THESIS

SIM-1 UAS: A FRAMEWORK FOR RAPID PROTOTYPING OF MATLAB DEVELOPED FLIGHT TEST CODE

by

Juncun Su

September 2023

Thesis Advisor:	Isaac I. Kaminer
Co-Advisor:	Sean P. Kragelund

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** September 2023	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE SIM-1 UAS: A FRAMEWORK FOR RAPID PROTOTYPING OF MATLAB DEVELOPED FLIGHT TEST CODE	**5. FUNDING NUMBERS**
6. AUTHOR(S) Juncun Su	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.	**12b. DISTRIBUTION CODE** A

13. ABSTRACT (maximum 200 words)

Unmanned aerial systems (UAS) like the ScanEagle have been employed in both military and research applications. Despite the ScanEagle's low cost and operational flexibility, its utility for autonomy research is limited due to its proprietary hardware and software. With this in mind a new UAS, named SIM-1, was procured. It employs open-source hardware and software, making it suitable for research and development.

In this thesis, the SIM-1 UAS was assembled and successfully flown in simulated and actual test flights. Simulations were conducted using Gazebo software, which employs a physics-based virtual environment, while flight tests were carried out at local flying fields. A basic MATLAB control algorithm was developed for SIM-1, and flight paths were planned through the ground control station (GCS). Notably, two methods of flight path planning were explored in this thesis. The first method uses open-source GCS software, QGroundControl, while the second method uses algorithms developed, tested, and ported to C++ code using MATLAB/Simulink. This second method provided an avenue for guidance, navigation and control (GNC) prototype algorithms to be flight tested rapidly. Finally, SIM-1 was also configured with a machine learning algorithm for object detection using its onboard camera payload, which opens up opportunities for more advanced research with this UAS platform.

14. SUBJECT TERMS UAV, UAS, drone, autonomous, unmanned, SITL simulation, HITL simulation, guidance, navigation, control	**15. NUMBER OF PAGES** 89
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

SIM-1 UAS: A FRAMEWORK FOR RAPID PROTOTYPING OF MATLAB DEVELOPED FLIGHT TEST CODE

Juncun Su
Military Expert 5, Republic of Singapore Air Force
BME, National University of Singapore, 2010

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 2023

Approved by: Isaac I. Kaminer
Advisor

Sean P. Kragelund
Co-Advisor

Brian S. Bingham
Chair, Department of Mechanical and Aerospace Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Unmanned aerial systems (UAS) like the ScanEagle have been employed in both military and research applications. Despite the ScanEagle's low cost and operational flexibility, its utility for autonomy research is limited due to its proprietary hardware and software. With this in mind a new UAS, named SIM-1, was procured. It employs open-source hardware and software, making it suitable for research and development.

In this thesis, the SIM-1 UAS was assembled and successfully flown in simulated and actual test flights. Simulations were conducted using Gazebo software, which employs a physics-based virtual environment, while flight tests were carried out at local flying fields. A basic MATLAB control algorithm was developed for SIM-1, and flight paths were planned through the ground control station (GCS). Notably, two methods of flight path planning were explored in this thesis. The first method uses open-source GCS software, QGroundControl, while the second method uses algorithms developed, tested, and ported to C++ code using MATLAB/Simulink. This second method provided an avenue for guidance, navigation and control (GNC) prototype algorithms to be flight tested rapidly. Finally, SIM-1 was also configured with a machine learning algorithm for object detection using its onboard camera payload, which opens up opportunities for more advanced research with this UAS platform.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

3D	Three Dimensional
3DOF	3 Degree of Freedom
AGL	Above Ground Level
API	Application Programming Interface
AR	Aspect Ratio
CAVR	Center for Autonomous Vehicle Research
CNN	Convolutional Neural Network
COTS	Commercial Off-the-Shelf
CPU	Central Processing Unit
CRUSER	Consortium for Robotics and Unmanned Systems Education and Research
ENU	East, North, Up
EPO	Expanded Polyolefin
GCS	Ground Control Station
GNC	Guidance, Navigation and Control
GPS	Global Positioning System
GPU	Graphics Processing Unit
GUI	Graphical User Interface
IMU	Inertial Measurement Unit
ISR	Intelligence Surveillance and Reconnaissance
LiPo	Lithium Polymer
LTP	Local Tangent Plane
ML	Machine Learning
NN	Neural Network
NPS	Naval Postgraduate School
OS	Operating System
R&D	Research and Development

RC	Remote Control
RF	Radio Frequency
ROS	Robot Operating System
SITL	Software-in-the-Loop
UAS	Unmanned Aerial System
YOLO	You Only Look Once

ACKNOWLEDGMENTS

First, I would like to thank my thesis advisors, Professor Kaminer and Professor Kragelund, for the guidance and mentorship in this thesis journey. I am grateful for the unique opportunity to delve into the developmental aspects of an unmanned aerial system (UAS); it has been fun and educational. I also fondly recall the brainstorming sessions and the flight tests that we went through together. I wish you all the best in your stay in the Naval Postgraduate School (NPS) and hope to see you continue pushing boundaries in the realm of unmanned systems.

Second, I would like to thank my friends and family for your companionship and support, and for shaping the person I am today. I look forward to the moments we will share in the future again.

Finally, my deepest thanks go to my wife, Swee Yan, and my son, Matthias, for their love and company during my time in NPS. This thesis would not have come to fruition without them and their support.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. MOTIVATION

Unmanned aerial systems (UAS) such as the ScanEagle have been employed by the military for intelligence, surveillance, and reconnaissance (ISR) purposes. Users such as the United States Marine Corps, the United States Navy and the Singapore Navy have incorporated the use of ScanEagle to augment their ISR capabilities [1], [2]. Part of the ScanEagle's "popularity" could be attributed to its low cost, and operational flexibility, as pointed out by Gettinger from the Center for the Study of Drone at Bard College. Gettinger stated that the ScanEagle drove the development and adoption of unmanned technologies due to these reasons [3].

In addition to its operational use, the ScanEagle has also been used for UAS research in the Naval Postgraduate School (NPS). Notably, the NPS Center for Autonomous Vehicle Research (CAVR) is dedicated to educating students on unmanned vehicle technologies and advancing Naval autonomous vehicle operations. Within CAVR, a diverse range of unmanned systems, including the ScanEagle, is deployed for research purposes by NPS faculty and students [4]. Benjamin Keegan, for example, used the ScanEagle for his masters thesis exploring autonomously positioning UAS as wireless communication nodes for optimal communications [5].

Despite its advantages, the ScanEagle does have its own limitations. Like other proprietary products, ScanEagle's design, technology and intellectual property rights belong to Insitu. Inc. and Boeing as they are its designer and manufacturer [6]. As such, user autonomy over the design and functions of their ScanEagle is limited. With this in mind, a new UAS, named SIM-1, developed by Chesi UAS Solutions [7], was procured. SIM-1 is a commercial off-the-shelf (COTS) product, costs significantly less than the ScanEagle, and employs open-source hardware and software. These qualities make it suitable for research and development (R&D) purposes. Additionally, the SIM-1 UAS can also be equipped with newer technologies such as machine learning (ML) object detection algorithm, for more advanced research.

B. PROBLEM STATEMENT

With these motivations, this thesis focuses on answering the following question:

- How can a low cost, COTS UAS like the SIM-1 UAS be best utilized for education (in the form of supporting coursework and laboratory instruction) and advanced research (in flight testing novel sensing and control algorithms)?

C. THESIS OBJECTIVES

To assess the utility of the SIM-1 UAS for these purposes, this thesis evaluates and documents the following:

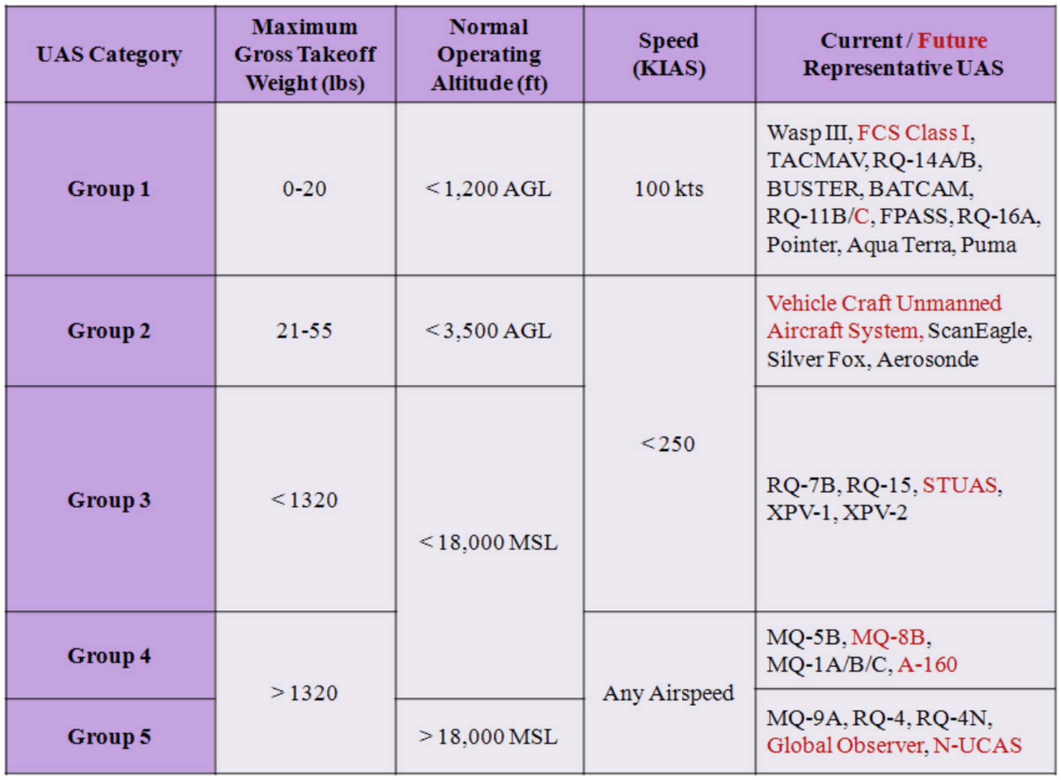
1. Assemble the SIM-1 UAS with the help of Chesi UAS Solutions.
2. Develop and integrate basic control laws into the SIM-1 UAS.
3. Perform flight path planning on the SIM-1 UAS.
4. Conduct simulated and real-life flight tests of the SIM-1 UAS.
5. Explore the integration of ML algorithm and utilize the onboard camera for object detection.

II. SYSTEM OVERVIEW, HARDWARE AND SOFTWARE

A. SYSTEM OVERVIEW

The SIM-1 UAS is a fixed-wing UAS, with the following key specifications: takeoff weight of 8.4 lbs (3.8 kg), normal operating altitude of below 1200 ft above ground level (AGL), and normal operating speed of 29.7 kts. Based on these specifications, it is a Group 1 UAS as per the United States Joint UAS Group classifications shown in Figure 1 [8]. For comparison, the ScanEagle is a Group 2 UAS based on the same categorization.

The SIM-1 UAS (Figure 2) is a COTS product from Chesi UAS Solutions, and it is significantly more cost effective than the ScanEagle. It employs open-source hardware and software, which allows greater flexibility in its parts and choice of software. This lower cost and flexibility make it ideal for R&D purposes.



UAS Category	Maximum Gross Takeoff Weight (lbs)	Normal Operating Altitude (ft)	Speed (KIAS)	Current / Future Representative UAS
Group 1	0-20	< 1,200 AGL	100 kts	Wasp III, FCS Class I, TACMAV, RQ-14A/B, BUSTER, BATCAM, RQ-11B/C, FPASS, RQ-16A, Pointer, Aqua Terra, Puma
Group 2	21-55	< 3,500 AGL	< 250	Vehicle Craft Unmanned Aircraft System, ScanEagle, Silver Fox, Aerosonde
Group 3	< 1320	< 18,000 MSL		RQ-7B, RQ-15, STUAS, XPV-1, XPV-2
Group 4	> 1320		Any Airspeed	MQ-5B, MQ-8B, MQ-1A/B/C, A-160
Group 5		> 18,000 MSL		MQ-9A, RQ-4, RQ-4N, Global Observer, N-UCAS

Figure 1. Joint UAS Group Classifications. Source: [8].

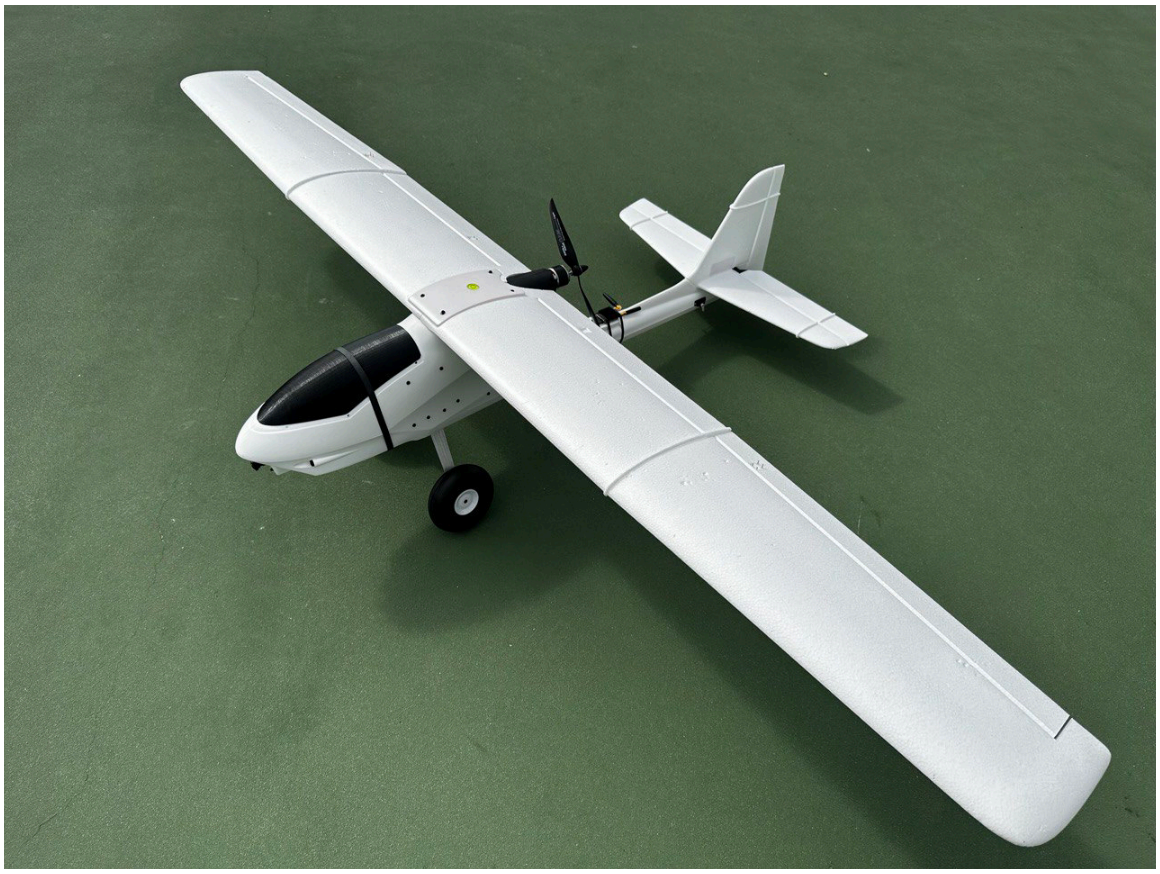


Figure 2. Photo of a Fully Assembled SIM-1 UAS

B. SIM-1 UAS HARDWARE SETUP

The hardware of the SIM-1 UAS can be broadly categorized into two categories: external parts and internal components. The external assembly comprises the fuselage, the lift generation and control surfaces, the tail assembly, the landing gear system and various auxiliary parts.

Internally, the SIM-1 UAS is enabled by key components including the autopilot flight controller, remote control (RC) radio transmitter/receiver, onboard computer, telemetry radio and lithium polymer (LiPo) battery.

1. Fuselage

The fuselage of the SIM-1 UAS is fabricated through injection molding using plastic. This fabrication technique and choice of material give lightweight and relative durability properties to the fuselage and the UAS as a whole. Notably, the fuselage was built with mounting points for the integration of other components. A partially disassembled SIM-1 UAS, with its main parts detached from the fuselage is shown in Figure 3.

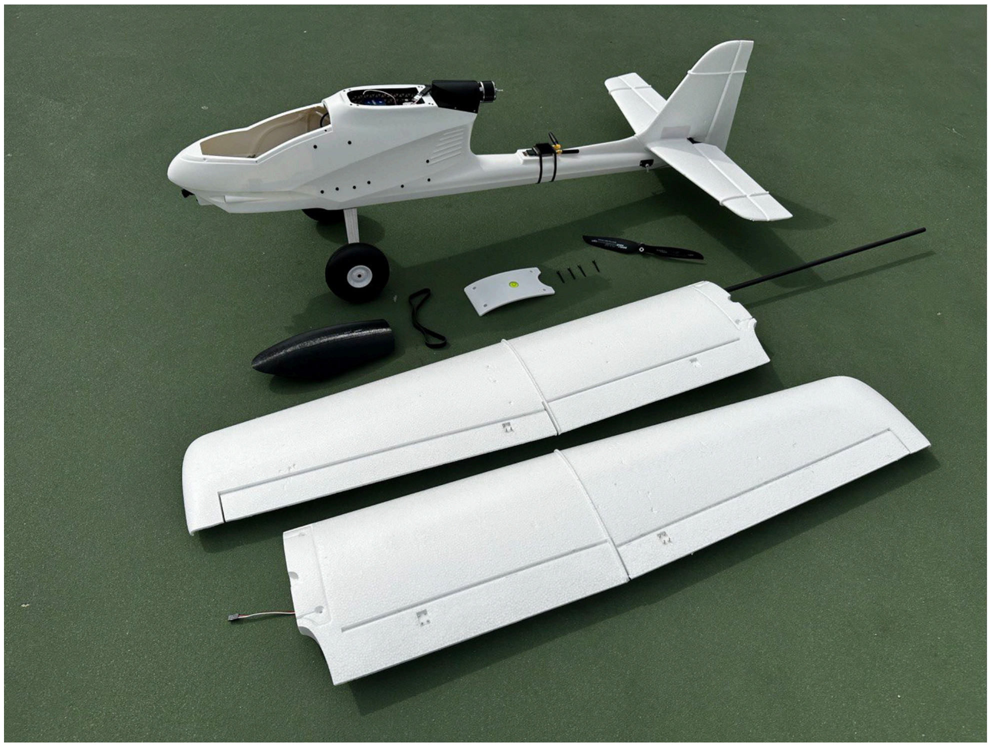


Figure 3. Photo of a Partially Disassembled SIM-1 UAS

2. The Lift Generation and Control Surfaces and the Tail Assembly

The primary lift generating surface of the SIM-1 UAS is its main wing. The main wing comprises two pieces, the port half and starboard half, and measures 6.5 ft (1.98 m) when assembled. This length is the default length supplied by Chesi UAS Solutions. Chesi UAS Solutions also offers wings with a longer wingspan, which translates to a larger aspect ratio (AR), which is the ratio of the wingspan over the wing chord length. Generally, fixed-wing aircraft whose wings have higher AR, will generate more lift [9], thereby giving the aircraft better glide performance. However, the longer wingspan will also lead to reduced maneuverability due to the increased moment of inertia that the control surfaces must overcome. The main wing was molded from lightweight and durable expanded polyolefin (EPO) material. Prior to flight, the two main wing halves are assembled and positioned at the top of the fuselage, which is then secured using the wing cover and four screws at the corresponding mounting points, as depicted in Figure 4.

The tail assembly of the SIM-1 UAS uses a conventional design, comprising two horizontal stabilizers and a single vertical stabilizer. These stabilizers share the same material as the main wing.

The SIM-1 UAS is equipped with control surfaces typical of most fixed-wing aircraft. These include ailerons on the main wing, and elevators and rudders on the tail assembly to provide roll, pitch and yaw controls respectively. They are controlled by the flight control computer via actuators. These control surfaces can be seen in Figure 5 and Figure 6 respectively.

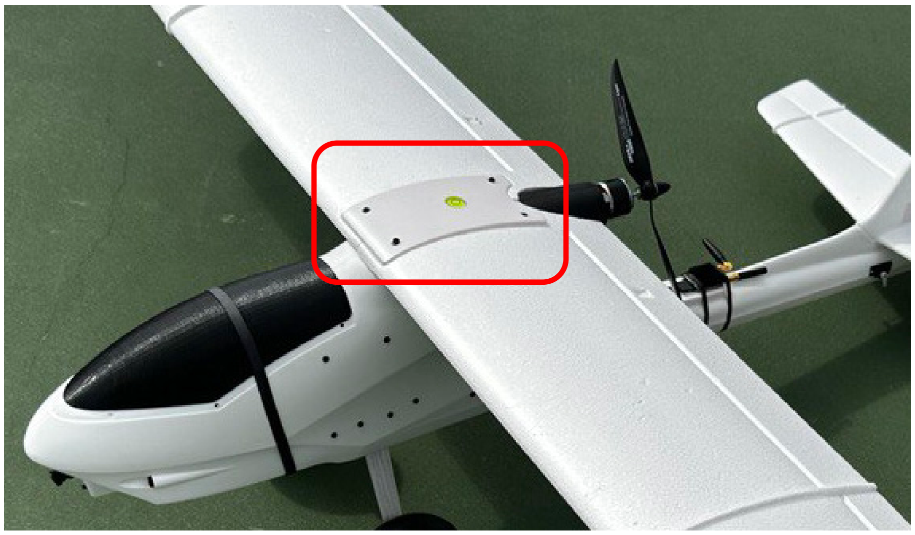


Figure 4. Photo of the SIM-1 UAS Wing Cover and Mounting Screws (Shown in Red Box)

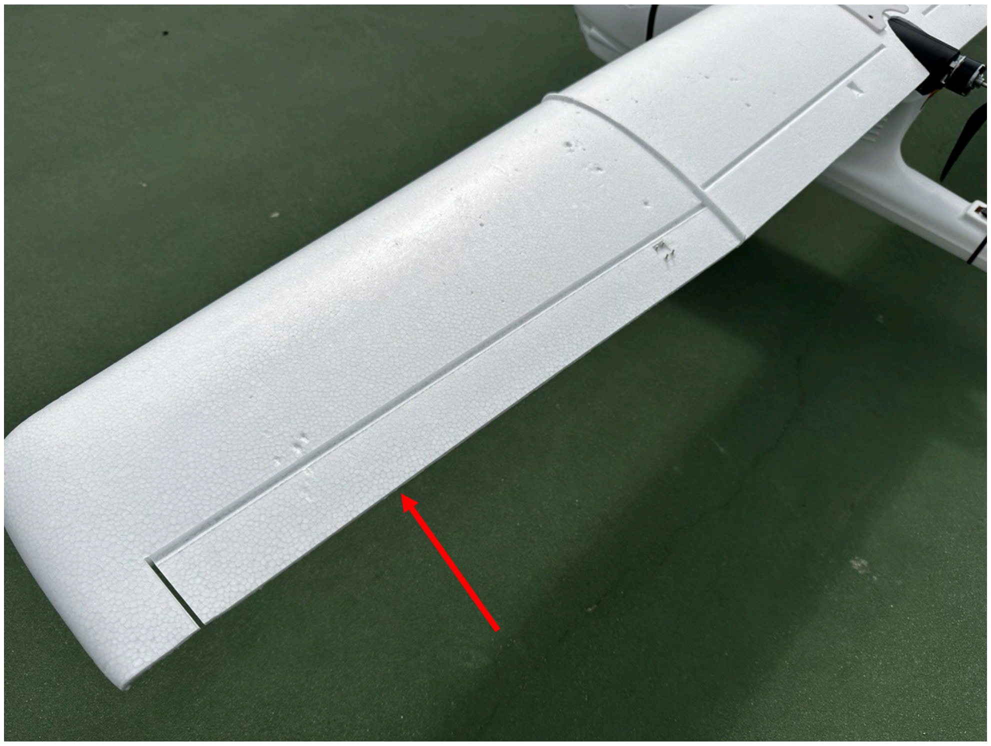


Figure 5. Photo of the SIM-1 UAS Aileron (Port Side) (Indicated by Red Arrow)

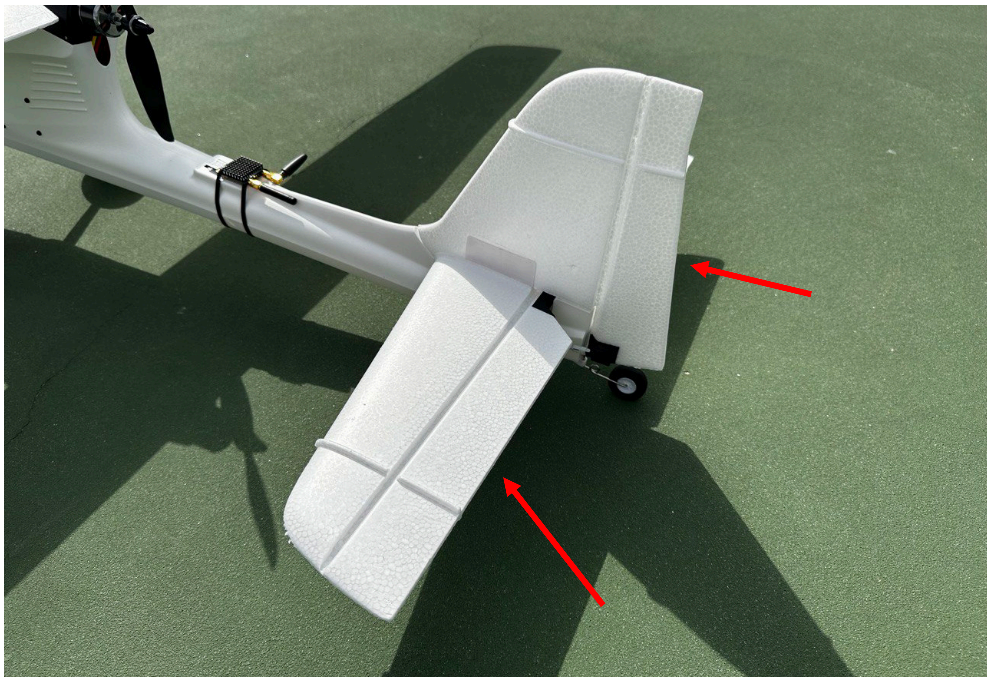


Figure 6. Photo of the SIM-1 Elevators and Rudder (Indicated by Red Arrows)

3. Landing Gear System

The landing gear system adopts a tail dragger configuration, also known as the conventional configuration. In this configuration, the main landing gears wheels are situated in front of the center of gravity, and a single smaller tailwheel is mounted on the tail. Schmidt highlighted one advantage of the tail dragger landing gear configuration – an overall reduction in weight due to a relatively compact tail gear [10]. Such weight saving is especially important to a small UAS like the SIM-1 UAS, where power is limited.

Like other tail dragger aircraft, the SIM-1 UAS rests on the ground with a pitch up attitude. This gives the SIM-1 UAS more clearance between the front fuselage and the ground and makes the UAS and the front-mounted payload less susceptible to damage from ground obstacles.

However, the tail dragger landing gear configuration poses challenges when the aircraft is landing in crosswind condition as compared to a tricycle configuration. Schmidt explained that the forces generated by the main wheel cause a destabilizing moment in the tail dragger configuration [10]. Figure 7 illustrates the phenomenon.

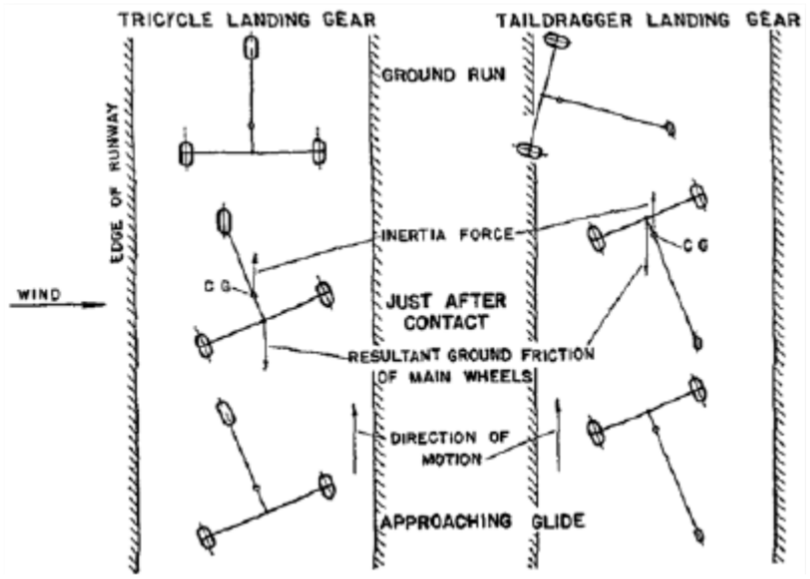


Figure 7. Stabilization and Destabilization during Crosswind Landing for the Tricycle and Taildragger Landing Gear. Source: [10].

4. Other Parts

Other notable parts are the payload, the propellers, and the cockpit cover. The front-mounted payload is a camera system (model: 8MP HDR MIPI CSI-2) from e-con Systems, and is capable of capturing 4K resolution images [11]. Figure 8 shows the front-mounted camera payload [11].

The propeller is a 2-bladed fixed pitch propeller, mounted to the electrical motor, behind the cockpit and the main wing. Figure 9 shows a photo of the propeller. The electrical motor turns the propeller, driving the SIM-1 UAS forward. During takeoff, when the forward speed exceeds a certain threshold, enough lift will be generated for the UAS to leave the ground.

Lastly, the cockpit cover (shown in Figure 10) is fitted over the cockpit compartment to help the fuselage maintain an aerodynamic profile and protect the components that are housed within.

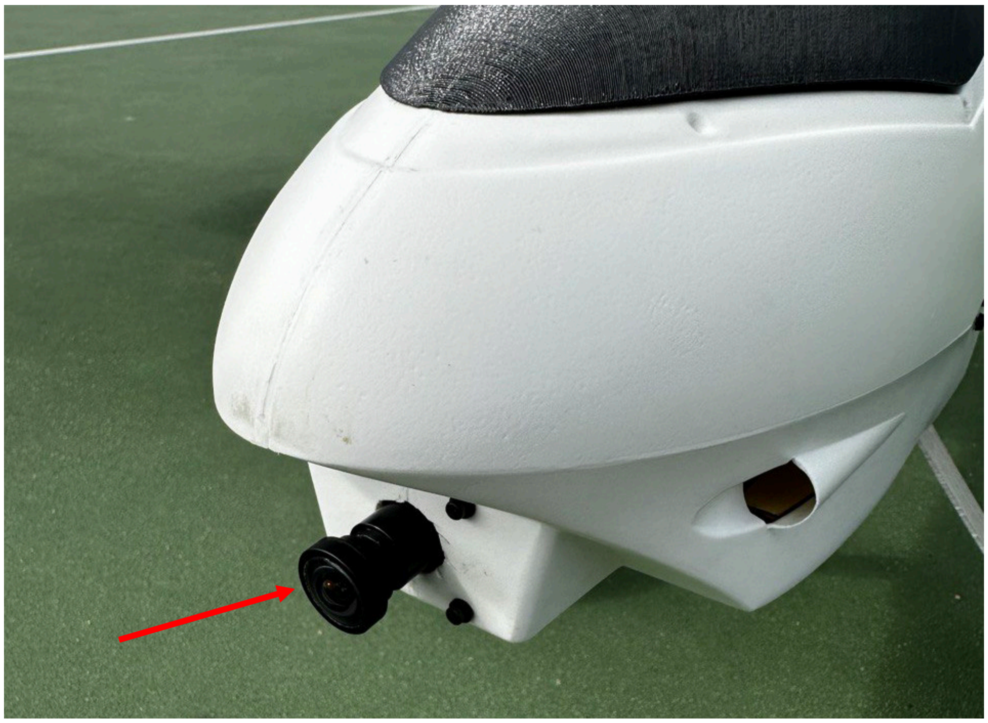


Figure 8. Photo of SIM-1 Front-Mounted Payload (Indicated by Red Arrow)

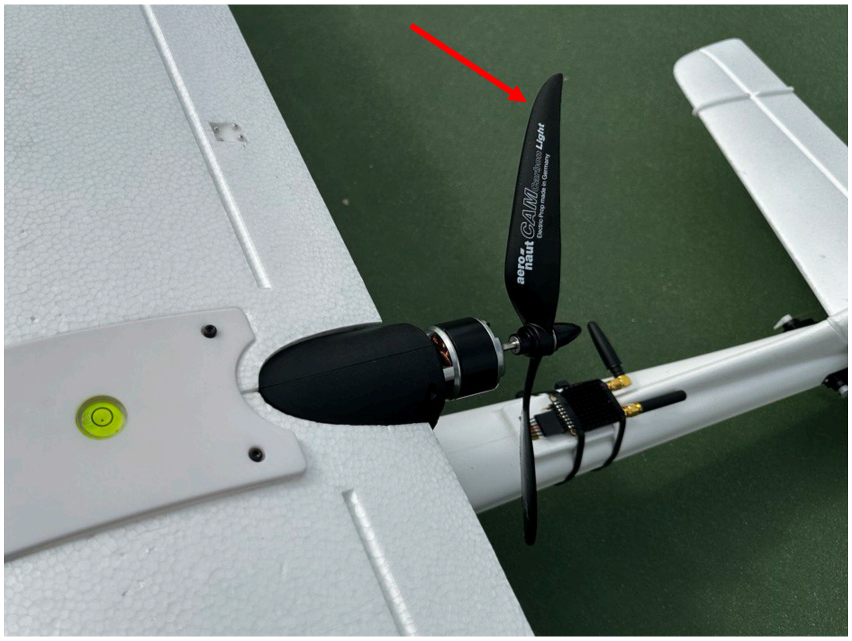


Figure 9. Photo of SIM-1 Propeller (Indicated by Red Arrow)

Figure 10. Picture of SIM-1 Cockpit Cover

5. Internally Installed Components

The key components installed in the cockpit compartment of the SIM-1 UAS are the autopilot flight controller, RC radio transmitter/receiver, onboard computer, telemetry radio and LiPo Battery.

a. Autopilot Flight Controller

The autopilot flight controller installed in SIM-1 UAS is the PixHawk 2.1 Blue Cube (shown in Figure 11). It is manufactured by CubePilot in the USA. The PixHawk 2.1 Blue Cube functions as the brain and sensors of the UAS: sensing the SIM-1 UAS's attitude and altitude and flying the UAS by sending appropriate commands to the motors and actuators. The sensing is done through temperature-controlled and shock absorbing inertial measurement units (IMUs) which contain accelerometers, gyroscope sensors and barometer, to sense accelerations, rate of rotation and altitude respectively. Together with the base board, the PixHawk 2.1 Blue Cube provides interfaces with other hardware such as antenna, servos, and telemetry modules. Based on the product specifications, the PixHawk 2.1 Blue Cube supports different flight modes such as loiter, altitude hold, autonomous with waypoints, etc, for a wide variety of UAS. It also supports different ground control station (GCS) software such as QGroundControl [12]. These are useful for developmental flight tests.

Figure 11. PixHawk 2.1 Blue Cube Autopilot Flight Controller. Source: [12].

b. RC Radio Transmitter/Receiver

For remote control operation, the SIM-1 UAS utilizes a RC radio receiver (model: Graupner GR-16 8CH 2.4GHz HoTT) (Figure 12) in the cockpit compartment. This receiver functions to receive inputs from the RC radio transmitter (model: Graupner mz-

12 PRO 12) (Figure 13) on the ground. The transmitter functions like a game controller, reading the stick inputs from the user, and sending the signals to the receiver via radio frequency (RF) and passing this information to the flight controller to make SIM-1 fly correspondingly [15]. This receiver/transmitter set is designed to work within line-of-sight range.

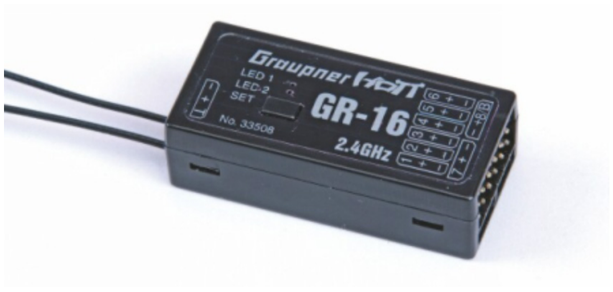


Figure 12. RC Radio Receiver (Model: Graupner GR-16 8CH 2.4GHz HoTT). Source: [13].

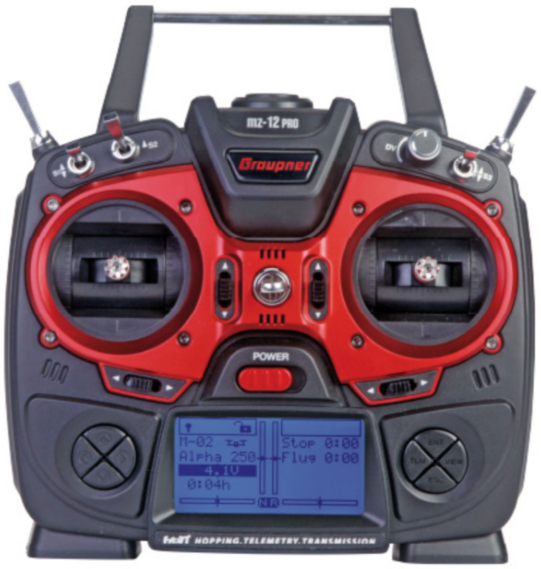


Figure 13. RC Radio Transmitter (Model: Graupner mz-12 PRO 12). Source: [14]

c. *Onboard Computer*

The SIM-1 UAS comes installed with an onboard computer, the NVIDIA Jetson NANO developer kit (shown in Figure 14), that has an integrated 128-core Graphics Processing Unit (GPU), quad-core Central Processing Unit (CPU) and 4GB of memory. It provides dedicated processing power for edge computing so that more advanced tasks such as image classification and object detection can be done onboard the UAS [16].

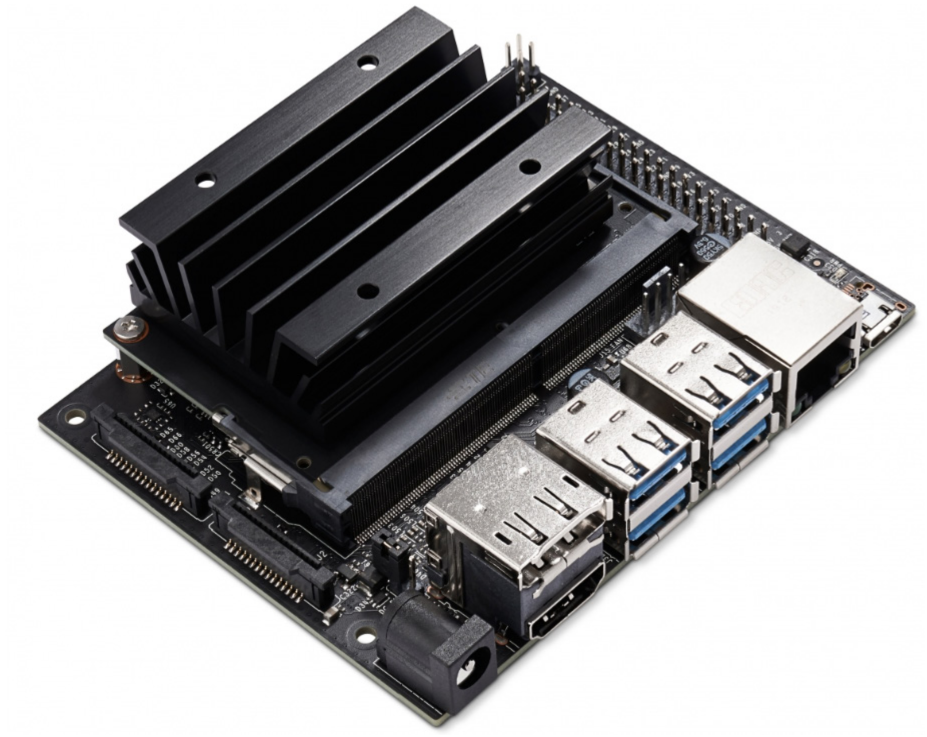


Figure 14. NVIDIA Jetson NANO Developer Kit. Source: [16].

d. *Telemetry Radio*

The SIM-1 UAS also includes a telemetry radio (model: RF900 Radio (Tx/Rx)) (shown in Figure 15). The matching set of radio and antenna is connected to the GCS via an USB port. The main function of the telemetry radio is for two-way communication between the UAS and the GCS, so that the real-time flight data can be transmitted from the UAS to the GCS for display, monitoring and planning. Based on the product specifications, the telemetry radios have a range of 9.3 to 24.9 miles (15 to 40 km) [17].

Figure 15. Telemetry Radio. Source: [17].

e. Lithium Polymer (LiPo) Battery

The components in the SIM-1 UAS are powered by a 5000 mAh LiPo battery (model: TP5000-4SE55) (see Figure 16). A fully charged battery will provide the SIM-1 UAS an endurance of about 45 minutes with payloads switched on. Based on Chesi UAS Solutions, options for using larger capacity batteries are possible. This would provide the SIM-1 UAS with more endurance. However, a larger battery also adds additional weight to the UAS, and is only advisable for more advanced users as a crash would result in greater impact damage to the fuselage and UAS structure.

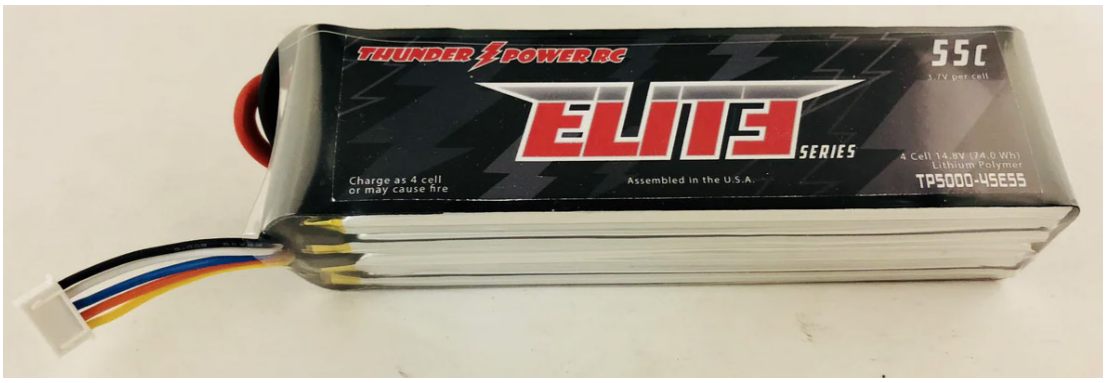


Figure 16. TP5000-ASE55 LiPo Battery. Source: [18].

C. GROUND CONTROL STATION

The SIM-1 GCS is an MSI gaming laptop with Intel i7 CPU, and dedicated NVIDIA GPU (see Figure 17). The GCS uses the Ubuntu 20.04 LTS Linux Operating

System (OS). To facilitate communications with the SIM-1 UAS onboard computer, and other developmental algorithms, the GCS utilizes the Robot Operating System (ROS), QGroundControl GCS software, PX4 autopilot firmware and Gazebo simulation software.

The GCS communicates with the SIM-1 UAS via telemetry and RF signals. Through QGroundControl, some of the key flight parameters can be displayed in near real-time on the GCS. The GCS can be used to control the SIM-1 UAS, send new mission/flight plans to the UAS, etc. The GCS can also be used to provide live video footage from the UAS's camera.

The GCS comes with a game controller interface that can be used to simulate control inputs in a virtual environment such as Gazebo.

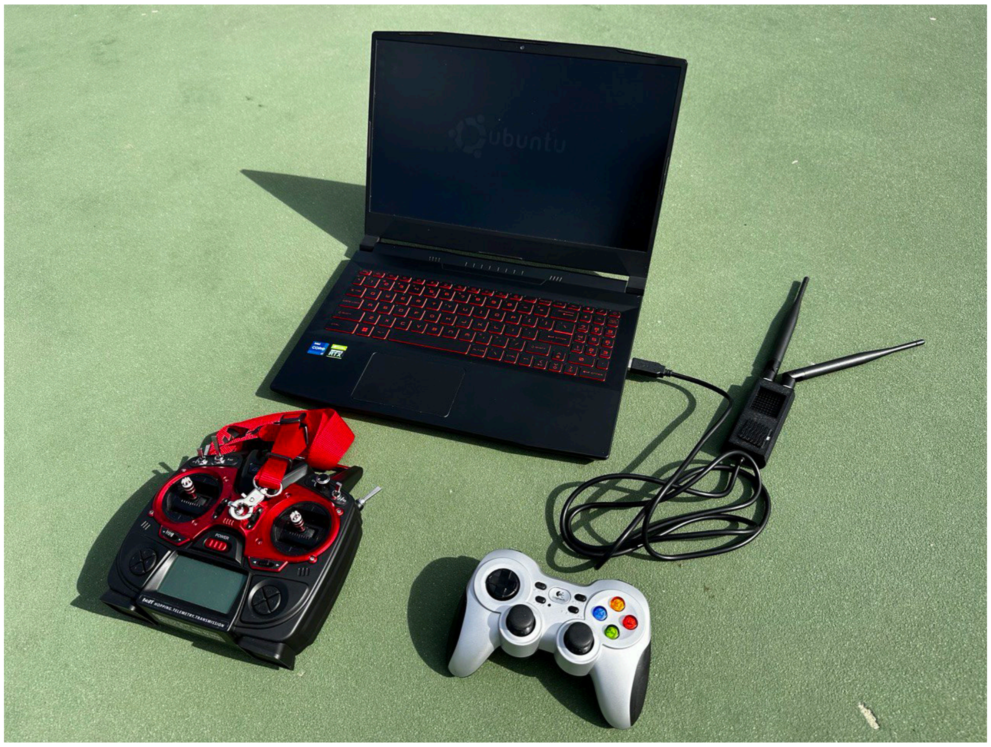


Figure 17. Photo of the GCS and Peripheral Interfaces

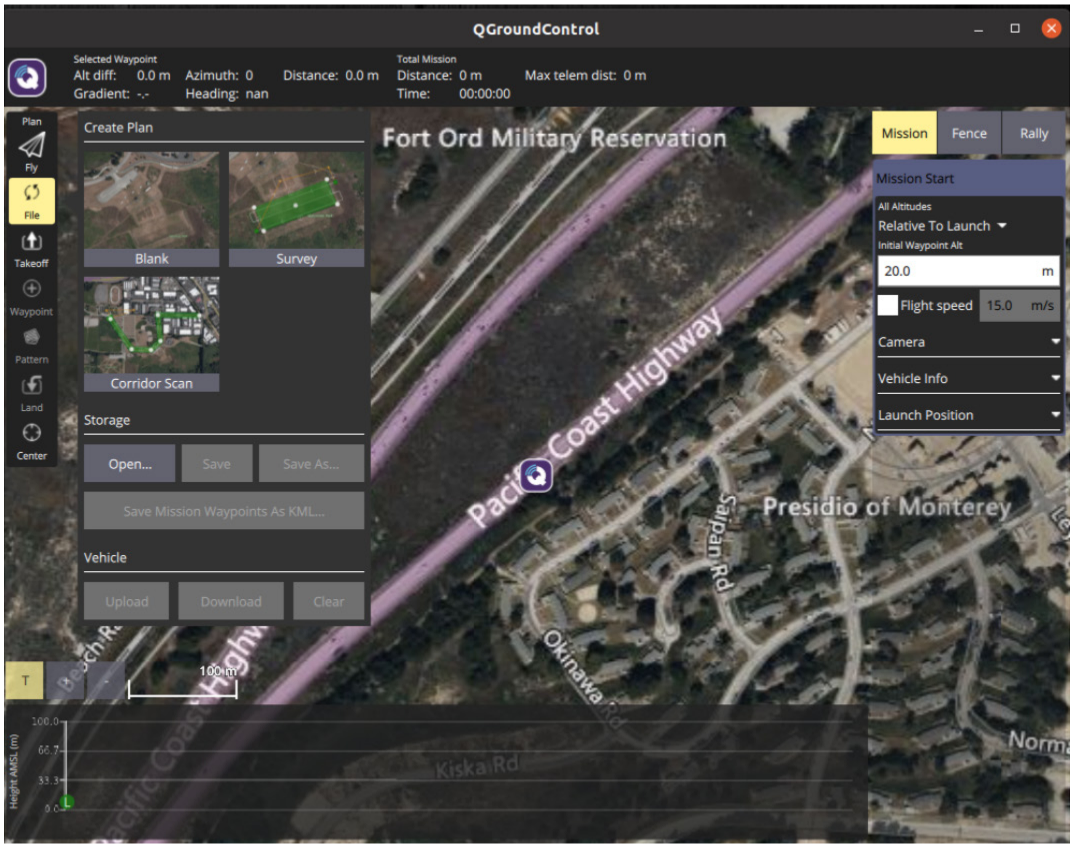
D. SOFTWARE AND SOFTWARE INTERFACE

1. ROS

The GCS is installed with ROS 1, an open-source software development kit, which functions as middleware that runs on top of the installed Linux OS. It provides a set of tools and libraries to help developers build robotics software applications. One of the key utilities of ROS is its communication messaging system, which is useful for intra and inter robot communications. This is done by sending messages between communication nodes via a publish/subscribe model [19]. The version installed is the ROS 1 Noetic Ninjemys, which was developed primarily for Ubuntu 20.04, the OS utilized by the GCS laptop.

2. QGroundControl

The GCS controls the SIM-1 UAS using QGroundControl (version 4.2.4), open-source GCS software that provides a graphical user interface (GUI) for UAS flight control and mission planning. See Figure 18 for a snapshot of the QGroundControl GUI. QGroundControl supports open-source autopilot firmware such as PX4 (installed in the SIM-1 GCS) and ArduPilot. Through QGroundControl, some of the key flight parameters can be displayed in near real-time on the GCS. One of the key utilities of QGroundControl is the ability for users to plan missions by defining waypoints and setting flight paths such as loiter patterns etc. During flight, QGroundControl can also be used to monitor flight parameters and alter the flight path as necessary [20].



Selected Waypoint
Alt diff: 0.0 m Azimuth: 0 Distance: 0.0 m
Gradient: -.- Heading: nan

Total Mission
Distance: 0 m Max telem dist: 0 m
Time: 00:00:00

All Altitudes: Relative To Launch
Initial Waypoint Alt: 20.0 m
Flight speed: 15.0 m/s

Figure 18. A Snapshot of the QGroundControl GUI

3. PX4 Autopilot Firmware

The SIM-1 GCS uses PX4 (version 1.14.0beta) open-source autopilot firmware. It is compatible with flight controller hardware such as the Pixhawk series of autopilot flight controllers which the SIM-1 UAS is installed with. One of the key utilities is that PX4 can work together with flight controllers to provide advanced flight controls such as stabilization, attitude control and autonomous flights [21].

4. Gazebo

The GCS is installed with Gazebo (version 11.13.0), open-source simulation software widely used for robotics development and testing. Gazebo employs a three-dimensional (3D) physics-based environment to simulate the behavior of robots interacting with realistic forces such as gravity and friction [22]. This is useful for testing robotic software to understand its behaviors before operating in an actual environment. In this thesis, the SIM-1 UAS was "flown" in Gazebo, before conducting actual flight tests.

5. Software Interfaces

In summary, the GCS is installed with ROS, QGroundControl GCS software, PX4 autopilot firmware and Gazebo simulation software, and collectively, they provide the necessary tools and avenues for basic flying and more advanced algorithm development using the SIM-1 UAS.

At the backbone of the GCS laptop is the Linux OS that manages the hardware and software resources within the laptop. ROS then sits on top of the Linux OS as a middleware by providing a communication messaging system that sends messages across nodes which are akin to 'software modules' according to Quigley et al. [23]. This enables other software (such as QGroundControl, PX4 and Gazebo) to perform their functions. Figure 19 illustrates the software interfaces among the software installed on the GCS. For example, the MAVROS package enables communications between ROS, MAVLink [1] -enabled autopilots, and MAVLink-enabled GCS, by translating ROS messages into MAVLink messages and vice versa. This allows the ROS nodes and the corresponding MAVLink enabled software to communicate with one another [24]. The ROS/Gazebo/QGroundControl integration with PX4 follows Figure 20, where the PX4 communicates with the Gazebo simulator to receive simulated sensor data from the virtual world and in response sends motor and actuator commands. PX4 then communicates with QGroundControl and an application programming interface (API) (ROS in this case) to send/receive data to/from the virtual environment [25].

[1] MAVLink is lightweight messaging protocol developed specifically for communications with drones and between onboard drone components. Data streams are sent / published as topics while configuration data such as the mission protocol or parameter protocol are point-to-point with retransmission [26].

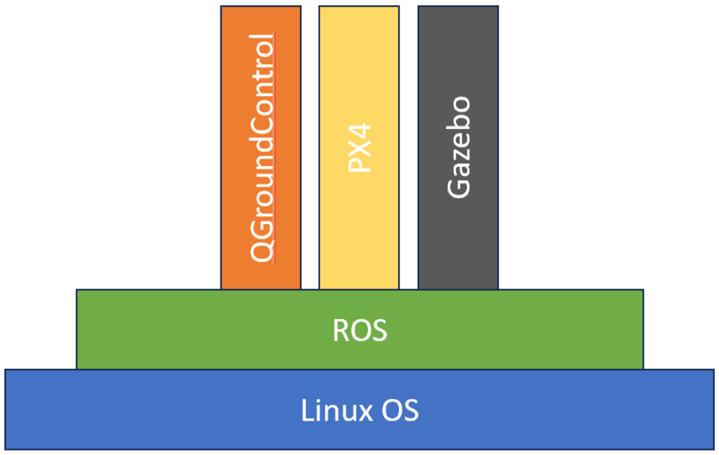


Figure 19. Software Interface in the GCS

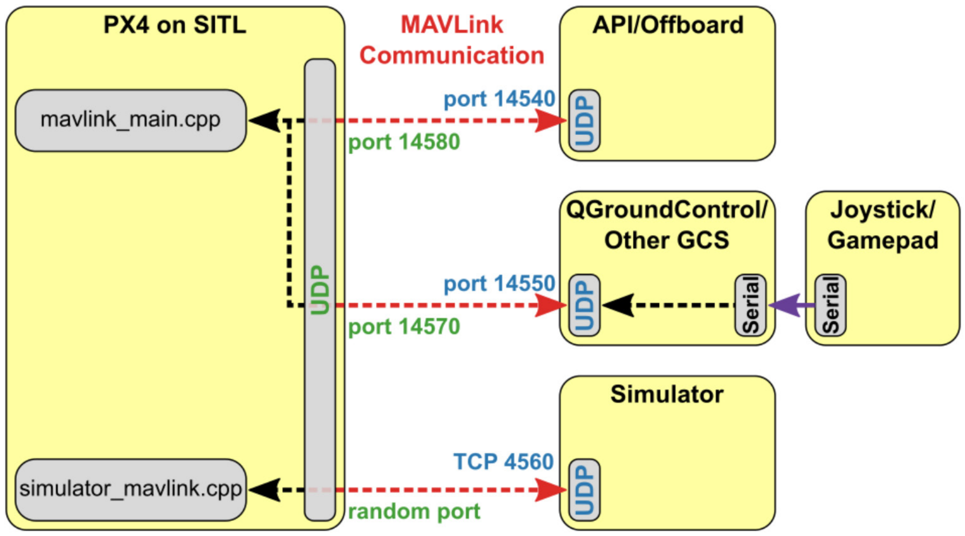


Figure 20. Communication Links between Software.

THIS PAGE INTENTIONALLY LEFT BLANK

III. DEVELOPMENT AND INTEGRATION OF BASIC FLIGHT PATH PLANNING AND CONTROL

A. DEVELOPMENT AND INTEGRATION OF BASIC CONTROLS

Developing and integrating a control algorithm into the SIM-1 UAS can be done via various approaches. This thesis studied a method which involves creating a control algorithm using MATLAB/Simulink. From this algorithm, a corresponding set of C++ codes can then be generated by Simulink. These codes are then integrated into a ROS node and executed. This method is useful for students to prototype, simulate and refine algorithms as part of their coursework and/or laboratory exercises, before proceeding to actual flight tests.

Alternatively, control algorithms can also be directly coded using programming languages such as C++ or Python. These algorithms can then be incorporated into a ROS node for utilization within the ROS environment. Such approaches are detailed in ROS.org tutorials [27], [28].

As a proof of concept, the following steps were carried out in this thesis:

1. Create a basic Simulink model and control algorithm
2. Generate C++ code from Simulink
3. Incorporate the C++ code into ROS node
4. Test and validate concept

1. Create a Basic Simulink Model and Control Algorithm

A basic Simulink model (Figure 21) was created by Chesi UAS Solutions for use on the SIM-1 UAS, and the related files can be found in GitLab via https://gitlab.nps.edu/nps_sim_uas/nps-drones/-/tree/master/nodes/px4_simulink_mavros/src/matlab_simulink [29]. Based on the Simulink file, the model consists of two main blocks, the "Simulator_Sub" block and the "GNC_Sub" block. The "Simulator_Sub" block is used to simulate the system dynamics via a simplified 3 Degree of Freedom (3DOF) model, where the UAS is represented as a point mass. The "GNC_Sub" block contains two further sub-blocks, “guidance_sub” and “controller_sub” (Figure 22), which does the Guidance and Control tasks for the UAS, respectively. These tasks are achieved by referencing the UAS’s current position and velocity vectors, and a matrix of waypoints, to determine the next waypoint that the UAS should head towards, and calculate the amount of control force required. Proportional and Derivative controllers are implemented in the “controller_sub” sub-block. Navigation is not required in this thesis as the UAS’s positions are assumed to be known. However, for example in a global positioning system (GPS) denied environment, this assumption would become invalid and an alternative form of navigation would be required. The flag_detect field is set up to enable object detection as described in Chapter V.

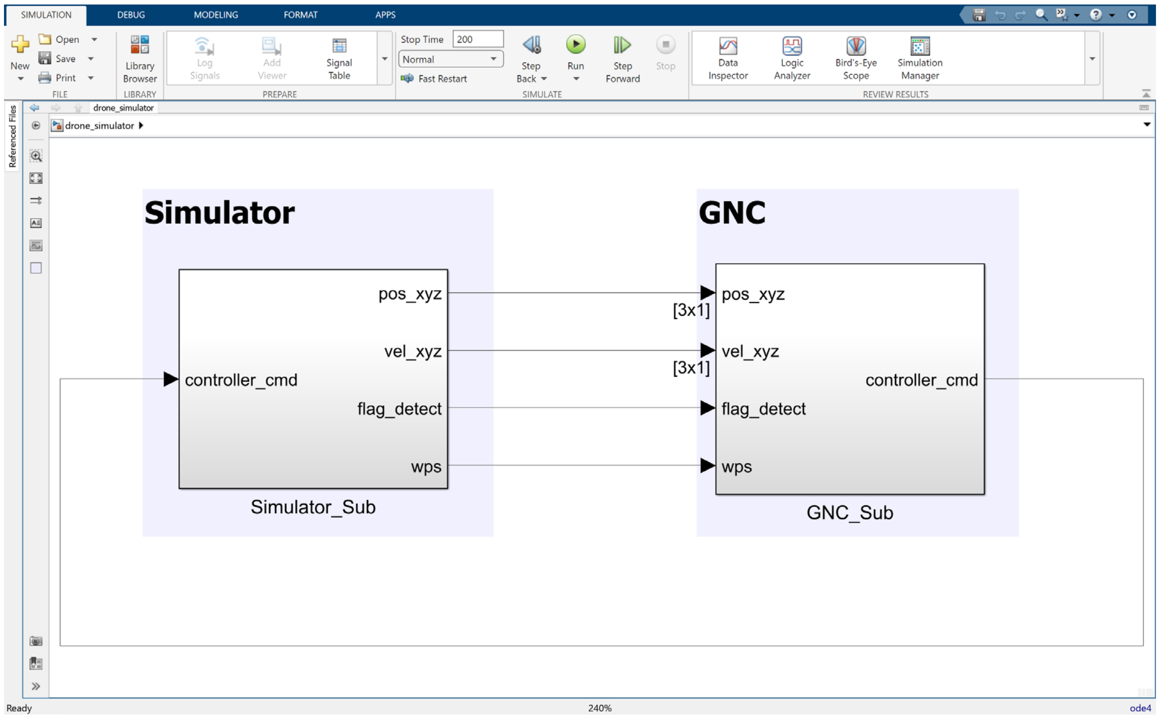


Figure 21. Simulink Model for SIM-1 UAS

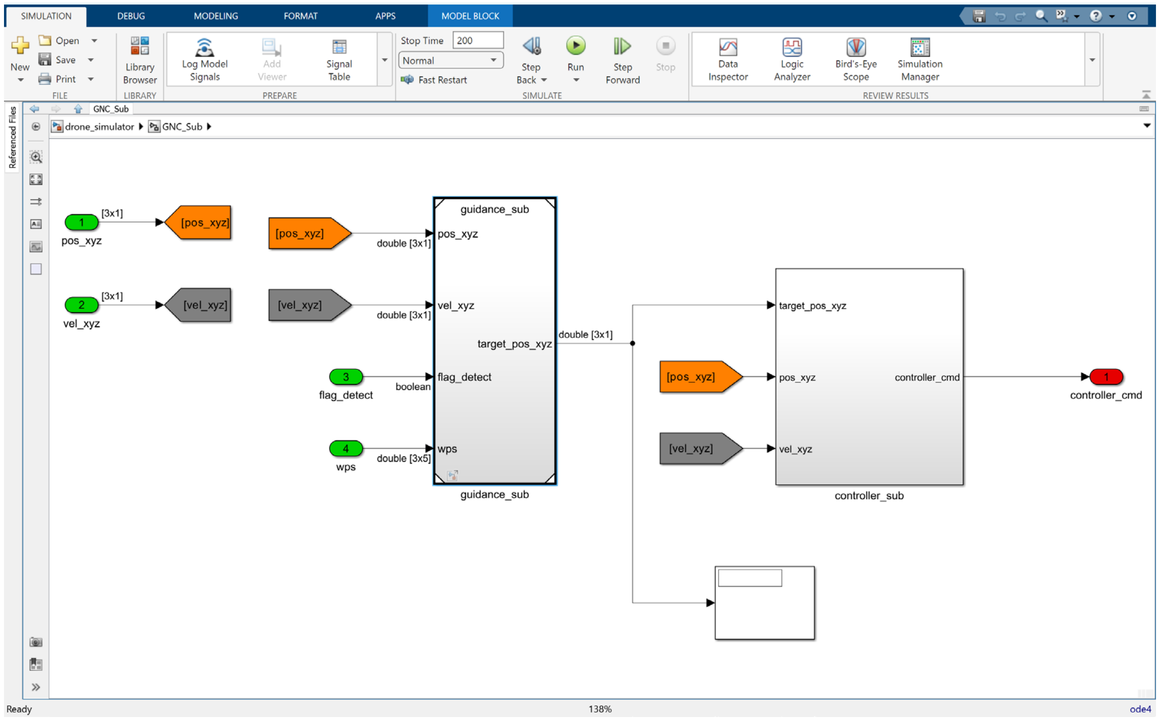


Figure 22. Inside GNC_Sub Block

2. Generate C++ Code from Simulink

Prior to the generation of the C++ codes, it is necessary to run the "guidance_simulink_init.m" file to initialize the key parameters such as the control gains, and the list of desired waypoints. This is also one of the ways to generate and upload a list of waypoints to the UAS for flight path planning. After initializing the parameters, it is crucial that the configurations in Table 1 are set correctly for the "guidance_sub" block. This will ensure that the code is properly generated for subsequent use. The menu to do so can be accessed via Modeling -> Model Settings.

Table 1. Simulink Code Generation Configurations

S/N	Configurations
1	Solver -> Solver selection -> Type: Fixed-step
2	Solver -> Solver selection-> Solver: ode4 (Runge-Kutta)
3	Hardware Implementation -> Device Type: x86-64 (Linux 64)
4	Code Generation -> Target selection -> System target file: ert.tlc[2]
5	Code Generation -> Target selection -> Language: C++
6	Code Generation -> Build Process -> Generate code only (checked)
7	Code Generation -> Report -> Create code generation report (checked)
8	Code Generation -> Report -> Open report automatically (checked)

After setting the configurations, the next step is to remove old code files (if any) by deleting the folder "guidance_sub_ert_rtw" in the same MATLAB folder. Lastly, use the Build button (shown in Figure 23) or the keyboard command Ctrl-B to generate the C++ codes. The Code Generation Report (Figure 24) can be used to verify that the code has been built correctly, e.g., by comparing the input ports and output ports with the Simulink model. In this case, the model was designed to have 5 waypoints for SIM-1 to follow, and the resulting dimensions of the waypoints (stated as wps in the model) is 3 by 5 as seen in Figure 24.

[2] Based on Mathworks, the ert.tlc is for embedded systems such as when hardware-in-the-loop testing is involved [30].

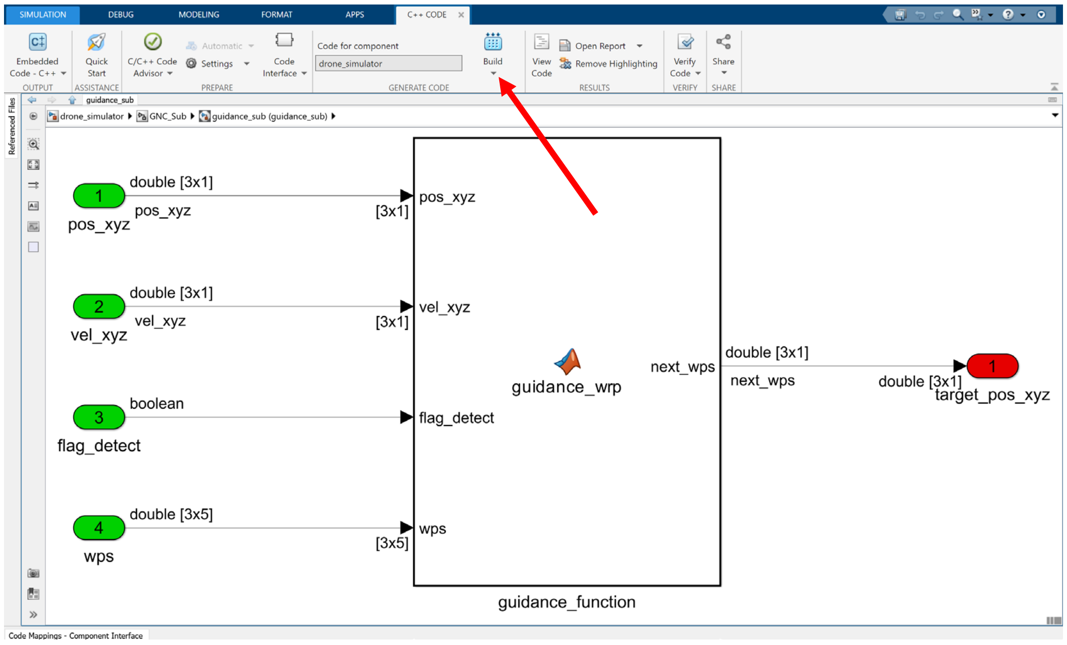
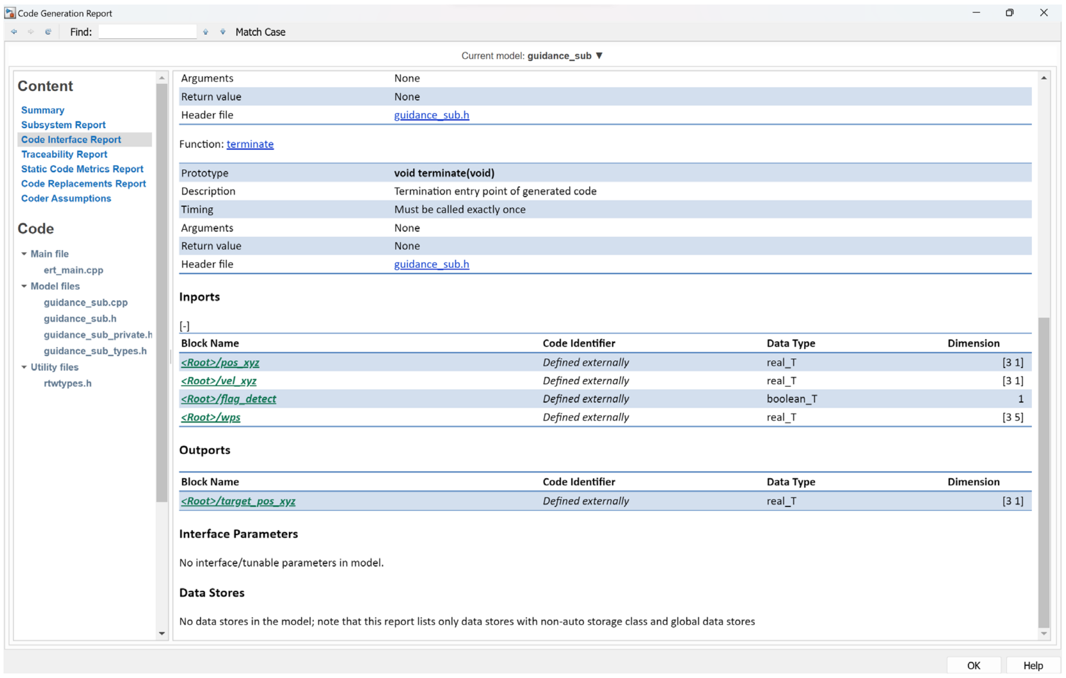


Figure 23. Screenshot of guidance_sub Block with Build Button (Indicated by Red Arrow)



Code Generation Report

Find: Match Case

Current model: **guidance_sub** ▼

Content

- Summary
- Subsystem Report
- Code Interface Report
- Traceability Report
- Static Code Metrics Report
- Code Replacements Report
- Coder Assumptions

Code

- Main file
  - ert_main.cpp
- Model files
  - guidance_sub.cpp
  - guidance_sub.h
  - guidance_sub_private.h
  - guidance_sub_types.h
- Utility files
  - rtwtypes.h

Arguments	None
Return value	None
Header file	guidance_sub.h

Function: terminate

Prototype	**void terminate(void)**
Description	Termination entry point of generated code
Timing	Must be called exactly once
Arguments	None
Return value	None
Header file	guidance_sub.h

Inports

[-]

Block Name	Code Identifier	Data Type	Dimension
<Root>/pos_xyz	*Defined externally*	real_T	[3 1]
<Root>/vel_xyz	*Defined externally*	real_T	[3 1]
<Root>/flag_detect	*Defined externally*	boolean_T	1
<Root>/wps	*Defined externally*	real_T	[3 5]

Outports

Block Name	Code Identifier	Data Type	Dimension
<Root>/target_pos_xyz	*Defined externally*	real_T	[3 1]

Interface Parameters

No interface/tunable parameters in model.

Data Stores

No data stores in the model; note that this report lists only data stores with non-auto storage class and global data stores

OK Help

Figure 24. Sample Code Generation Report

3. Incorporate C++ Code into ROS Node

The build process will generate a list of files (see left side of Figure 24) in the folder, “guidance_sub_ert_rtw” in the same MATLAB folder. Of importance to this step are the “guidance_sub.cpp” and “guidance_sub.h,” which contains the control algorithm, and data structure respectively [31]. These are to be copied into the “px4_simulink_mavros” package folder in the ROS catkin workspace previously created on the GCS. Specifically, “guidance_sub.cpp” is to be copied into the “catkin_ws/src/px4_simulink_mavros/src” folder, while “guidance_sub.h” is to be copied into the “catkin_ws/src/px4_simulink_mavros/include” folder. Next, navigate to the root directory of the workspace (“~/catkin_ws”) and run the “catkin build” command. This will build and compile the “px4_simulink_mavros” package.

Chesi UAS Solutions wrote a C++ code for a ROS node, “drone_controller.cpp” to link the guidance_sub files generated from Simulink to the PX4 flight controller. This node integrates the guidance and control algorithm from the Simulink model and communicates the commands to PX4 firmware on SIM-1 via MAVROS, which implements communications with the autopilot and GCS via MAVLink protocol. The specific code can be found in GitLab via https://gitlab.nps.edu/nps_sim_uas/nps-drones/-/tree/master/nodes/px4_simulink_mavros/src [29].

4. Test and Validate Concept

After incorporating the C++ codes in the ROS node per the previous step, the node can then be executed via the “rosrun px4_simulink_mavros px4_simulink_mavros_node” command. SIM-1 will then fly based on the commands from the Simulink model and the control algorithm. QGroundControl and Gazebo can be used to validate that this has been implemented successfully. Figure 25 shows a snapshot in QGroundControl where SIM-1 was diverted from its original flight path (after following a series of pre-planned waypoints, it had begun its landing approach) to head towards a single new waypoint (middle of the figure) where it began loitering. This diversion occurred as soon as the ROS node was executed, demonstrating that the guidance and control algorithm developed in Simulink had been successfully implemented in ROS.

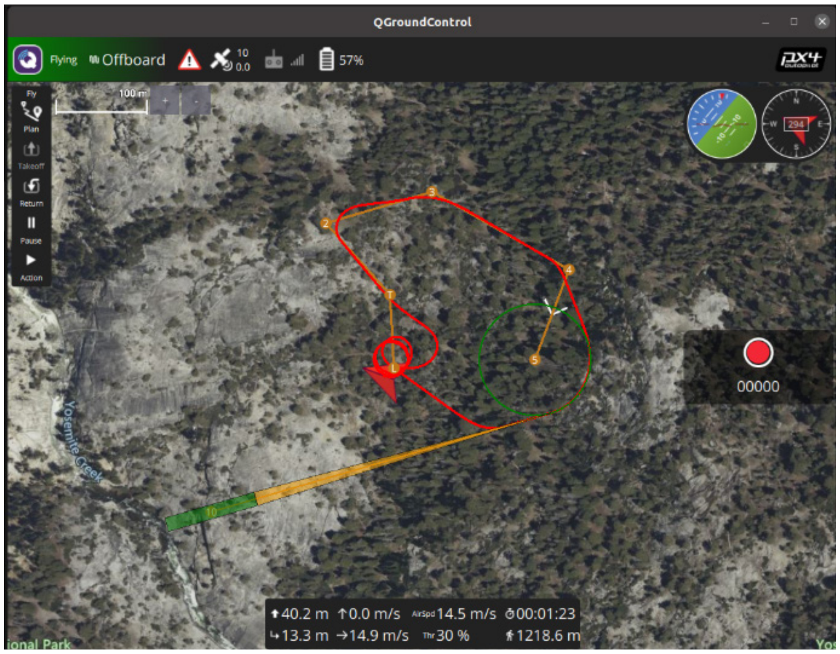


Figure 25. Diversion of SIM-1 from Its Original Flight Plan

B. PLANNING OF FLIGHT PATH

In this thesis, two methods of flight path planning were explored: basic flight path planning using QGroundControl software, and waypoint generation and following using Simulink-generated C++ codes described in Chapter III.A.

1. Basic Flight Path Planning Using QGroundControl

The first method of flight path planning utilizes the QGroundControl software interface to manually define waypoints, set flight parameters (such as altitude and speed), and plan a mission for the SIM-1 UAS. It is summarized below, and the QGroundControl user guide can be referred to should more complex planning be required [20].

1. Initialize programs
2. Create flight plan (take-off, waypoints and landing)

3. Set flight parameters

4. Save and upload flight plan

5. Fly SIM-1

a. Initialize Programs

First, the GCS software QGroundControl is launched, via the "~/git_repos/nps-drones/utilities/QGroundControl.AppImage" command, and QGroundControl will display the GCS location per Figure 26. Next, navigate to ~/catkin_ws/ in the terminal window, and run the "roslaunch flight_simulator single_plane_yosemite.launch" command. For convenience, Gazebo, PX4 and MAVROS have been included in the launch file (Figure 27 and Figure 28) so they can all be executed by a single command. Notably, all three of them are required for the software simulation to work. Once this is done, QGroundControl will update itself (Figure 29) to correspond to the Gazebo world location (Yosemite National Park[3] in this case) specified in the launch file. The Gazebo simulation view is shown in Figure 30.

[3] Yosemite National Park, was used arbitrarily as the area has been modelled by others for use in Gazebo [32].

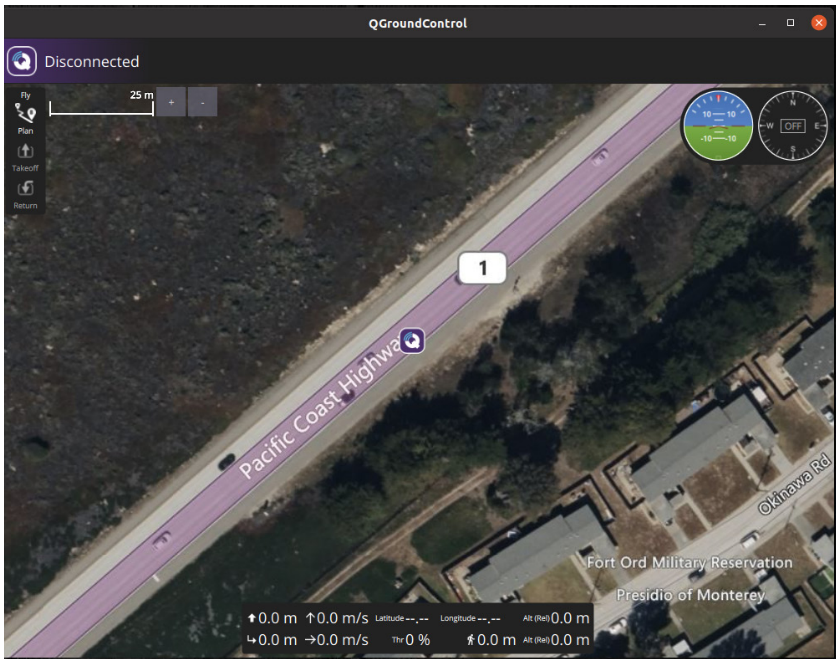
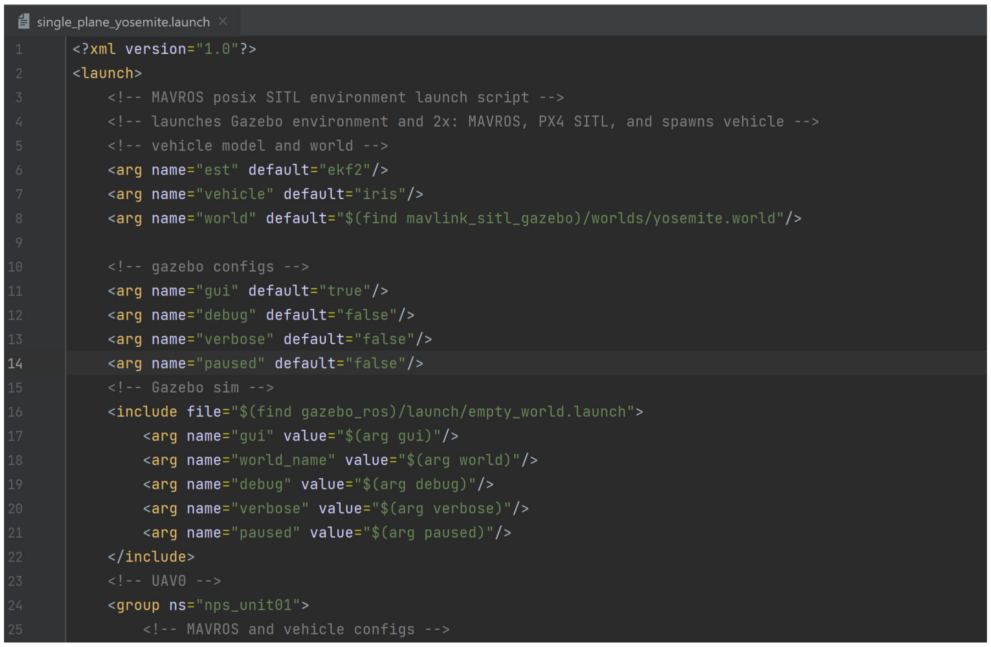

QGroundControl
Disconnected
Fly
Plan
Takeoff
Return
25 m
1
Pacific Coast Highway
Okinawa Rd
Fort Ord Military Reservation
Presidio of Monterey
0.0 m 0.0 m/s Latitude --.-- Longitude --.-- Alt (Rel) 0.0 m
0.0 m 0.0 m/s Thr 0 % 0.0 m Alt (Rel) 0.0 m

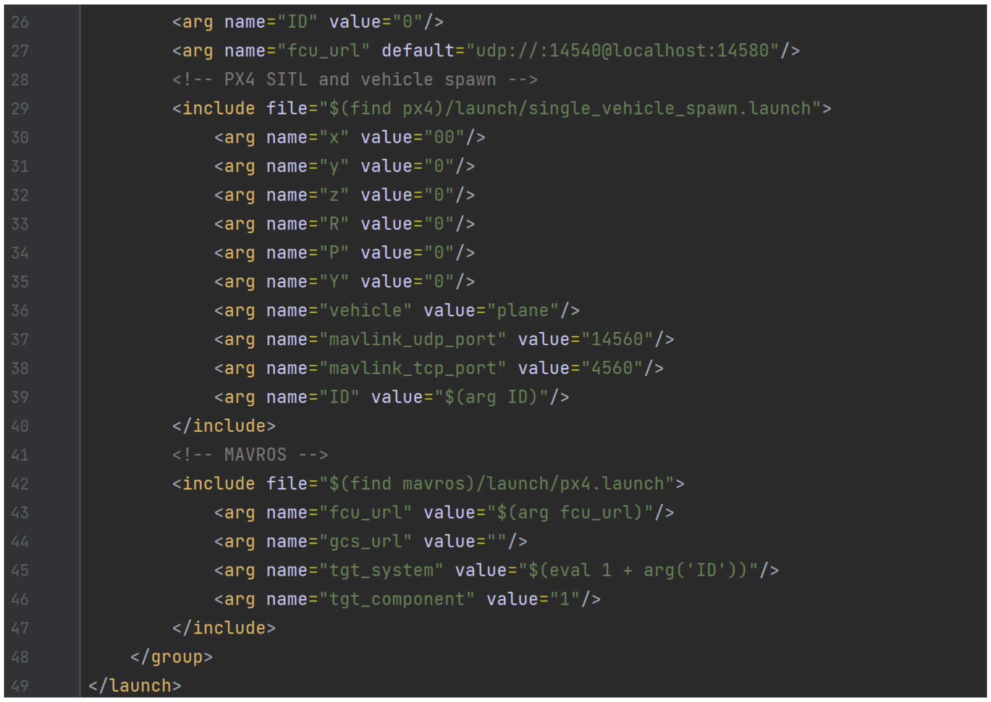
Figure 26. Initialize QGroundControl Software



single_plane_yosemite.launch

```
<?xml version="1.0"?>
<launch>
    <!-- MAVROS posix SITL environment launch script -->
    <!-- launches Gazebo environment and 2x: MAVROS, PX4 SITL, and spawns vehicle -->
    <!-- vehicle model and world -->
    <arg name="est" default="ekf2"/>
    <arg name="vehicle" default="iris"/>
    <arg name="world" default="$(find mavlink_sitl_gazebo)/worlds/yosemite.world"/>

    <!-- gazebo configs -->
    <arg name="gui" default="true"/>
    <arg name="debug" default="false"/>
    <arg name="verbose" default="false"/>
    <arg name="paused" default="false"/>
    <!-- Gazebo sim -->
    <include file="$(find gazebo_ros)/launch/empty_world.launch">
        <arg name="gui" value="$(arg gui)"/>
        <arg name="world_name" value="$(arg world)"/>
        <arg name="debug" value="$(arg debug)"/>
        <arg name="verbose" value="$(arg verbose)"/>
        <arg name="paused" value="$(arg paused)"/>
    </include>
    <!-- UAV0 -->
    <group ns="nps_unit01">
        <!-- MAVROS and vehicle configs -->
```

Figure 27. “single_plane_yosemite.launch” File (First Part)



```
        <arg name="ID" value="0"/>
        <arg name="fcu_url" default="udp://:14540@localhost:14580"/>
        <!-- PX4 SITL and vehicle spawn -->
        <include file="$(find px4)/launch/single_vehicle_spawn.launch">
            <arg name="x" value="00"/>
            <arg name="y" value="0"/>
            <arg name="z" value="0"/>
            <arg name="R" value="0"/>
            <arg name="P" value="0"/>
            <arg name="Y" value="0"/>
            <arg name="vehicle" value="plane"/>
            <arg name="mavlink_udp_port" value="14560"/>
            <arg name="mavlink_tcp_port" value="4560"/>
            <arg name="ID" value="$(arg ID)"/>
        </include>
        <!-- MAVROS -->
        <include file="$(find mavros)/launch/px4.launch">
            <arg name="fcu_url" value="$(arg fcu_url)"/>
            <arg name="gcs_url" value=""/>
            <arg name="tgt_system" value="$(eval 1 + arg('ID'))"/>
            <arg name="tgt_component" value="1"/>
        </include>
    </group>
</launch>
```

Figure 28. “single_plane_yosemite.launch” File (Second Part)

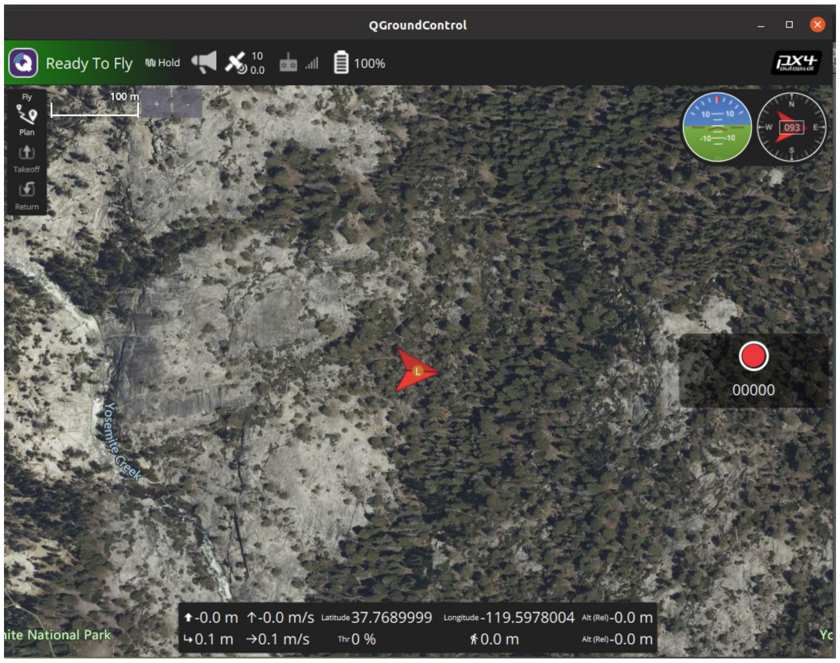

QGroundControl
Ready To Fly
Hold
10
0.0
100%
Fly
Plan
Takeoff
Return
100 m
093
00000
Yosemite Creek
ite National Park
-0.0 m -0.0 m/s Latitude 37.7689999 Longitude -119.5978004 Alt (Rel) -0.0 m
0.1 m 0.1 m/s Thr 0 % 0.0 m Alt (Rel) -0.0 m

Figure 29. Updated QGroundControl Showing SIM-1 at Yosemite National Park Area

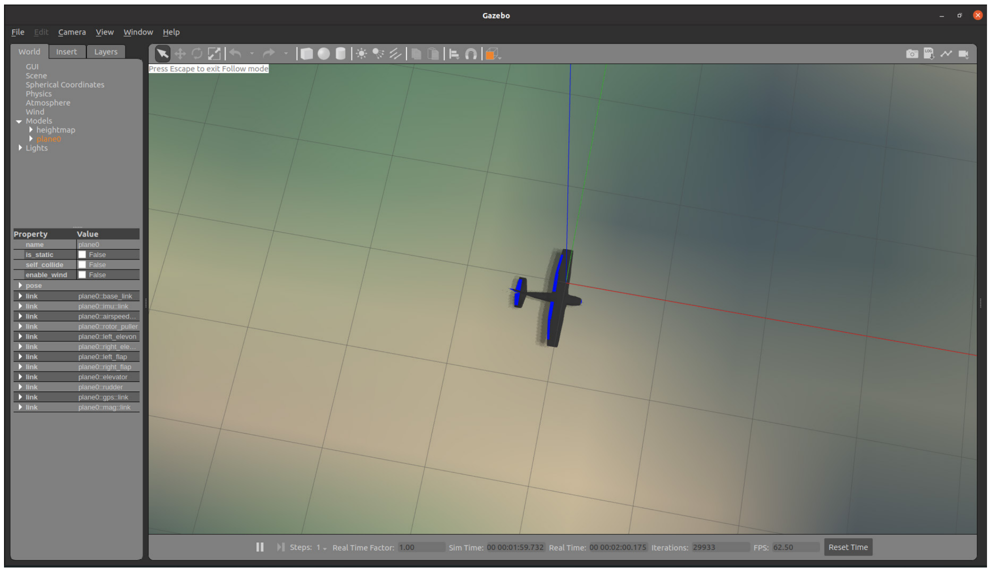
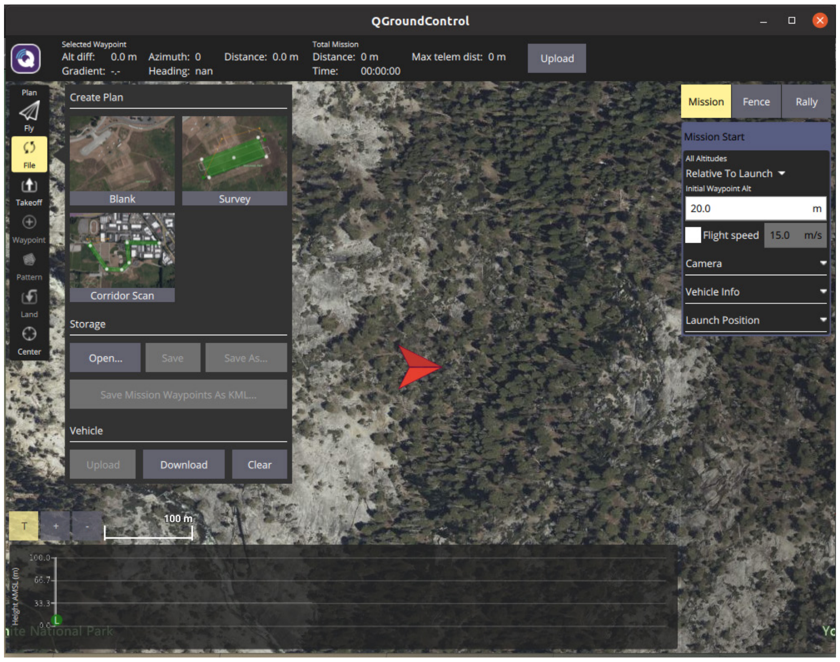


Figure 30. Gazebo Showing SIM-1 in Yosemite National Park Virtual World

b. Create Flight Plan (Take-off, Waypoints and Landing)

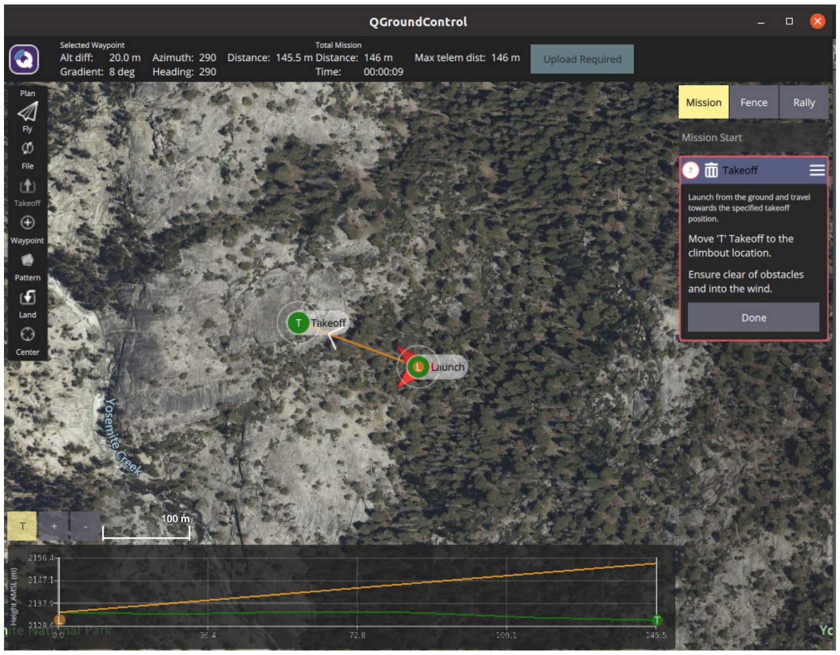
The first step to creating a flight plan is to use the “Plan” icon in the top left-hand corner in QGroundControl. This will open up various options such as loading an existing flight plan or designing a plan from scratch (Figure 31). To design a plan from scratch, click on “Blank” plan. The second step is to set a takeoff position (Figure 32), followed by adding the desired waypoints, and the landing point (Figure 33). It should be noted that the SIM-1 UAS (like other fixed-wing aircraft) will require a landing approach to land properly. This is made easy in QGroundControl which will automatically set a loiter waypoint and a landing approach. In such a setting, the fixed wing drone will loiter at the loiter waypoint until the landing conditions are met, before proceeding with its landing approach to the desired landing point.


Selected Waypoint
Alt diff: 0.0 m Azimuth: 0 Distance: 0.0 m
Gradient: -.- Heading: nan

Total Mission
Distance: 0 m Max telem dist: 0 m
Time: 00:00:00

Mission Start
All Altitudes: Relative To Launch
Initial Waypoint Alt: 20.0 m
Flight speed 15.0 m/s

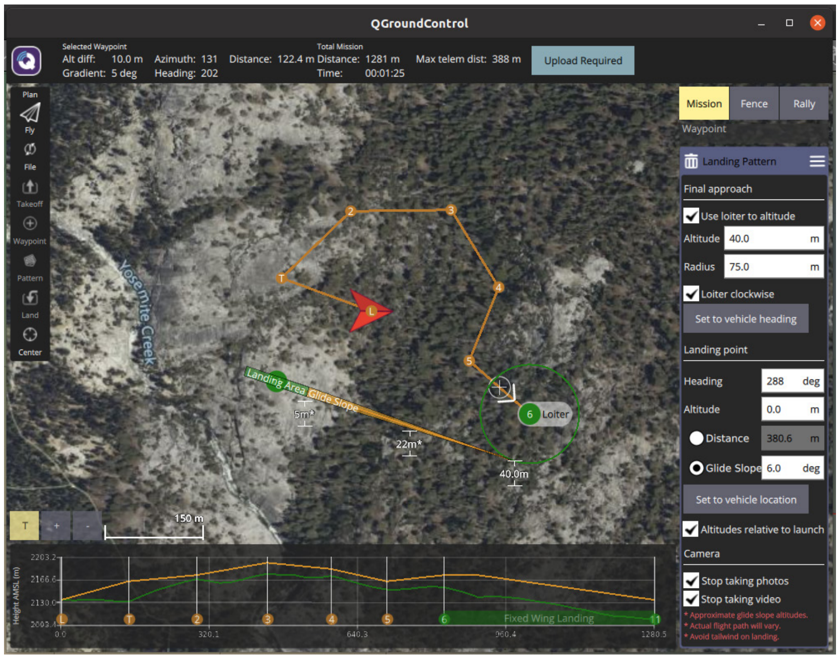
Figure 31. Creating a Flight Plan in QGroundControl


Selected Waypoint
Alt diff: 20.0 m Azimuth: 290 Distance: 145.5 m
Gradient: 8 deg Heading: 290

Total Mission
Distance: 146 m Max telem dist: 146 m
Time: 00:00:09

Takeoff
Launch from the ground and travel towards the specified takeoff position.
Move 'T' Takeoff to the climbout location.
Ensure clear of obstacles and into the wind.

Figure 32. Setting a Takeoff Point in QGroundControl



Selected Waypoint
Alt diff: 10.0 m Azimuth: 131 Distance: 122.4 m
Gradient: 5 deg Heading: 202

Total Mission
Distance: 1281 m Max telem dist: 388 m
Time: 00:01:25

Upload Required

Landing Pattern

Final approach

- [✓] Use loiter to altitude
- Altitude: 40.0 m
- Radius: 75.0 m
- [✓] Loiter clockwise
- Set to vehicle heading

Landing point

- Heading: 288 deg
- Altitude: 0.0 m
- Distance: 380.6 m
- [✓ Glide Slope]: 6.0 deg
- Set to vehicle location
- [✓] Altitudes relative to launch

Camera

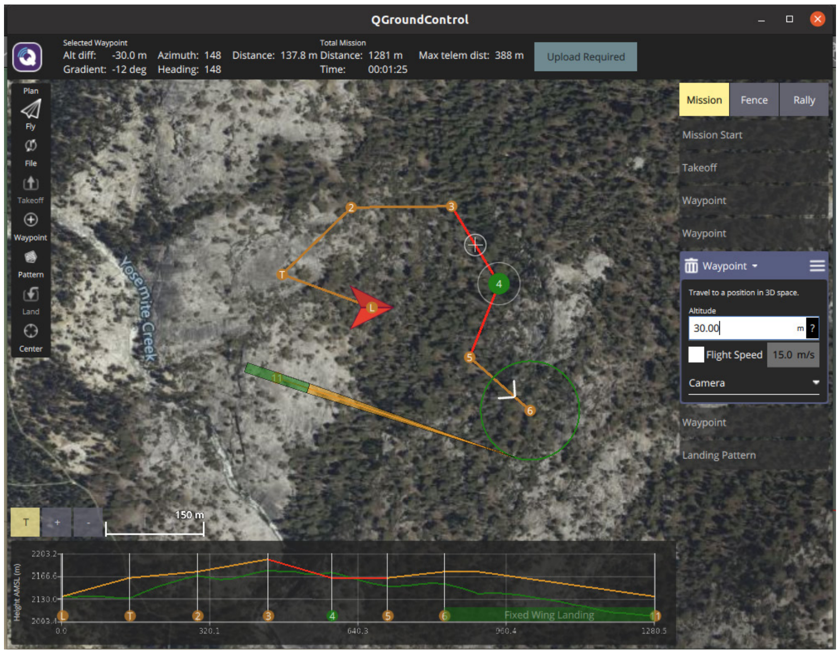
- [✓] Stop taking photos
- [✓] Stop taking video

* Approximate glide slope altitudes.
* Actual flight path will vary.
* Avoid tailwind on landing.

Figure 33. Setting Waypoints and Landing Point in QGroundControl

c. *Set Flight Parameters*

One of the most important aspects of designing a flight plan for the SIM-1 UAS (or any other fixed-wing aircraft) is to ensure that the flight path is clear of the terrain using the Height AMSL graph at the bottom of Figure 33. The key here is to set the flight parameters such as the altitude such that SIM-1 stays clear of the terrain. In such a condition, the entire flight path will be orange, otherwise the flight path will be red. An example is in Figure 34 where the altitude of waypoint 4 was set incorrectly, and the corresponding segments of the flight path are shown in red. Such unsafe flight plans are not allowed to be uploaded for use by the autopilot.



Selected Waypoint
Alt diff: -30.0 m Azimuth: 148 Distance: 137.8 m
Gradient: -12 deg Heading: 148

Total Mission
Distance: 1281 m Max telem dist: 388 m
Time: 00:01:25

Waypoint
Travel to a position in 3D space.
Altitude 30.00 m
Flight Speed 15.0 m/s

Figure 34. Invalid Flight Plan with Some Red Segments

d. Save and Upload Flight Plan

Once the flight plan is complete, it can be saved for subsequent use. This makes simulation convenient as the same flight plan can be used repeatably. To ensure that the flight plan is being used, it must be uploaded to the SIM-1 autopilot, by pressing "Upload Required." When successfully uploaded, "Done" will be shown near the top of the QGroundControl window (Figure 35), and SIM-1 will be ready to fly.

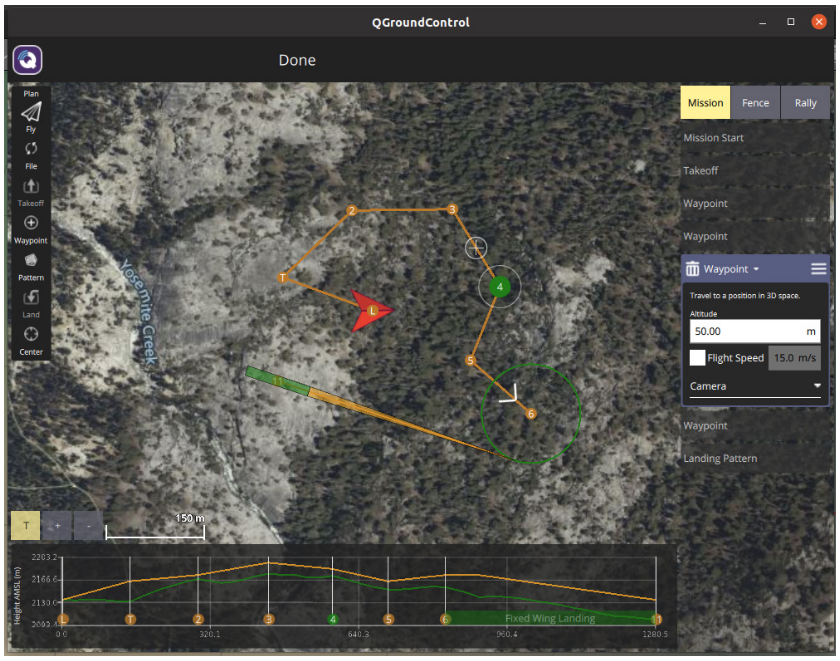
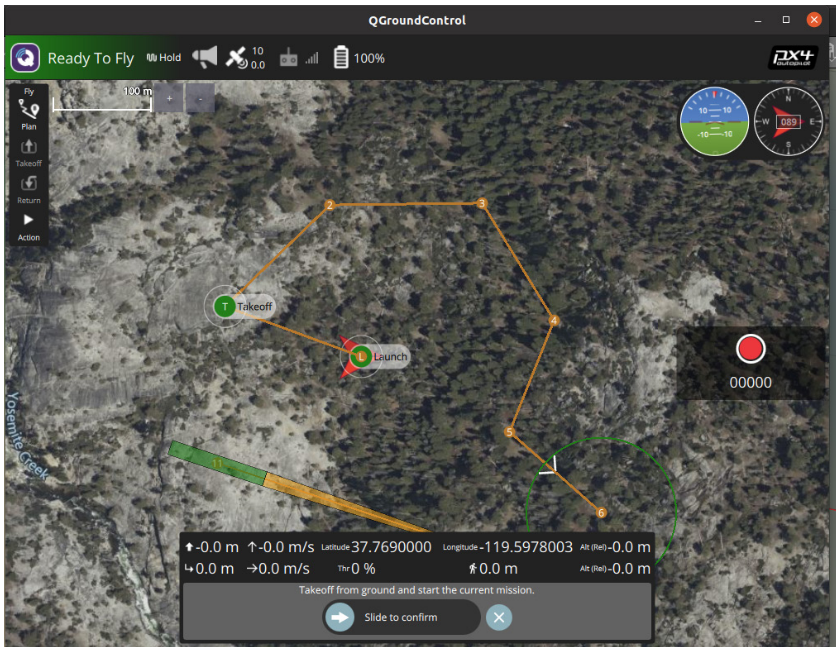


Figure 35. Flight Plan Successfully Uploaded to SIM-1, with "Done" Being Displayed at the Top

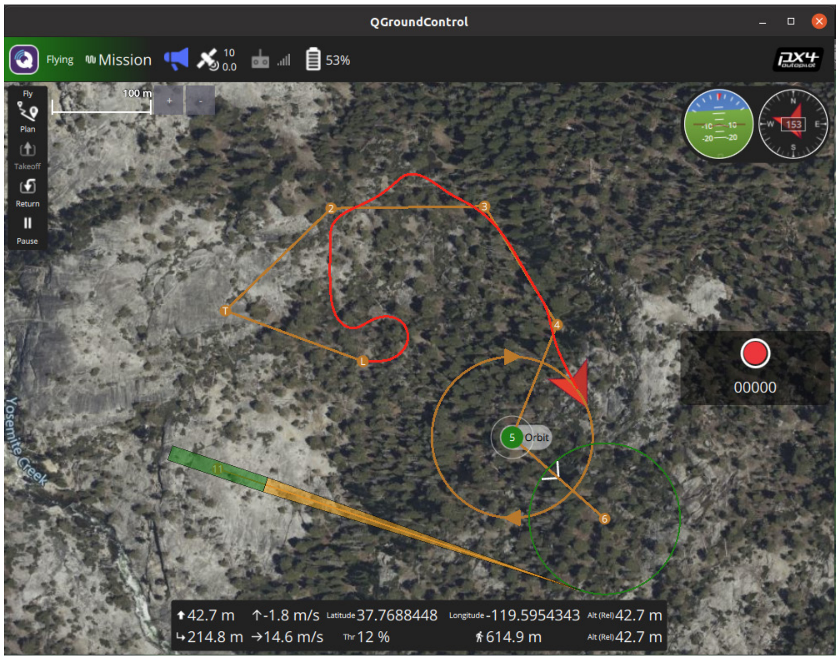
e. Fly SIM-1

Once the flight plan is uploaded successfully to SIM-1 (actual or simulated), it is ready to fly. Flying is done by dragging the cursor "Slide to confirm" (Figure 36). SIM-1 will take off from its starting position and head towards the various waypoints (Figure 37), before making a landing approach. The simulated SIM-1 in Gazebo will also fly accordingly (Figure 38).



-0.0 m -0.0 m/s Latitude 37.7690000 Longitude -119.5978003 Alt (Rel) -0.0 m
0.0 m 0.0 m/s Thr 0 % 0.0 m Alt (Rel) -0.0 m
Takeoff from ground and start the current mission.

Figure 36. “Slide to Confirm” to Fly SIM-1



42.7 m -1.8 m/s Latitude 37.7688448 Longitude -119.5954343 Alt (Rel) 42.7 m
214.8 m 14.6 m/s Thr 12 % 614.9 m Alt (Rel) 42.7 m

Figure 37. SIM-1 Flying According to the Flight Plan

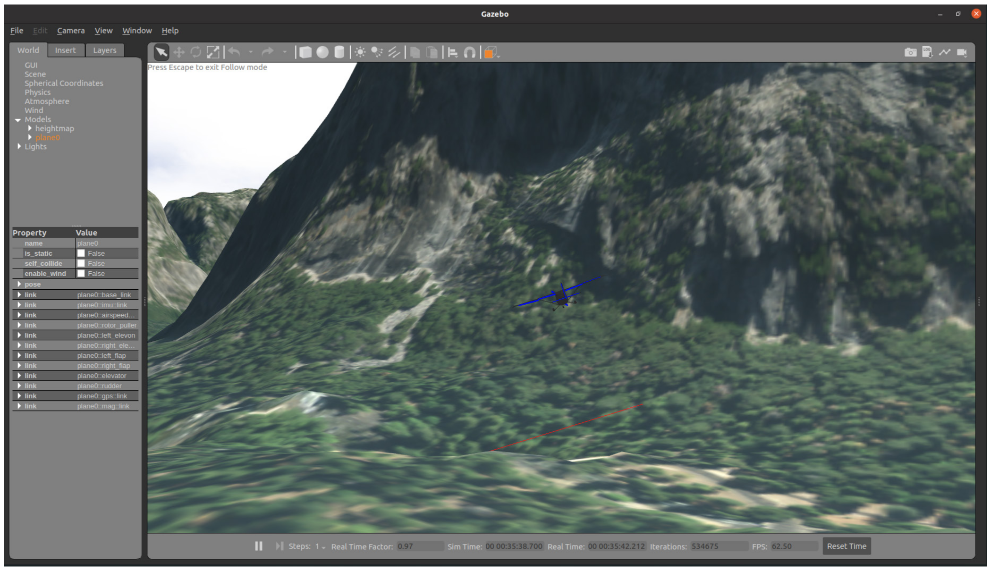


Figure 38. SIM-1 Flying in Gazebo

2. Waypoint Generation and Following Using Simulink Generated C++ Codes

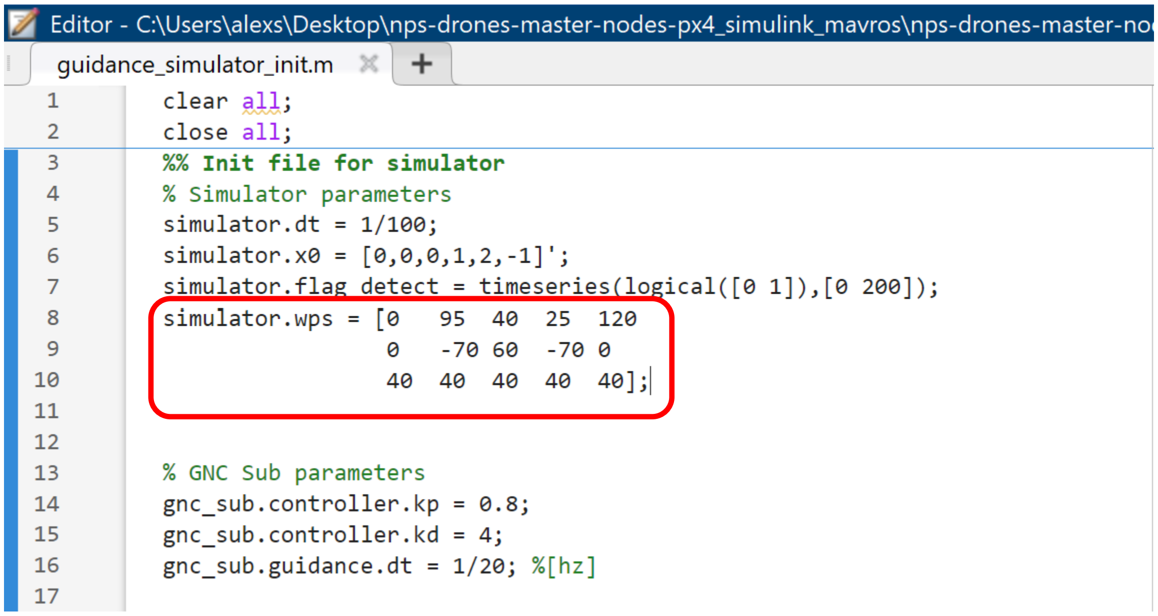
The second method of flight path planning that was explored in this thesis leverages the Simulink model that was described in Chapter III.A. This method also involves manually defining waypoints and uploading them to the SIM-1 UAS. As an example, a new flight path comprising 5 waypoints is used. Broadly, this flight path planning method involves the following steps:

1. Define waypoints and update MATLAB and Simulink files.
2. Generate and Incorporate C++ code from Simulink into ROS node.
3. Execute node.

a. Define Waypoints and Update MATLAB and Simulink Files

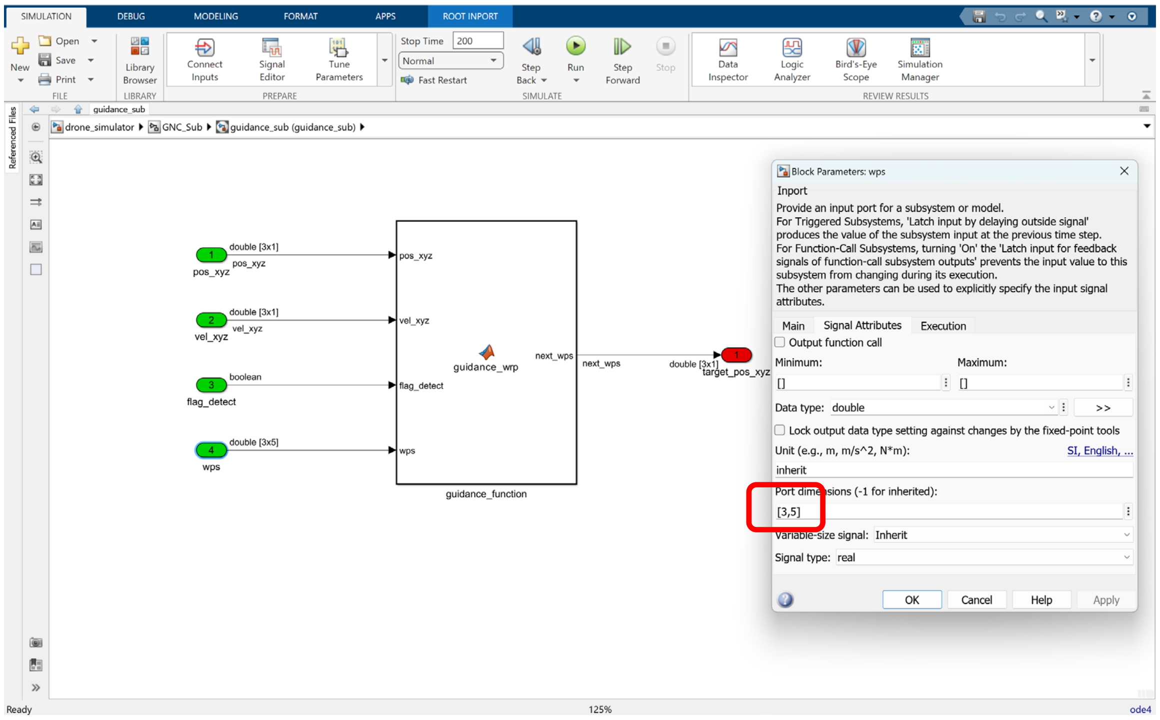
This method is based on the premise that the waypoint coordinates of the flight path are known. The coordinates can stem from operational needs or simulation and testing requirements. The coordinate system that is used in the Simulink model follows a Local Tangent Plane (LTP) coordinate system, specifically the East, North, Up (ENU) coordinate system. Of note, the origin of the coordinate system is the position of the

SIM-1 UAS when it connects with the GCS and becomes active in QGroundControl. The waypoint coordinates are set relative to this origin. The associated MATLAB and Simulink files are to be updated with these waypoints coordinates and the waypoint matrix dimensions. Specifically, the "guidance_simulator_init.m" is to be updated with the waypoint coordinates, as shown in Figure 39 for the five waypoints in this example. Each column represents the x, y and z coordinates of a waypoint. A total of five columns corresponds to five waypoints. The corresponding waypoint signal port dimensions in "guidance_sub" needs to be updated as well (Figure 40). These steps will ensure that the specified waypoint coordinates are built into the C++ code for subsequent use.


Editor - C:\Users\alexs\Desktop\nps-drones-master-nodes-px4_simulink_mavros\nps-drones-master-no
guidance_simulator_init.m

```
clear all;
close all;
%% Init file for simulator
% Simulator parameters
simulator.dt = 1/100;
simulator.x0 = [0,0,0,1,2,-1]';
simulator.flag_detect = timeseries(logical([0 1]),[0 200]);
simulator.wps = [0   95  40  25  120
                 0   -70 60  -70 0
                 40  40  40  40  40];


% GNC Sub parameters
gnc_sub.controller.kp = 0.8;
gnc_sub.controller.kd = 4;
gnc_sub.guidance.dt = 1/20; %[hz]
```

Figure 39. Updated "guidance_simulator_init.m" with the Waypoint Coordinates



Block Parameters: wps

Inport

Provide an input port for a subsystem or model.
For Triggered Subsystems, 'Latch input by delaying outside signal' produces the value of the subsystem input at the previous time step.
For Function-Call Subsystems, turning 'On' the 'Latch input for feedback signals of function-call subsystem outputs' prevents the input value to this subsystem from changing during its execution.
The other parameters can be used to explicitly specify the input signal attributes.

Main | Signal Attributes | Execution

☐ Output function call

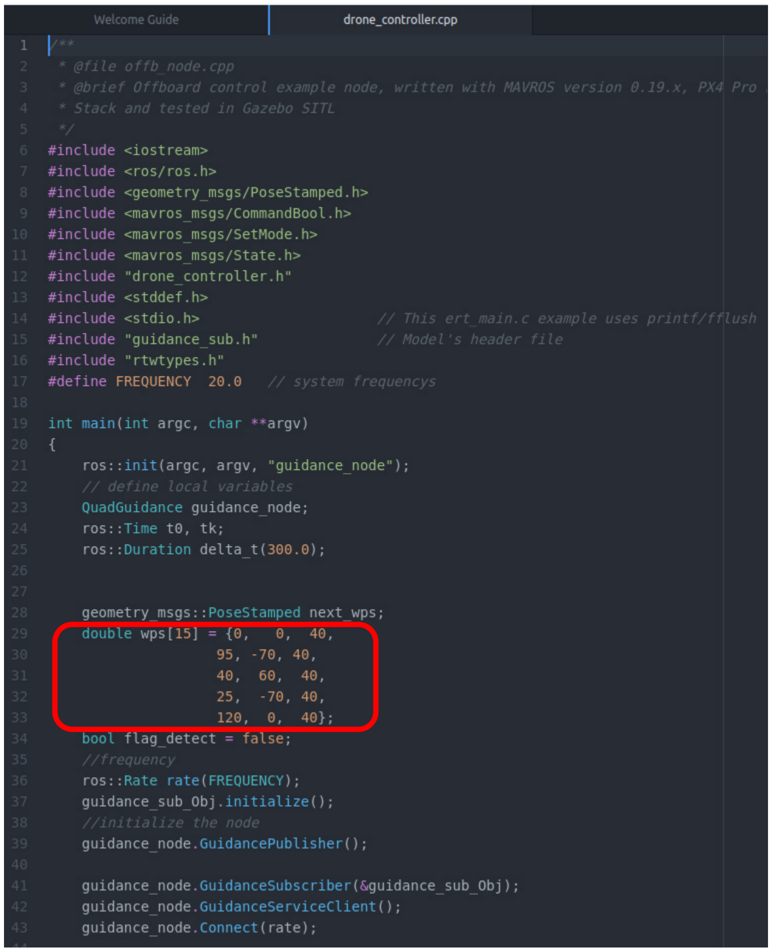
Minimum: []
Maximum: []
Data type: double
☐ Lock output data type setting against changes by the fixed-point tools
Unit (e.g., m, m/s^2, N*m): inherit
SI, English, ...
Port dimensions (-1 for inherited): [3,5]
Variable-size signal: Inherit
Signal type: real

OK | Cancel | Help | Apply

Figure 40. Updated Waypoint Signal Port Dimensions

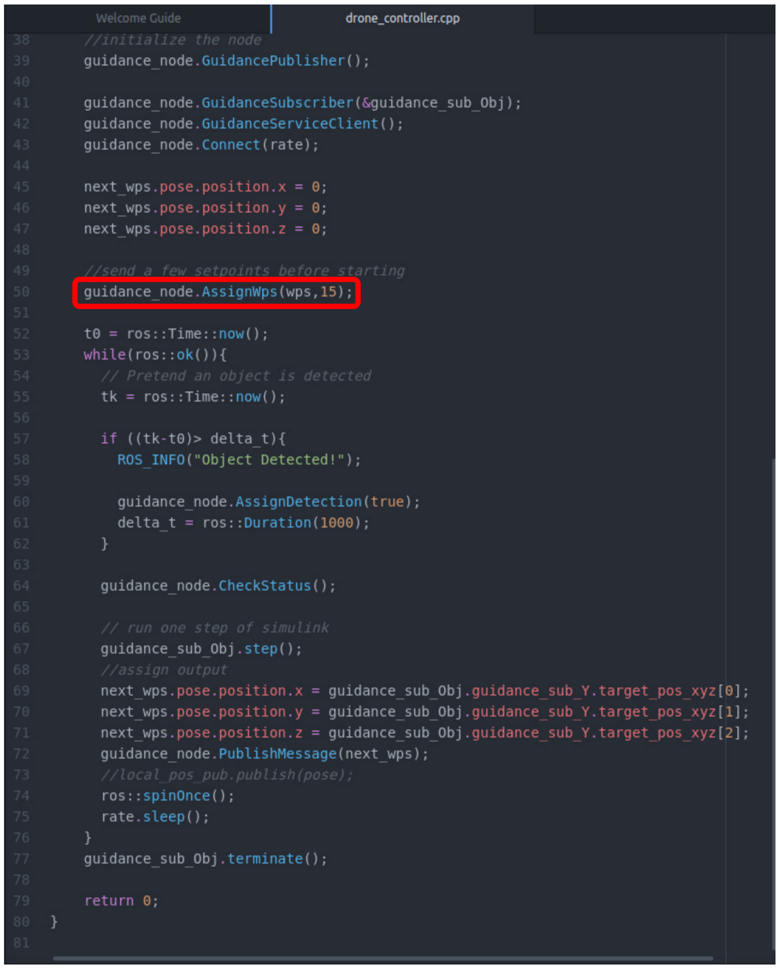
b. Generate and Incorporate C++ Code from Simulink into ROS Node

This step is to generate C++ code from the updated Simulink model, and follows the steps in Chapter III.A.2. Next, the C++ code from the updated Simulink model is incorporated into a ROS node, following the steps in Chapter III.A.3 plus an additional step to update the "drone_controller.cpp" code with the coordinates of the five waypoints and increasing the number of elements in the waypoint matrix to 15, as shown in Figure 41 and Figure 42 respectively. It should be noted that in the C++ code, each <u>row</u> represents the x, y and z coordinates of a waypoint. Since MATLAB stores these coordinates in a column, the waypoint matrix needs to be transposed in the C++ code.



```cpp
/**
 * @file offb_node.cpp
 * @brief Offboard control example node, written with MAVROS version 0.19.x, PX4 Pro
 * Stack and tested in Gazebo SITL
 */
#include <iostream>
#include <ros/ros.h>
#include <geometry_msgs/PoseStamped.h>
#include <mavros_msgs/CommandBool.h>
#include <mavros_msgs/SetMode.h>
#include <mavros_msgs/State.h>
#include "drone_controller.h"
#include <stddef.h>
#include <stdio.h>                    // This ert_main.c example uses printf/fflush
#include "guidance_sub.h"             // Model's header file
#include "rtwtypes.h"
#define FREQUENCY  20.0   // system frequencys

int main(int argc, char **argv)
{
    ros::init(argc, argv, "guidance_node");
    // define local variables
    QuadGuidance guidance_node;
    ros::Time t0, tk;
    ros::Duration delta_t(300.0);


    geometry_msgs::PoseStamped next_wps;
    double wps[15] = {0,   0,  40,
                    95, -70, 40,
                    40,  60,  40,
                    25,  -70, 40,
                    120,  0,  40};
    bool flag_detect = false;
    //frequency
    ros::Rate rate(FREQUENCY);
    guidance_sub_Obj.initialize();
    //initialize the node
    guidance_node.GuidancePublisher();

    guidance_node.GuidanceSubscriber(&guidance_sub_Obj);
    guidance_node.GuidanceServiceClient();
    guidance_node.Connect(rate);
```

Figure 41. Updated Waypoints' Coordinates in "drone_controller.cpp"

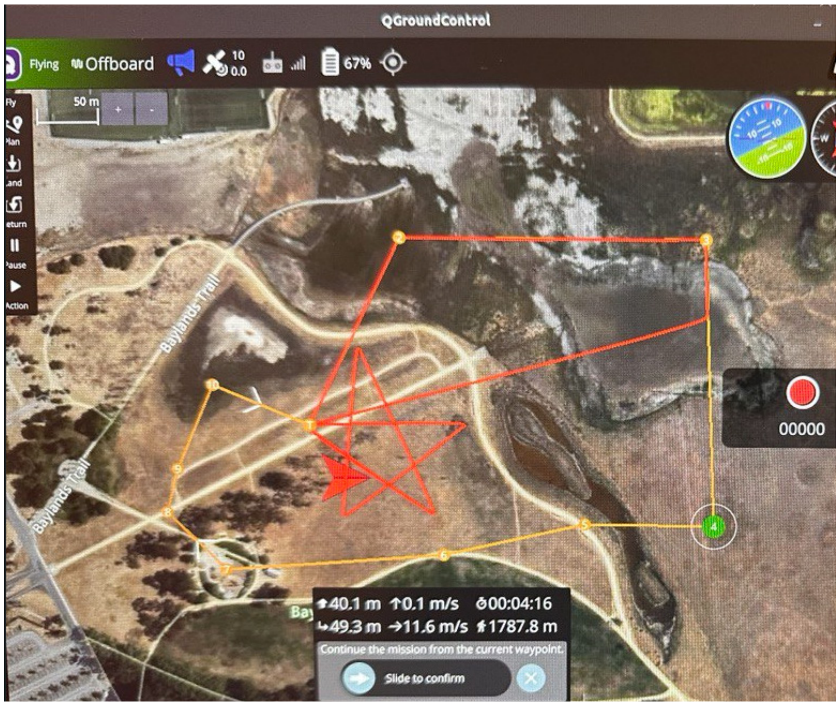


```
    //initialize the node
    guidance_node.GuidancePublisher();

    guidance_node.GuidanceSubscriber(&guidance_sub_Obj);
    guidance_node.GuidanceServiceClient();
    guidance_node.Connect(rate);

    next_wps.pose.position.x = 0;
    next_wps.pose.position.y = 0;
    next_wps.pose.position.z = 0;

    //send a few setpoints before starting
    guidance_node.AssignWps(wps,15);

    t0 = ros::Time::now();
    while(ros::ok()){
      // Pretend an object is detected
      tk = ros::Time::now();

      if ((tk-t0)> delta_t){
        ROS_INFO("Object Detected!");

        guidance_node.AssignDetection(true);
        delta_t = ros::Duration(1000);
      }

      guidance_node.CheckStatus();

      // run one step of simulink
      guidance_sub_Obj.step();
      //assign output
      next_wps.pose.position.x = guidance_sub_Obj.guidance_sub_Y.target_pos_xyz[0];
      next_wps.pose.position.y = guidance_sub_Obj.guidance_sub_Y.target_pos_xyz[1];
      next_wps.pose.position.z = guidance_sub_Obj.guidance_sub_Y.target_pos_xyz[2];
      guidance_node.PublishMessage(next_wps);
      //local_pos_pub.publish(pose);
      ros::spinOnce();
      rate.sleep();
    }
    guidance_sub_Obj.terminate();

    return 0;
}
```

Figure 42. Updated Number of Waypoint Elements in “drone_controller.cpp”

c. *Execute Node*

This step involves executing the updated ROS node follows the steps in Chapter III.A.4. Upon executing the “rosrun px4_simulink_mavros px4_simulink_mavros_node” command, the SIM-1 UAS (actual or simulated) will divert from its original flight path and fly the new flight path comprising five waypoints instead. Figure 43 displays a QGroundControl screenshot from a simulated flight when SIM-1 diverted from its original flight path shortly after waypoint three, and followed a 5-point star-shaped flight path.



Continue the mission from the current waypoint.

Figure 43. SIM-1 Diverted from Original Flight Path and Followed a 5-Point Star-Shaped Flight Path

3. Advantages and Disadvantages of Each Method of Flight Path Planning

Each of these two flight path planning methods has advantages and disadvantages.

The first method of utilizing the QGroundControl interface to perform basic flight path planning ensures that an executable flight path is generated. For example, QGroundControl automatically creates a proper landing approach based on the UAS hardware, and QGroundControl prevents users from ignoring the effects of terrain (described in Chapter III.B.1.c.). Additionally, flight path planning in QGroundControl is intuitive and users can visualize the flight path against the terrain easily. However, the manual flight path planning with QGroundControl may be cumbersome for complex flight paths comprising large number of waypoints.

On the other hand, the second method uses algorithms developed, tested, and ported to C++ code using Simulink. As a result, users must have a better understanding of flight requirements such as take-off and landing, to properly design a set of viable flight path waypoints, a task which QGroundControl handles automatically. Using Simulink and C++ code is also less intuitive than the graphical approach used in QGroundControl. However, the Simulink and C++ code method allows more sophisticated guidance, navigation and control (GNC) algorithms to be integrated. Hence, it also provides an avenue for GNC prototype algorithms to be swiftly integrated and flight tested rapidly. Consequently, the SIM-1 UAS can then be utilized for education in the form of supporting UAS coursework and advanced research by flight testing novel GNC algorithms at NPS.

In summary, the first method using QGroundControl for flight path planning is more suitable for beginners, while the second method using C++ code gives more user autonomy and capability but is also more complex.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. SOFTWARE-IN-THE-LOOP SIMULATION AND ACTUAL FLIGHT TEST RESULTS

A. SOFTWARE-IN-THE-LOOP SIMULATION

For this thesis, basic software-in-the-loop (SITL) simulations were carried out to validate the following basic requirements:

- The SIM-1 UAS can fly in a stable manner with basic control laws in place.
- The communication links between the SIM-1 UAS and the GCS are configured and working properly.

These simulations were carried out before actual flight testing. The advantages of conducting simulations are that they are relatively easy to execute and can be utilized to reveal and resolve any software related issues.

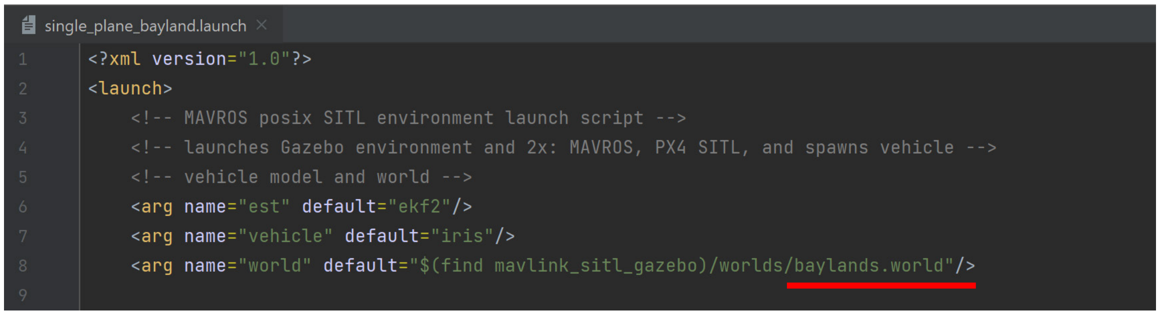
1. Preparations

To facilitate SITL simulations, the following steps are required: 1) Setting up the Ground Control Station (Chapter II.C.), 2) Setting up the software (Chapter II.D.), 3) Developing and incorporating basic controls (Chapter III.A.) and 4) Planning of flight path (Chapter III.B.). The details of these steps are found in the respective chapters in this thesis.

Additionally, the Gazebo simulation environment must be set up. This includes choosing or creating a Gazebo virtual world for the simulation. For this thesis, a total of four worlds were used, 1) Yosemite National Park, 2) Baylands Park, 3) Rancho San Antonio Flying Field, and 4) Monterey Bay Academy. The Yosemite National Park and Baylands Parks worlds were created by J. Lim and are available online in GitHub [32], [33]. The Rancho San Antonio Flying Field was included in this thesis as flight testing was conducted there. Finally, Monterey Bay Academy was included because this area can support more regular flight test activities, due to partnership agreement with the NPS Consortium for Robotics and Unmanned Systems Education and Research (CRUSER).

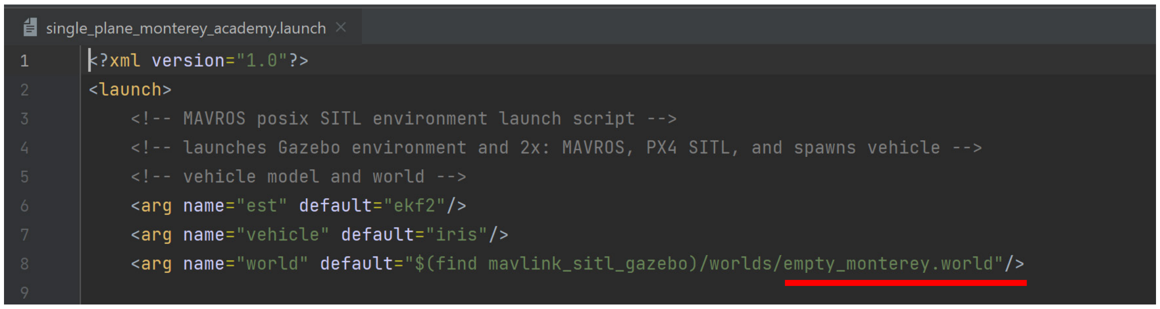
However, it should be noted that the third and fourth worlds are empty worlds that have not yet been populated in Gazebo. Since there is ample air space in each area without obstructions, exact virtual replicas of these areas are not crucial. However, when more regular flight testing takes place at Monterey Bay Academy, future student can model this area so that more representative flight simulations can be conducted.

A launch file was created to run Gazebo, PX4 and MAVROS at the same time. As mentioned in Chapter III.B.1.a., the launch file starts Gazebo using the Yosemite National Park world model, together with the other software modules. In order to run simulations in other areas, a new launch file can be created to specify the desired world, e.g. “baylands.world” (Figure 44) or “empty_monterey.world” (Figure 45) if a pre-built Gazebo world is not available. In the specified world file, the latitude and longitude of its LTP coordinate frame origin must be stipulated (Figure 46) so that the correct location will be loaded. The latitude and longitude shown in Figure 46 are located at the Monterey Bay Academy.


single_plane_bayland.launch

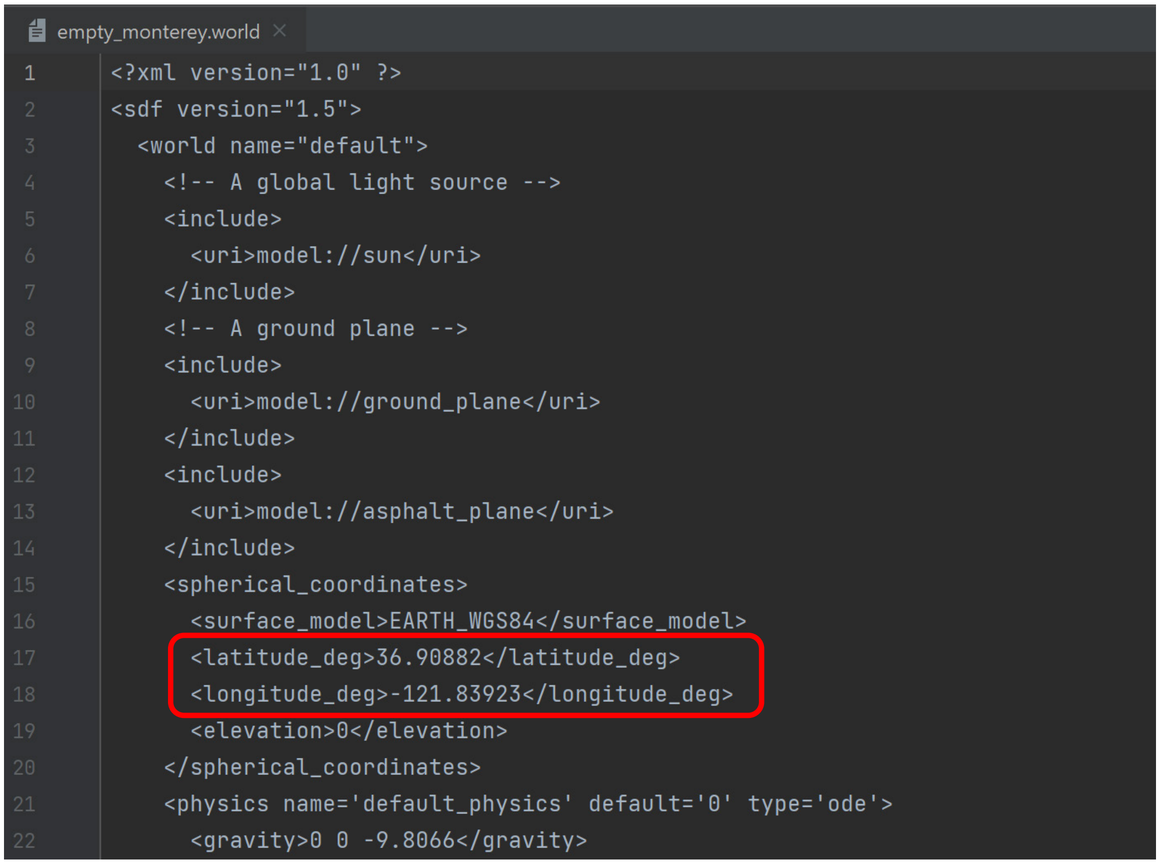
```
<?xml version="1.0"?>
<launch>
    <!-- MAVROS posix SITL environment launch script -->
    <!-- launches Gazebo environment and 2x: MAVROS, PX4 SITL, and spawns vehicle -->
    <!-- vehicle model and world -->
    <arg name="est" default="ekf2"/>
    <arg name="vehicle" default="iris"/>
    <arg name="world" default="$(find mavlink_sitl_gazebo)/worlds/baylands.world"/>
```

Figure 44. “bayland.launch” File which Loads the “bayland.world”


single_plane_monterey_academy.launch

```
<?xml version="1.0"?>
<launch>
    <!-- MAVROS posix SITL environment launch script -->
    <!-- launches Gazebo environment and 2x: MAVROS, PX4 SITL, and spawns vehicle -->
    <!-- vehicle model and world -->
    <arg name="est" default="ekf2"/>
    <arg name="vehicle" default="iris"/>
    <arg name="world" default="$(find mavlink_sitl_gazebo)/worlds/empty_monterey.world"/>
```

Figure 45. Launch File which Loads a New World



```
<?xml version="1.0" ?>
<sdf version="1.5">
  <world name="default">
    <!-- A global light source -->
    <include>
      <uri>model://sun</uri>
    </include>
    <!-- A ground plane -->
    <include>
      <uri>model://ground_plane</uri>
    </include>
    <include>
      <uri>model://asphalt_plane</uri>
    </include>
    <spherical_coordinates>
      <surface_model>EARTH_WGS84</surface_model>
      <latitude_deg>36.90882</latitude_deg>
      <longitude_deg>-121.83923</longitude_deg>
      <elevation>0</elevation>
    </spherical_coordinates>
    <physics name='default_physics' default='0' type='ode'>
      <gravity>0 0 -9.8066</gravity>
```

Figure 46. Snippets of the New World File, Stipulating the Latitude and Longitude of Monterey Bay Academy

2. Simulation Specifics and Simulation Results

For this thesis, basic simulations were done to validate: 1) SIM-1's ability to perform waypoints following and loitering behaviors in various Gazebo worlds, 2) communication links between the SIM-1 UAS and the GCS by diverting SIM-1 from its original flight plan to a new flight plan uploaded from the GCS, and 3) SIM-1's correct response to manual controls inputs via a game controller.

a. *Yosemite National Park*

Once QGroundControl and the launch file are run, QGroundControl (with the pre-loaded flight plan) and Gazebo windows will appear (Figure 29 and Figure 30 respectively) for the Yosemite National Park simulation. Figure 47 (top left) shows the basic flight plan that SIM-1 will execute in the simulation. The simulation was carried out, and SIM-1 was able to follow the intended flight path closely without drifting and was able to hold a stable loiter of radius 50m (Figure 47 top right). Next, Figure 47 (bottom left) shows SIM-1 being diverted from its original flight path to a new waypoint at its starting position. This was executed using the method described in Chapter III.B.2. Lastly, Figure 47 (bottom right) shows that SIM-1 was being controlled manually via inputs from an external game controller. SIM-1 was seen performing a left bank which corresponded to a left stick input on the right joystick.

Similar simulations were done in the other three virtual worlds: Baylands Park, Rancho San Antonio Flying Field, and Monterey Bay Academy. Their respective simulation flight plan and simulation results are shown in Figures 48, 49 and 50 respectively.

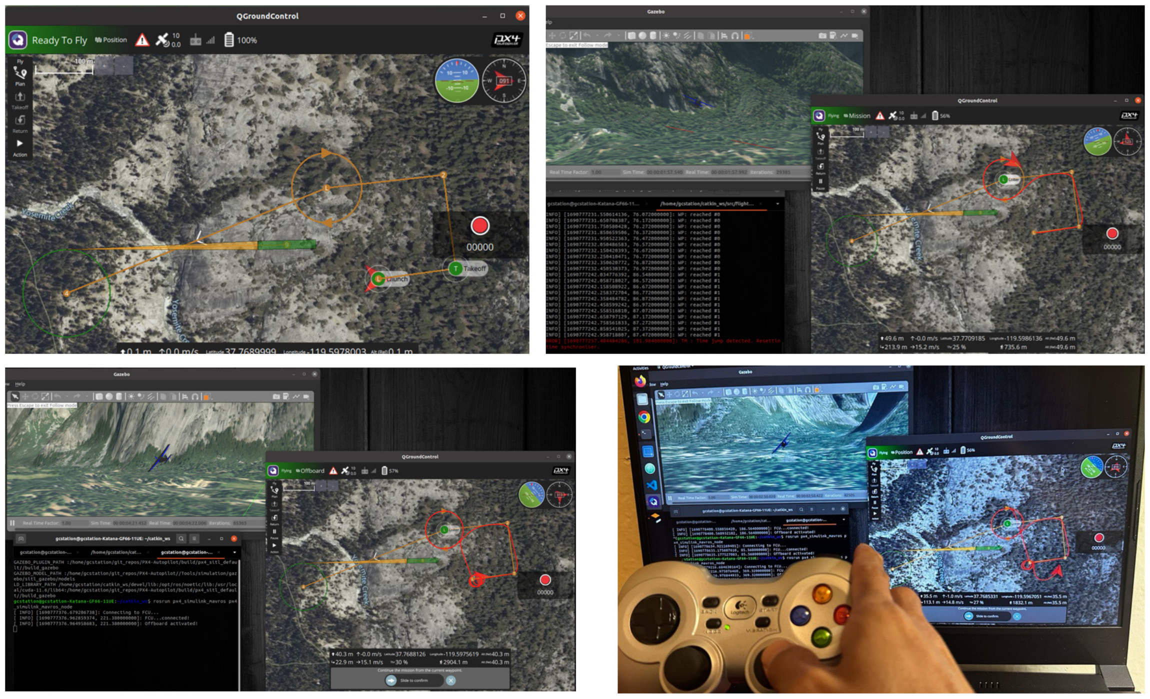


Figure 47. SIM-1 Simulation Flight Plan and Results for Yosemite National Park

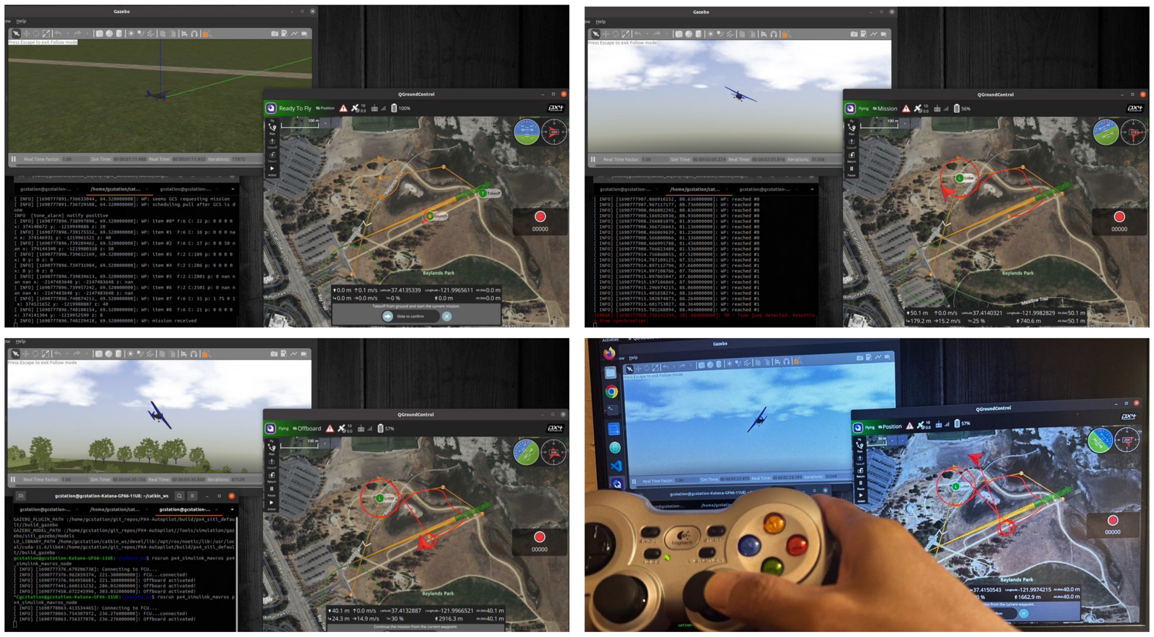


Figure 48. SIM-1 Simulation Flight Plan and Results for Baylands Park

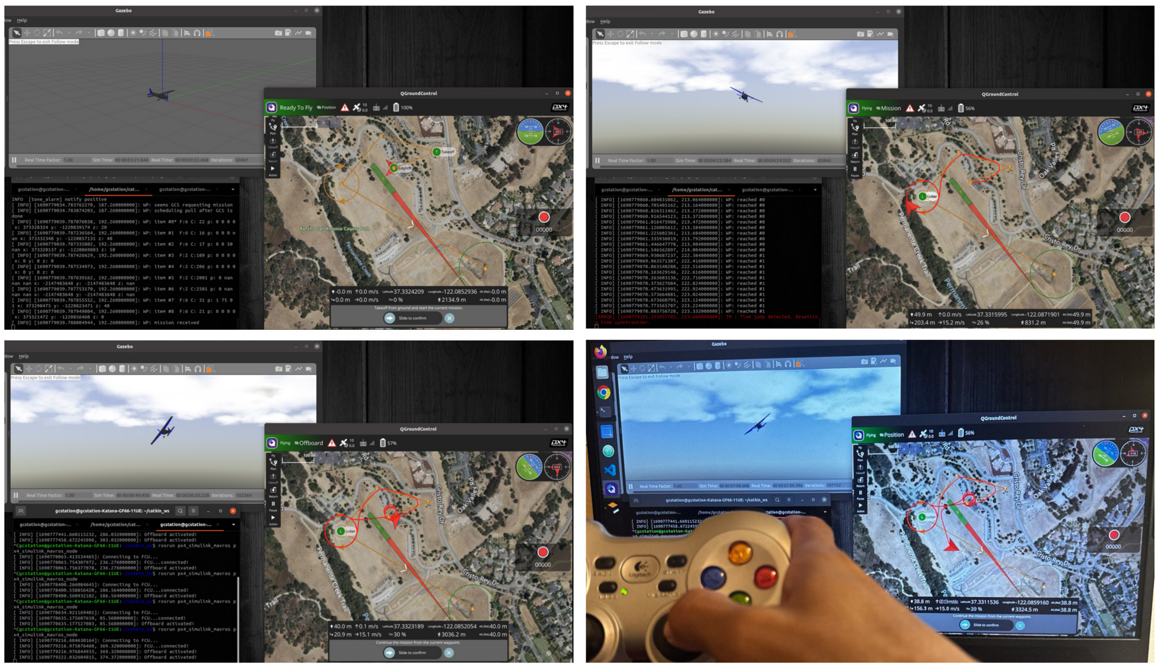


Figure 49. SIM-1 Simulation Flight Plan and Results for Rancho San Antonio Flying Field

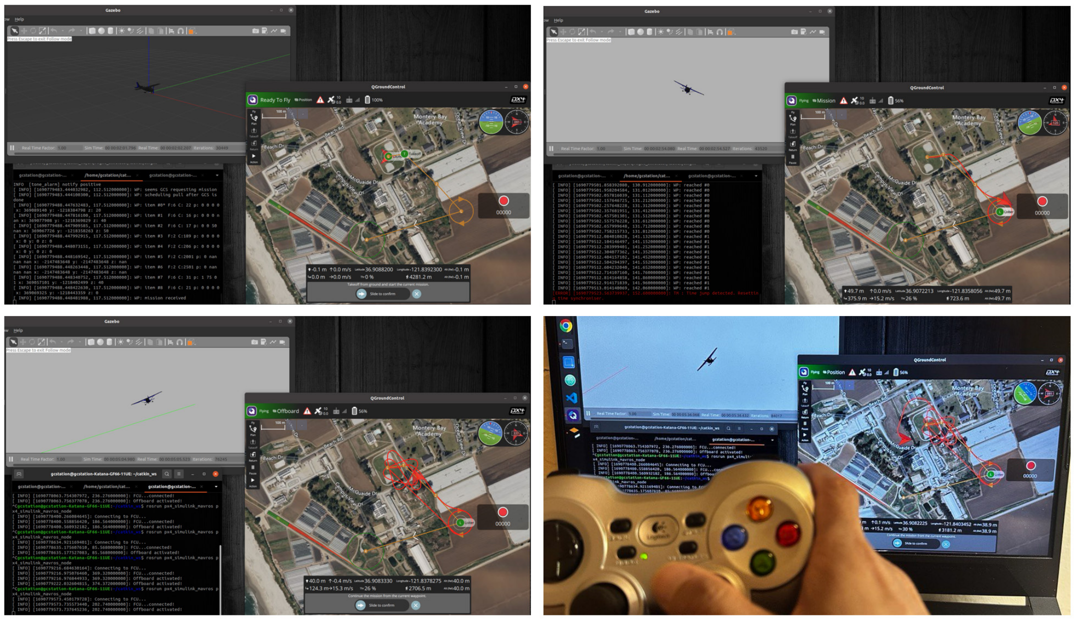


Figure 50. SIM-1 Simulation Flight Plan and Results for Monterey Bay Academy

3. Analysis of Simulation Results and Validation of Simulations

Table 2 summarizes the simulation results. The simulation results are largely satisfactory, as most of the requirements are met. Notably, SIM-1 managed to maintain a stable loiter and respond correctly to external control inputs in all four simulations. This was expected as the simulation environment has no significant effect on SIM-1's ability to fly unless it contains obstacles such as trees or mountains that may affect the flight path.

For the Baylands Park and Rancho San Antonio Flying Field simulations, it was noted that SIM-1 transited to its second waypoint before reaching its specified takeoff point. A review of the QGroundControl documentations and open source discussions did not yield clear reasons for why this happened. However, the author postulates that the algorithm may have been designed to optimize the flight path for a fixed wing UAS to approach the next waypoint during takeoff. Specifically, for Baylands Park and Rancho San Antonio Flying Field, the angle that the second waypoint makes with the takeoff 'trajectory' is an acute angle, whereas the same angle for the other two simulation areas were at least 90 degrees. Perhaps the standard flight path planner deliberately causes the

UAS to head for the next waypoint to achieve a shorter and hence faster path, instead of flying all the way to the stipulated takeoff point before making a wider turn towards the second waypoint. While this seems like a trivial and beneficial feature of the QGroundControl software, it may cause potential issues with the steeper altitude gradient when such unintended 'diversions' occur. However, based on observations of these simulations, the flight trajectories are smooth. As such, it is likely that the software algorithm has taken care of this aspect.

In terms of diversion to a new flight plan, with the exception of the Monterey Bay Academy simulation, the other three simulations did not complete the full flight path which involved flying into a 5-point star-shaped flight path. The SIM-1 UAS in these three simulations were observed to be making small loiter around the first waypoint set at (0, 0, 0) (the starting point), and did not proceed to the next four waypoints. Similar simulations were conducted on a quadcopter model under varying conditions, and these aircraft were able to achieve the desired 5-point star-shaped flight path. An example of the quadcopter simulation can be found in Figure 51. After a review of the C++ code setup for flight path diversion, the author postulated that the issue could be with the threshold that was set before the UAS satisfies the condition to fly to the next waypoint. This threshold is a comparison of the distance of the drone to the current waypoint. The difference between the quadcopter and the fixed wing drone flights is that the former will fly directly to the current waypoint while the latter would loiter around that same waypoint. And in a loiter, the distance between the drone and its waypoint does not decrease; hence, since the condition to fly to the next waypoint cannot be met, the loiter continues indefinitely. This behavior is unsatisfactory in a real-life operational scenario, as the UAS would not be able to reach the stipulated waypoint(s) and perform its intended role. To resolve this issue, the waypoint generation method (described in Chapter III.B.2) should be improved so that the waypoint following algorithm can be more robust. It should be noted that the basic requirements of generating waypoints and diverting SIM-1 to the new flight path were still achieved.

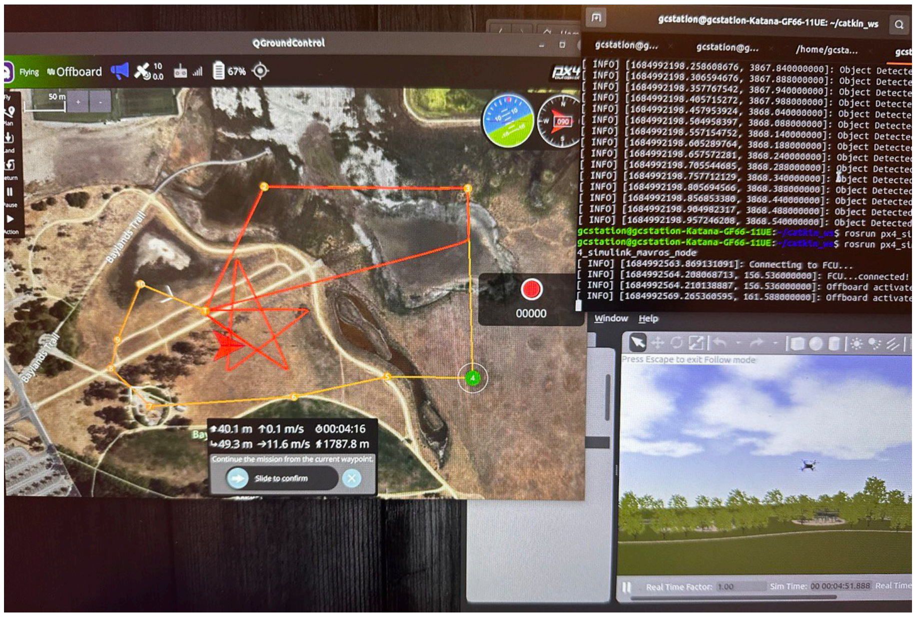
In summary, the simulations showed that SIM-1 UAS can fly in a stable manner with basic control laws in place, and the communication links between the SIM-1 UAS and the GCS worked properly. The simulations also identified a few minor issues that can be improved upon in the future.

Table 2. Summary of Simulation Results

Simulation Area	Ability to Fly		Communication Links between GCS and SIM-1	
	Waypoint Following	Loiter	Diversion to a New Flight Plan	Response to External Control Inputs
Yosemite	🟩 Adhere to waypoints closely	🟩 Able to maintain stable loiter	🟧 Diverted to only the first waypoint	🟩 SIM-1 banked left according to stick input
Baylands	🟧 Adhere to waypoints, except for takeoff point	🟩 Able to maintain stable loiter	🟧 Diverted to only the first waypoint	🟩 SIM-1 banked left according to stick input
Rancho San Antonio Flying Field	🟧 Adhere to waypoints, except for takeoff point	🟩 Able to maintain stable loiter	🟧 Diverted to only the first waypoint	🟩 SIM-1 banked left according to stick input
Monterey Bay Academy	🟩 Adhere to waypoints closely	🟩 Able to maintain stable loiter	🟩 Diverted and flew to all 5 waypoints	🟩 SIM-1 banked left according to stick input

Green means that the objective was fully met.

Orange means that the objective was partially met.



```
gcstation@gcstation-Katana-GF66-11UE: ~/catkin_ws
gcstation@g...    gcstation@g...    /home/gcsta...    gcst
[ INFO] [1684992198.258608676, 3867.840000000]: Object Detecte
[ INFO] [1684992198.306594676, 3867.888000000]: Object Detecte
[ INFO] [1684992198.357767542, 3867.940000000]: Object Detecte
[ INFO] [1684992198.405715272, 3867.988000000]: Object Detecte
[ INFO] [1684992198.457953924, 3868.040000000]: Object Detecte
[ INFO] [1684992198.504958397, 3868.088000000]: Object Detecte
[ INFO] [1684992198.557154752, 3868.140000000]: Object Detecte
[ INFO] [1684992198.605289764, 3868.188000000]: Object Detecte
[ INFO] [1684992198.657572281, 3868.240000000]: Object Detecte
[ INFO] [1684992198.705544685, 3868.288000000]: Object Detecte
[ INFO] [1684992198.757712129, 3868.340000000]: Object Detecte
[ INFO] [1684992198.805694566, 3868.388000000]: Object Detected
[ INFO] [1684992198.856853380, 3868.440000000]: Object Detected
[ INFO] [1684992198.904982317, 3868.488000000]: Object Detected
[ INFO] [1684992198.957246208, 3868.540000000]: Object Detected
gcstation@gcstation-Katana-GF66-11UE:~/catkin_ws$ rosrun px4_si
gcstation@gcstation-Katana-GF66-11UE:~/catkin_ws$ rosrun px4_si
4_simulink_mavros_node
[ INFO] [1684992563.869131091]: Connecting to FCU...
[ INFO] [1684992564.208068713, 156.536000000]: FCU...connected!
[ INFO] [1684992564.210138887, 156.536000000]: Offboard activate
[ INFO] [1684992569.265360595, 161.588000000]: Offboard activate
```

Figure 51. Simulated Quadcopter Diverted to a 5-Point Star-Shaped Flight Path

B. ACTUAL FLIGHT TEST RESULTS

Following the SITL simulations, actual flight tests were conducted. According to Gregory et al., flight testing is conducted to determine the true performance of an aircraft and uncover any unforeseen issues [34]. For this thesis, flight tests were conducted to validate the same requirements as in the simulations, i.e., that SIM-1 can fly in a stable manner, and that the communication links worked properly. Flight tests were conducted at the Baylands Parks (Figure 52) and the Rancho San Antonio Flying Field (Figure 53) using an incremental approach. To facilitate subsequent flight tests, checklists were created in Appendix B for reference.

The first set of flight tests were done at Baylands Park, with basic objectives to check that SIM-1 was able to take off, fly relatively low and near, and land safely. Initially, these flights, especially take offs and landings, were conducted manually via the RC radio transmitter. This was to minimize the amount of risk on SIM-1 in case of unforeseen circumstances. The flight altitude was capped at about 98 ft (approximately 30 m), and the flight range was within 328 ft (approximately 100 m) of the RC pilot. These flight tests were carried out successfully. Despite manual takeoff and landings, SIM-1 was able to fly smoothly without any erratic movements and maintain a stable loiter (pre-set via QGroundControl) in the operating area. Additionally, QGroundControl was also used to adjust the loiter zone in terms of flight altitude and position during the flight. SIM-1 flew to the amended loiter waypoint uneventfully after uploading it via the GCS.

The second set of flight tests were done at the Rancho San Antonio Flying Field, with extended objectives to check that SIM-1 was able to fly higher and further while maintaining similar stable flight characteristics. The flight altitude was capped at about 197 ft (approximately 60 m), and the flight range was within 500 ft (approximately 150 m) of the RC pilot. Again, these flight tests were carried out successfully, as SIM-1 flew smoothly without any erratic movements and maintained a stable loiter (pre-set via QGroundControl) in the area. Figure 54 shows SIM-1 maintaining a stable loiter. SIM-1 also flew to an amended loiter waypoint (with different coordinates and altitude) uneventfully after uploading it via the GCS. Figure 55 and Figure 56 show the two-dimensional (2D) and 3D flight path taken by SIM-1 in this flight test. The flight path visualizations were created from log files downloaded from SIM-1 after the flight, using Flight Review analysis software developed by PX4 Team [35]. This flight test also verified that SIM-1's endurance was greater than 30 minutes. It has been estimated that the 5000 mAh battery would give SIM-1 an endurance of about 45 minutes.

The flight test results showed that SIM-1 was able to fly smoothly and maintain a stable loiter. This is evidence that the autopilot and the flight controls in place were configured correctly. The flight tests also demonstrated that the communication links between SIM-1, the GCS and the RC radio transmitter were fully functional, since the RC radio and telemetry links worked during the flight tests.

In summary, the flight tests were carried out successfully as SIM-1 demonstrated stable flight and all communication links functioned properly. There were some minor issues uncovered in the process, but they did not affect the basic requirements of SIM-1 and could be resolved in a straightforward manner.

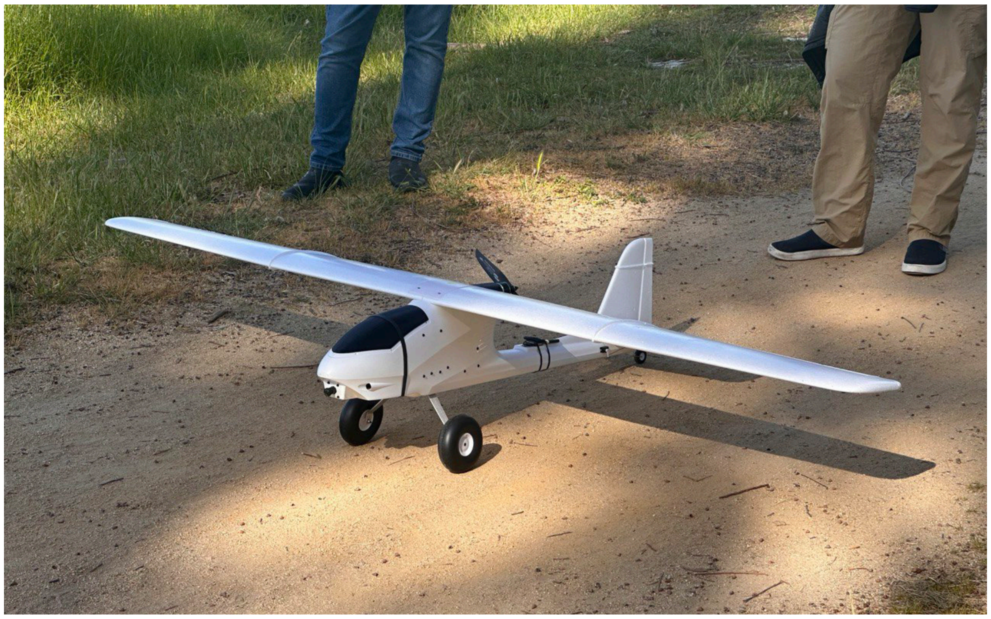


Figure 52. Photo of SIM-1 before Take Off at Baylands Park

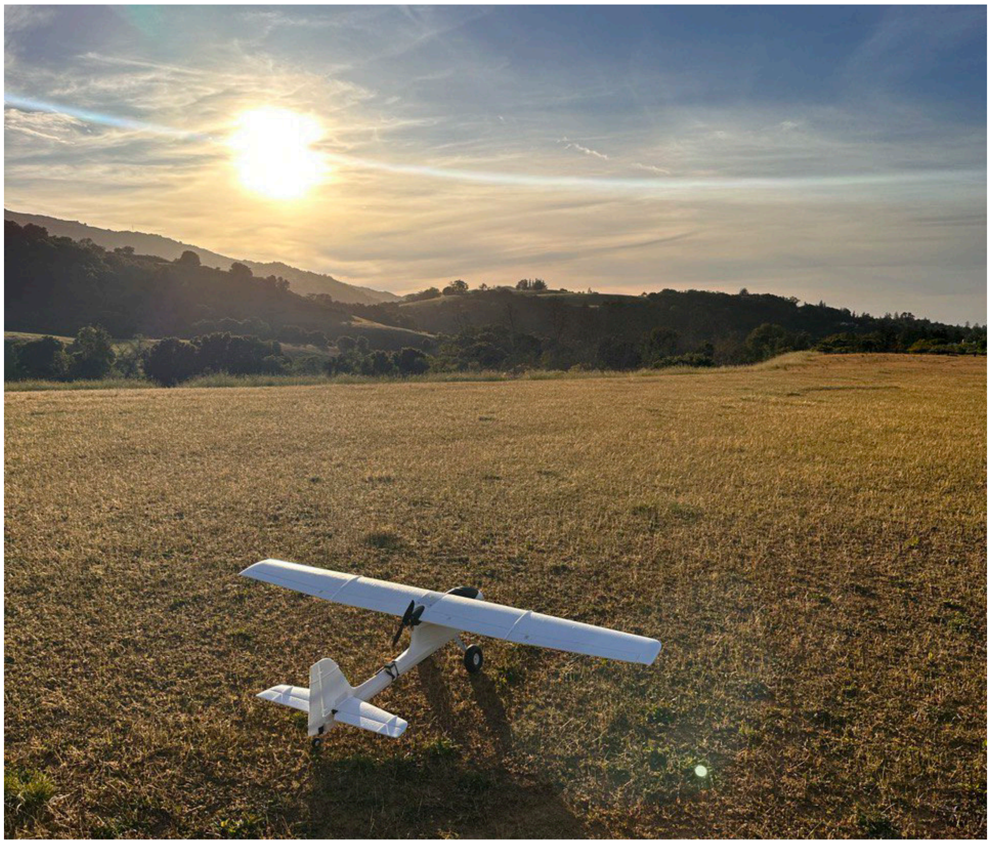


Figure 53. Photo of SIM-1 at Rancho San Antonio Flying Field

Figure 54. Photo of SIM-1 (Indicated by Red Arrow) in a Stable Loiter at Rancho San Antonio Flying Field

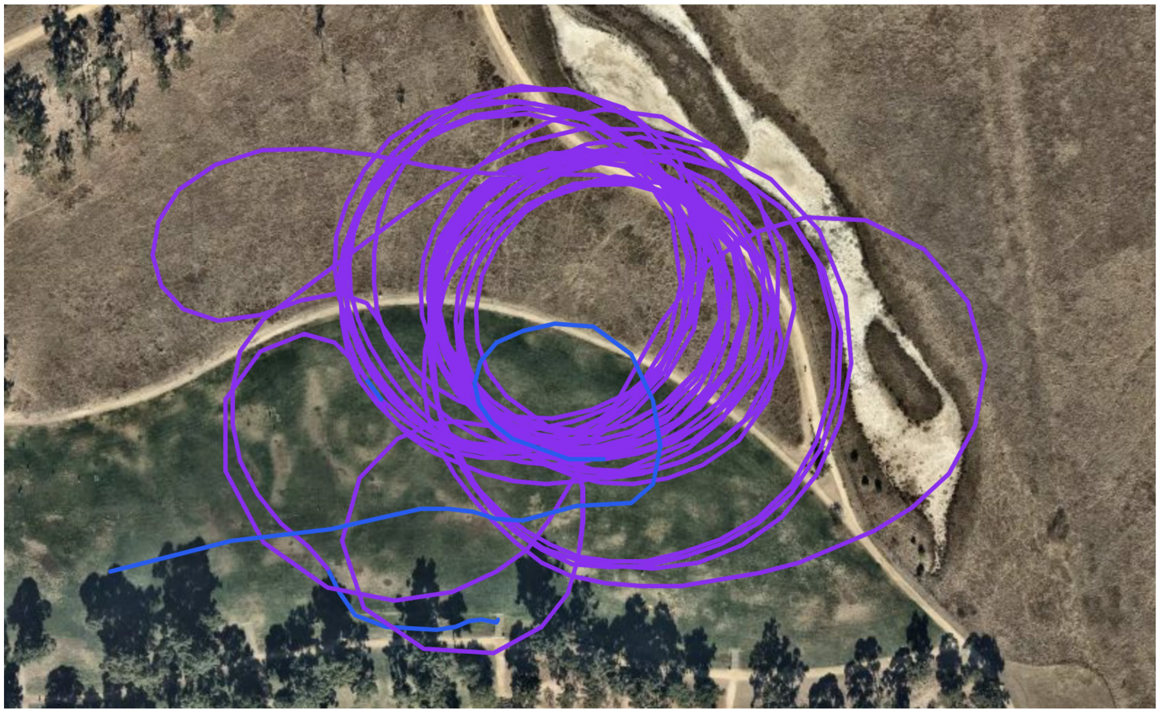


Figure 55. SIM-1 2D Flight Path at Rancho San Antonio Flying Field Flight Test

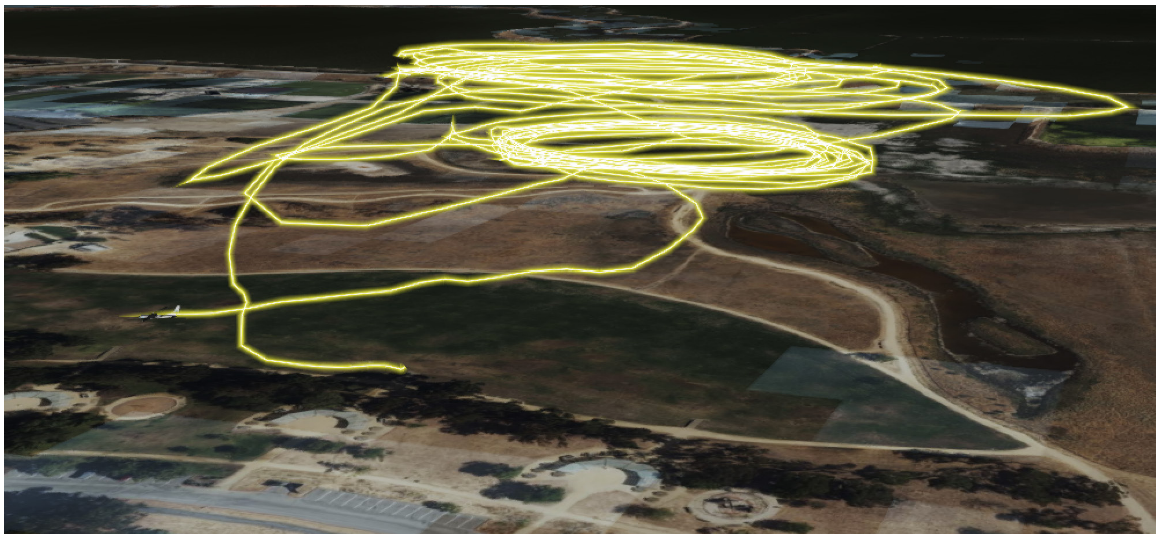


Figure 56. SIM-1 3D Flight Path at Rancho San Antonio Flying Field Flight Test; Flight Path Shows Loiter at Two different Altitudes.

V. SETTING UP FOR MACHINE LEARNING

A. SETTING UP THE CAMERA NODE

In order to utilize SIM-1's onboard camera, Chesi UAS Solutions set up the GCS with a ROS node called "nps_drone_cam." This node was configured to acquire images from the camera connected to SIM-1. This node was tested using the GCS laptop's front facing camera as well. Two associated launch files were also developed to specify the desired camera via the "roslaunch nps_drone_cam drone_camera.launch" and "roslaunch nps_drone_cam gcstation_camera.launch" commands, respectively. These launch files were also utilized to set the desired camera settings such as frames per second, image height and width, etc. The relevant files can also be found in GitLab [29].

B. SETTING UP THE MACHINE LEARNING (ML) ALGORITHM NODE

According to Kim, "Machine Learning is a kind of Artificial Intelligence and Deep Learning is a kind of Machine Learning." He also pointed out that Deep Learning is a ML technique that employs deep neural networks[4] (NN) which have two or more hidden layers of neurons [36]. Aggarwal explains that convolutional neural networks (CNN) are a group of deep NN, which were designed to work with grid-structured inputs, and one of the applications of CNN is to carry out object detection on images. Object detection is akin to multiple object localizations in a single image [37].

One research objective is to enable SIM-1 to perform object detection tasks while it is in flight. This would allow SIM-1, after being deployed in an area for surveillance tasks, to detect specific objects such as vehicles, triggering further actions. The Simulink model elaborated in Chapter III.A incorporated a feature where a detection flag could trigger a "stop flying" action. Theoretically, SIM-1 would enter a loiter mode around the location where the detection event occurred. To test this capability, the CNN algorithm

[4] Aggarwal explained that NNs are ML techniques that simulate the mechanism of learning in brain networks in biological organisms. For the case of ML NN, output(s) is computed from inputs by propagating the values from the input neurons to the output neurons through the NN weights and biases. Learning occurs when the weights and biases are adjusted such that the predicted output value(s) converges to the intended output value(s) as in the case of supervised learning. [33]

"You Only Look Once" (YOLO) was implemented on the SIM-1 UAS. According to Redmon et al., YOLO utilizes only one CNN on images to predict the objects' presence and locations in the image, unlike other 2-stage CNNs where these two tasks are done separately. As such, YOLO is relatively fast compared to other CNN algorithms, and it can work in almost real-time [38]. These advantages make YOLO a suitable starting algorithm for SIM-1. However, YOLO does suffer in terms of accuracy as compared to other 2-stage algorithms [38], and hence another algorithm should be considered if accuracy is the priority. Based on GitHub, other object detection algorithms (such as DetectNet) have been implemented successfully using ROS as well [39]. As such, these algorithms should be compatible with SIM-1 as well.

To perform object detection using YOLO, a ROS node was executed on the GCS. The node source codes and related details are available on GitHub via the Robotics Systems Lab [40]. The node runs the YOLO algorithm on camera images and videos, once the "roslaunch darknet_ros yolo_v3.launch" command is run. Specifically, object detection will be performed on individual camera images, and the detected objects are published on a ROS topic that describes the type of object, its location in the image, and the accuracy of the detection [29]. This capability was tested on the ground using the GCS laptop camera, and object detection was performed as shown in Figure 57. Along with the bounding boxes around the objects that were detected in the image, their detection accuracy is also displayed in the terminal. Similar steps can be carried out on SIM-1 in flight to test out new sensing and guidance capabilities. Unfortunately, there was a lack of opportunity to conduct flight tests using this capability, and this task will be explored in future works instead.

In summary, to perform vision-based object detection using the SIM-1 UAS ROS framework, a camera node and a separate node to launch the CNN algorithm were required. The capability to perform object detection was tested on ground, but not in flight. However, successful flight tests demonstrating the communication links between SIM-1 and its GCS, and successful object detection tests on the ground, strongly suggest that object detection can be successfully performed in flight. However, there might be other issues that could surface in flight. One potential issue could be that the instability of the non-gimballed camera negatively impacts the quality of the images taken in flight which may in turn affect the accuracy of object detection.

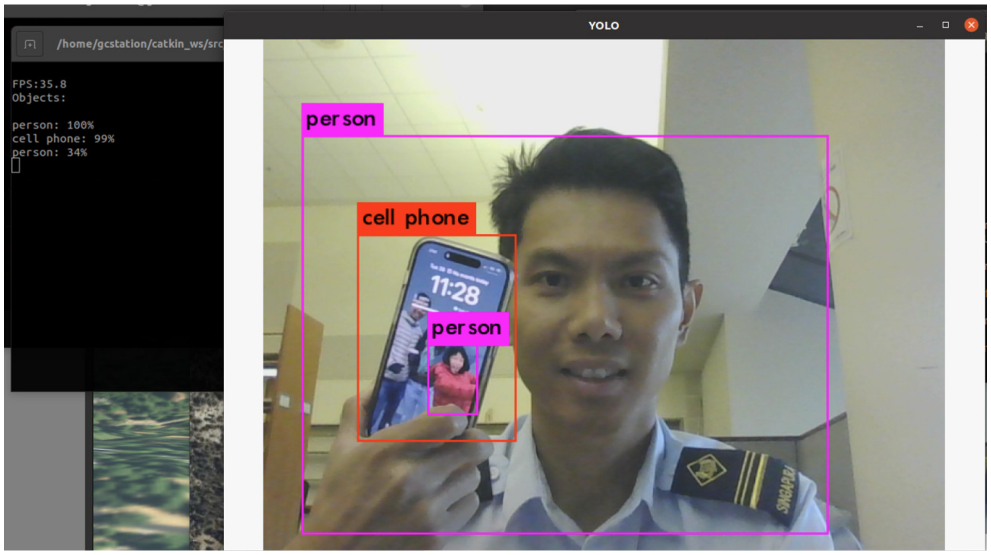

```
/home/gcstation/catkin_ws/src

FPS:35.8
Objects:

person: 100%
cell phone: 99%
person: 34%
```

Figure 57. Object Detection Performed on a Photo, Accuracy of Detection Shown on Left Terminal

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSION

A. SUMMARY

In this thesis, a new Group 1 fixed-wing UAS, named SIM-1, was assembled, configured and then flown successfully in simulated and actual test flights. Extensive simulations were conducted to validate a GNC algorithm development pipeline from MATLAB/Simulink to ROS to Gazebo. Actual flight tests were conducted to verify the SIM-1 UAS functionality. The flight tests also identified several practical considerations regarding the SIM-1 UAS, as detailed in this thesis.

Notably, this thesis explored a waypoint generation and following method that uses algorithms developed, tested and ported to C++ code using MATLAB/Simulink. This method provided an avenue and framework for flight testing prototype GNC algorithms using the SIM-1 UAS. This experimentation framework can be utilized to support NPS coursework in unmanned systems and controls, as well as supporting advanced research into novel GNC algorithms.

Lastly, a ML object detection algorithm was incorporated into the system and shown to work on the ground. This testing strongly indicates that this capability will function during flight using SIM-1's onboard camera. As such, the basic objectives of this thesis were achieved. This also laid the foundation for future research.

B. RECOMMENDATION FOR FUTURE WORKS

Based upon the testing conducted for this thesis, we recommend future research in the following areas:

1. More Elaborate Guidance, Navigation and Control Algorithm

The basic waypoint following algorithm used in SIM-1 could be expanded to include full nonlinear path following algorithms using a more accurate 6-DOF model of the UAS. The enhanced algorithm can then be incorporated in SIM-1 and validated through flight tests in different conditions such as in an area with obstacles.

2. In-flight Objection Detection

In-flight object detection could be conducted in future flight tests, and this would open up many options for future research. Several possibilities include the design of specific responses and/or behaviors such as loitering above a detected object, tracking its motion, etc.

3. Multiple SIM-1 UASs

QGroundControl can be launched with several drones, and hence could also be utilized to explore multi-robot coordination or interaction of multiple SIM-1 UAS. So far, NPS has a fleet of two SIM-1 UAS and one GCS. As such, some possibilities for future work could be to plan and flight test formation flying involving the two SIM-1 UAS and then extending to include some unique interactions between the two UAS.

APPENDIX. FLIGHT CHECKLIST

A. PRE-FLIGHT CHECKLIST

1. Ensure all hardware (SIM-1, GCS and peripherals) are brought.
2. Ensure GCS and SIM-1 batteries, including spare ones, are charged up.
3. Pre plan flight test flight paths if possible.
4. Assemble SIM-1 by the following: 1) attach main wings and connect the aileron control cables, 2) secure the main wings with wing cover and four screws, 3) attach battery in cockpit compartment and connect battery cable, 4) attach cockpit cover, and 5) attach and secure propeller.
5. Switch on GCS, and attach telemetry radio antenna to GCS via USB port.
6. Run QGroundControl, ensure that SIM-1 is active and upload flight plan.
7. Take off SIM-1 manually or autonomously.

B. POST-FLIGHT CHECKLIST

1. Check condition of SIM-1.
2. Download flight log from QGroundControl while SIM-1 is active.
3. Disconnect and disassemble SIM-1.
4. Ensure all hardware retrieved and packed.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] Naval Technology. *ScanEagle – Mini-UAV (Unmanned Aerial Vehicle)*, (Aug. 22, 2007). Available: https://www.naval-technology.com/projects/scaneagle-uav/

[2] Asian Military Review. *Eyes in the Sky*, (Mar. 31, 2021). Available: https://www.asianmilitaryreview.com/2021/03/eyes-in-the-sky/?amp

[3] Bard College. *ScanEagle: A Small Drone Making a Big Impact,* (Jan. 6, 2014). Accessed: Jul. 15, 2023. Available: https://dronecenter.bard.edu/scaneagle-drone/

[4] Naval Postgraduate School, "Center for Autonomous Vehicle Research," Accessed Jul. 20, 2023. Available: https://nps.edu/web/cavr

[5] B. P. Keegan, "UAV Position Optimization for Wireless Communications," Masters thesis, Dept. of Mech. Eng., Naval Postgraduate School, Monterey, CA, USA, 2018.

[6] Boeing. *ScanEagle*. Accessed: Jul. 15, 2023. Available: https://www.boeing.com/defense/autonomous-systems/scaneagle/index.page

[7] Chesi UAS Solutions, "Drones for R&D," Accessed Jul. 20, 2023. Available: https://chesiuassol.com/

[8] *United States Air Force Unmanned Aircraft Systems Flight Plan 2009–2047*. United States Air Force, Washington, DC, USA, 2009. Available: https://www.govexec.com/pdfs/072309kp1.pdf

[9] A. K. Kundu, *Aircraft Design*. New York, NY, USA: Cambridge University Press, 2010.

[10] R. K. Schmidt, *The Design of Aircraft Landing Gear*. Warrendale, PA, USA: SAE International, 2021.

[11] e-con Systems, "4K HDR Camera for NVIDIA Jetson Xavier NX/ TX2 NX/ Nano," Accessed: Aug. 2, 23. Available: https://www.e-consystems.com/nvidia-cameras/jetson-tx2-nx-cameras/ar0821-8mp-hdr-camera.asp

[12] CubePilot, "The Cube Orange+ Standard Set," Accessed Jul. 16, 2023. Available: https://www.cubepilot.org/#/cube/specs

[13] Control Hobbies, "Graupner GR-16 8CH 2.4 GHz HoTT Receiver," Accessed Aug. 8, 2023. Available: https://www.controlhobbies.com/33508.html

[14] Control Hobbies, “Graupner mz-12 PRO 12 Channel Telemetry Radio System,” Accessed Aug. 8, 2023. Available: https://www.controlhobbies.com/s1002-pro.html

[15] Drone Nodes, “Drone Transmitter and Receiver – Radio Control System Guide,” Accessed Jul. 17, 2023. Available: https://dronenodes.com/drone-transmitter-receiver-fpv/

[16] NVIDIA Developer, “Jetson Nano Developer Kit,” Accessed Jul. 17, 2023. Available: https://developer.nvidia.com/embedded/jetson-nano-developer-kit

[17] IT-Lock, “RFD900x-US Telemetry Bundle (FCC approved), “ Accessed Jul. 17, 2023. Available: https://irlock.com/collections/telemetry-2/products/rfd900-telemetry-bundle

[18] Thunder Power RC, “TP5000-4SE55,” Accessed Jul. 17, 2023. Available: https://www.thunderpowerrc.com/collections/5000-mah-2/products/tp5000-4se55?variant=31080883060800

[19] Open Robotics, “ROS – Robot Operating System,” Accessed Jul. 17, 2023. Available: https://www.ros.org/

[20] Dronecode, “QGroundControl User Guide,” Accessed Jul. 18, 2023. Available: https://docs.qgroundcontrol.com/master/en/

[21] Dronecode, “PX4 Software Overview,” Accessed Jul. 18, 2023. Available: https://px4.io/software/software-overview/

[22] Open Robotics, “Gazebo,” Accessed Jul. 18, 2023. Available: https://gazebosim.org/features

[23] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng, “ROS: An Open-Source Robot Operating System,” ICRA Workshop on Open Source Software, Jan. 2009, vol. 3.

[24] Dronecode, “MAVROS,” Accessed Jul. 19, 2023. Available: https://dev.px4.io/v1.10_noredirect/en/ros/mavros_installation.html

[25] Dronecode, “ROS with Gazebo Simulation,” Accessed Jul 19, 2023. Available: https://dev.px4.io/v1.10_noredirect/en/simulation/ros_interface.html

[26] Dronecode, “MAVLink Developer Guide,” Accessed Aug. 11, 2023. Available: https://mavlink.io/en/

[27] Open Robotics, “roscpp_tutorials/Tutorials/WritingPublisherSubscriber,” Accessed Jul. 27, 2023. Available: http://wiki.ros.org/roscpp_tutorials/Tutorials/WritingPublisherSubscriber

[28] Open Robotics, "ROSNodeTutorialPython," Accessed Jul. 27, 2023. Available: http://wiki.ros.org/ROSNodeTutorialPython

[29] S. Chesi, "Drone Software Configuration and Installation Manual," unpublished.

[30] Mathworks, "ConfigureToolchain(ToolchainInfo) or Template Makefile Build Process," Accessed Jul. 28, 2023. Available: https://www.mathworks.com/help/rtw/ug/program-builds.html#f1158912

[31] Mathworks, "Manage File Packaging of Generated Code Modules," Accessed Jul. 28, 2023. Available: https://www.mathworks.com/help/ecoder/ug/generate-code-modules.html

[32] J. Lim, "PX4-SITL_gazebo-classic/worlds/Yosemite.world," Accessed Jul. 30, 2023. Available: https://github.com/PX4/PX4-SITL_gazebo-classic/blob/main/worlds/yosemite.world

[33] J. Lim, "PX4-SITL_gazebo-classic/worlds/baylands.world," Accessed Jul. 30, 2023. Available: https://github.com/PX4/PX4-SITL_gazebo-classic/blob/main/worlds/baylands.world

[34] J. W. Gregory, T. Liu, *Introduction to Flight Testing*. Hoboken, NJ, USA: John Wiley & Sons, 2021.

[35] PX4 Team, "Flight Review," Accessed Aug. 17, 2023. Available: https://logs.px4.io/

[36] P. Kim, MATLAB Deep Learning: With Machine Learning, Neural Networks and Artificial Intelligence. New York, NY, USA: Springer, 2017.

[37] C.C. Aggarwal, *Neural Networks and Deep Learning*. Cham, Switzerland: Springer Nature, 2018.

[38] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. *You only look once: Unified, real-time object detection*. IEEE Conference on Computer Vision and Pattern Recognition, 2016.

[39] H. Shah, J. Singh, "Isaac ROS Object Detection," Accessed Aug. 8, 2023. Available: https://github.com/NVIDIA-ISAAC-ROS/isaac_ros_object_detection

[40] Robotics Systems Lab, "YOLO ROS: Real-Time Object Detection for ROS," Accessed Aug. 8, 2023. Available: https://github.com/leggedrobotics/darknet_ros

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Fort Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California