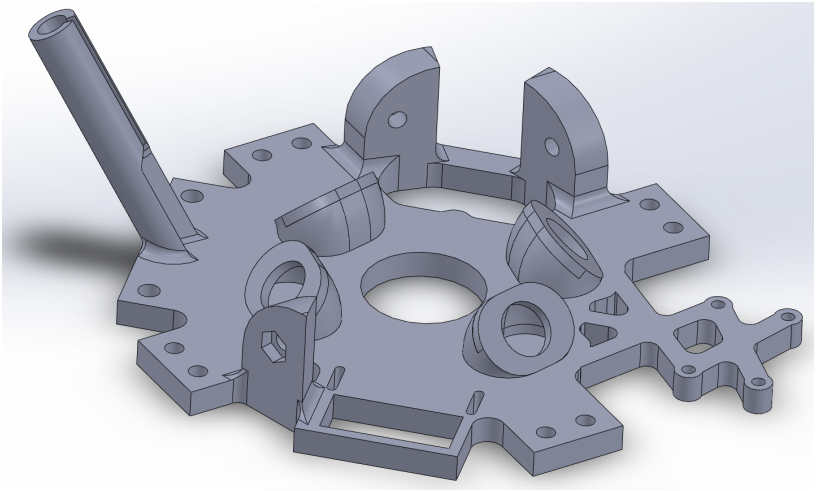


Figure 15: Dampening Plate

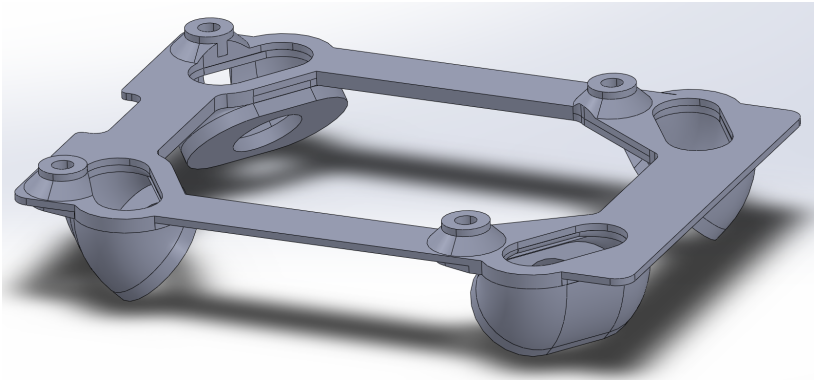
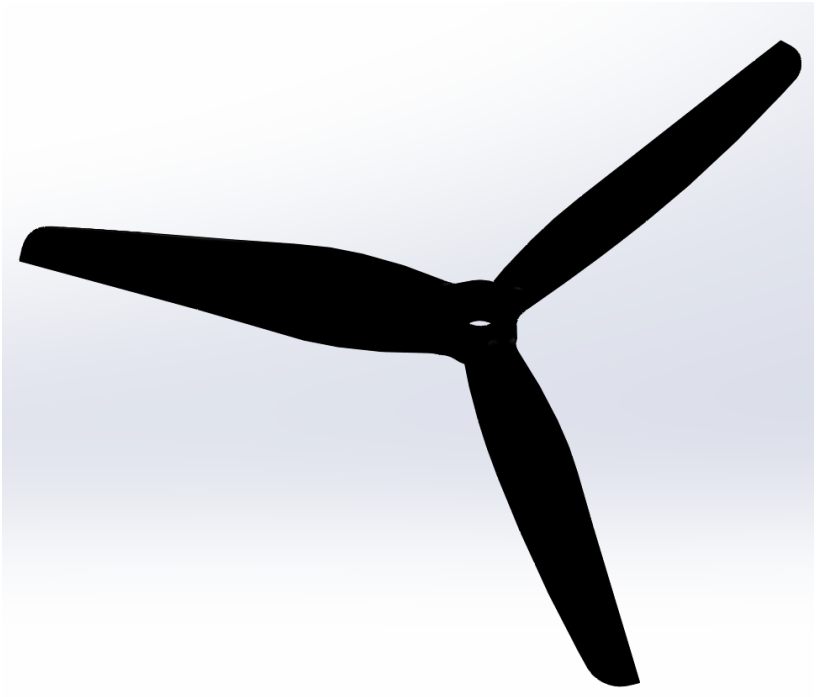


Figure 16: Propeller



B Electrical Components

Figure 17: Raspberry Pi 3B

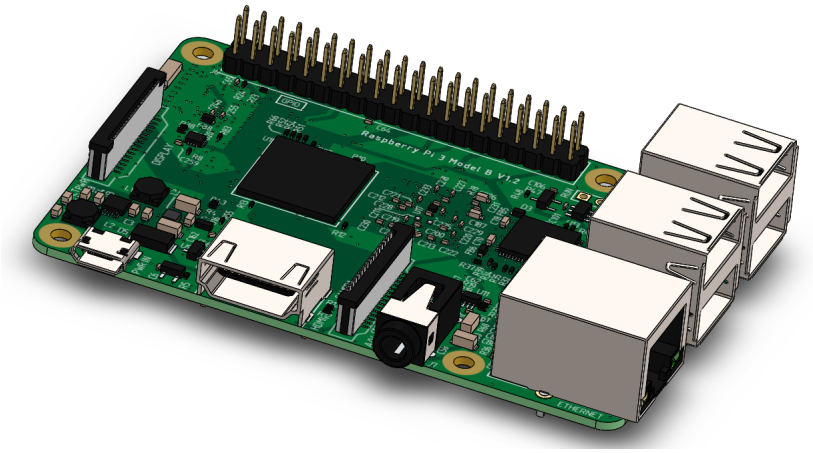


Figure 18: Navio2

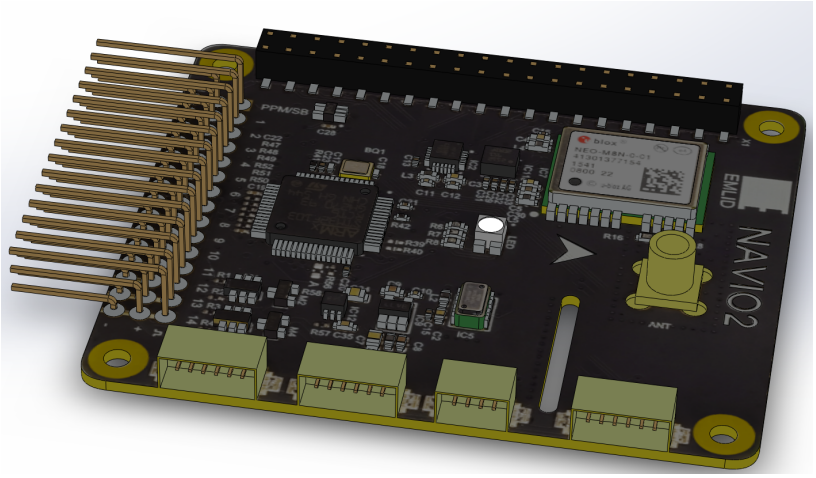


Figure 19: Radio Antenna and Receiver

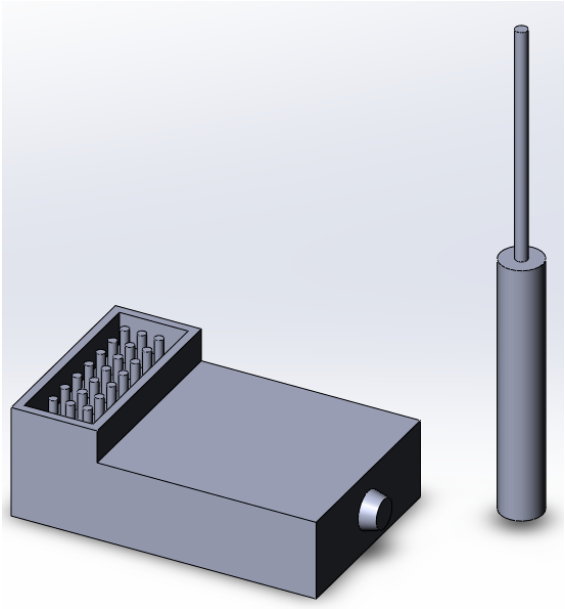


Figure 20: Camera

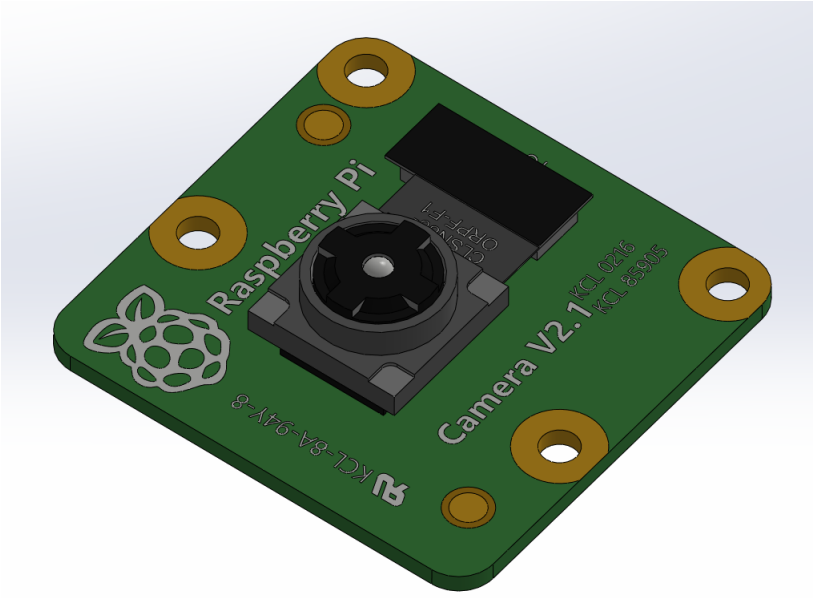


Figure 21: Electronic Speed Controller

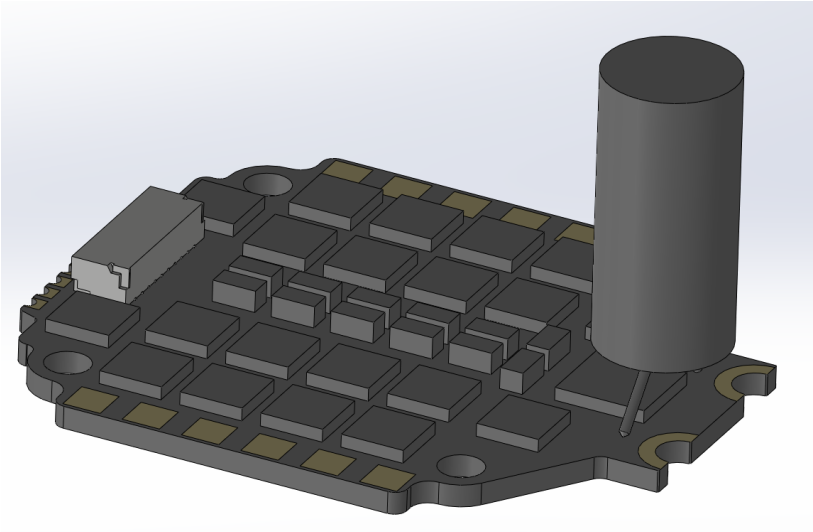


Figure 22: GPS

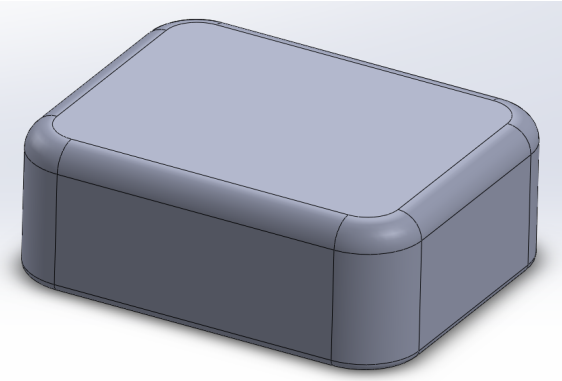


Figure 23: Brushless DC Motor

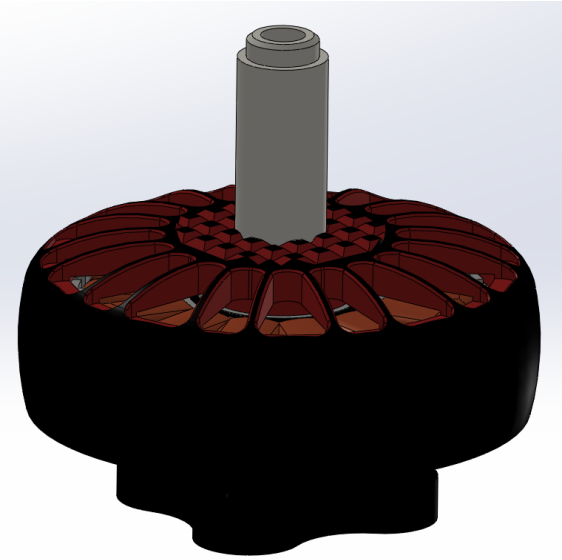


Figure 24: LiPo Battery

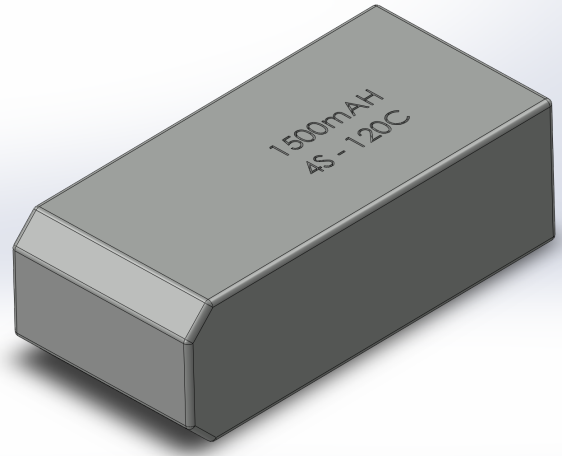
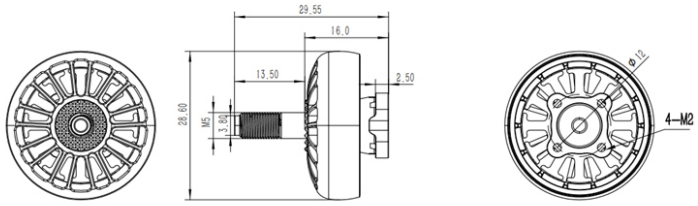


Figure 25: Motor Specs

iFight XING 2205 2300KV

Technical Datas	
KV	2300
Configu-ration	12N14P
Stator Diamter	22.0MM
Stator Length	5.0MM
Shaft Diameter	5mm
Motor Dimension(Dia.*Le	Φ28.6*29.55mm
Weight(g)	13 g
Idle current(10)@10V(A	≦1.35
No.of Cells(Lipo)	4~6S
Max Continuous Power(W)60S	451.33 W
Internal Resistance	72.41mΩ
Max Current(60S)	17.91A



Prop (inch)	Voltages (V)	Throttle (%)	Load Currency (A)	Pull(g)	Power(W)	Efficiency(g/W)	Temperature(in full throttle load 60S)
3042	25.2	50%	4.97	297.65	125.24	2.38	60℃
		60%	6.62	374.12	166.82	2.24	
		70%	8.65	476.66	217.98	2.19	
		80%	12.11	612.13	305.17	2.01	
		90%	15.43	765.41	388.84	1.97	
		100%	17.91	865.33	451.33	1.92	

C Reflection

Due to the limited resources available for the product, improvements to the product could be made if the resource constraint is removed. Such improvements are as follows:

1. Improved camera dampeners to minimize vibrations on the camera during flight.
2. Improved camera quality in terms of resolution and frames per second.
3. Larger drone size and more powerful motors to allow for easier assembly, easier design, increased stability under inclement weather, and having room for for a secondary GPS, secondary camera, and additional sensors.
4. Attach a wifi antenna onto the drone for increased connectivity range and switch camera communication to use a radio Video Transmitter (VTX) for increased bandwidth and speed.
5. Larger battery capacity for increased flight times.
6. More powerful edge compute device as compared to a Raspberry Pi 3B. Also requires switching the flight controller to a PixHawk instead of a Navio2, which is the industry standard for flight controllers and also provides support for additional edge compute devices.
7. Substitute PLA for TPU for better vibration dampening qualities of the frame.
8. Implement non-GPS localization for a redundant localization method, such as optical flow.
9. Implement downward facing range finders, such as LiDAR or infrared, to give more accurate height readings and improve landing capabilities.
10. Attach ultrasonic sensors in cardinal directions of the drone for object avoidance during flight.

Multiple design solutions were also considered during the development of the ParkingLotHawk. The major alternatives are described within Table 17.

Table 17: Design Alternatives

Design Alternative	Advantages	Disadvantages	Reason for Not Using Design Alternative
Create a land rover.	1. Longer battery life. 2. Lower maintenance. 3. Operable in more diverse environmental conditions.	1. Greater interaction with physical environment, therefore creates a driving hazard and potential to crash into other land objects. 2. Negatively impacts occupants of parking lot and could cause increased traffic.	The negative impact on the occupants of the parking lot is the main reason for not proceeding with this design. By increasing traffic or distracting the drivers, this goes against the purpose of using the product.
Create in-house flight controller hardware.	1. Cheaper to create. 2. More flexibility in design.	1. Dramatically increases complexity of product. 2. Is not compatible with established open source controller software.	Although the alternative would be cheaper, decreased complexity of the product is the main reason for buying a premade flight controller.
Use Operator's PC as main computation controller and send all commands to the drone, as opposed to having the computation controller on the drone.	1. Increased computation resources. 2. Allows for more resource intensive algorithms.	1. Increased latency in drone commands. 2. If signal is lost between the Operator's PC and the drone, the drone cannot continue operation.	The increased robustness by having the computation controller be onboard the drone is the main reason for foregoing this design alternative. Due to the risk of losing connection to the Operator's PC, an inoperable drone is deemed to be too high a risk.
Use 4 separate ESCs as opposed to a 4 in 1.	1. Lower cost. 2. Easier maintenance.	1. Increased space occupied. 2. Increased wire complexity.	The increased space and wiring is the main cause for using the 4 in 1 ESC, as the small frame chosen already has limited space, and this reduction is worth the extra costs.