

AERO-SENTINEL

System Design Document

Farhan Rouf (*rouff*), **Michael Lu** (*lum33*), **Noor Alam** (*alamn8*),
Safwan Hossain (*hossam18*), **Muhtasim Rahman** (*rahmam70*)

Issued Date	Changed by	Revision
<i>22nd December, 2023</i>	Muhtasim Rahman, Safwan Hossain, Noor Alam, Micheal Lu, Farhan Rouf	0
<i>21st March, 2024</i>	Muhtasim Rahman, Safwan Hossain, Noor Alam, Micheal Lu, Farhan Rouf	1

TABLE OF CONTENTS

LIST OF FIGURES.....	4
LIST OF TABLES.....	5
1. INTRODUCTION.....	6
1.1 Background.....	6
1.2 Project Description.....	6
1.3 Scope of Project.....	6
1.4 Purpose of Document.....	6
1.5 Defining Notion and Nomenclature.....	6
2. MODULE GUIDE.....	7
2.1 System Decomposition Tree.....	7
2.2 Module Hierarchy.....	11
2.3 Traceability Matrix.....	12
3. MODULES (Module Guide, MIS, and MID).....	13
3.1 CAMERA SYSTEM.....	13
3.1.1 Video Stream (ADT) Abstract Data Type (M1).....	13
3.1.2 Classification Module (M2).....	16
3.1.3 Locator Module (M3).....	18
3.1.4 Outliner Module (M4).....	19
3.1.5 Tracking Module (M5).....	21
3.2 SERVER (GUI).....	23
3.2.1 Socket Handler (ADT) Abstract Data Type (M6).....	23
3.2.2 Serial Port Handler (ADT) Abstract Data Type (M7).....	26
3.2.3 Communication Manager (Controller Class) (M8).....	28
3.2.4 Server Runner (Runner Class) (M9).....	30
3.2.5 User Client (ADT) Abstract Data Type (M10).....	31
3.2.6 Chart Handler (ADT) Abstract Data Type (M11).....	33
3.2.7 Drone State (ADT) Abstract Data Type (M12).....	35
3.2.8 Data Parser (Utility Class) (M13).....	37
3.2.9 User Controller(Controller Class) (M14).....	39
3.2.10 Client Runner(Runner Class) (M15).....	41
3.2.11 Camera Client(ADT) Abstract Data Type (M16).....	42
3.2.12 Communication Data (ADT) Abstract Data Type (M17).....	44
3.2.13 Event Tags Module (M18).....	47
3.2.14 Config Module (M19).....	49
3.2.15 Serial Port Module (M20).....	50
3.2.16 Server Module (M21).....	50
3.2.17 Socket IO Module (M22).....	50

3.2.18 Event Emitter Module (M23).....	51
3.3. DRONE (MIS & MID).....	51
3.3.1 Sensor Data Module (Utility Class) (M24).....	51
3.3.2 Sensor Fusion Module (Utility Class) (M25).....	53
3.3.3 Radio Control Module (Utility Class) (M26).....	55
3.3.4 Control Module (Utility Class) (M27).....	56
3.3.5 Motor Mixing Module (Utility Class) (M28).....	57
3.3.6 Transmitter Module (M29).....	59
3.3.7 Receiver Module M30.....	61
4. ENGINEERING DRAWINGS.....	64
4.1 CAMERA SYSTEM.....	64
4.1.1 Exploded View.....	64
4.1.2 Gimbal Base.....	64
4.1.3 Base 2.....	65
4.1.4 Sensor Holder.....	66
4.1.5 Anticipated Changes for Camera System.....	67
4.2 DRONE.....	68
5. DRONE FLIGHT CONTROLLER.....	68
6. ANTICIPATED CHANGES.....	70
7. DESIGN DECISIONS NOT LIKELY TO CHANGE.....	71
8. REFERENCES.....	72

LIST OF FIGURES

Figure 1: Cartesian coordinate variables

Figure 2: Level I Decomposition

Figure 3: Level II Decomposition of Drone Piloting

Figure 4: Level II Decomposition of Drone Monitoring

Figure 5: Level II Decomposition of Drone Tracking

Figure 6: Functional Decomposition Diagram (Overall View)

Figure 7: Camera System Exploded View

Figure 8: Camera System Gimbal Base

Figure 9: Camera System Base 2

Figure 10: Camera System Sensor Holder

Figure 11: Camera System Drone Dimensions

Figure 12: Drone Drawing

Figure 13: Drone Flight Controller High Level Circuit Diagram

LIST OF TABLES

Table 1: Module Hierarchy

Table 2: Traceability Matrix

Table 3: Video Stream Module Exported Access Programs

Table 4: Classification Module Exported Access Programs

Table 5: Locator Module Exported Access Programs

Table 6: Outliner Module Exported Access Programs

Table 7: Tracking Module Exported Access Programs

Table 8: Socket Handler Exported Access Programs

Table 9: Serial Port Handler Exported Access Programs

Table 10: Communications Manager Exported Access Programs

Table 11: User Client (ADT) Exported Access Programs

Table 12: Chart Handler Exported Access Programs

Table 13: Drone State Exported Access Programs

Table 14: Data Parser Exported Access Programs

Table 15: User Controller (Controller Class) Exported Access Programs

Table 16: Camera Client Exported Access Programs

Table 17: Communication Data (ADT)

Table 18: Sensor Data Module Exported Access Programs

Table 19: Sensor Fusion Module Exported Access Programs

Table 20: Radio Control Module Exported Access Programs

Table 21: Control Module Exported Access Programs

Table 22: Motor Mixing Module Exported Access Program

Table 23: Transmitter Module Exported Access Program

Table 24: Receiver Module Exported Access Program

Table 25: Flight Controller Parts

Table 26: Anticipated Changes Traceability Matrix

1. INTRODUCTION

1.1 Background

Drones have the potential to revolutionize industries such as transportation, healthcare, agriculture, public safety and beyond, however, their impact is currently undermined by their limited battery life. Our client VanWyn Inc. has proposed a radical new method to extend the operation time of drones by wirelessly transmitting power to drones from the ground, which would extend the battery life indefinitely.

1.2 Project Description

Our capstone team, The Aero-Sentinels, was entrusted with the ambitious goal of developing a UAV platform that comprises a ground station system that can communicate with the UAV as well as track the UAV in real-time, and along with the ground station a UAV system that can transmit mission-critical data.

1.3 Scope of Project

The scope of this project involves developing a fully stabilized UAV, a tracking system that can find and track the UAV, and an interface that establishes communication with the UAV and monitors essential UAV data to the user on the ground.

1.4 Purpose of Document

The purpose of this document is to provide a comprehensive overview of The Aero-Sentinels' capstone project, outlining the ambitious objective of developing a groundbreaking UAV platform in collaboration with our client, VanWyn Inc. This document aims to define system interactions, behaviors, and performance criteria, to ensure clarity in project requirements as well as alignment between course requirements, project requirements and stakeholder requirements. Module Guide, Module Interface Specification, Module Internal Design, Scheduling, System Decomposition Tree and more will be provided in this document to better explain the Component and System Design of our project. They will serve as a guide for the systematic design, implementation, and evaluation of the project, ensuring clarity and coherence in our approach to achieving the set goals.

1.5 Defining Notion and Nomenclature

- UAV - Unmanned Aerial Vehicle (the term UAV and Drone are used interchangeably in this document)

- **Cartesian Coordinate** variables will be used to describe the motion of the drone's position and orientation.

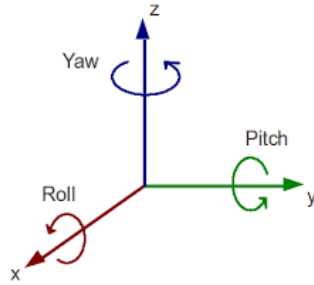


Figure 1: Cartesian coordinate variables

- **PPM - Pulse Position Modulation** a form of signaling wherein each transmitted symbol represents more than one bit. It is a standard signal encoding for radio control systems
- **MIS - Module Interface Specification** specifies the externally observable behavior of a module's access routines.
- **MID - Module Internal Design** specifies the internal structure of a module.

2. MODULE GUIDE

2.1 System Decomposition Tree

The functional decomposition will outline the high-level functions of the ***Drone Monitoring, Piloting and Tracking System***. This high-level system is then broken down into three sub-tasks: Drone Piloting, Drone Monitoring, and Drone Tracking.

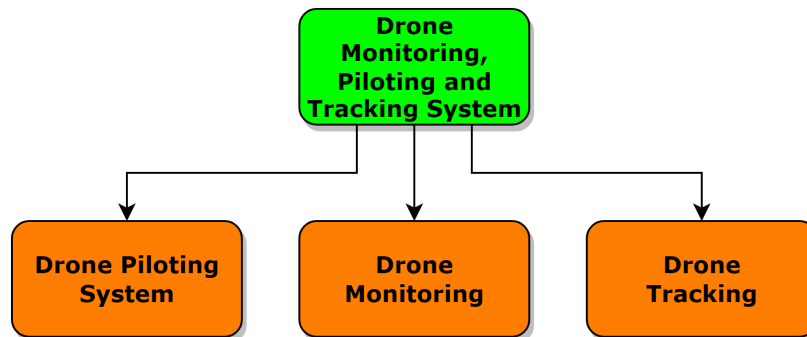


Figure 2: Level I Decomposition

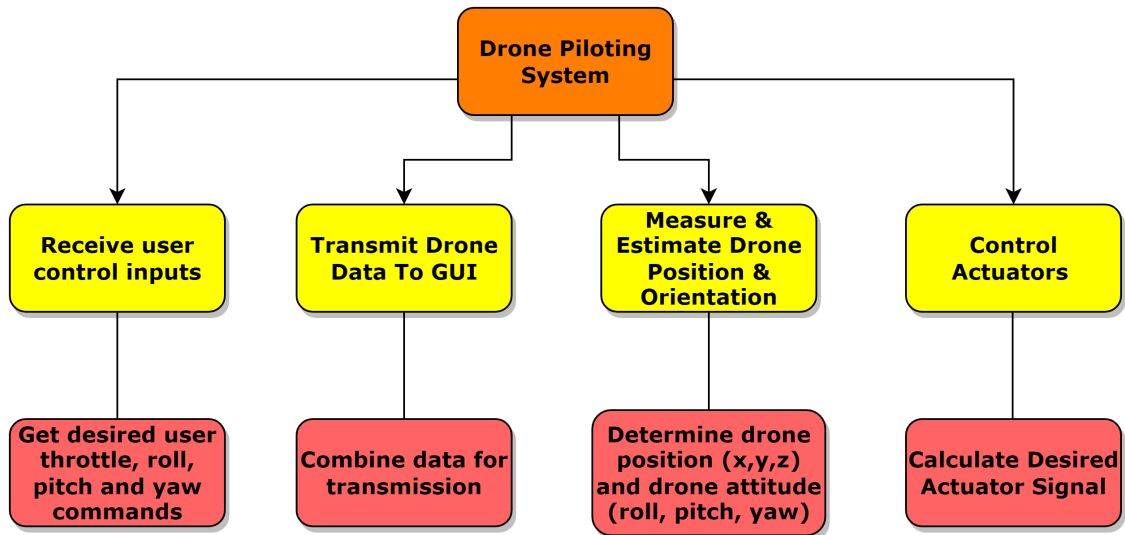


Figure 3: Level II Decomposition of Drone Piloting

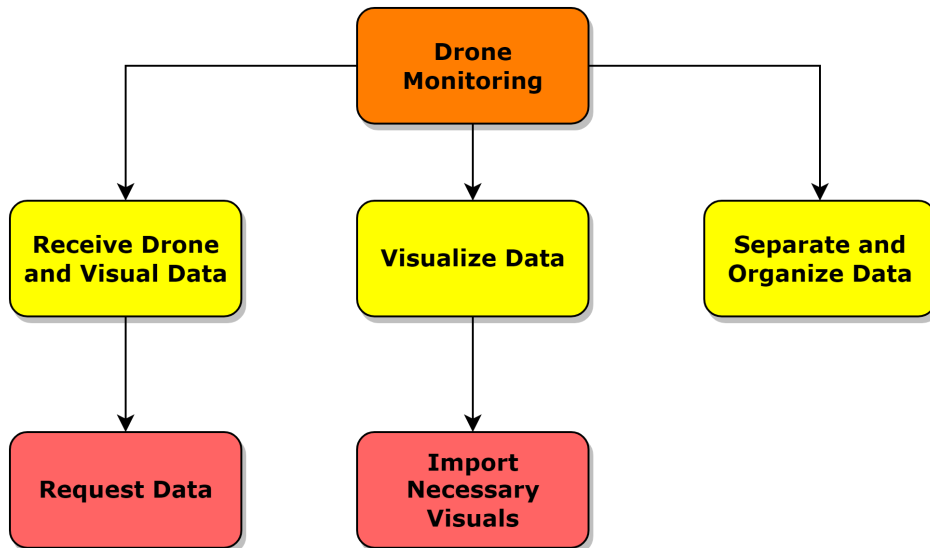


Figure 4: Level II Decomposition of Drone Monitoring

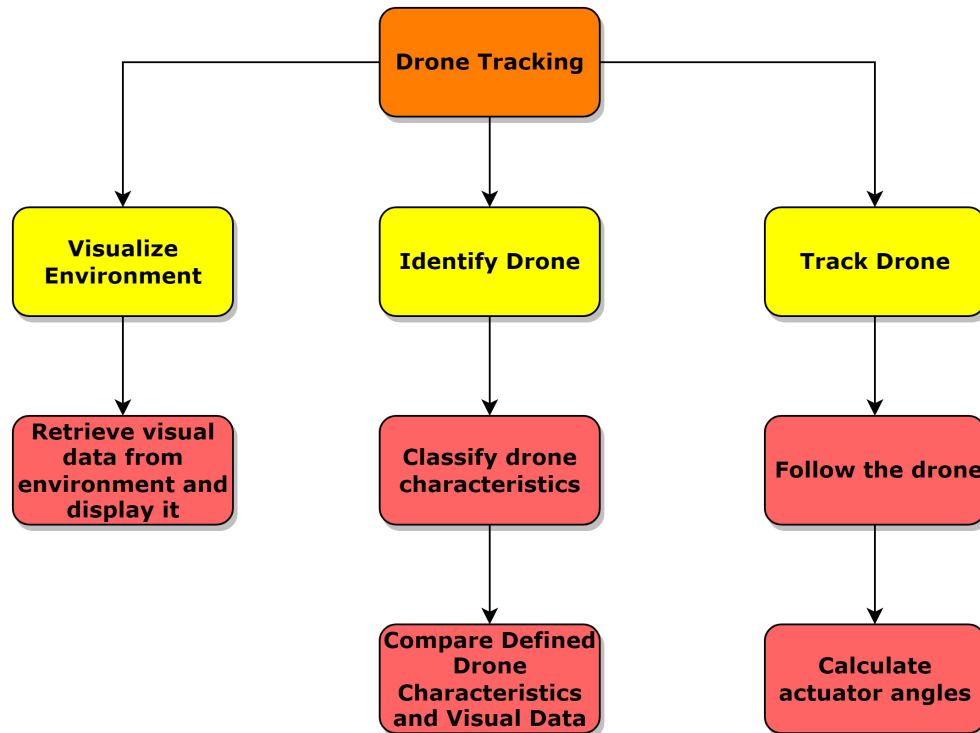


Figure 5: Level II Decomposition of Drone Tracking

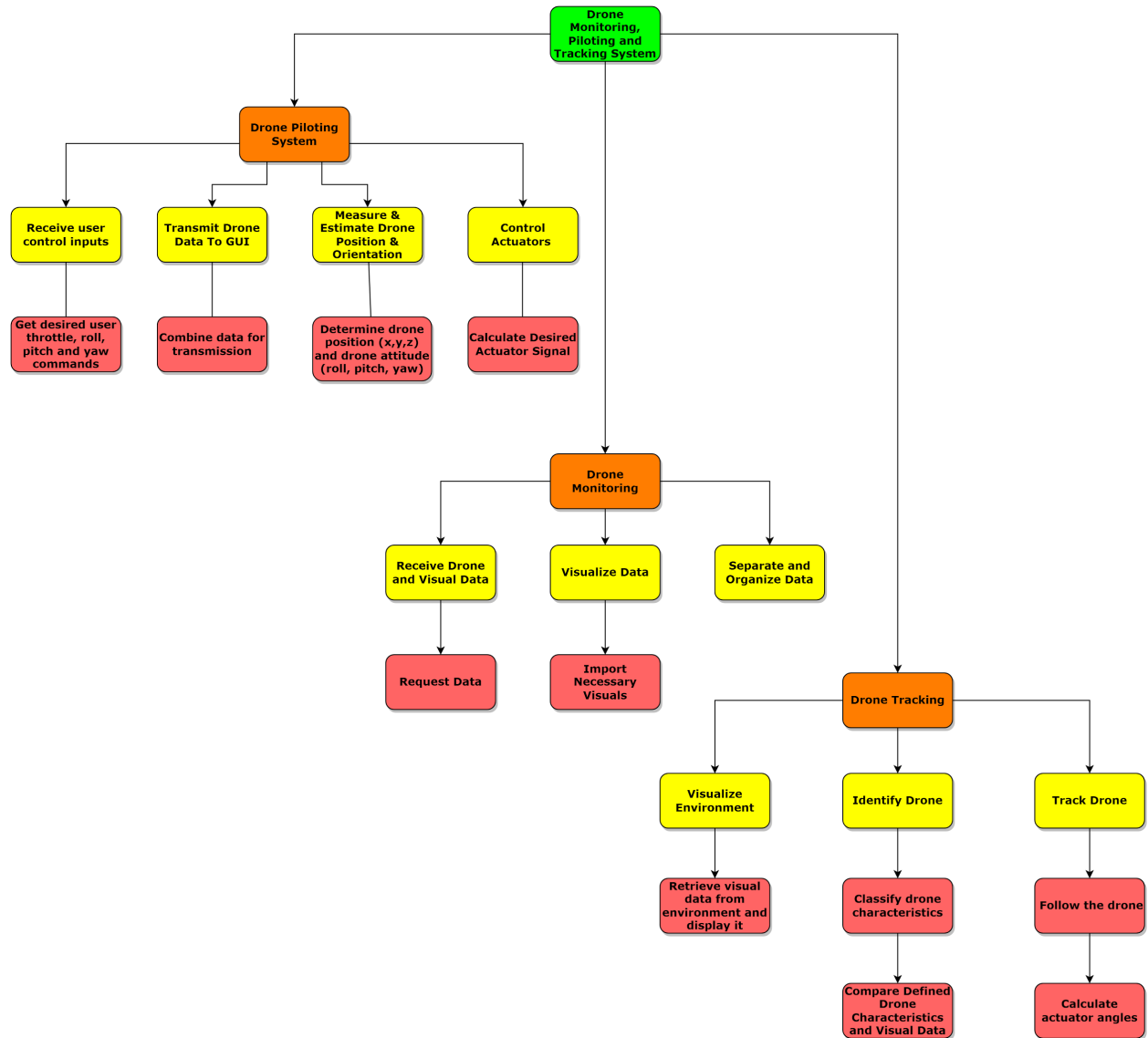


Figure 6: Functional Decomposition Diagram (Overall View)

Based on the system decomposition tree a list of modules were made for each of the three main high-level systems: Drone Tracking (Camera System), Drone Monitoring (Server/GUI), Drone Piloting (Drone System). Modules are grouped into different hierarchies to facilitate modularity and reusability, scalability, and for ease of maintenance and debugging.

2.2 Module Hierarchy

Level 1	Level 2	Module Number
CAMERA SYSTEM	Video Stream Module	M1
	Classification Module	M2
	Locator Module	M3
	Outliner Module	M4
	Tracking Module	M5
SERVER (GUI)	Socket Handler	M6
	Serial Port Handler	M7
	Communications Manager	M8
	Server Runner	M9
	User Client (ADT)	M10
	Chart Handler	M11
	Drone State	M12
	Data Parser	M13
	User Controller (Controller Class)	M14
	Client Runner	M15
	Camera Client	M16
	Communication Data (ADT)	M17
	Event Tags Module	M18
	Config Module	M19
	Serial Port Module	M20
	Server Module	M21
	Socket IO Module	M22
	Event Emitter Module	M23

DRONE	Sensor Data Module	M24
	Sensor Fusion Module	M25
	Radio Control Module	M26
	Control Module	M27
	Motor Mixing Module	M28
	Transmitter Module	M29
	Receiver Module	M30

Table 1: Module Hierarchy

Note: All modules in Table 1 are in yellow to reflect module hierarchy in Fig 6. Not all of the 26 modules are represented in Fig 6 due to space constraint.

2.3 Traceability Matrix

The following table below shows a module traceability matrix. This matrix is used to establish and manage the relationships between our different system modules throughout its lifecycle. Mainly, this is used to trace the functional requirement from their origin to ensure each requirement is addressed by the appropriate modules and that changes to requirements can be tracked and managed effectively.

Note: All modules (Level 1 listed on the right) correspond to a functional requirement listed on the left under Req ID.

Req ID	Requirement Description	Corresponding Module
<i>1. Unmanned Aerial System Requirements</i>		
FR-001	UAV shall be able to establish a connection to the ground station from a minimum distance of 2m from the test stand	Server (GUI) Module Drone System Module
FR-002	User must be able to remotely control UAV orientation (ϕ, θ, ψ)	Drone System Module
FR-003	UAV must be able to self stabilize its orientation with a damping ratio between $\zeta = 0.5 \sim 0.6$ and a settling time ($T_s < 4 \text{ sec}$)	Drone System Module
FR-004	UAV shall transmit real-time telemetry data to the	Drone System Module

	ground station	
<i>2. Tracking System Requirements</i>		
FR-005	Tracking system must be able to identify the UAV in the field of view that is between 2-5 m away	Camera System Module
FR-006	Tracking system must be able to keep the UAV within Field of View while moving up to 1.4 m/s	Camera System Module
<i>3. Communication System</i>		
FR-007	Connection between the drone and communication system should be maintained for at least 30 minutes	Drone System Module Server (GUI) Module
FR-008	Communication system must be able to receive 1 signal per second from the UAV with information about its telemetry	Server (GUI) Module
<i>4. Graphical User Interface</i>		
FR-009	GUI must be able to display data received from the communication system within 100 ms of receiving the data	Server (GUI) Module
FR-010	GUI must be able to send data to the tracking system	Server (GUI) Module
FR-011	GUI must be able to display data received from the tracking system	Server (GUI) Module
FR-012	GUI must be able to notify the user when it is unable to display data accurately.	Server (GUI) Module

Table 2: Traceability Matrix

3. MODULES (Module Guide, MIS, and MID)

3.1 CAMERA SYSTEM

3.1.1 Video Stream (ADT) Abstract Data Type (M1)

VideoStream

MODULE GUIDE

Description

The VideoStream module is in charge of retrieving video frames at the required resolution and frame rate from the camera. It is initialized at the beginning and runs until the program is stopped.

Secret

Video Retriever

MIS

Uses

Cv2

Imported Constants

None

Imported Types

None

Imported Functions

None

Exported Constants

None

Exported Types

VideoStream: Class

Frame: int [] []

Exported Access Programs

Routine Name	In	Out	Exceptions
new VideoStream	Int (), int	VideoStream	IllegalArgument Exception
start		VideoStream	
stop		Boolean	

read		Frame	
------	--	-------	--

Table 3: Video Stream Module Exported Access Programs

Assumptions

Routine new VideoStream is always called before all other exported access programs.

MID

Semantics

State Variables

current_frame : Frame datatype that holds the current frame being observed by the camera.

grabbed : Boolean value that holds whether the most recent frame was retrieved successfully.

stopped : Boolean value that holds whether the videostream has been stopped or not.

State Invariants

$0 \leq \text{current_frame}[i][j] \leq \text{maxPixelVal}$

Access Routine Semantics

VideoStream(resolution, framerate)

Transition:

- Initializes a videostream object, retrieving video data from the camera in the given resolution of type int tuple and framerate which is type int. Raises an `IllegalArgumentException` if arguments are of wrong type or if $\text{resolution} \notin \text{valid_resolutions}[]$ || $(\text{framerate} < 0 \ \&\& \ \text{framerate} > \text{maxFrameRate})$.
- Reads first frame and grabbed from stream. grabbed set to True if read() successfully. grabs frame and frame stores the frame retrieved.
- *stopped* set to False.

Exception: `IllegalArgumentException`

start()

Transition:

- Starts to read frames from stream if *stopped* is False.
- Updates *grabbed* and *current_frame* from videostream.
- Checks whether *current_frame*[i][j] is in the range of 0 to maxPixelVal.
- If *current_frame*[i][j] < 0 => *current_frame*[i][j] = 0.
- If *current_frame*[i][j] > maxPixelVal => *current_frame*[i][j] = maxPixelVal.

read()

Output:

- Returns most recent frame.

stop()

Transition:

- stopped set to True.

3.1.2 Classification Module (M2)

MODULE GUIDE

Description

Takes the frame as input and runs an object detection algorithm to return the boxes, classes, scores for each object detected.

Secret

Classification Algorithm

MIS

Uses

VideoStream

Imported Constants

None

Imported Types

Frame, Interpreter

Imported Functions

None

Imported Libraries

Interpreter

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine Name	In	Out	Exception
classify	Frame	Interpreter	FrameInvalidException
getboxes	Interpreter	Int [] []	

getclasses	Interpreter	string []	
getscores	Interpreter	Int []	

Table 4: Classification Module Exported Access Programs

Assumptions/Considerations

classify(frame) is called before all other functions in the module.

MID

Semantics

State Variables

Boxes : 2D Array containing start and end x and y coordinates for each box of object detected in the form [x_start, y_start,

Classes : Array of strings containing Name of each Detected Object

Scores : Array of floats containing confidence percentage of each detected object

State Invariants

None

Access Routine Semantics

classify(frame)

Transition:

- Checks if the frame array length is 0. If empty list return InvalidFrameException
- Runs an object detection function using the Interpreter class and the current frame.

Output:

- Returns an object Interpreter that contains all information of detected objects in the frame.

Exception:

- InvalidFrameException for empty frames.

getboxes()

Output:

- Returns a 2D int array containing the start and end points of the boxes around the detected objects.

getclasses()

Output:

- Returns a 1D string array containing the names of every object detected.

getscores()

Output:

- Returns a 1D float array containing the scores of each object detected.

3.1.3 Locator Module (M3)

MODULE GUIDE

Description

Takes the name of the desired object to be located and if the object is detected an array of X and Y coordinates of the object are returned.

Secret

Object Location Calculator

MIS

Uses

Classification Module

Imported Constants

None

Imported Types

Interpreter

Imported Functions

None

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine Name	In	Out	Exception
getobjectcoordinates	Frame, Enumeration	Int []	

Table 5:

Locator Module Exported Access Programs

Assumptions

None

MID

Semantics

State Variables

$x_coordinate$: integer

$y_coordinate$: integer

State Invariants

$0 \leq x_coordinate \leq \text{maxWidth}$

$0 \leq y_coordinate \leq \text{maxHeight}$

Access Routine Semantics

getobjectcoordinates(Interpreter, class)

Transition:

- Returns an array containing x and y coordinates of a detected object if the class specified has been detected.

- If $\text{getclasses}()[i] = \text{class}$

$$x_coordinate = \frac{\text{getboxes}()[i][0] + \text{getboxes}()[i][2]}{2}$$

$$y_coordinate = \frac{\text{getboxes}()[i][1] + \text{getboxes}()[i][3]}{2}$$

Where $\text{getboxes}[i] = [x_start, y_start, x_end, y_end]$

- If $x_coordinate < 0 \Rightarrow x_coordinate = 0$
- If $x_coordinate > \text{maxWidth} \Rightarrow x_coordinate = \text{maxWidth}$
- If $y_coordinate < 0 \Rightarrow y_coordinate = 0$
- If $y_coordinate > \text{maxHeight} \Rightarrow y_coordinate = \text{maxHeight}$

Output:

- $[x_coordinate, y_coordinate]$

3.1.4 Outliner Module (M4)

MODULE GUIDE

Description

This module takes the frame, a 2D array detailing the boxes to be drawn with their starting and ending points, and the name of the box.

Secret

Object Outlining Algorithm

MIS**Imported Constants**

None

Imported Types

frame

Imported Functions

None

Imported Libraries

cv2

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine Name	In	Out	Exception
drawboxes	frame, int [] [], string	frame	

*Table 6: Outliner Module Exported Access Programs***Assumptions**

None

MID**Semantics****State Variables**

None

State Invariants

None

Access Routine Semantics*drawboxes(frame,boxes,class)*

Transition:

- Returns a frame that draws rectangles using the coordinates in *boxes[i]* for every i and labels the rectangles with the name *class* using functions from the cv2 library (cv2.rectangle, cv2.putText).

3.1.5 Tracking Module (M5)

MODULE GUIDE

Secret

Motor Signal Calculator

Description

This module will calculate the necessary motor signals and moves the camera accordingly towards the object to be tracked until it is in the center of the frame. This is done by calculating the x and y differences between the object and the center of the frame and then moving the motors in the direction of the object until the x and y differences are 0.

MIS

Uses

Locator Module, VideoStream Module

Imported Constants

None

Imported Types

None

Imported Functions

None

Imported Libraries

SERVO

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine Name	In	Out	Exception
track	Int []		

Table 7: Tracking Module Exported Access Programs

Assumptions

None

MID

Semantics

State Variables

Both state variables are initialized at 0

turn_angle_x : float

turn_angle_y : float

State Invariants

$0 \leq \text{turn_angle_x} \leq \text{maxHorizontalRotation}$

$0 \leq \text{turn_angle_y} \leq \text{maxHorizontalRotation}$

Access Routine Semantics

track(class)

Transition:

- $\text{turn_angle_x} = \text{turn_angle_x} - \frac{(\text{getobjectcoordinates}(\text{class})[0] - \text{horizontal_midpoint})}{\text{horizontal_midpoint}}$
- $\text{turn_angle_y} = \text{turn_angle_y} + \frac{(\text{getobjectcoordinates}(\text{class})[1] - \text{vertical_midpoint})}{\text{vertical_midpoint}}$
- If $\text{turn_angle_x} < 0 \Rightarrow \text{turn_angle_x} = 0$
- If $\text{turn_angle_y} < 0 \Rightarrow \text{turn_angle_y} = 0$

- If turn_angle_x > maxHorizontalRotation => turn_angle_x = maxHorizontalRotation
- If turn_angle_y > maxVerticalRotation => turn_angle_y = maxVerticalRotation
- Rotate servo responsible for the horizontal plane by turn_angle_x and the servo responsible for the vertical plane by turn_angle_y

3.2 SERVER (GUI)

3.2.1 Socket Handler (ADT) Abstract Data Type (M6)

Template Module inherits EventEmitter

SocketHandler

MODULE GUIDE

Description

The SocketHandler class manages socket connections between a server, camera systems, and user clients in a networked environment. It uses socket.io for real-time, bidirectional, and event-based communication.

Secrets:

The information and methodology for a server to interact with users and the camera system.

MIS

Uses

EventEmitter, Server, IO, CommunicationData

Syntax

Imported Constants

EventTags

Imported Types

Server, CommunicationData, IO

Exported Access Programs

Routine name In Out Exceptions

Routine Name	In	Out	Exceptions
--------------	----	-----	------------

new SocketHandler	Server	SocketHandler	IO Exception
startListening			IO Exception
sendDataToUser	CommunicationData		IO Exception
sendDataToCamera	CommunicationData		IO Exception
registerCamera	Socket		
registerUser	Socket		
getIO		IO	
isUserConnected		Boolean	
isCameraConnected		Boolean	
isSocketConnected	Socket	Boolean	

Table 8: Socket Handler Exported Access Programs

MID

Semantics

State Variables

io: IO

userSocket: Socket

cameraSocket: Socket

State Invariant

None

Assumptions

The programmer will not use SocketHandler the startListening() routine has been called.

Considerations

This class sets the CORS policy to allow all origins (security implications should be considered).

Access Routine Semantics

Constructor: new SocketHandler(server)

Transition:

- io, userSocket, cameraSocket := new Server(server), null, null

Output:

- out := self

Exception:

- Throw IO Exception if an unexpected IO error occurs (numerous unpredictable factors may disrupt connectivity)

startListening()

Transition:

- Listens for incoming connection requests from sockets

Exception:

- Throw IO Exception if an unexpected IO error occurs (numerous unpredictable factors may disrupt connectivity)

sendDataToUser(eventTag, data)

Transition:

- Emits an event with the specified eventTag and data to the user.

Exception:

- Throw IO Exception if an unexpected IO error occurs (numerous unpredictable factors may disrupt connectivity)

sendDataToCamera(eventTag, data)

Transition:

- Emits an event with the specified eventTag and data to the camera.

Exception:

- Throw IO Exception if an unexpected IO error occurs (numerous unpredictable factors may disrupt connectivity)

registerCamera(socket)

Transition:

- Registers a camera socket and sets up event listeners for camera-related events.

registerUser(socket)

Transition:

- Registers a user socket and sets up event listeners for user-related events.

getIO()

Output:

- out := io

isUserStillConnected()

Output:

- out := true if the user is still connected to the server, false otherwise.

isCameraConnected()

Output:

- out := true if the camera is still connected to the server, false otherwise.

3.2.2 Serial Port Handler (ADT) Abstract Data Type (M7)

Template Module inherits EventEmitter

SerialPortHandler

MODULE GUIDE

Description

The SerialPortHandler class provides functionalities to manage serial port connections, specifically for Arduino devices. It includes methods to list available ports, connect to a specified port, disconnect from the current port, and handle data communication.

Secrets:

The information and methodology for a server to read from serial input

MIS

Uses

EventEmitter, SerialPort

Syntax

Imported Constants

EventTags

Imported Types

SerialPort, PortInfo

Exported Access Programs

Routine name In Out Exceptions

Routine Name	In	Out	Exceptions
new SerialPortHandler	String[]	SerialPortHandler	

getListOfPorts		PortInfo[]	IO Exception
tryConnectingToPort	String		IO Exception
tryDisconnectingFromCurrentPort		Boolean	
isPortConnected		Boolean	

Table 9: Serial Port Handler Exported Access Programs

MID

Semantics

State Variables

serialPort: SerialPort

options: String[]

State Invariant

None

Assumptions

None

Considerations

A valid serial device is needed to successfully connect to the serial port

Access Routine Semantics

Constructor: new SerialPortHandler(customSettings)

Transition:

- options, serialPort := customSettings, serialPort

Output:

- out := self

getListOfPorts()

Transition:

- Retrieves a list of available serial ports

Output:

- Array of available ports with port information
- Identifies and marks ports that are already connected to the handler

Exception:

- Throw IO Exception if an unexpected IO error occurs (numerous unpredictable factors may disrupt connectivity)

tryConnectingToPort(path)

Transition:

- `serialPort := new SerialPort({path: path, ...this.options});`

Exception:

- Throw IO Exception if an unexpected IO error occurs (numerous unpredictable factors may disrupt connectivity)

tryDisconnectingFromCurrentPort()

Transition:

- Attempts to disconnect from the currently connected serial port

Output:

- **True** on successful disconnection, **False** otherwise

Exception:

- Throw IO Exception if an unexpected IO error occurs

isPortConnected(path)

Output:

- `out := serialPort ≠ null ∧ serialPort.isOpen()`

3.2.3 Communication Manager (Controller Class) (M8)

CommunicationManager

MODULE GUIDE

Description

The CommunicationManager class coordinates communication across different system components. It listens for events from serial and socket connections, processes incoming data, and routes it to the appropriate destination. It handles user inputs, camera data, and serial data from devices like an Arduino.

Secrets:

- The methodology for handling different inputs from different systems (user, camera, serial)
- The communication routing between the different systems (e.g. how serial data reaches the user)

MIS

Uses

SocketHandler, SerialPortHandler

Syntax**Imported Constants**

EventTags

Imported Types

SocketHandler, SerialPortHandler

Exported Access Programs

Routine name In Out Exceptions

Routine Name	In	Out	Exceptions
new CommunicationManager	Server, String[]	CommunicationManager	
startListening			
sendSerialPortListToUser			

Table 10: Communications Manager Exported Access Programs

MID**Semantics****State Variables**

socketHandler: SocketHandler

serialPortHandler: SerialPortHandler

State Invariant

None

Assumptions

None

Considerations

None

Access Routine Semantics

Constructor: new CommunicationManager(server, customOptions = {})

Transition:

- socketHandler := new SocketHandler(server),
- serialPortHandler := new SerialPortHandler(customOptions)

Output:

- out := self

startListening()

Transition:

- socketHandler.startListening();
- Listens for incoming connection requests from sockets

Exception:

- Throw IO Exception if an unexpected IO error occurs (numerous unpredictable factors may disrupt connectivity)

sendSerialPortListToUser()

Transition:

- Fetches the list of available serial ports and sends it to the connected client.

3.2.4 Server Runner (Runner Class) (M9)

ServerRunner

MODULE GUIDE

Description

The ServerRunner application starts the server and initializes all relevant variables.

Secrets:

Contains Run Sequence for Server

MIS

Uses

CommunicationManager, Server

Syntax

Imported Constants

None

Imported Types

CommunicationManager, Server

Exported Access Programs

None

MID

Semantics

State Variables

communicationManager: CommunicationManager

State Invariant

None

Assumptions

Server Runner will be used to initialize the server and start an instance of CommunicationManager to listen for socket and serial connections.

Considerations

None

Access Routine Semantics

None

3.2.5 User Client (ADT) Abstract Data Type (M10)

UserClient

MODULE GUIDE

Description

The UserClient class facilitates client-side socket communication for the user. It connects to a server using Socket.io, handles various socket events, processes incoming data (including Arduino data and image data), and manages serial port information.

Secrets:

The information and methodology for a user to interact with the server and camera system.

MIS

Uses

Socket.io, CommunicationData

Syntax

Imported Constants

EventTags, Config

Imported Types

DataParser, IO, CommunicationData

Exported Access Programs

Routine Name	In	Out	Exceptions
new UserClient		UserClient	
tryConnectingToServer			IO Exception
registerEvents			
requestSerialPortData			
disconnectCurrentSerialPort			
sendDataToServer	CommunicationData		

Table 11: User Client (ADT) Exported Access Programs

MID

Semantics

State Variables

socket: Socket

State Invariant

None

Assumptions

None

Considerations

None

Access Routine Semantics

Constructor: new UserClient(server)

Transition:

- socket := null

Output:

- out := self

tryConnectingToServer()

Transition:

- socket := io(Config.server.address)

Exception:

- Throw IO Exception if failed to connect to server

registerEvents()

Transition:

- Registers events for handling incoming data and server communication.

requestSerialPortData()

Transition:

- Requests the list of serial ports from the server
- The server is expected to return the serial port data in a separate event that will be handled internally

disconnectCurrentSerialPort()

Transition:

- Sends a request to the server to disconnect the current serial port

sendDataToServer(CommunicationData)

Transition:

- Emits a specified event with data to the server.
- This data is handled internally on the server side

3.2.6 Chart Handler (ADT) Abstract Data Type (M11)

ChartHandler

MODULE GUIDE

Description

The ChartHandler class is responsible for storing and updating the charts and graphs on the client-side user interface. It handles the visual representation of drone data on the GUI.

Secrets:

- Data management for chart visualization

MIS

Uses

Chart.js

Syntax

Imported Constants

None

Imported Types

Chart

Exported Access Programs

Routine Name	In	Out	Exceptions
new ChartHandler		ChartHandler	
addChart	String, String		
updateChart	String, String		

Table 12: Chart Handler Exported Access Programs

MID

Semantics

State Variables

charts: { label : String, chart : Chart | (label , chart)} # Map containing chart data in format (String, Chart)

State Invariant

None

Assumptions

None

Considerations

None

Access Routine Semantics

Constructor: new ChartHandler()

Transition:

- `charts := <>`

Output:

- `out := self`

`addChart(label, type)`

Transition:

- `charts' := 'charts & {label , new Chart(type)} # Append a new chart to map charts`
- The label acts as the identifier for the chart. When a chart needs to be updated, its label is used to identify it.

`updateChart(label, data)`

Transition:

- Updates a chart that is linked to a specified label with the new data
- The label acts as the identifier for the chart. When a chart needs to be updated, its label is used to identify it.

3.2.7 Drone State (ADT) Abstract Data Type (M12)

DroneState

MODULE GUIDE

Description

The DroneState class encapsulates state information regarding a drone. It has state information such as ID number, location coordinates, acceleration, speed, and battery percentage.

Secrets:

DroneState Formatter

MIS

Uses

None

Syntax

Imported Constants

None

Imported Types

None

Exported Access Programs

Routine Name	In	Out	Exceptions
new DroneState		DroneState	
setName	String		
setIp	String		
setThreatLevel	Enumeration		
setLocation	Integer[]		
setVelocity	Integer[]		
setAcceleration	Integer[]		
setBatteryPercentage	Integer		

Table 13: Drone State Exported Access Programs

MID

Semantics

State Variables

name: String

ip: String

threatLevel: String

location: Integer[]

velocity: Integer[]

acceleration: Integer[]

batteryPercentage: Integer

State Invariant

None

Assumptions

None

Considerations

None

Access Routine Semantics

Constructor: new DroneState()

Transition:

- name, ip, threatLevel, location, velocity, acceleration, batteryPercentage := null, null, null, \diamond , \diamond , \diamond , 0

Output

- out := self

setName(name)

Transition:

- self.name := name

setIp(ip)

Transition:

- self.ip := ip

setThreatLevel(threatLevel)

Transition:

- self.threatLevel := threatLevel
- Options: “Hostile”, “Friendly” or “Unknown”

setLocation(location)

Transition:

- self.location := location

setAcceleration(acceleration)

Transition:

- self.acceleration := acceleration

setVelocity(velocity)

Transition:

- self.velocity := velocity

setBatteryPercentage(batteryPercentage)

Transition:

- self.batteryPercentage := batteryPercentage

3.2.8 Data Parser (Utility Class) (M13)

DataParser

MODULE GUIDE

Description

The DataParser class is a utility class responsible for parsing raw data received from the drone. The raw data is converted to a more structured and usable format. It is responsible for breaking down a data string into components such as name, IP, threat level, location, velocity, and other drone-specific metrics.

Secrets:

Methodology for transforming raw data into structured, formatted information

MIS**Description****Uses**

DroneState

Syntax**Imported Constants**

DataIndexes

Imported Types

DroneState

Exported Access Programs

Routine Name	In	Out	Exceptions
parseData	String	DroneState	InvalidArgumentException

Table 14: Data Parser Exported Access Programs

MID**Semantics****State Variables**

None

State Invariant

None

Assumptions

None

Considerations

None

Access Routine Semantics

parseData(rawDroneData)

Output:

- An instance of DroneState that represents the inputted drone data

Exception:

- Throw InvalidArgument Exception if inputted drone data is not parsable

3.2.9 User Controller(Controller Class) (M14)

UserController

MODULE GUIDE

Description

UserController coordinates the client-side functionalities of this system. It uses UserClient, ChartHandler, and DataParser to handle incoming data and display the data to the user. It also allows for user input, allowing the user to interact with the camera system, server, and GUI.

Secrets:

The methodology for handling different inputs from the server

MIS

Uses

UserClient, ChartHandler, DataParser

Syntax

Imported Constants

None

Imported Types

UserClient, Chart, DroneState

Exported Access Programs

Routine Name	In	Out	Exceptions
new UserController		UserController	
connectToLocalServer			IO Exception
sendUserInputToServer	CommunicationData		IO Exception

Table 15: User Controller (Controller Class) Exported Access Programs

MID

Semantics

State Variables

userClient: UserClient

State Invariant

None

Assumptions

None

Considerations

None

Access Routine Semantics

Constructor: new UserClient()

Transition:

- userClient := new UserClient();

Output:

- out := self

connectToLocalServer()

Transition:

- userClient.tryConnectingToServer()
- Tries to connect to the local server

Exception:

- Throw IO Exception if failed to connect to server

sendUserInputToServer(communicationData)

Transition:

- `userClient.sendDataToServer(communicationData)`

Exception:

- Throw IO Exception if failed to connect to server

3.2.10 Client Runner(Runner Class) (M15)

ClientRunner

MODULE GUIDE

Description

The ClientRunner application initializes all the relevant variables on the client-side.

Secrets:

Order of Modules to Run

MIS

Uses

UserController

Syntax

Imported Constants

None

Imported Types

UserController

Exported Access Programs

None

MID

Semantics

State Variables

`userController`: UserController

State Invariant

None

Assumptions

None

Considerations

None

Access Routine Semantics

None

3.2.11 Camera Client(ADT) Abstract Data Type (M16)

CameraClient

MODULE GUIDE

Description

CameraClient is a class designed to manage a camera client in a network environment. It handles socket communications to send and receive data, particularly managing image updates and user input events.

Secrets:

The information and methodology for a user to interact with the server and user.

MIS

Uses

Socket.io, base64

Syntax

Imported Constants

EventTags

Imported Types

Socketio, Client, base64

Exported Access Programs

Routine Name	In	Out	Exceptions
new CameraClient		CameraClient	
connect	String		IO Exception
disconnect			
update_image	String		

Table 16: Camera Client Exported Access Programs

MID

Semantics

State Variables

sio: Socketio.Client

State Invariant

None

Assumptions

None

Considerations

None

Access Routine Semantics

Constructor: new CameraClient(server)

Transition:

- sio := null

Output:

- out := self

connect(server_address)

Transition:

- sio.connect(server_address)

Exception:

- Throw IO Exception if unexpected network error occurs

disconnect()

Transition:

- `sio.disconnect()`

Exception:

- Throw IO Exception if unexpected network error occurs

update_image(raw_image)

Transition:

- `sio.send_data(encode_image(raw_image))`
- Encodes the image and sends it along with supplementary camera data to the server

Local Functions (private methods)

`encode_image` : Object \rightarrow String

`encode_image(raw_image) \equiv base64.b64encode(raw_image).decode('utf-8')`

`send_data`: Object \rightarrow Void

`send_data(data) \equiv sio.emit('video', data)`

3.2.12 Communication Data (ADT) Abstract Data Type (M17)

CommunicationData

MODULE GUIDE

Description

The CommunicationData class is designed for managing and formatting communication data, primarily by storing the type of request and generating a timestamped string representation of the data. This functionality is useful for tracking and logging communication requests in a system, ensuring that each request is clearly identified and time-stamped for effective monitoring and debugging.

Secrets:

CommunicationData Formatter

MIS

Uses

None

Syntax**Imported Constants**

None

Imported Types

None

Exported Types

```
TARGETS = {  
    SERIAL,  
    CAMERA,  
    USER  
}
```

```
ACTIONS = {  
    GENERAL: {  
        TARGET_LOCK,  
        NEW_CONNECTION  
    },  
    USER: {  
        SEND_IMAGE_DATA,  
        SEND_PORT_LIST  
    },  
    SERIAL: {  
        CONNECT,  
        RECONNECT,  
        DISCONNECT,  
        GET_PORT_LIST  
    },  
    CAMERA: {  
        CONFIG_CHANGE  
    }  
}
```

```

REQUEST_TYPES = {
    CONFIG_UPDATE,
    DATA_REQUEST,
    DATA_UPDATE
}

```

Exported Access Programs

Routine Name	In	Out	Exceptions
new CommunicationData		CommunicationData	
setRequestType	String		
setAction	String		
setTarget	String		
setValue	String		

Table 17: Communication Data (ADT)

MID

Semantics

State Variables

requestType: String

action: String

target: String

value: String

State Invariant

None

Assumptions

None

Considerations

None

Access Routine Semantics

Constructor: new CommunicationData()

Transition:

- requestType, action, target, value := null, null, null, null

Output

- out := self

setRequestType(requestType)

Transition:

- self.requestType := requestType

setAction(action)

Transition:

- self.action := action

setTarget(target)

Transition:

- self.target := target

setValue(value)

Transition:

- self.value := value

3.2.13 Event Tags Module (M18)

EventTags

MODULE GUIDE

Description

EventTags are custom tags that will be used to emit certain events based on an identifier value

Secrets:

EventTag Dictionary

MIS

Uses

None

Syntax

Exported Constants

```
SERVER_EVENT_TAGS = {  
    CAMERA_IMAGE_DATA: 'camera_image_data_to_server',
```

```
CAMERA_SYSTEM_DATA: 'camera_system_data_to_server',  
USER_INPUT: 'user_input_to_server',  
USER_SYSTEM_DATA: 'user_system_data_to_server',  
SERIAL_DATA: 'serial_data_to_server',  
}
```

```
CLIENT_EVENT_TAGS = {  
  CAMERA_IMAGE_DATA: 'camera_image_data_to_client',  
  CAMERA_SYSTEM_DATA: 'camera_system_data_to_client',  
  USER_INPUT: 'user_input_to_client',  
  USER_SYSTEM_DATA: 'user_system_data_to_client',  
  SERIAL_DATA: 'serial_data_to_client',  
}
```

Exported Types

None

Exported Access Programs

None

MID

Semantics

State Variables

None

State Invariant

None

Assumptions

None

Considerations

None

Access Routine Semantics

None

3.2.14 Config Module (M19)

Config

MODULE GUIDE

Description

This module contains configuration settings that can be used

Secrets:

Configuration Setup

MIS

Uses

None

Syntax

Exported Constants

```
Config = {  
  server: {  
    arduinoPort: "COM4",  
    serverPort: 3000,  
    address: 'localhost'  
  },  
  parser: {  
    numOfDecimalsToDisplay: 1,  
    useMessageMarkingFlags: true,  
    messageBeginFlag: '!',  
    messageEndFlag: '@'  
  }  
}
```

Exported Types

None

Exported Access Programs

None

MID**Semantics****State Variables**

None

State Invariant

None

Assumptions

None

Considerations

None

Access Routine Semantics

None

3.2.15 Serial Port Module (M20)

SerialPort

Considerations

Third party library as described in <https://developer.mozilla.org/en-US/docs/Web/API/SerialPort>

3.2.16 Server Module (M21)

Server

Considerations

Third party library as described in <https://nodejs.org/api/http.html#class-httpserver>

3.2.17 Socket IO Module (M22)

SocketIO

Exported Type

SocketIO,
IO

Considerations

Third party library as described in <https://socket.io/docs/v4/>

3.2.18 Event Emitter Module (M23)

EventEmitter

Considerations

Third party library as described in <https://nodejs.org/api/events.html#class-eventemitter>

3.3. DRONE (MIS & MID)**3.3.1 Sensor Data Module (Utility Class) (M24)**

MIS

Secret: Data Reader, Data Formatter

Description

This module is responsible for reading data from sensor registers as well as converting the data from the format it was stored as in the registers to a more usable format ie. floats, integer, boolean etc.

Imported Constants

None

Imported Types

None

Exported Constants:

None

Exported Functions:

gyro_setup(), gyro_signal(), accel_setup(), accel_signal()

Exported Types:

None

Exported Access Programs

Routine Name	In	Out	Exceptions
Sensor Data	Environmental Input	Angular Rates, Acceleration	
gyro_setup	Sensor Register Address		RegisterWriteExcepti on
gyro_signal	Sensor Register Address	float gyro_rates[roll_rates, pitch_rates, yaw_rates]	RegisterReadExcepti on RegisterWriteExcepti on
accel_setup	Sensor Register Address		RegisterWriteExcepti on
accel_signal	Sensor Register Address	float acceleration[ax, ay, az]	RegisterReadExcepti on RegisterWriteExcepti on

Table 18: Sensor Data Module Exported Access Programs

Assumptions:

None

Considerations:

None

MID

State Variables

None

Access Routine Semantics

Constructor: new Sensor Data()

Transition:

- Detects the sensor connected to device and sets global sensor register variables to their respective values

gyro_setup()

Transition:

- Setup up gyro sensor sensitivity, accuracy and setup I2C clock speed

Exception:

- Prints an error message if sensor registers are not detected

gyro_signal()

Transition:

- Starts the I2C communication, switch on low-pass filter, set sensor sensitivity scale factor, access registers storing gyro measurements

Exception:

- Prints error message if I2C communication fails to start
- Prints error message if register storing measurements can not be accessed

`accel_setup()`

Transition:

- Setup up accel sensor sensitivity, accuracy and setup I2C clock speed

Exception:

- Prints an error message if sensor registers are not detected

`accel_signal()`

Transition:

- Starts the I2C communication, switch on low-pass filter, set sensor sensitivity scale factor, access registers storing accel measurements

Exception:

- Prints error message if I2C communication fails to start
- Prints error message if register storing measurements can not be accessed

3.3.2 Sensor Fusion Module (Utility Class) (M25)

MIS

Secret: Sensor Fusion Algorithm (For now: Kalman Filter)

Description

This module will fuse together all the sensor data in a manner which will minimize the variance in the final orientation estimation. For the time being we are implementing a Kalman filter to fuse the sensor data but this algorithm is subject to change.

Imported Constants

None

Imported Types

None

Exported Constants:

None

Exported Functions:

Kalman(KalmanState, KalmanUncertainty, KalmanInput, KalmanMeasurement)

Exported Types:

None

Exported Access Programs

Routine Name	In	Out	Exceptions
Kalman()	KalmanState, KalmanUncertainty, KalmanInput, KalmanMeasurement	KalmanState, KalmanUncertainty, KalmanGain	

*Table 19: Sensor Fusion Module Exported Access Programs***Assumptions:**

None

Considerations:

None

MID**State Variables**

None

Access Routine Semantics

Kalman(KalmanState, KalmanUncertainty, KalmanInput, KalmanMeasurement)

Transition:

- Given the inputs *KalmanState*, *KalmanUncertainty*, *KalmanInput*, *KalmanMeasurement* the following algorithm is conducted to update the KalmanGain, KalmanMeasurement and KalmanUncertainty

Kalman Filter Algorithm

```

Ts = 0.004;
gyro_rate_error = 16; // Provided in the MPU6050 Datasheet
accel_measurement_error = 9; // Provided in the MPU6050 Datasheet
KalmanState = KalmanState + Ts*KalmanInput;
KalmanUncertainty = KalmanUncertainty + Ts*Ts*gyro_rate_error;
KalmanGain = KalmanUncertainty *1/(KalmanUncertainty + accel_measurement_error);
KalmanState = KalmanState + KalmanGain*(KalmanMeasurement-KalmanState);
KalmanUncertainty =(1-KalmanGain)*KalmanUncertainty;

```

3.3.3 Radio Control Module (Utility Class) (M26)

MIS

Secret: Receiver Setup, Radio Data Parser

Description

This module is responsible for reading data received from radio transmitters and parsing the data in order to separate them into the various transmitter channels.

Uses

#include <PulsePosition.h> // PulsePosition can transmit and receive PPM (Pulse Position Modulated) signals commonly used to control RC aircraft and servo motors. Up to 8 simultaneous input and/or output PPM streams may be used, with each stream conveying up to 16 signals¹

Imported Constants

None

Imported Types

None

Exported Constants:

None

Exported Functions:

receiver_setup(), receiver_parser()

Exported Types:

None

Exported Access Programs

Routine Name	In	Out	Exceptions
receiver_setup()	None	None	NoReciverDetectedException
receiver_parser()	Raw Radio PPM signal	Parsed Transmitter Channel Data	Missing Channel Data Exception

Table 20: Radio Control Module Exported Access Programs

¹ Pulse Position Library information received from Teensy Manufacturer Website: [PulsePosition Library, for multiple high-res PPM encoded signal streams to and from Teensy 3.1 \(pjrc.com\)](http://www.pjrc.com/teensy/pulse_position_library/)

Assumptions:

None

Considerations:

None

MID**State Variables with Types**

None

Access Routine Semantics

receiver_setup()

Transition:

- Setup radio receiver channel

Exception:

- Prints an error message if radio receiver is not detected

receiver_parser()

Transition:

- Setup radio receiver channel

Exception:

- Prints an error message if there parser detects missing channel data or incorrect sized data packet

3.3.4 Control Module (Utility Class) (M27)**MIS**

Secret: PID Control Algorithm

Description

This module hosts the PID control algorithm used in the overall drone controller.

Imported Constants

None

Imported Types

None

Exported Constants:

None

Exported Functions:

pid(Error, Kp, Ki, Kd, Prev_Error, Prev_Ki), pid_reset()

Exported Types:

None

Exported Access Programs

Routine Name	In	Out	Exceptions
pid()	Error, Kp, Ki, Kd, Prev_Error, Prev Ki	PID signal	Integral Windup Exception
pid_reset()	PID Constants (Kp, Ki, Kd)		

Table 21: Control Module Exported Access Programs

MID

State Variables with Types

None

Assumptions:

None

Considerations:

None

Access Routine Semantics

pid(Error, Kp, Ki, Kd, Prev_Error, Prev_Ki)

Transition:

- Using the user defined PID constants compute the aggregated PID signal

Exception:

- If integral windup is detected an error message will be printed to the screen

pid_reset()

Transition:

- PID constants are set to zero

3.3.5 Motor Mixing Module (Utility Class) (M28)

MIS

Secret: Motor Mixing Algorithm

Description

This module is responsible for initializing the motors, actuating motors and also providing the appropriate motor commands based on the quadcopter configurations used.

Forces and torques on the quadrotor can be written in matrix form such as:
 k_1 and k_2 are constants that need to be determined experimentally, δ is the motor command signal.

$$\begin{pmatrix} F \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{pmatrix} = \begin{pmatrix} k_1 & k_1 & k_1 & k_1 \\ 0 & -\ell k_1 & 0 & \ell k_1 \\ \ell k_1 & 0 & \ell k_1 & 0 \\ -k_2 & k_2 & -k_2 & k_2 \end{pmatrix} \begin{pmatrix} \delta_f \\ \delta_r \\ \delta_b \\ \delta_l \end{pmatrix} \triangleq \mathcal{M} \begin{pmatrix} \delta_f \\ \delta_r \\ \delta_b \\ \delta_l \end{pmatrix}$$

The control matrix is implemented in code as follows with the controllability matrix specifying the forces and torques..

$$\begin{pmatrix} \delta_f \\ \delta_r \\ \delta_b \\ \delta_l \end{pmatrix} = \mathcal{M}^{-1} \begin{pmatrix} F \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{pmatrix}.$$

Imported Constants

None

Imported Types

None

Exported Constants:

None

Exported Functions:

motor_init(), motor_mixing()

Exported Types:

None

Exported Access Programs

Routine Name	In	Out	Exceptions
motor_init()	Motor Pins		MotorNotConnected Exception
motor_mixing()	Drone Commands	Motor Signals	

Table 22: Motor Mixing Module Exported Access Program

Assumptions:

None

Considerations:

None

MID

State Variables

None

Access Routine Semantics

motor_init()

Transition

- Starts the I2C communication, switch on low-pass filter, set sensor sensitivity scale factor, access registers storing gyro measurements

Exception:

- Prints error message if I2C communication fails to start
- Prints error message if register storing measurements can not be accessed

motor_mixing()

Transition

- Calculate the appropriate motor commands based the selected drone configuration selected and output those motor signals to the electronic speed controller circuit

3.3.6 Transmitter Module (M29)

Module Guide

Secret

Transmission Protocol

Description

The code configures the module for two-way communication, generates random data for transmission and utilizes acknowledgement mechanisms to ensure successful delivery. It alternates between transmitting data and waiting for a response, creating a basic and reliable point-to-point communication system.

MIS

Uses

SPI, nRF24l01, RF24, Arduino

Imported Constants

None

Imported Types

None

Imported Libraries

SPI, nRF24l01, RF24, Arduino

Imported Functions

None

Access Routine

Routine Name	In	Out	Exceptions
setup			
transmit_loop			

Table 23: Transmitter Module Exported Access Program

Assumptions

The setup function and RF24 libraries are always initialized before the transmission and monitoring process

MID

State variables

radio: This is an instance of ‘RF42’ class, and it represents the radio communication interface with the NRF24L01 module.

address: An array of addresses used for communication. It stores the addresses of the nodes involved in the communication used by the writing and the reading pipes.

State Invariants

$\forall a \in \text{address}, 0 < |a| < 6$

Each address is a string literal of 5 characters and the compiler adds the null terminator.

Access Routine Semantics

setup()

Transition:

- The RF24 object ('radio') is initialized, specifying CE (Chip-Enable) and CSN (Chip Select Not) pin
- Transmit power and data rate are configured to the desired level
- Writing and reading pipes are opened for communication with the receiver node
- NRF24L01 module is initialized, and various parameters such as transmit power level, data rate, and communication channel are configured to the desired level.
- Writing pipe is opened for communication with the receiver. The transmitter node is set to stop listening, indicating it is ready to transmit data.

transmit_loop()

Transition:

- An unsigned data type (string or char or integers) will be initialized with the essential data we want to send to the receiver.
- The NRF24L01 module is configured for transmitting mode to write the data for the receiver to read.
- The module is then configured for receiving mode to receive a response or modified data.
- If a response is received, the data is read and sent and received data are printed.
- The loop repeats every 1 second.

3.3.7 Receiver Module M30

Module Guide

Secret

Receiving Protocol

Description

The code configures the module for two-way communication, generates random data for transmission and utilizes acknowledgement mechanisms to ensure successful delivery. It alternates between transmitting data and waiting for a response, creating a basic and reliable point-to-point communication system.

MIS

Uses

SPI, nRF24L01, RF24, Arduino

Imported Constants

None

Imported Types

None

Imported Libraries

SPI, nRF24l01, RF24, Arduino

Imported Functions

None

Access Routine

Routine Name	In	Out	Exceptions
setup			
receiving_loop			

Table 24: Receiver Module Exported Access Program

Assumptions

The setup function and RF24 libraries are always initialized before the transmission and monitoring process

MID

State variables

radio : This is an instance of ‘RF42’ class, and it represents the radio communication interface with the NRF24L01 module.

address : An array of addresses used for communication. It stores the addresses of the nodes involved in the communication used by the writing and the reading pipes.

State Invariants

$\forall a \in \text{address}, 0 < |a| < 6$

Each address is a string literal of 5 characters and the compiler adds the null terminator.

Access Routine Semantics

setup()

Transition:

- The RF24 object (‘radio’) is initialized, specifying CE (Chip-Enable) and CSN (Chip Select Not) pin

- Receiver power and data rate is configured to the desired level
- Writing and reading pipes are opened for communication with the receiver node
- NRF24L01 module is initialized, and various parameters such as transmit power level, data rate, and communication channel are configured to the desired level.

receive_loop()

Transition:

- The loop checks if there is any available data from the transmitter
- Once the data is available, the loop enters a while loop to read the data from the NRF24L01 buffer
- After reading the data, the NRF24L01 module is temporarily set to stop listening.
A modified data or response is sent back to the transmitter\
- The module is set back to start listening for the next incoming data
- The loop prints a message to the serial monitor indicating the response sent back

4. ENGINEERING DRAWINGS

4.1 CAMERA SYSTEM

The camera system is composed of a total of 3 different 3D modeled parts and acts as a skeleton design to enable the two servo motors to rotate 180 degrees along the parallel planes. The following shows the exploded view drawing of the camera system along with 3 other sheets detailing the dimensioning of each part. Note that Item 2 is the servo motor themselves and their part drawings are not included here as we did not design those.

4.1.1 Exploded View

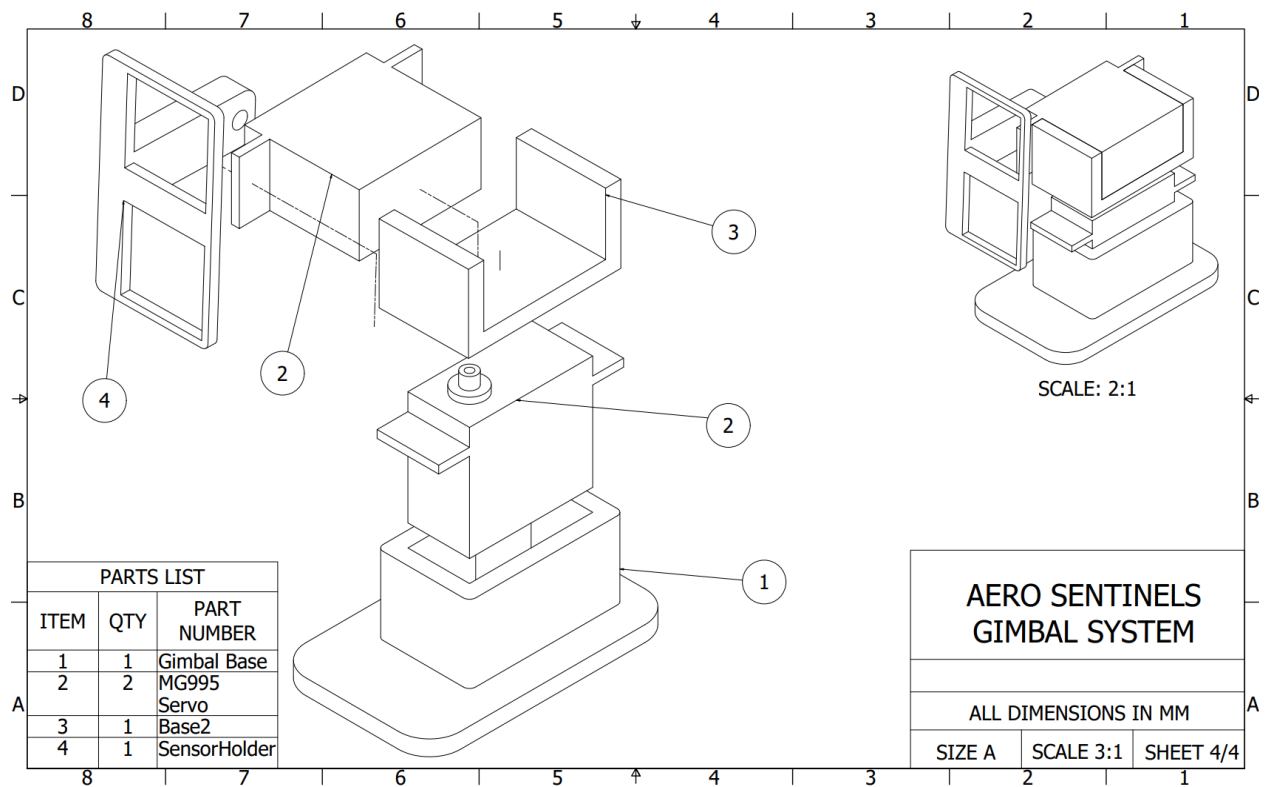


Figure 7: Camera System Exploded View

4.1.2 Gimbal Base

The gimbal base serves as housing for the first servo motor allowing the system to pan the horizontal plane.

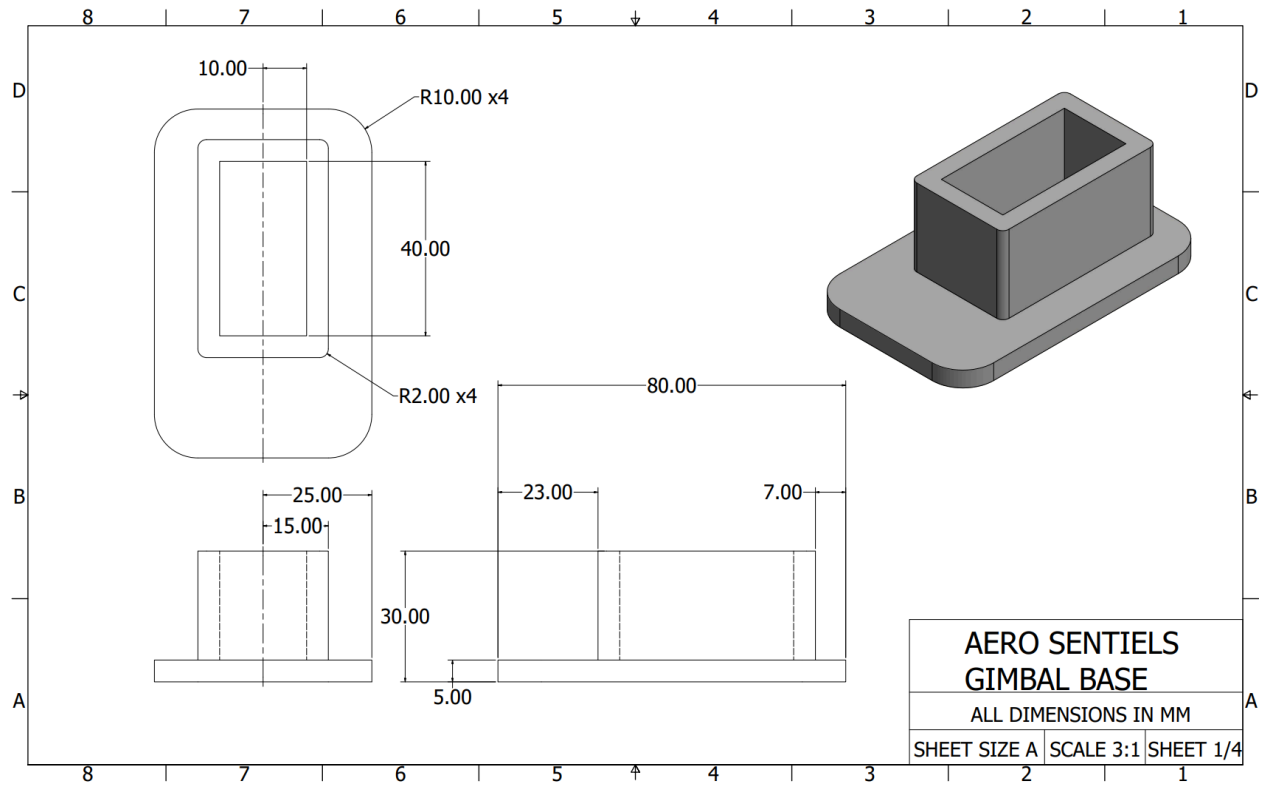


Figure 8: Camera System Gimbal Base

4.1.3 Base 2

Gimbal base 2 is the connection between motor 1 and motor 2 allowing vertical panning.

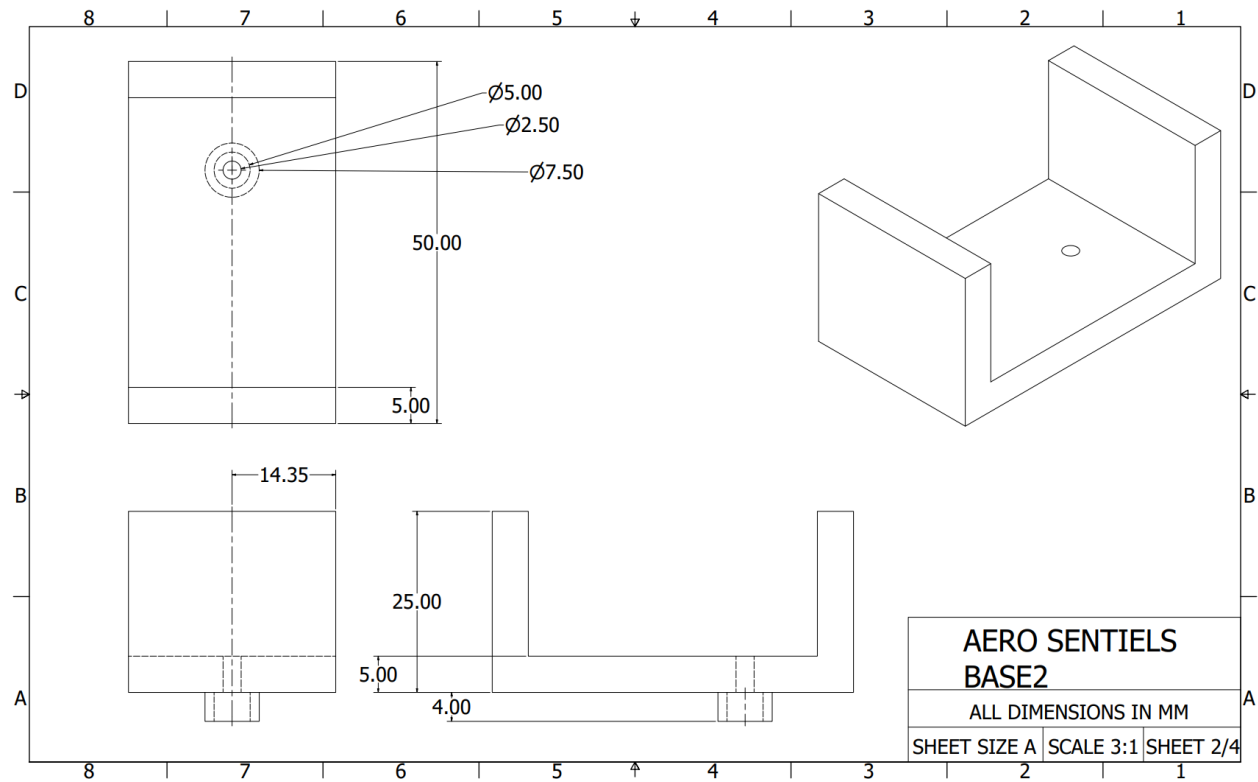


Figure 9: Camera System Base 2

4.1.4 Sensor Holder

The sensor holder is the part responsible for holding the physical camera module. 2 Slots are made depending on the type of camera module to be used in the final revision.

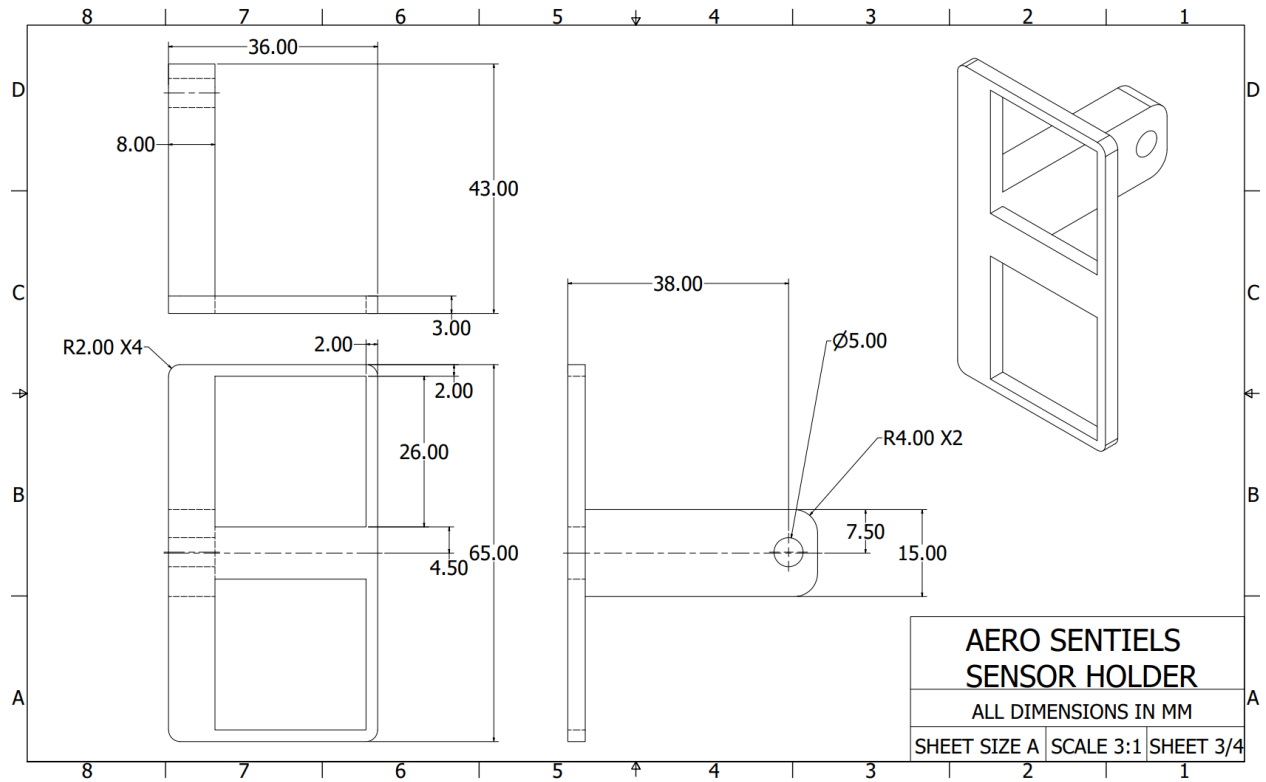


Figure 10: Camera System Sensor Holder

4.1.5 Anticipated Changes for Camera System

Depending on the type and size of sensor being used in our final revision, there might be slight modifications to the overall dimensions of the camera system. In the case of using a larger sensor, the Sensor Holder part is prone to change to accommodate for new sizing. Depending on the weight distribution of a new sensor, the Gimbal base might also be adjusted to account for center of balance ([Refer to Section 6 for full list of Anticipated Changes](#)).

4.2 DRONE



The entire drone model consists of three main components: the drone arms, main drone frame and the landing legs.



Figure 11: Drone Design

5. DRONE FLIGHT CONTROLLER

The drone flight controller consists of the following parts:

Description	Part Number	Image
Inertial Measurement Unit (IMU)	MPU6050	
Microcontroller (Teensy 4.0)	MIMXRT1062DVL6B	




Flysky 2.4GHz 6 Channel Radio Receiver	FS-iA6B	
30A BLHeli_S ESC	N/A	
E-Flite Lipo Battery		

Table 25: Flight Controller Parts

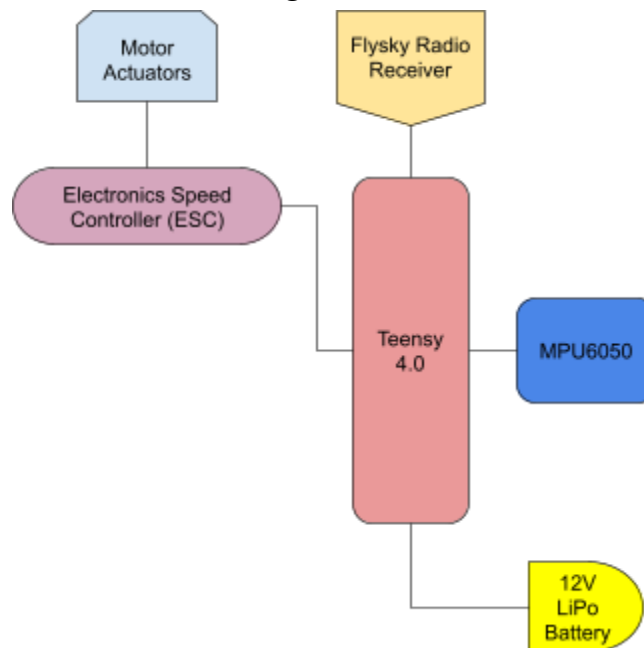


Figure 13: High Level Circuit Diagram

6. ANTICIPATED CHANGES

1. Camera

May be changed to improve system performance.

2. Gimbal Design

May be changed according to camera and to smooth tracking.

3. Classification Algorithm

Algorithms may change to incorporate different objects or to improve classification speed.

4. Motor Signal Calculation Method

The calculation method may be further optimized for smoother tracking.

5. Communication Hardware

Different communication methods may be considered in the future like bluetooth or internet as such the appropriate communication hardware will need to be implemented

6. Communication Protocol

The communication protocol used in the future may change to better fit our requirements

7. Drone Sensor Payload

Better sensors can be used onboard the drone for more accurate orientation and position measurement

8. Drone Control Algorithm

A more robust controller may be considered down the line in order to improve quadcopter performance

9. Sensor Fusion Algorithm

More robust sensor fusion algorithm can be considered to reduce computational complexity and reduce overall control loop speed

ANTICIPATED CHANGES TRACEABILITY MATRIX

Anticipated Change#	Modules Affected
1	Camera System Module
2	Gimbal System Physical Hardware Design
3	Camera System Module
4	Camera System Module
5	Communication Hardware (Eg; new hardware such as bluetooth modules, receivers, etc.)

6	Server (GUI) Module
7	Drone Design Hardware
8	Drone System Module
9	Drone System Module

Table 26: Anticipated Changes Traceability Matrix

7. DESIGN DECISIONS NOT LIKELY TO CHANGE

1. Choice of language (c#, javascript, python, html, css)
2. Basic Aerodynamic Design - the drone will remain a quadcopter design.
3. Power Source The requirement for a power source, typically batteries, which must be sufficient to power the drone for its intended flight duration.
4. Control System
5. Payload Capacity
6. Weather and Environmental Resistance - we do not consider weather as part of our design.
7. Wireless Communication - the drone will need to communicate wirelessly with the ground station.
8. Privacy Considerations - we do not consider privacy or security concerns.

8. REFERENCES

1. [1] MPU6050 Inertial Measurement Unit Datasheet: [MPU-6000-Datasheet1.pdf \(tdk.com\)](#)
2. [2] Teensy Manufacturer Website: [Teensy® 4.0 \(pjrc.com\)](#)