UTD Raytheon AUX Showcase

UTDesign Spring 2022

UTDesign Team # 1370
Alexander Doudnikov (CE)
William Greenfield (EE)
William Mullican (CE)
Sean Njenga (CE)
Noah Parker (EE)
Ian Falk (CE)
Preetika Kondepudi (EE)

Corporate Sponsor:
Raytheon Technologies - Christine Nezda
Raytheon Technologies - Alfonso Lopez

Faculty Advisor:
Dr. Marco Tacca

Instructors:
Dr. Marco Tacca
Dr. Neal Skinner

April 23, 2022

Abstract

The Unmanned Aerial System (UAS) Innovation Showcase, in partnership with Raytheon Technologies, is a competition involving UTD CS and ECE and five other Texas Universities. The objective of this competition is for students to use their creativity and innovation to research, develop, integrate, and test a drone that will autonomously compete in four challenges. Our drone design is a hexacopter that incorporates GPS, stereo cameras, and LiDAR to detect objects and navigate obstacles. The drone uses a Cube Orange flight controller, Nvidia Jetson TX2, and ArduPilot firmware to integrate components and set flight plans. Our Hexacopter design provides increased camera stability and payload capacity allowing us to carry heavier, higher quality sensors. Our team-oriented effort, project management, and problem-solving skills allowed us to come in first place against four other universities and deliver a reliable and accurate drone.

Table of Contents

List of Figures

Acknowledgements

The entire team would like to take this time to express our gratitude to Raytheon Technologies and the UTDesign department for providing us with the funds and tools to be able to work on this project. Furthermore our professor and mentor Dr. Tacca for helping us throughout this year-long process and providing us with exceptional feedback and support when needed during our research and design process. Lastly Christine Nezda and Alfonso Lopez at Raytheon who were instrumental in providing feedback on not only the engineering side of the project but project and team management throughout the duration of the project.

1 Introduction

The Raytheon Drone Showcase is an Unmanned Aerial System (UAS) competition between UTD and five other Texas universities. The purpose of this competition is for students to research, develop, integrate, and test UAS hardware and software. Each university was given a budget of $5,000 and a list of four challenges to complete. Teams were graded on their design, speed, and accuracy.

2 Challenge Approaches

2.1 Challenge 1

2.1.1 Challenge 1 Description

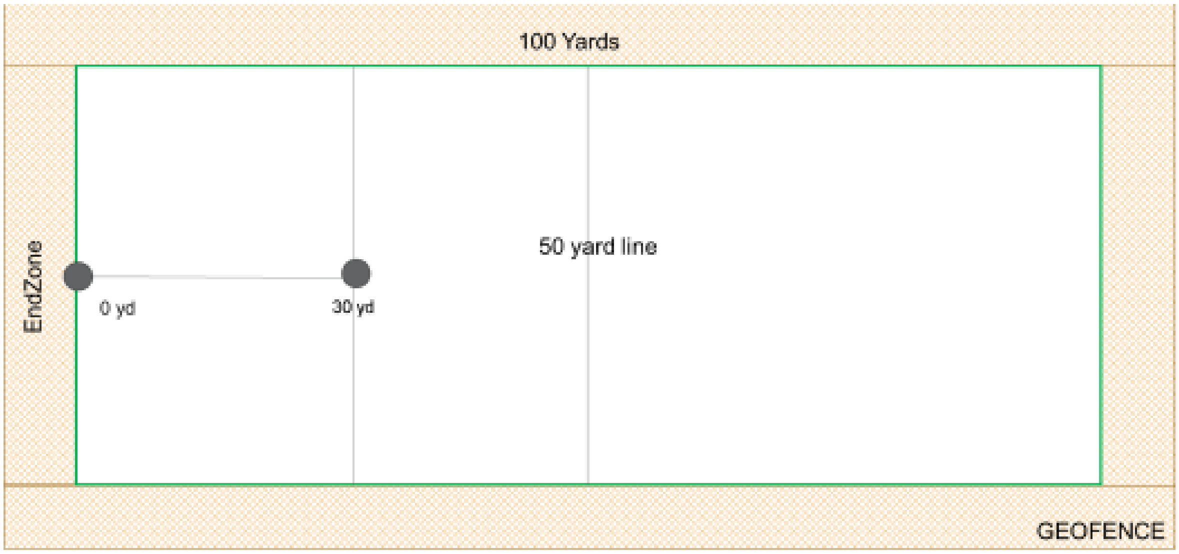


Figure 1: Challenge 1 illustration

The purpose of the first innovation showcase is to verify basic autonomy functionality of the UAS.

The UAS is expected to:

- Autonomously lift off at the Zero yard line on an End Zone
- Autonomously stay within a minimum of 20 feet and a maximum elevation of 30 feet
- Autonomously move across the field and land on the 30 yard line

Innovation Showcase Evaluation Criteria will be:

- How long did it take for the UAS to complete its task (timer starts on liftoff)
- Accuracy (i.e. distance) to landing spot
- Staying within elevation

2.1.2 Challenge 1 Approach

Our original approach to this challenge was to use our object detection with our onboard camera to detect the 30 yard line to land on. However, that was deemed too complicated and we went for a simpler approach using a GPS waypoint. We used Real-time kinematic positioning (RTK) to enhance GPS positioning. Unlike traditional GPS RTK GPS uses two modules, one that is installed on the vehicle and a base station placed at a surveyed location. The base station is able to send realtime location corrections to the vehicle's module. Thus the approach used for challenge 1 was to use an RTK GPS in order to get centimeter level accuracy and land directly on the 30 yard line.

2.2 Challenge 2

2.2.1 Challenge 2 Description

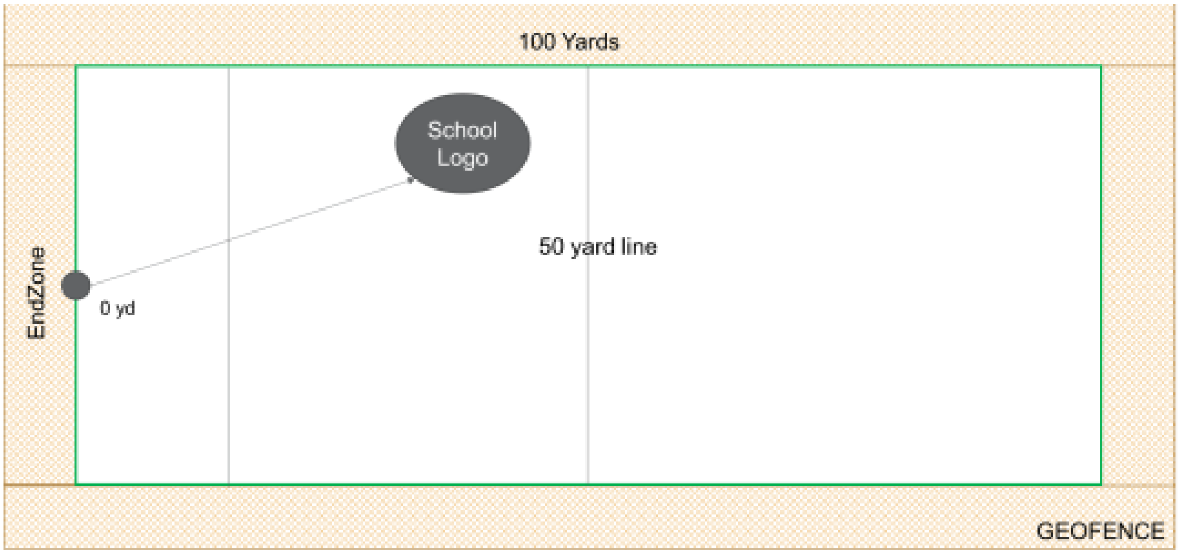


Figure 2: Challenge 2 illustration

The purpose of the second Innovation Showcase is for the drone to autonomously lift off and land on a designated spot (school logo) which will be randomly placed by the corporate sponsor before the 50- yard line. Other school logos may be on the field at the same time.

The UAS is expected to:

- Autonomously lift off at the Zero yard line on an EndZone (timer starts on liftoff)
- Autonomously stay within a minimum of 20 feet and a maximum elevation of 50 feet while searching for the logo
- Autonomously land on school logo

Innovation Showcase Evaluation Criteria will be:

- How long did it take for the UAS to complete its task (timer starts on liftoff)
- Accuracy (i.e. distance) to landing spot
- Staying within elevation
- See logo criteria in rules section for definition and constraints on the logo

2.2.2 Challenge 2 Approach

An example of the logo we designed for this challenge can be seen below in **Figure 3,** the logo was designed primarily as an Aruco marker which is quick and easy to detect using OpenCV. The pink around the outside allowed the use of a hue filter, so that colors other than pink would be ignored.

The algorithm for finding this logo is outlined as so:

1. Take off to the center of the half of the football field with the camera at 45°.
2. Do a 360-yaw spin to scan the entirety of the football field around the drone, noting geo-coordinates relative to the drone of Hue POIs (positions of interest). It knows a fairly accurate GPS coordinate of the POI by knowing the camera angle, camera lens data, and current drone GPS location from the flight controller.
3. Fly to each POI and scan the Aruco code, having the camera facing down.
4. If it's our logo then land! If not repeat step 3.

Additionally, a redundant approach that is not reliant on hue detection was implemented:

1. Rotate camera down
2. Fly in a configurable raster pattern over the entire search area
3. If logo is detected below, initiate landing process

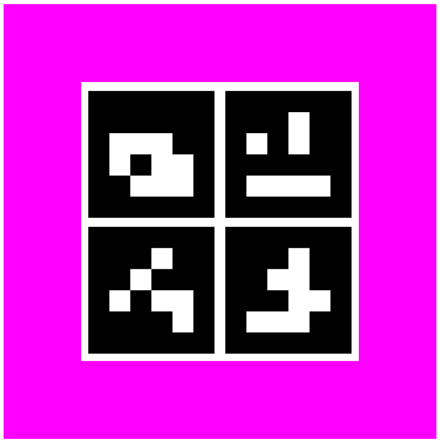


Figure 3: Aruco marker logo used for challenge 2

2.3 Challenge 3

2.3.1 Challenge 3 Description

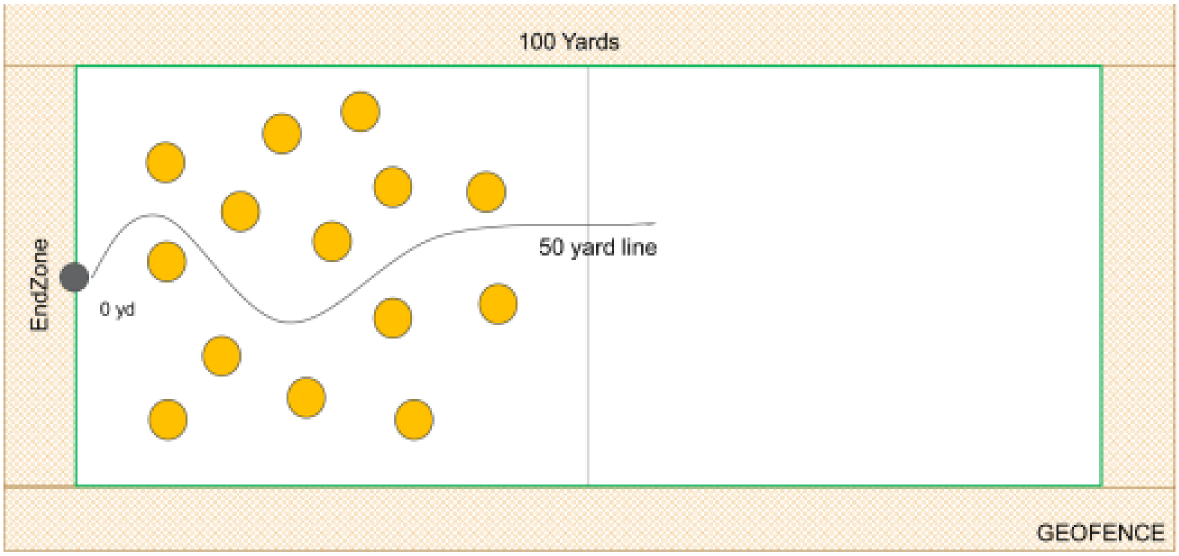


Figure 4: Challenge 3 illustration

The purpose of the third Innovation Showcase is for the drone to autonomously navigate through obstacles in its path then crossing a “finish line”.

The UAS is expected to:

- Autonomously lift off at the Zero yard line on an EndZone (timer starts on liftoff)
- Autonomously stay within a maximum elevation of the height of stationary obstacles o e.g. the drone should not be able to “see” over the agility stick, and should stay slightly lower than the height of the agility stick
  - The height has been determined at approximately 5 ft
  - The obstacles will consist of soccer agility sticks covered by red pool noodles (approximate width 3.5 inches)

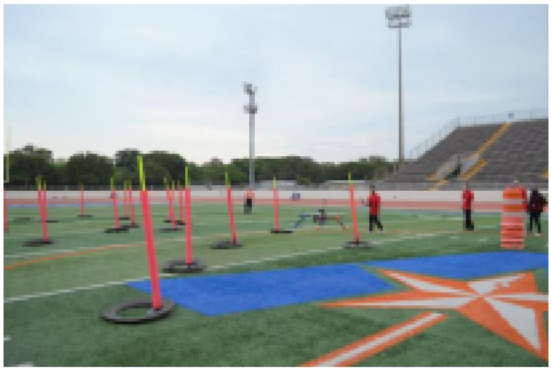


Figure 5: Example of agility sticks used for obstacle course

Innovation Showcase Evaluation Criteria will be:

- How long did it take for the UAS to complete its task (timer starts on liftoff)
- Obstacle Avoidance
- Staying within elevation

2.3.2 Challenge 3 Approach

The original approach for this challenge was to use the stereo camera for depth sensing. We developed a script to find the object with the closest distance in each frame, and report that distance to the drones flight computer. This script did work, however, we opted to use a 350 degree LIDAR to complete this challenge. We also used a downwards facing LIDAR to help maintain our altitude, compensating for altitude GPS drift.

2.4 Challenge 4

2.4.1 Challenge 4 Description

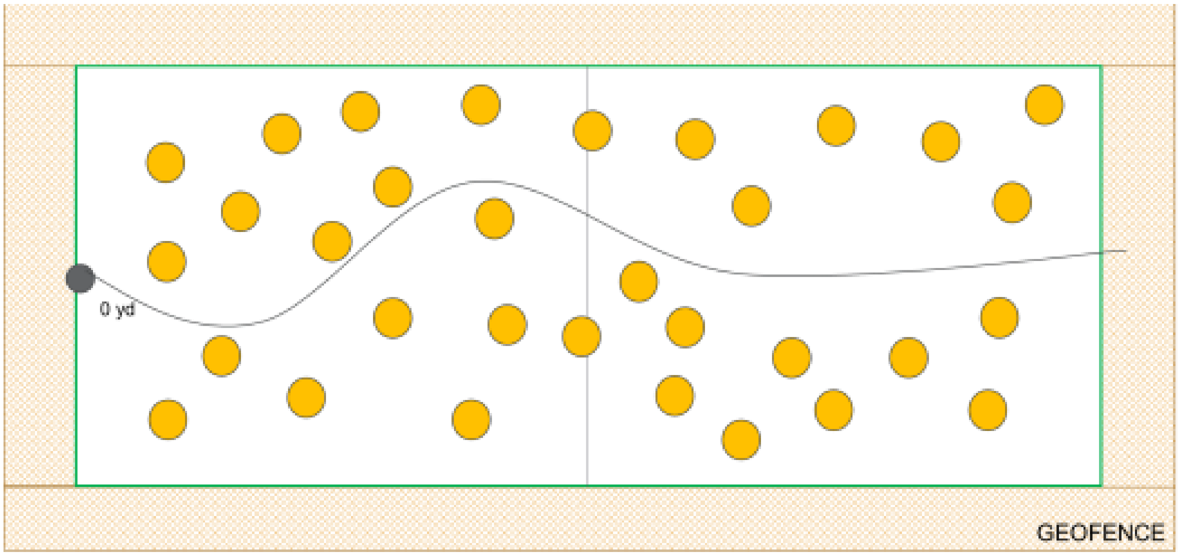


Figure 6: Challenge 4 illustration

This showcase is expected to take place inside a facility that has limited to no GPS signal available. The drone will need to understand its location relative to the area being flown in, and understand its mission to navigate from one point to another while avoiding obstacles at a certain height.

The UAS is expected to:

- Autonomously lift off at the "start line" of the enclosed space (timer starts on liftoff) • Autonomously stay within a maximum elevation of the height of stationary obstacles
    - e.g. the drone should not be able to "see" over the agility stick, and should stay slightly lower than the height of the agility stick
    - The height has been determined at approximately 5 ft

- ○ The obstacles will consist of soccer agility sticks covered by red pool noodles (approximate width 3.5 inches). These are the same obstacles as in Challenge 3

Innovation Showcase Evaluation Criteria will be:

- How long did it take for the UAS to complete its task
- Obstacle Avoidance
- Staying within elevation
- GPS Disabled

2.4.2 Challenge 4 Approach

For challenge 4 we had originally planned the same solution as challenge 3, use the stereo camera for depth sensing but ultimately used the 350 degree LIDAR. However, since we were not allowed any GPS data we also used the stereo camera to send visual odometry and pose information to the flight computer. Odometry allows the drone to know its change in position from the start, and pose allows the drone to know its orientation.

3 Simulation

3.1 Ardupilot SITL

Software in the loop (SITL) allows Ardupilot firmware and flight simulation without drone hardware. This is used to test scripts in a controlled and low risk environment before proceeding with development. SITL can be used on several operating systems to simulate different vehicles but this section will cover multi-rotor aircraft simulated on Ubuntu. It is important to point out SITL simulation only provides the user with flight controller simulation in an empty world. This allows us to simulate basic tasks like GPS waypoint missions, and flight procedures.

With SITL we are able to emulate an ardupilot flight controller, which we then run our Python Dronekit scripts on. We started with this to learn how to use Dronekit, then moved on to simulating Challenge 1. After we completed this simulation we used what we learned to create a framework Python script that all challenges would use, simplifying our development for interfacing with the drone.

3.2 Gazebo and Robot Operating System (ROS)

Gazebo is an open source 3D robotics simulator that enables users to simulate complete 3D worlds and sensors. We used this in conjunction with Ardupilot SITL to simulate a drone flying in a 3D world.

Robot Operating System (ROS) enabled us to simulate various sensors like a 360 LiDAR and cameras for Challenge 2 simulation. This worked by handling the sensor data from Gazebo, and translating it to the Ardupilot protocol, MAVLink, using the plugin MAVROS which sent the data to the flight controller. Once this was complete we were able to mount a generic 360 LiDAR onto the simulation drone and start navigating obstacle courses using Ardupilots built in obstacle avoidance.

3.3 Blender and Gazebo Worlds

Natively, Gazebo does not support an easy way to create custom worlds with custom textures within the software. We wanted to create our own, to scale, football field in Gazebo to simulate on. For this reason we turned to using Blender (an open source 3D computer graphics toolset) to create our Football field, then importing it using a Blender plugin named blender_gazebo. Initially we started with a generic football field with computer made textures, then later when working with cameras we replaced the field with a satellite image of the football field we would be competing on.

3.4 Challenge 1 Simulation

Challenge 1 was initially simulated using Ardupilot SITL. Once the SITL drone was running we ran our Python script, which connected to the drone and commanded it to liftoff, move 30 yards forward, then land. This was considered a successful simulation. Later on, once we had Gazebo, we redid this simulation in a 3D environment, but there was no benefit to be found.

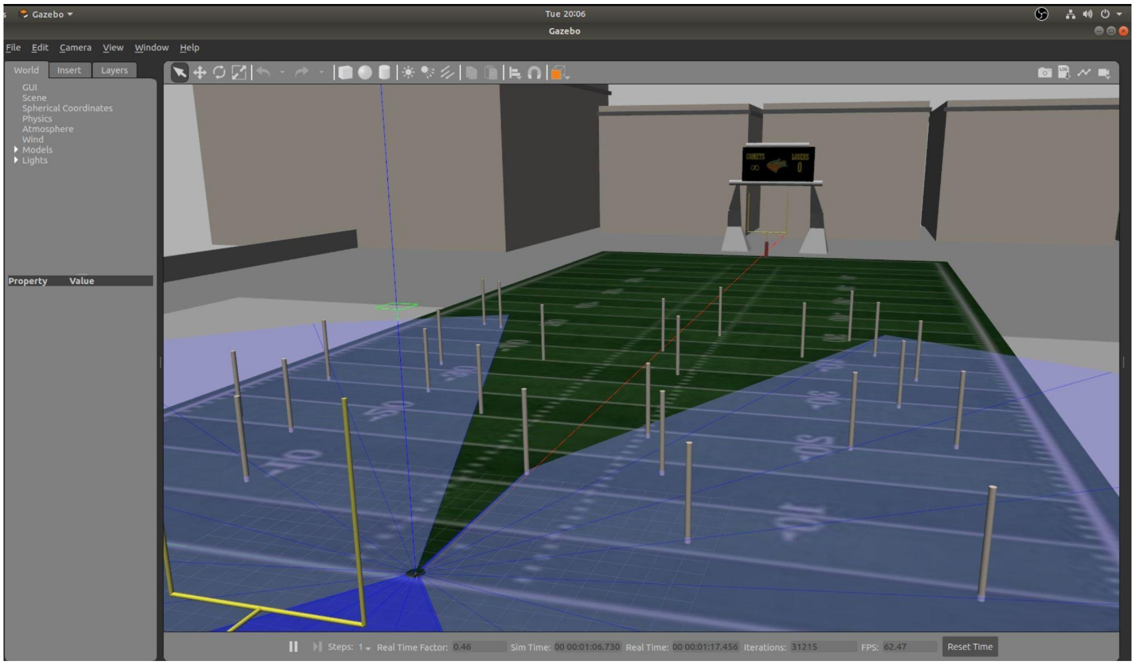
3.5 Challenge 2 Simulation

Challenge 2 required SITL, Gazebo, ROS, and a world with target logos placed in various locations. At the time, it was assumed that two cameras would be used so the built in Iris drone model was modified with an additional camera. Unfortunately, to use our python 3 code with ROS required the setup and configuration of ROS Noetic on a separate computer than our existing ROS Melodic system. Once configured, with the aid of specially built log analysis tools the hue search and visit algorithm was debugged and tested successfully with multiple targets. The world had multiple markers added to it, and by changing the target IDs we were able to verify that only the designed logo would be landed on.

3.6 Challenge 3 & 4 LiDAR Simulation

Challenge 3 required SITL, Gazebo, ROS, and a world with obstacles for the drone to avoid. We initially started with a simple world that contained a large cylinder to avoid so we could get the sensors working. Once we had the pipeline of 360 LiDAR feeding its data to Ardupilots obstacle avoidance, we focused on creating a realistic football field with the obstacles we would need it to avoid.

Our custom blender world allowed us to randomly place a set amount of obstacles that were the same dimensions of the pool noodles. Not manually placing the obstacles in the simulation world sped up our simulation process. Simulating this challenge gave us various Ardupilot parameters that we would use once we shifted to physical simulation.



Steps: 1 Real Time Factor: 0.46 Sim Time: 00 00:01:06.730 Real Time: 00 00:01:17.456 Iterations: 31215 FPS: 62.47

Figure 7: Challenge 3 Gazebo World with Obstacles

3.7 Challenge 3 & 4 ZED Simulation

Challenge 3 and 4 required some way to detect obstacles, our initial implementation was using the ZED stereo camera to detect obstacles and send the distance data to the FCU. We used depth image processing with depth maps generated by the ZED. From those depth maps we were able to generate a point cloud for each image and calculate the distance to the closest obstacle in the middle third of the image as shown below in the middle three sectors of **Figure 8**. The implementation calculates the distance for every pixel in the “x” or width of the image along a set “y” or height, since the objects we had to detect were small in width but tall so we did not need to do calculations along the entire “y” or height of the image. For positional tracking we used the built in visual odometry and pose data acquisition functions built into the ZED.

Using Robot OS and MAVROS we then published the data packets in a format that the flight controller could interpret for obstacle distance, visual odometry, and pose data from the ZED stereo camera. We were able to successfully send the obstacle distance data over to the FCU and have the distances show up in Mission Planner’s proximity viewer, however we ran into complications towards competition day.

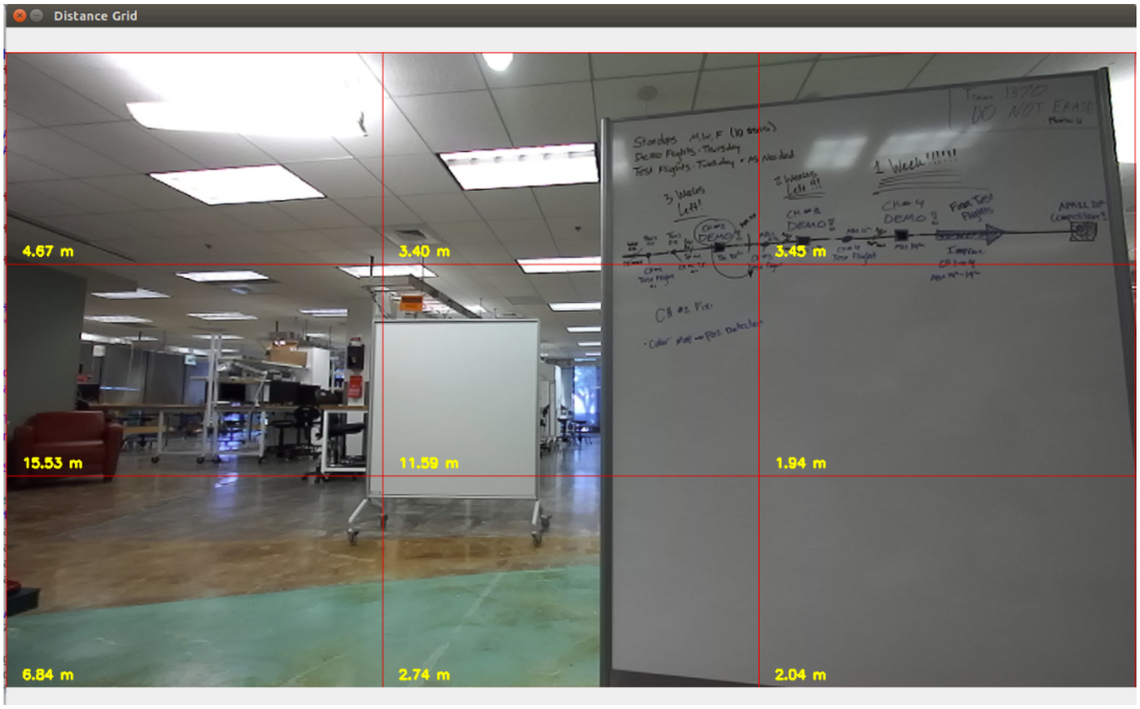


Figure 8: ZED Depth Sensing Object Detection (Old Run of the Script)

4 Hardware Integration

4.1 Design Criteria

This section explores the design considerations required to build a robust UAS and explains the decisions our team made in our drone's design.

4.1.1 Powertrain

Through testing with an initial quadcopter prototype, our team learned that a reliable powertrain is the most important system on an autonomous drone. If the powertrain fails in any way, has a slow PID response, or induces excessive vibrations, none of the drone's sensors would provide useful data. Through an iterative design approach, our team landed on a hexacopter design that provided a stable, reliable, and maneuverable base for our sensors and compute package.

There are a lot of variables that go into a powertrain's design, and each must be optimized to find a sweet spot between efficiency and performance. To tackle this problem, our team created a powertrain philosophy to validate our design: payload drives frame size which restrains propeller length driving motor spec which determines battery spec that must be adjusted to optimize flight time and maneuverability per unit weight of battery.

Walking through each step:

1. **Payload:** First we needed to determine the components that make up our drone's flight systems and sensor suite. This will be discussed in detail in the following sections. We then approximated the weight of our drone's components for the heaviest scenario feasible. This included weights of components we did not have in the design at the time, such as a large camera lens and gimbal system in the case that we needed to upgrade our camera system and further reduce vibrations. We could not afford the time or cost of a complete powertrain redesign for a heavier payload, so we used large margins to ensure our drone could handle anything we decided to strap to it. While this approach was more expensive in the cost of a single drone, the tradeoff for a robust design was well worth the cost of larger motors and higher capacity batteries.

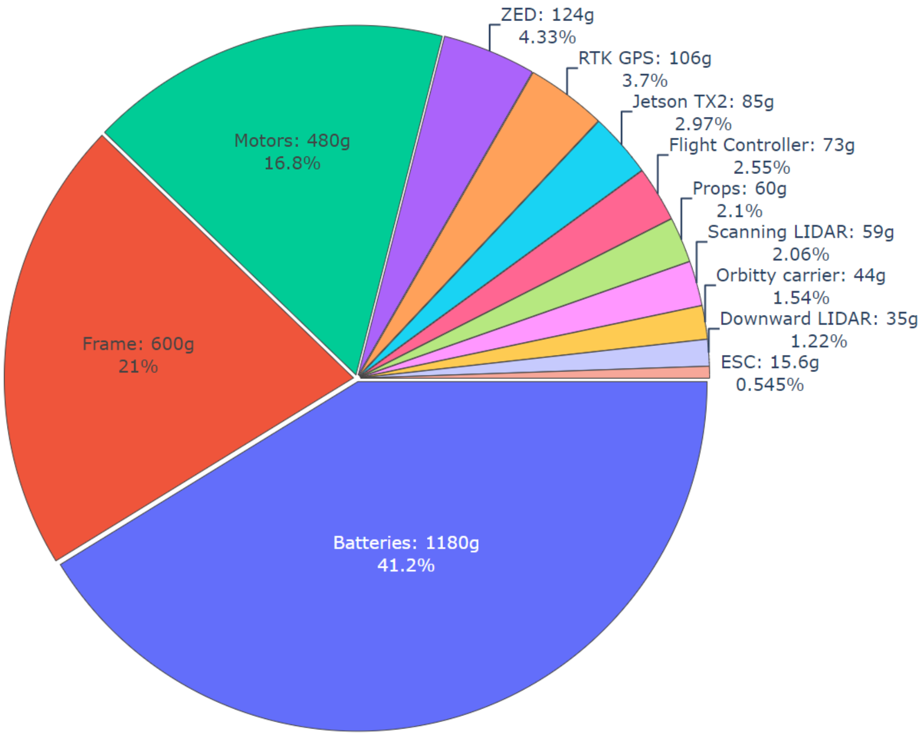


Figure 9: Pie Chart of All Components Weight

2. **Frame Size:** We visually calculated the payload's formfactor through CAD design and concluded that everything would fit on a 250mm plate. We then looked for hexacopter frames we could purchase and found that a frame with a 250mm plate has a 690mm formfactor. This is the measurement between opposite motors from the center of one stator to the other.

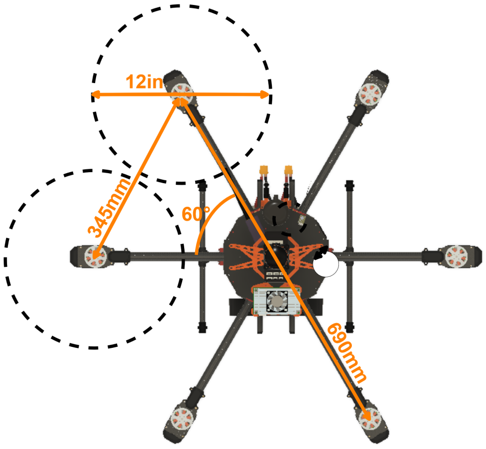
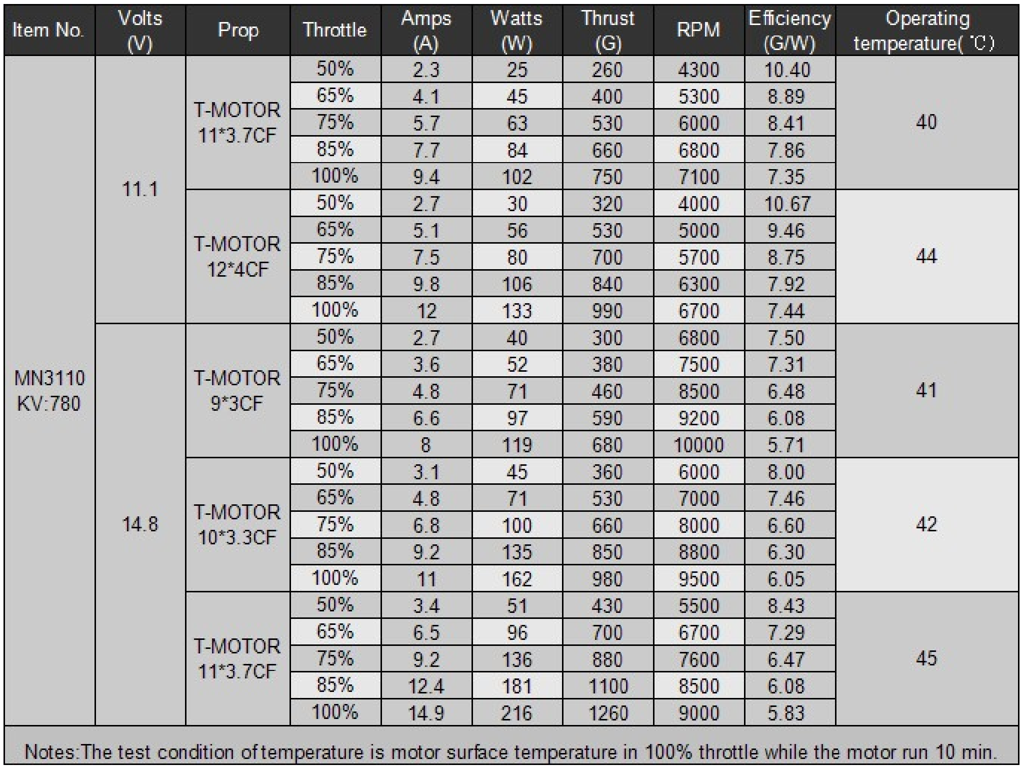


Figure 10: Drivetrain Measurements

3. **Propeller Length:** Given our frame size of 690mm, the distance between two adjacent motors is 345mm (approximately 13 inches). leaving a margin for propeller clearance, the maximum propeller size we could use was 12 inches. Now we could find a propeller with a length of 12 inches or less and a pitch steep enough to generate sufficient thrust.

4. **Motor Spec:** With our propeller limits defined, we could start looking at motor data sheets to find a motor that could generate enough thrust to give our hexacopter a 3:1 thrust to weight ratio. A thrust to weight ratio under 3:1 would not provide sufficient maneuverability and would limit our maximum pitch. Maximum pitch is important for airspeed and maintaining ground speed in high winds. A 3:1 thrust to weight ratio puts our targeted thrust at 12 pounds after adding generous weight margins for heavier hardware and battery weight. According to the manufacturers data sheet (figure #), T-Motor's MN3110 motor driving an 11 inch propeller with a 3.7 inch pitch can generate 1260 grams of thrust at full throttle, and 430 grams at half throttle when supplied 14.8 volts. At full throttle, the motors draw 14.9 amps each. It's also important to note that the motor is more efficient at lower RPM's, so the lower our throttle is at hover, the longer our flight time. Using all 260g of thrust from each of the 6 motors, we would have 7,560g (16 pounds) of thrust at our disposal. This significantly exceed our design requirements, but provided extra efficiency and a robust design.

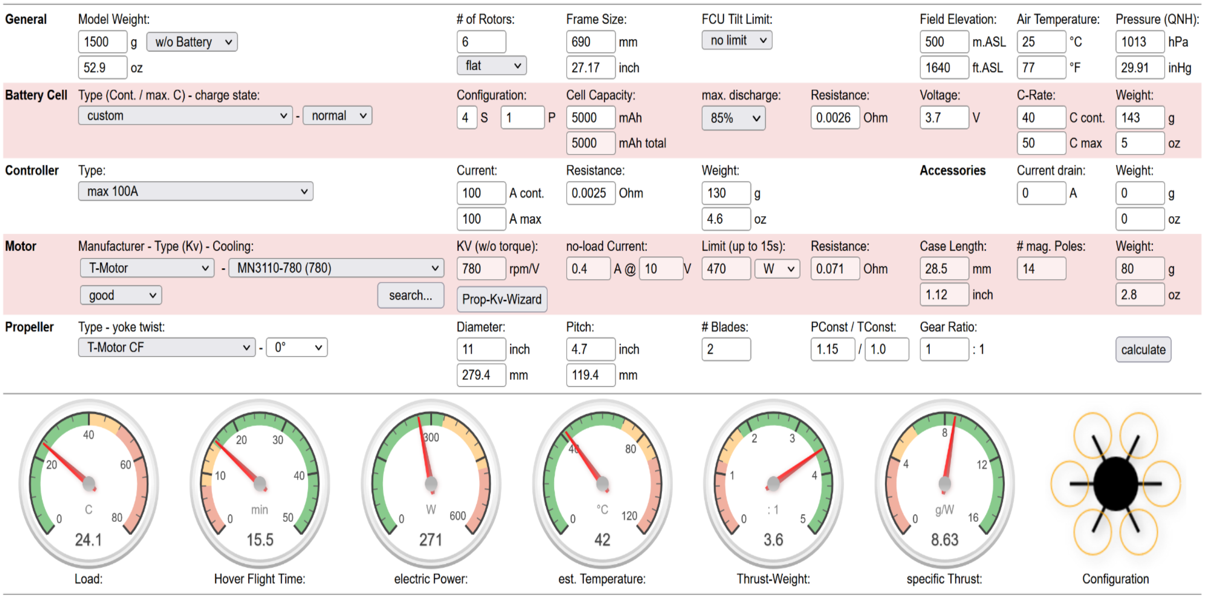


Item No.	Volts (V)	Prop	Throttle	Amps (A)	Watts (W)	Thrust (G)	RPM	Efficiency (G/W)	Operating temperature(℃)
MN3110 KV:780	11.1	T-MOTOR 11*3.7CF	50%	2.3	25	260	4300	10.40	40
			65%	4.1	45	400	5300	8.89	
			75%	5.7	63	530	6000	8.41	
			85%	7.7	84	660	6800	7.86	
			100%	9.4	102	750	7100	7.35	
		T-MOTOR 12*4CF	50%	2.7	30	320	4000	10.67	44
			65%	5.1	56	530	5000	9.46	
			75%	7.5	80	700	5700	8.75	
			85%	9.8	106	840	6300	7.92	
			100%	12	133	990	6700	7.44	
	14.8	T-MOTOR 9*3CF	50%	2.7	40	300	6800	7.50	41
			65%	3.6	52	380	7500	7.31	
			75%	4.8	71	460	8500	6.48	
			85%	6.6	97	590	9200	6.08	
			100%	8	119	680	10000	5.71	
		T-MOTOR 10*3.3CF	50%	3.1	45	360	6000	8.00	42
			65%	4.8	71	530	7000	7.46	
			75%	6.8	100	660	8000	6.60	
			85%	9.2	135	850	8800	6.30	
			100%	11	162	980	9500	6.05	
		T-MOTOR 11*3.7CF	50%	3.4	51	430	5500	8.43	45
			65%	6.5	96	700	6700	7.29	
			75%	9.2	136	880	7600	6.47	
			85%	12.4	181	1100	8500	6.08	
			100%	14.9	216	1260	9000	5.83	
Notes:The test condition of temperature is motor surface temperature in 100% throttle while the motor run 10 min.									

Figure 11: Motor Spec Sheet

5. **Battery:** Now that we knew the power demands from each of our motors, we calculated the battery requirements for our drone. We wanted a 15 minute flight time and a 50% margin on the batteries max continuous current draw. Our drone's motors draw approximately 15 amps each for a total of 90 amps at 14.8 volts, or 1330 watts. The power draw from our sensor suite is negligible. This power requirement can be met by standard and readily available 4-cell lithium polymer batteries with a capacity of 5000mAh and a C-rating of 50.

6. **Design Validation with eCalc:** Before purchasing parts, our team used a simulation software called eCalc that uses a mathematical model to estimate the expected performance of our drones powertrain.



General — Model Weight: 1500 g w/o Battery; 52.9 oz — # of Rotors: 6; flat — Frame Size: 690 mm; 27.17 inch — FCU Tilt Limit: no limit — Field Elevation: 500 m.ASL; 1640 ft.ASL — Air Temperature: 25 °C; 77 °F — Pressure (QNH): 1013 hPa; 29.91 inHg

Battery Cell — Type (Cont. / max. C) - charge state: custom - normal — Configuration: 4 S 1 P — Cell Capacity: 5000 mAh; 5000 mAh total — max. discharge: 85% — Resistance: 0.0026 Ohm — Voltage: 3.7 V — C-Rate: 40 C cont.; 50 C max — Weight: 143 g; 5 oz

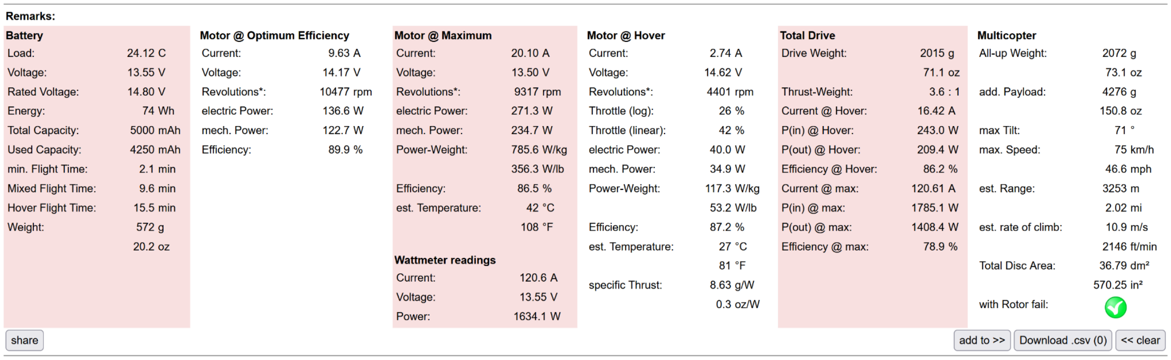
Controller — Type: max 100A — Current: 100 A cont.; 100 A max — Resistance: 0.0025 Ohm — Weight: 130 g; 4.6 oz — Accessories — Current drain: 0 A — Weight: 0 g; 0 oz

Motor — Manufacturer - Type (Kv) - Cooling: T-Motor - MN3110-780 (780); good; search... — KV (w/o torque): 780 rpm/V; Prop-Kv-Wizard — no-load Current: 0.4 A @ 10 V — Limit (up to 15s): 470 W — Resistance: 0.071 Ohm — Case Length: 28.5 mm; 1.12 inch — # mag. Poles: 14 — Weight: 80 g; 2.8 oz

Propeller — Type - yoke twist: T-Motor CF - 0° — Diameter: 11 inch; 279.4 mm — Pitch: 4.7 inch; 119.4 mm — # Blades: 2 — PConst / TConst: 1.15 / 1.0 — Gear Ratio: 1 : 1 — calculate

Load:	Hover Flight Time:	electric Power:	est. Temperature:	Thrust-Weight:	specific Thrust:	Configuration
24.1 C	15.5 min	271 W	42 °C	3.6 : 1	8.63 g/W	

Figure 12: eCalc Powertrain Estimate

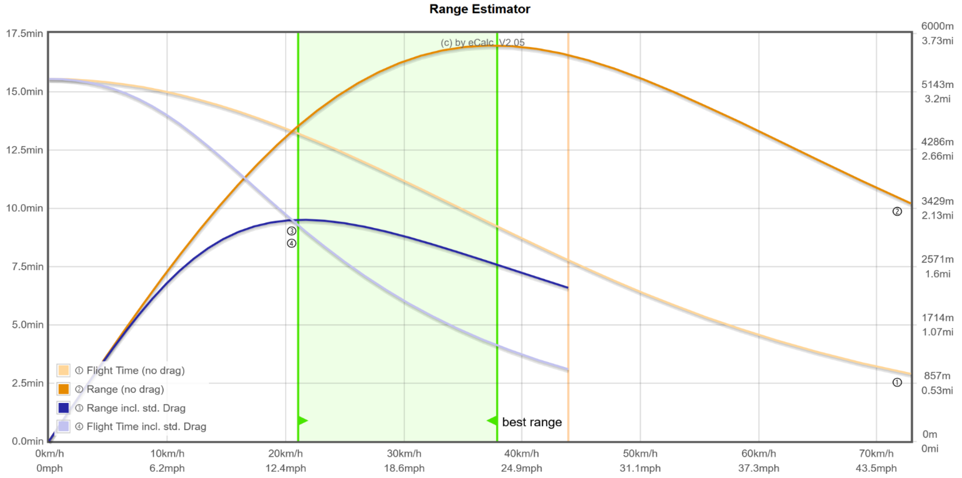


Remarks:

Battery		Motor @ Optimum Efficiency		Motor @ Maximum		Motor @ Hover		Total Drive		Multicopter	
Load:	24.12 C	Current:	9.63 A	Current:	20.10 A	Current:	2.74 A	Drive Weight:	2015 g	All-up Weight:	2072 g
Voltage:	13.55 V	Voltage:	14.17 V	Voltage:	13.50 V	Voltage:	14.62 V		71.1 oz		73.1 oz
Rated Voltage:	14.80 V	Revolutions*:	10477 rpm	Revolutions*:	9317 rpm	Revolutions*:	4401 rpm	Thrust-Weight:	3.6 : 1	add. Payload:	4276 g
Energy:	74 Wh	electric Power:	136.6 W	electric Power:	271.3 W	Throttle (log):	26 %	Current @ Hover:	16.42 A		150.8 oz
Total Capacity:	5000 mAh	mech. Power:	122.7 W	mech. Power:	234.7 W	Throttle (linear):	42 %	P(in) @ Hover:	243.0 W	max Tilt:	71 °
Used Capacity:	4250 mAh	Efficiency:	89.9 %	Power-Weight:	785.6 W/kg	electric Power:	40.0 W	P(out) @ Hover:	209.4 W	max. Speed:	75 km/h
min. Flight Time:	2.1 min				356.3 W/lb	mech. Power:	34.9 W	Efficiency @ Hover:	86.2 %		46.6 mph
Mixed Flight Time:	9.6 min			Efficiency:	86.5 %	Power-Weight:	117.3 W/kg	Current @ max:	120.61 A	est. Range:	3253 m
Hover Flight Time:	15.5 min			est. Temperature:	42 °C		53.2 W/lb	P(in) @ max:	1785.1 W		2.02 mi
Weight:	572 g				108 °F	Efficiency:	87.2 %	P(out) @ max:	1408.4 W	est. rate of climb:	10.9 m/s
	20.2 oz					est. Temperature:	27 °C	Efficiency @ max:	78.9 %		2146 ft/min
				Wattmeter readings			81 °F			Total Disc Area:	36.79 dm²
				Current:	120.6 A	specific Thrust:	8.63 g/W				570.25 in²
				Voltage:	13.55 V		0.3 oz/W			with Rotor fail:	✓
				Power:	1634.1 W						

share — add to >> — Download .csv (0) — << clear

Figure 13: eCalc Remarks/Analysis



`

Figure 14: eCalc Estimated Velocity (Above)

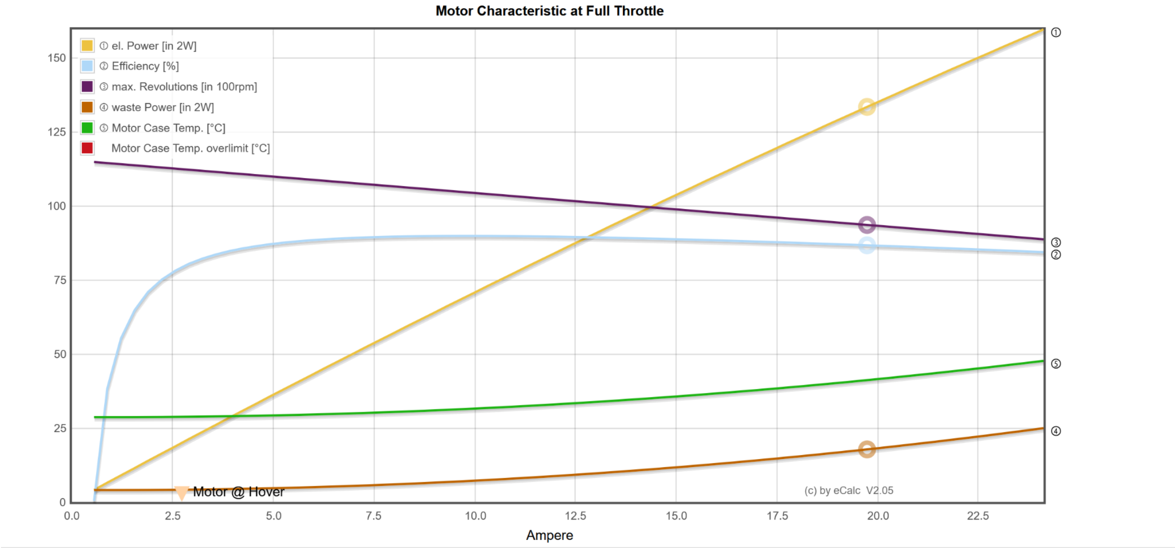


Figure 15: eCalc Motor Characteristic Diagram

These results matched our powertrain calculations, giving us 15 minutes of flight time and plenty of margin for our powertrains energy demands.

4.1.2 Flight Controller

A powertrain is only as reliable as its least reliable part, so it was important to have a high-quality flight controller with redundant internal measuring units (IMUs). The Pixhawk cube was the best choice with three redundant temperature controlled IMUs, two of which are mechanically vibration-isolated. The cube also has two barometers and one magnetometer to help it orient itself in space. An additional magnetometer is housed inside a separate GPS module for additional redundancy. The pixhawk cube is also compatible with Ardupilot, an open source autonomous vehicle firmware that we designed our sensor suite to be compatible with. Our flight controller acts as the brains for mechanical parts on our drone, taking instructions from our companion computer and translating them into instructions for our electronic speed controllers. The flight controller also connects our GPS, RC override receiver, mavlink telemetry, 360 lidar, lidar altimeter, battery voltage monitor, and servo motors. The Pixhawk Cube running Ardupilot proved to be extremely reliable and could take over a controlled landing of our drone in the case of errors in our companion computer's code through development.

4.1.3 Camera

After testing with the Raspberry Pi HQ camera, we determined we needed a camera with a larger field of view, better auto exposure, higher resolution, and more software controllable options. Additionally, we determined that using an off the shelf optical odometry system would be best for challenge 4. As such, we selected the ZED stereo camera as it meets all of these criteria while providing an extremely easy to use python SDK. The ZED camera provides high quality images with minimal configuration, while also exposing powerful configuration options when needed to take the best advantage of its excellent hardware.

4.1.4 Companion Computer

After testing with a Raspberry Pi 4, we knew we would need as much visual processing power as possible to use the full capabilities of the ZED 2 camera which requires NVIDIA CUDA. We opted to use an Nvidia Jetson TX2 mounted on an orbitty carrier board, as a backup we also bought an Nvidia Jetson Nano if we experienced any issues with the TX2. The only other requirements for the computer were that it would need to be able to run Python and Linux.

4.1.5 360 Degree LiDAR

Selecting the right LiDAR initially had the following criteria: cost, weight, and native Ardupilot integration. We started with a cheap RP LiDAR A1 mounted on our prototype drone that integrated natively with Ardupilots obstacle avoidance, which performed adequately. One major concern was that there was a lot of interference from the sun during the day, making it unusable so our outdoor LiDAR testing was done after the sunset.

Learning from this we bought the newer RP LiDAR A2, but after experiencing sun interference we quickly shifted to the Lightware SF45/B 350 Degree LiDAR, which also natively integrates with Ardupilot. The SF45/B provided much higher LiDAR resolution and range at the cost of 10 degrees, but we ignored 40 degrees behind the drone anyways for the GPS mast. It was also much lighter than any of the other scanning LiDAR products.

4.1.6 GPS Denied Altimeter

For Challenge 4 we would not be able to know our Altitude without the GPS. The Ardupilot flight controller uses a mixture of barometer (not very accurate) and GPS sensors to find its altitude. To remedy this we knew we would need some kind of downward facing sonar or LiDAR. Because sonar is disrupted by the drones prop wash we opted for a downward facing LiDAR.

By the time we selected this part we used all of our serial ports on the flight controller, requiring our altimeter to be I2C compatible. We chose to use a Lightware SF11/C, which was rated for reading distance up to one hundred meters away. While expensive it was very accurate, even with interference from the sun which many LiDAR sensors are prone to. Mounted facing downwards and configured in ardupilot it was an ideal solution.

4.2 Components

4.2.1 Subsystem 1 “Ground Control Station”

1. Computer with Mission Planner Software
2. Telemetry Radio
3. RTK Base Station

4.2.2 Subsystem 2 “Drone”

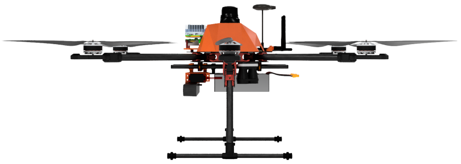


Figure 16: Side Render of Drone

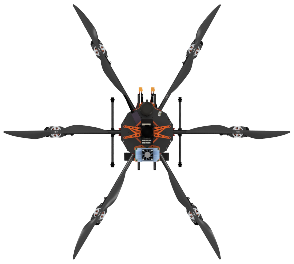


Figure 17: Top-down Render of Drone

1. 1x PixHawk flight controller
2. 1x Telemetry radio
3. 1x GPS Module
4. 6x Motors
5. 6x ESCs
6. 1x Nvidia Jetson TX2
7. 2x LiPo batteries
8. 1x ZED Stereo Camera
9. 1x 350 Degree Lightware LiDAR
10. 1x Servo Motor
11. 1x One directional Lightware LiDAR

4.3 Fabrication/Assembly process

4.3.1 Powertrain Assembly

The powertrain has several sensitive sensors that are negatively affected by vibrations, electromagnetic interference, and spikes in voltage. To get the best performance from our drone, we minimized vibrational coupling through custom 3D printed mounts, decreased voltage spikes induced by the motor's inductive loads with low ESR (equivalent series resistance) capacitors, and reduced electromagnetic interference with standoffs and short cable runs.

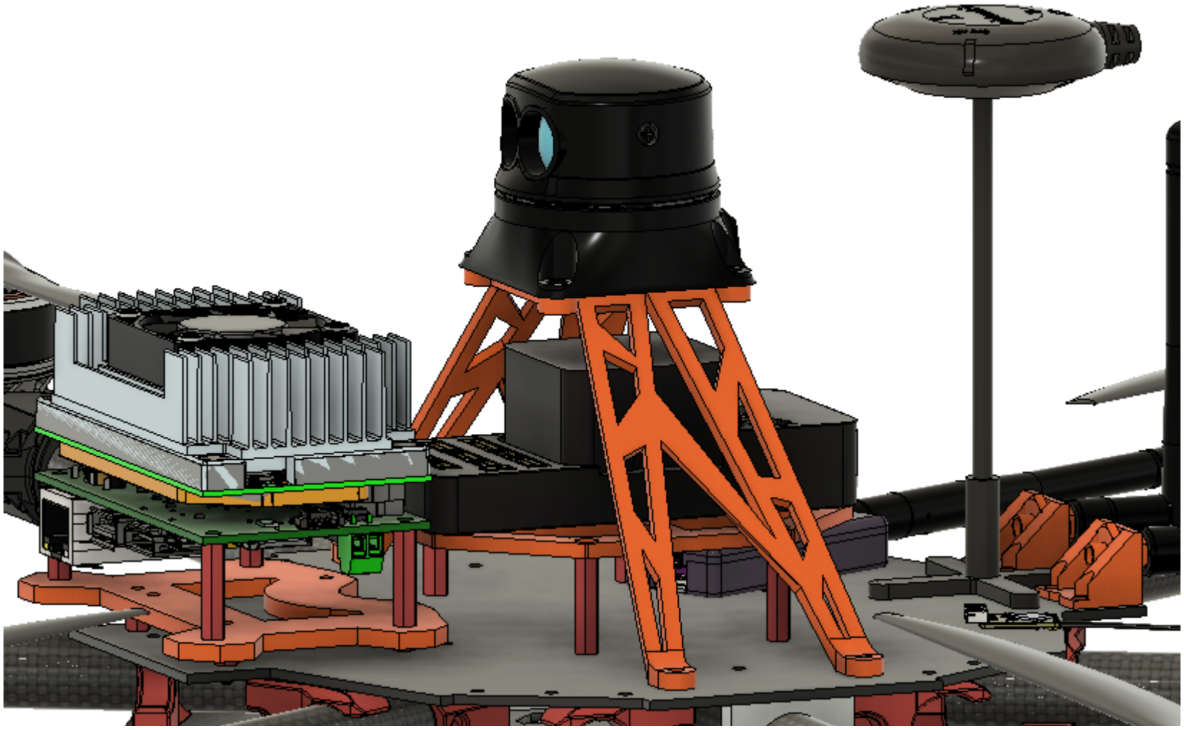


Figure 18: Powertrain Model in CAD

As shown in **Figure 18**, major sources of vibration, like the 360 lidar, are mechanically isolated from the flight controller and the GPS/compass module is on a high standoff to reduce electromagnetic interference. Capacitors rated for twice the drone's battery voltage (35V, 470uF) were wired in parallel across each of the six electronic speed controller's power terminals to absorb spikes in voltage from the motors.

4.3.2 Component Wiring

Figure 19 below outlines our complete wiring diagram, including all of our components and flight controller wiring harness connections.

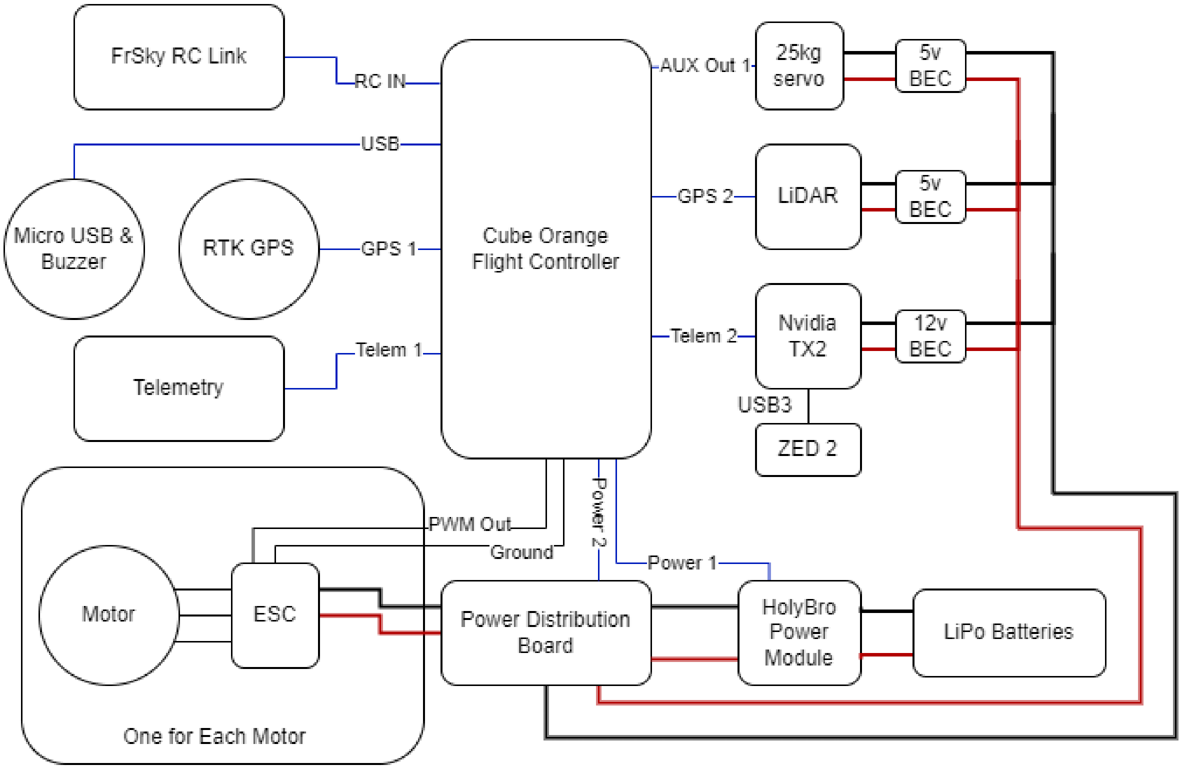


Figure 19: Hexacopter Wiring Diagram

5 Testing

5.1 Overview

The testing procedure was developed using the requirements and tasks of each challenge specified in the statement of work. There is one general test case that covers all primary flight hardware and should always be completed before any other non-simulation tests. Simulation tests were used to test program logic and integration with Ardupilot. All other subsystem tests are covered by completing the challenge test that the subsystem was designed for.

5.2 Test and Verification Plan

5.2.1 Challenge 1 Simulation

Step	Test Setup	Action	Normal Indication	Notes	Pass/Fail
1	At simulation computer	Open four terminals by hitting “Ctrl-Alt-T + Ctrl-Shift-T” four times	Four terminals open		PASS
2	In terminal 1	Input “roslaunch helium stadium_empty.luanch”	Gazebo virtual environment loads displaying virtual football field with obstacles		PASS
3	In terminal 2	Input “sim_vehicle.py -v ArduCopter -f gazebo-iris”	“APM: EKF2 IMU1 tilt alignment complete” will be displayed within the terminal. Initialization is complete once “APM: EKF2 IMU1 is using GPS” is displayed		PASS
4	In terminal 3	Input “./QGroundControl.AppImage”	Q Ground Control launches		PASS
5	In terminal 4	Input “python challenge1.py”	Challenge 1 procedure runs and completes within		PASS

			Gazebo window. The drone shall takeoff, fly to the 30-yard line, and land.		

5.2.2 General Function Test

Step	Test Setup	Action	Normal Indication	Notes	Pass/Fail
1	At the drone	Remove props			PASS
2		Attach and connect batteries	Drone Powers on and a chime is heard		PASS
3	At Ground Control Station(GCS)	Open “Mission Planner” and navigate to the top right of the screen. Input COM port (usually3) and BAUD rate (57600). Click Connect	The GCS connects to the drone and displays telemetry data within Mission Planner window		PASS
4		Navigate to setup -> Mandatory Hardware-> Accel Calibration. Click “Calibrate Accel” to begin. Click “Done” when each position is reached			PASS
5	At the Drone	Follow the directions provided on the Calibrate Accel screen. The positions are: level, on right side, left side, nose down, nose up and back.	Drone accelerometer is calibrated		PASS
6	At the Drone	place the drone on flat level ground			PASS

7	At the GCS	On the Accel Calibration page. Click “Calibrate Level”	Drone level is calibrated		PASS
8	At the GCS	Navigate to setup -> Mandatory Hardware-> Compass. Find “Onboard Mag Calibration” and click “Start”			PASS
9	At the Drone	Move along all three axes similar to the Accel Calibration step			PASS
10	At the GCS	Once Mag 1 and Mag 2 bars are filled click “accept” and follow the command to reboot.	Three rising tones are emitted, the aircraft reboots, and the compass direction is accurate		PASS
11	At the Drone	Reattach the props			PASS
12		Perform a low altitude Manual Test flight in “PosHold” mode	All motors arm. Drone hovers and holds at GPS position		PASS
13	At the GCS	Watch the Telemetry and Messages windows on Mission Planner	Drone displays the correct position on map, Arm status, and mode. No error messages or faults		PASS

5.2.3 Challenge 1 Test

Step	Test Setup	Action	Normal Indication	Notes	Pass/Fail
1	General Function Test complete and	Connect to the same wireless network as the drone. Open a “Putty” window and input	Putty window connects to TX2 and prompts for Username.		

	GCS connected. At Maintenance PC	“tx2.local” as the hostname and “22” (SSH) as the port			
2		Input “tx2” as username and the default team password	terminal opens to TX2 home directory		PASS
3		In the terminal input ”cd Desktop/Temoc-Air”	directory path changes to the specified folder		PASS
4	At the Drone	Place the drone on the 0 yard line, centered on the quarter field sector,and facing towards the 30 yard line. Verify pilot is ready to start.	Drone displays accurate location and compass heading on map.		PASS
5	At Maintenance PC	In the terminal input “python 3.9 Challenge1.py” . Watch terminal output for errors	Drone arms, flies to 20 ft. altitude, flies forward 30 yards, and lands on 30 yard line.		PASS
6	At the GCS	Monitor battery voltage, telemetry, and messages windows.	No faults or errors		PASS
7	At Maintenance PC	Upon completion or failure press Ctrl + C to end script	Script ends and returns to the Desktop/Temoc-Air directory		PASS

5.2.4 Challenge 2 Test

Step	Test Setup	Action	Normal Indication	Notes	Pass/Fail
1	General	Connect to the same wireless	Putty window connects to TX2 and prompts for		PASS

	Function Test complete and GCS connected. At Maintenance PC	network as the drone. Open a “Putty” window and input “tx2.local” as the hostname and “22” (SSH) as the port	Username.		
2		Input “tx2” as username and the default team password	terminal opens to TX2 home directory		PASS
3		In the terminal input ”cd Desktop/Temoc-Air”	directory path changes to the specified folder		PASS
4	At the Drone	Place drone on 0 yard line, centered on the quarter field sector,and facing towards the 50 yard line. Verify pilot is ready to start.	Drone displays accurate location and compass heading on map.		PASS
5	At Maintenance PC	In the terminal input “python 3.9 challenge_2_main.py” . Watch terminal output for errors	Drone arms, flies to 20 ft. altitude, begins grid search, detects logo, and lands on logo.		PASS
6	At the GCS	Monitor battery voltage, telemetry, and messages windows.	No faults or errors		PASS
7	At Maintenance PC	Upon completion or failure press Ctrl + C to end script	Script ends and returns to the Desktop/Temoc-Air directory		PASS

5.2.5 Challenge 3 Test

Step	Test Setup	Action	Normal Indication	Notes	Pass/Fail
1	General Function Test complete and GCS connected. At Maintenance PC	Connect to the same wireless network as the drone. Open a “Putty” window and input “tx2.local” as the hostname and “22” (SSH) as the port	Putty window connects to TX2 and prompts for Username.		PASS
2		Input “tx2” as username and the default team password	terminal opens to TX2 home directory		PASS
3		In the terminal input ”cd Desktop/Temoc-Air”	directory path changes to the specified folder		PASS
4	at the GCS	within Mission Planner navigate to the “Plan” tab.	Flight plan window is displayed		PASS
5		Right click on Map, select “Polygon->Draw Polygon” from the drop down.			PASS
6		On the right hand side of the screen find the dropdown that is displaying “MISSION” and change it to “FENCE”			PASS
7		Left click on the map to create a border that matches the boundaries of the obstacle course. Then click the polygon logon and select “Fence			PASS

		Inclusion” then click “Write”			
8		On the right hand side of the screen find the dropdown that is displaying “FENCE” and change it to “MISSION”			PASS
9		Left click point on map beyond obstacle course finish line with an altitude of 1.5 meters. Then Click “Write”			PASS
10	At the Drone	Place drone on 0 yard line, centered on the quarter field sector,and facing towards the obstacles. Verify pilot is ready to start.	Drone displays accurate location and compass heading on map.		PASS
11	At Maintenance PC	In the terminal input “vim challenge_3_4.py” . Ensure that “CURRENT_CHALLENGE” = 3.			PASS
12		In the terminal input “python 3.9 challenge_3_4.py” . Watch terminal output for errors	Drone arms, flies towards obstacle course, navigates through obstacle course and across finish line		PASS
13	At the GCS	Monitor battery voltage, telemetry, and messages windows.	No faults or errors		PASS
14	At Maintenance PC	Upon completion or failure press Ctrl + C to end script	Script ends and returns to the Desktop/Temoc-Air directory		PASS

15	At the Drone	Have the pilot manually land once the finish line is crossed	drone lands and disarms		PASS

5.2.6 Challenge 4 Test

Step	Test Setup	Action	Normal Indication	Notes	Pass/Fail
1	General Function Test complete and GCS connected. At Maintenance PC	Connect to the same wireless network as the drone. Open a “Putty” window and input “tx2.local” as the hostname and “22” (SSH) as the port	Putty window connects to TX2 and prompts for Username.		PASS
2		Input “tx2” as username and the default team password	terminal opens to TX2 home directory		PASS
3		In the terminal input ”cd Desktop/Temoc-Air”	directory path changes to the specified folder		PASS
4	at the GCS	within Mission Planner navigate to the “Plan” tab.	Flight plan window is displayed		PASS
5		Right click on Map, select “Polygon->Draw Polygon” from the drop down.			PASS
6		On the right hand side of the screen find the dropdown that is displaying “MISSION” and change it to “FENCE”			PASS
7		Left click on the map to create			PASS

		a border that matches the boundaries of the obstacle course. Then click the polygon logon and select “Fence Inclusion” then click “Write”			
8		On the right hand side of the screen find the dropdown that is displaying “FENCE” and change it to “MISSION”			PASS
9		Left click point on map beyond obstacle course finish line with an altitude of 1.5 meters. Then Click “Write”			PASS
10	At the Drone	Place the drone centered on the start line and facing towards the obstacles. Verify pilot is ready to start.	Drone displays accurate location and compass heading on map.		PASS
11	At Maintenance PC	In the terminal input “vim challenge_3_4.py” . Ensure that “CURRENT_CHALLENGE” = 4.			PASS
12		In the terminal input “python 3.9 challenge_3_4.py” . Watch terminal output for errors	Drone arms, flies towards obstacle course, navigates through obstacle course and across finish line	Drone did not complete obstacle course.	FAIL
13	At the GCS	Monitor battery voltage, telemetry, and messages windows.	No faults or errors		FAIL

14	At Maintenance PC	Upon completion or failure press Ctrl + C to end script	Script ends and returns to the Desktop/Temoc-Air directory		PASS
15	At the Drone	Have the pilot manually land once the finish line is crossed	drone lands and disarms		PASS

5.3 Evaluation of Results Relative to Design Criteria

The drone completed all necessary test cases prior to the competition except challenge 4. Challenge 1 can be tuned for accuracy using the RTK GPS setup procedure and properly calibrating the drone during the general flight test procedure. Additional parameters adjusting takeoff, landing, and flight speed may be used to improve the amount of time required to complete Challenge 1. While testing for Challenge 2 the team discovered that the ZED camera required specific calibration dependent on lighting. Inconsistent lighting such as partly cloudy skies can potentially lead to a test case failure because camera settings can not be adjusted dynamically. If challenge 2 test case fails the best course of action is to manually land and adjust parameters to aid the Zed camera with detection. Challenge 3 may also require parameter tuning based on environmental conditions such as wind. Parameters such as look ahead distances, the minimum distance to avoid, and the expiration of tracked objects can be used to achieve a successful test case.

6 Performance at Competition

The overall performance of the drone at the competition was more than pleasing, and the team ended up being awarded first place for it. Challenge 1 was completed in 30 seconds and landed with perfect accuracy in the center of the 30 yard line. For challenge 2 the drone managed to locate our logo when doing a grid search, but unfortunately was unable to land directly on it. Challenge 3 was completed in 3 minutes and 36 seconds, and avoided all obstacles in the course. Lastly, we were sadly unable to attempt challenge 4 due to time constraints.

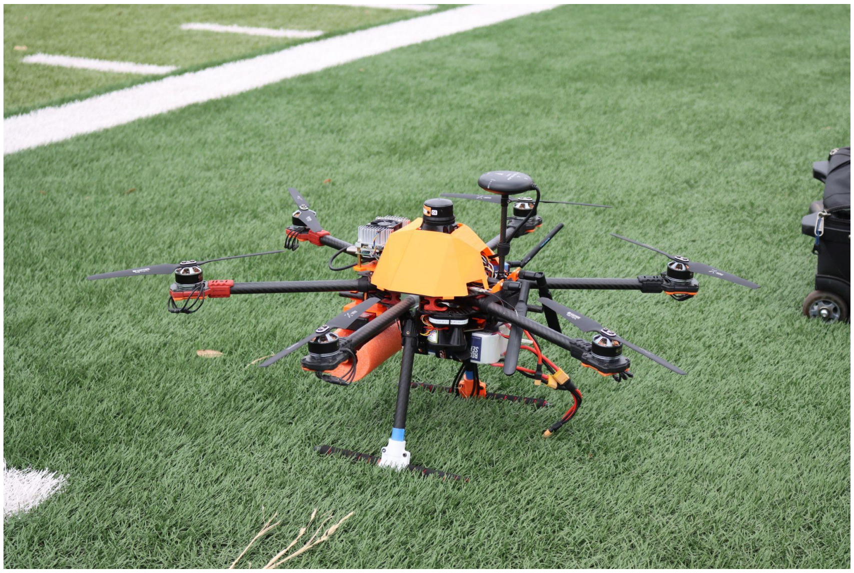


Figure 20: Hexacopter at Competition Field

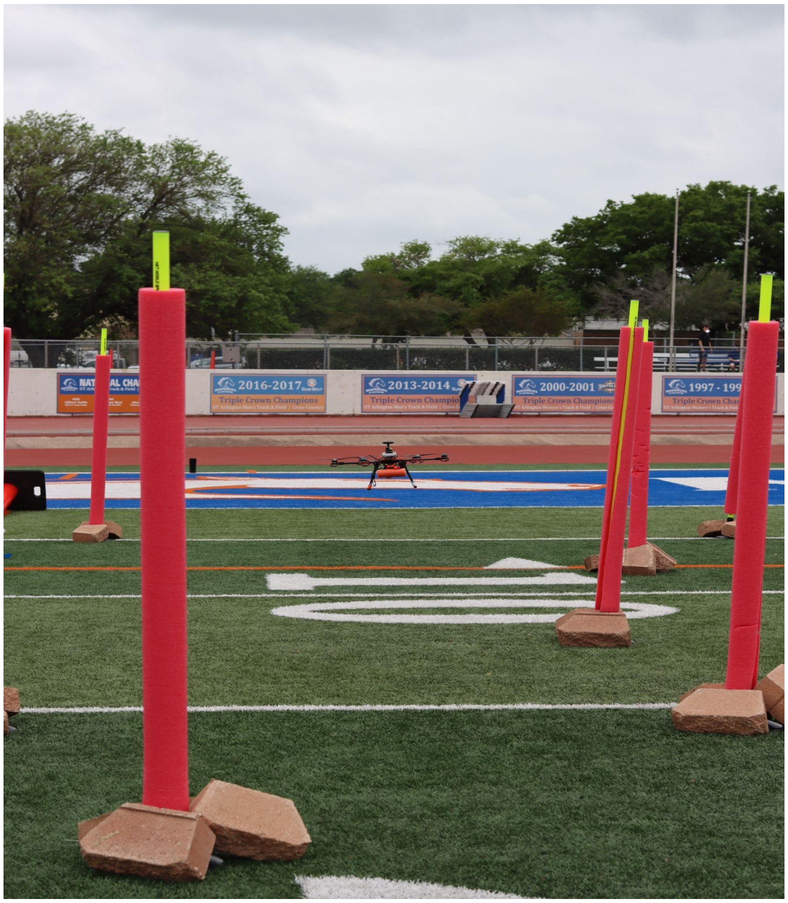


Figure 21: Hexacopter Testing Challenge 3

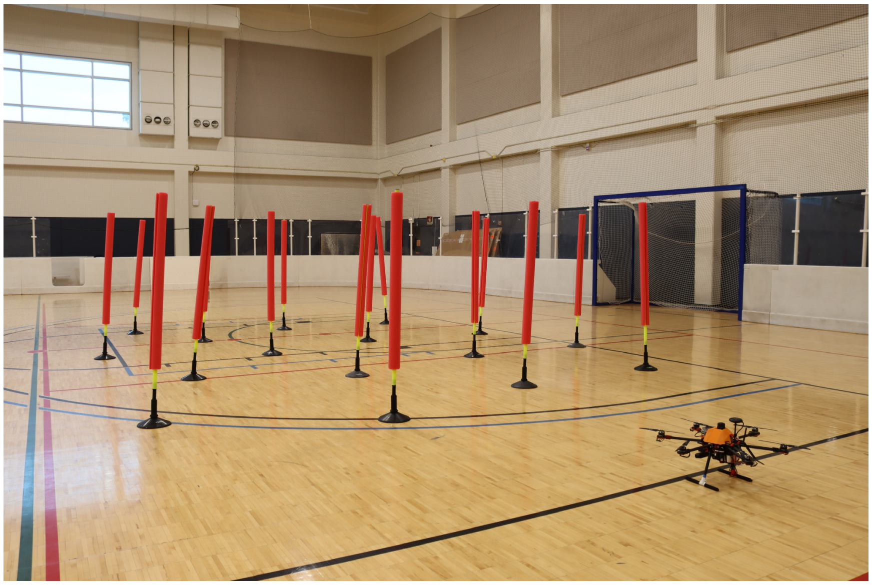


Figure 22: Hexacopter Preparing for Challenge 4

7 Conclusion

We placed first at both the Raytheon UAS Innovation Showcase ahead of five other Texas universities (UT, UTA, UT El Paso, A&M, and SMU) and UTDesign 2 Showcase. The drone's hardware and mechanical design allowed the drone to perform exceptionally well regardless of conditions like motor failure or high winds. Overall, even though we had issues the day of the competition, our preparation and knowledge of our drone's hardware and software allowed us to debug issues and make changes in short periods of time.

8 Future Work Recommendations

A few lessons learned that would be beneficial for future projects would be:

1. Start simulating as soon as possible, and once the framework is complete from there shift to integrating. Our bottleneck was not having enough time to integrate our simulation onto the real drone. "Test early and often".
2. Design each approach to be as simple and robust as possible by eliminating complexity. Only once it is working add features.
3. Our hardware was designed extremely well, but we did not spend enough time developing software. We do believe the ZED 2 is a solid replacement for LiDAR for Challenges 3&4 with enough software development time.
4. Integrate a few Mechanical Engineers onto the project to handle designing the drone, sensor mounts, and various other things. Our team had to learn CAD software to cover this deficiency.
5. This project being a real commitment and competition representing the university, better screening for individuals on the ECE team but mainly the CS team since they are only a semester would improve the overall experience. We think the CS team should be inclined to work on the project for two semesters rather than one as well.

9 Other Issues

9.1 Ethics

All of the source code and hardware design is original to our team. Safety is a huge concern for all things autonomous, that's why we implemented automatic and manual fail safes to minimize any potential damages. All software used was either open-source or used with the appropriate licensing. The intention behind this drone was to be used in a friendly competition to test our engineering skills and no way was used to to bring any form of harm upon others or business, but encourage people to pursue engineering in a new light.

9.2 Soft Skills

9.2.1 Lifelong Learning

The biggest takeaway from this project was being able to not only polish our previous skills but grow and learn new skills in different engineering domains, as well as collaboration and communication within a large team. Everyone in the team was able to walk away with improvements and growth in various aspects of the engineering process and encouraged us to keep pushing the boundary of learning.

9.2.2 Contemporary Issues

With the current global issues pertaining to war the implications of an autonomous drone have never been more at the forefront. Our project, while not being directly involved with Raytheon, itself raises concerns. This project was designed to be a friendly competition and test features of autonomous flight that can be used in applications that can further forward society in a positive direction.

All of the source code and hardware design is original to our team. Safety is a huge concern for all things autonomous, that's why we implemented automatic and manual fail safes to minimize any potential damages. All software used was either open-source or used with the appropriate licensing.

9.2.3 Multidisciplinary Teams

Our project was interdisciplinary, consisting of a hardware team from the Electrical and Computer Engineering department and a software team from the Computer Science department. The hardware team had the same seven members for both semesters but the software team was five to six members each semester. The UT Dallas Computer Science capstone project is only a semester long, meaning we got a new Computer Science team each semester. It was a unique challenge to lose half of the team then onboard a new half over winter break, but it did give us a chance to re-evaluate how we worked together and how the team was structured. Overall it did stagger our project development, but the Electrical and Computer Engineering team was able to pick up the slack.

10 Costs Estimate

10.1 Materials and Construction

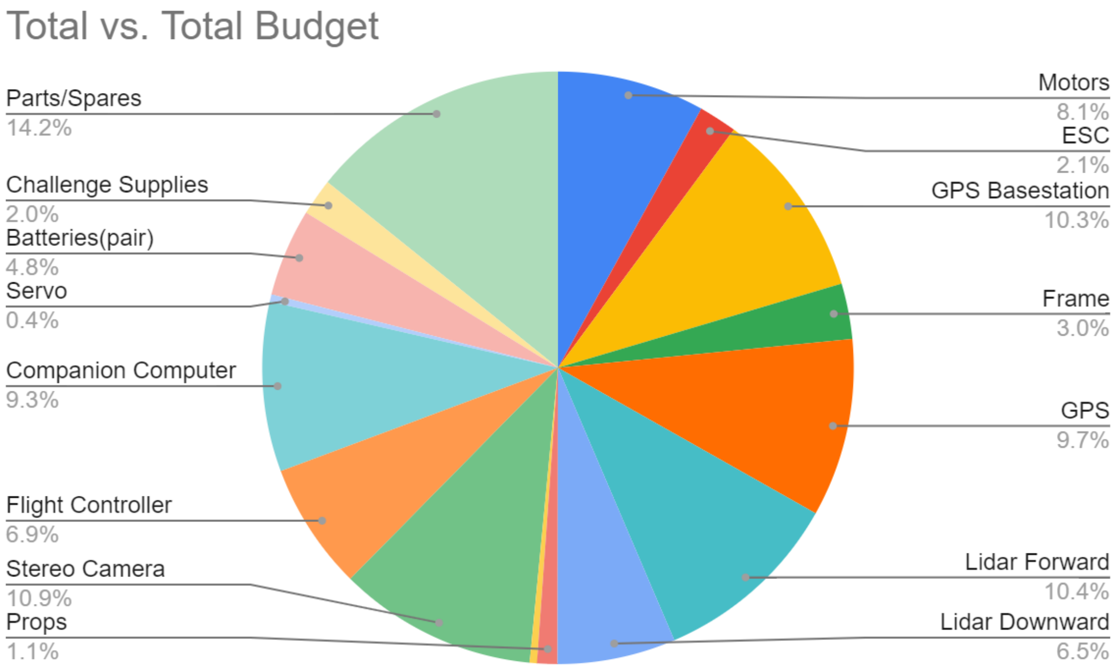


Figure 23: Pie Chart Budget Breakdown

Total Budget		Spent	Remaining	
$5,000.00		$4,319.73	$680.27	
Component	Model	Quantity	Cost	Total
Motors	T-Motor NM3110-15 780kv	6	$57.99	$347.94
ESC	EMAX Formula Series 32Bit 2-5S 45A ESC	6	$14.99	$89.94
GPS Basestation	RTK Basestation	1	$443.99	$443.99
Frame	Tarot FY690S CF Folding Hexacopter(TL68C01)	1	$130.97	$130.97
GPS	H-RTK F9P GNSS Series ROVER LITE	1	$420.09	$420.09
Lidar Forward	SF45/B Lidar	1	$449.00	$449.00
Lidar Downward	Lightware LiDAR SF11/C	1	$279.00	$279.00
Props	T-Motor MS1101 Props (pair)	3	$16.00	$48.00
Telemetry Radio	FrSky XM+ SBUS Mini Receiver	1	$16.99	$16.99

Stereo Camera	Zed 2 Stereolabs	1	$469.00	$469.00
Flight Controller	Cube Orange	1	$300.00	$300.00
Companion Computer	Nvidia Jetson TX2	1	$400.00	$400.00
Servo	LD-20MG Digital Servo	1	$16.99	$16.99
Batteries(pair)	TATTU 6000mAh 4s 35C XT60 Battery	1	$206.89	$206.89
Challenge Supplies	Noodles, Agility Sticks	1	$86.62	$86.62
Misc. Parts/Spares	N/A	1	$614.31	$614.31

Figure 24: Table Budget Breakdown

10.2 Estimate of Design Cost

Our estimated design cost for man hours is about 1700 hours across all 7 team members and travel time was 20 hours to test flights and the competition.

11 Project Management Summary

11.1 Team Breakdown

We split the team into three subteams for the challenges and general hardware. Alexander was the main team lead for all the sub teams, Will was the team lead for sub team 1 for General Hardware and Challenge 1, Noah was the team lead for subteam 2 for Challenge 2, and Sean was the team lead for subteam 3 for Challenges 3 and 4. We then split everyone else into a designated subteam, however team members' subteam placements would vary throughout the course of the project when focus shifted.

11.2 Agile

Our team was required to use the Agile methodology to assign, manage, and complete tasks. For this we used an online agile task board, JIRA.

11.2.1 Epics

First semester(Fall) we set our Epics to be researching and choosing parts for the following semester. They were laid out in this format:

- General Team Epic [August - November]: Build and flight test prototype quadcopter, researching drive train for final copter.
- Challenge Teams 1-4 Epic [August - Mid September]: Research approaches and pick parts to buy.
- Challenge Teams 1-4 [Mid September - December]: Simulate challenges with researched approach.
- General Team [November - December]: Order all parts for competition drone.

Second semester (Spring) we shifted our goals to be building the final drone, completing simulations, integrating sensors, and testing code.

- General Team [January - February]: Build competition drone, then test Challenge 1 as a benchmark.
- Challenge Team 2 [January - late March]: Complete simulation and test fly Challenge 2 on competition drone.
- Challenge Team 3 & 4 [January - mid April]: Test LiDAR approach in simulation and prototype quadcopter, then competition drone once parts arrive.
- Challenge Team 3 & 4 [January - mid April]: Develop and code ZED approach for Challenges 3 & 4, then flight test.

At the end of each Epic was a demo flight for our corporate advisor.

11.2.2 Sprints

Our team utilized weekly sprints that started Thursdays and ended Wednesdays at a full team meeting. During the week each subteam had their own standup call three times a week that helped keep them on track and all the members on the same page. In the standup calls, and full team meeting, each member would outline what they worked on, what they plan on working on, and what blockers they have, if any.

11.3 Facilities used

UTDesign Lab
UTDesign Makerspace
UTD Activity Center Gym
J.J. Pearce High School Football Field

11.4 Personnel

ECE Team:

Alexander Doudnikov: Team Lead

Sean Njenga: Challenge 3 Subteam Lead

Ian Falk: Challenge 4 Subteam Lead

Will Greenfield: Hardware Lead and Powertrain Design

Preetika Kondepudi: Challenge 1 Team Member

Noah Parker: Challenge 2 Subteam Lead

Wade Mullican: Challenge 2 Team Member

CS Team:

Matthew Lineberry

Ernesto Franco

Tulna Patel

Wasee Mahmood

Sulayman Hafizullah

Bisma Ahmed

James Baker

Mauricio Munoz

Nusrat Baset

Pei-Yun Tseng

Zainab Anwar

Advisors:

Marco Tacca: UTD

Christine Nezda: Raytheon

Alfonso Lopez: Raytheon

12 Bibliography

[1] Ardupilot, "ArduPilot Documentation," [Online]. Available: https://ardupilot.org/ardupilot/. [Accessed 2022].

[2] STEREOLABS, "Zed Documentation," [Online]. Available: https://www.stereolabs.com/docs/.

[3] NVIDIA, "Nvidia Documentation Center," Nvidia, [Online]. Available: https://docs.nvidia.com/. [Accessed 2022].

[4] eCalc, "xcopterCalc - Multicopter Calculator," eCalc, [Online]. Available: https://www.ecalc.ch/xcoptercalc.php. [Accessed 2022].

[5] K. Scott, "Installing and Configuring Your ROS Environment," ROS, 15 June 2020. [Online]. Available: http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment. [Accessed 2022].

[6] david0429, "Blender Gazebo," GitHub, 8 December 2019. [Online]. Available: https://github.com/david0429/blender_gazebo. [Accessed 2022].

[7] "MAVROS," GitHub, May 2022. [Online]. Available: https://github.com/mavlink/mavros/tree/master/mavros#installation. [Accessed 2022].

[8] "Stereolabs ZED Camera - ROS Noetic Ninjemis Integration," GitHub, April 2022. [Online]. Available: https://github.com/stereolabs/zed-ros-wrapper. [Accessed 2022].

[9] yanhwee, "Helium," GitHub, 27 July 2020. [Online]. Available: https://github.com/yanhwee/helium. [Accessed 2022].

[10] blender, "Blender Documentation," blender, [Online]. Available: https://docs.blender.org/. [Accessed 2022].

[11] eLinux, "Jetson/TX2 Cloning," eLinux, 5 February 2018. [Online]. Available: https://elinux.org/Jetson/TX2_Cloning. [Accessed 2022].

[12] dusty-nv, "Deploying Deep Learning," GitHub, 8 April 2022. [Online]. Available: https://github.com/dusty-nv/jetson-inference. [Accessed 2022].

[13] UTD ECE Team 1370, "Temoc-Air," GitHub, April 2022. [Online]. Available: https://github.com/UTD-UAX-2021-22/Temoc-Air. [Accessed 2022].

13 Appendices

13.1 Code

Our code is available to view on Github linked in the Bibliography section.

- GeneralDroneFunctions.py: Compilation of various drone related functions (like taking off, camera servo movement) to be used across all other scripts
- Challenge1.py: Runs the challenge 1 task
- challenge_2_main.py: Performs the grid search for the drone to locate the logo and precision landing
    - POI.py: Identifies the Aruco logo using OpenCV2 library. Hue and saturation can be tuned to account for lighting.
    - vehicleinfo.json: Defines the forward and downward camera positions
- challenge_3_4.py: Runs the scripts for challenge 3 & 4, for challenge 4 it also gets the visual odometry data and pose for the ZED
    - c4_distance.py: Runs the ZED depth image processing to detect closets obstacle
    - c4_mavlink.py: Sends various data packets to the FCU

13.2 Operation Instructions

13.2.1 Software Requirements on Personal Laptop

- SSH client - Windows: PuTTY
- FTP client - Windows: WinSCP
- Text editor - Windows: Notepad++

All of these Windows programs can be installed from https://ninite.com/ in the Developer Tools section, just check the boxes on the software you need and run ninite.

13.2.2 Pre-Flight Prep

- ☐ Charge:
    - ☐ All 3 pairs of hex batteries (4.2v)
    - ☐ Blue and white quad battery (4.2v)
    - ☐ Big black battery (4.2v)
    - ☐ Ground Control Station (GCS) Computer (Lenovo)
    - ☐ Personal Laptop
    - ☐ Radio Transmitter (white controller)
    - ☐ Battery Bank for Router (Currently Alex's personal bank. Any USB battery bank works)
    - ☐ Handheld radios

- ☐ Pack “Tools” box (Bottom to Top, Left to Right):
    - ☐ Bottom: Battery padding, spare frame plates
    - ☐ First row: Orange screwdriver kit, solder spool, both hardware boxes for M3 standoffs and screws, magnetic screw plate
    - ☐ Second row: Orange screwdriver kit, wide double sided tape with velcro inside, solder prowick, blue loctite, M3 hardware
    - ☐ Third row: 2 Male XT60 to 1 Female XT60 cable, 1 Male to 2 Female XT60 cable, Flux pen, Tweezers, metal solder sponge, maybe some other junk
    - ☐ Top: on top of screwdriver kit, green handled wire stripper, black and yellow handled wire cutter

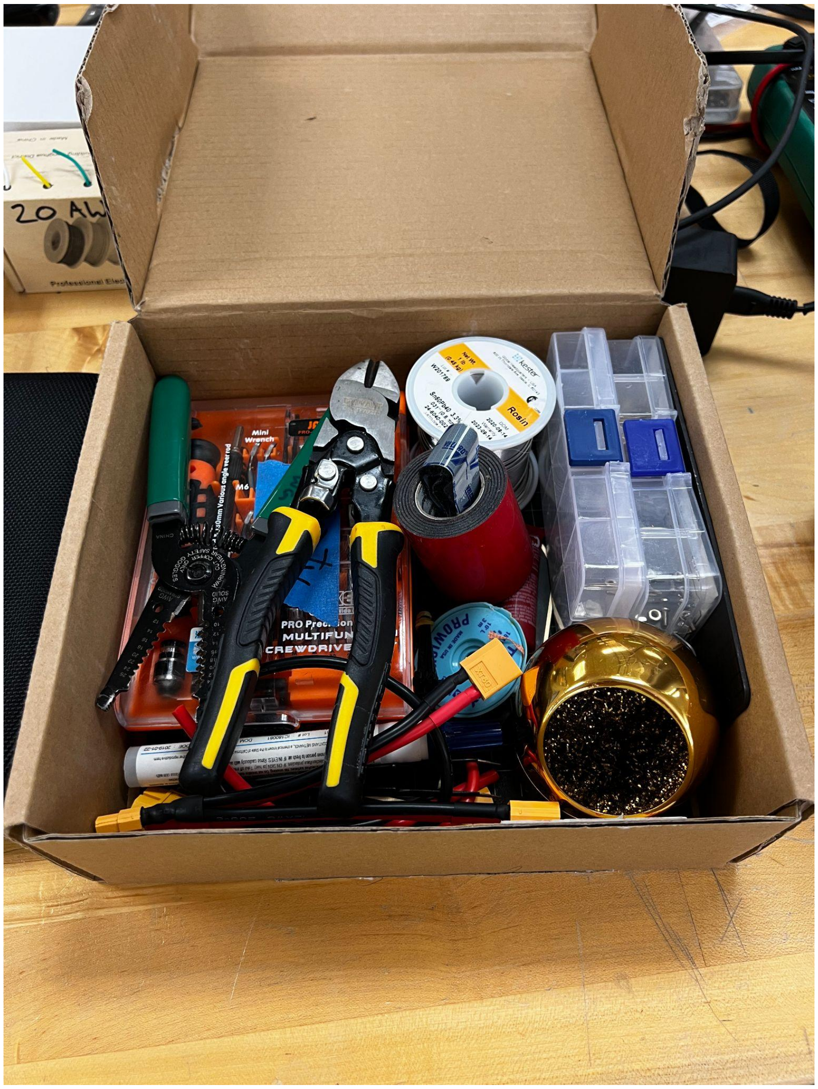


Figure 25: Packed “Tools” box

- ☐ Pack Green Field bag:

- □ Main pocket (things go into the bag in this order):
    - □ Tools Box, soldering iron next to this, lenovo charger, Wires and spare motors box in front of those two (marked wires), red multiwire box (contains zip ties too), multimeter, and across the top strap dual sided tape, masking tape, electrical tape. Lenovo laptop across the top.
    - □ Radios across the top attached to pockets/loops

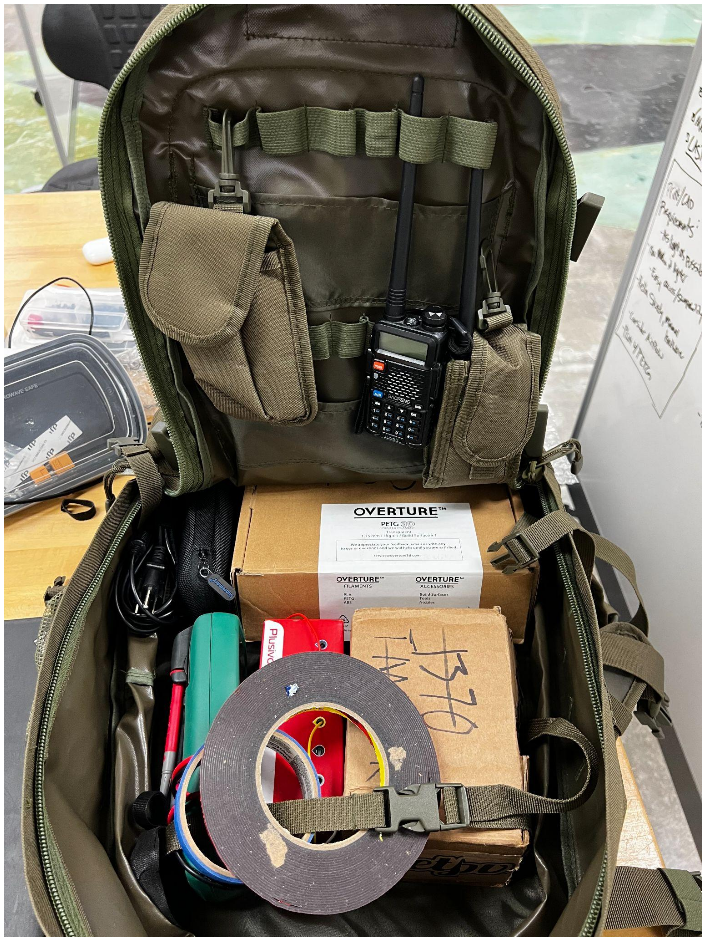


Figure 26: Packed bag without Lenovo Laptop

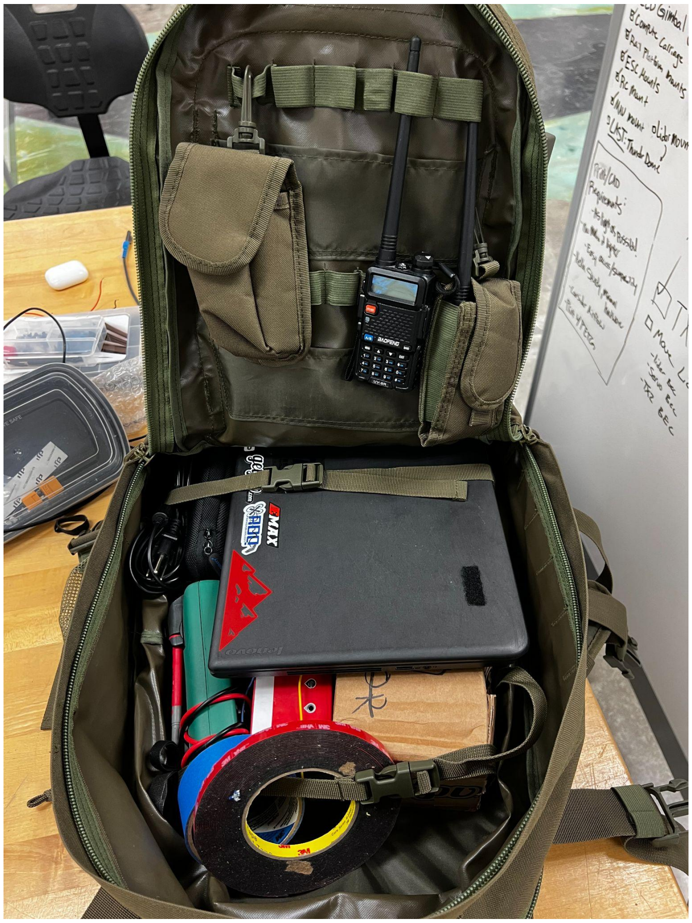


Figure 27: Packed bag with Lenovo Laptop

- ☐ Middle pocket:
  - ☐ Inside net: Spare 3D printed parts, spare servo, One Micro USB cable, One USB-C cable, One Ethernet Cable, Field TP-Link Router
  - ☐ Outside net: spare props

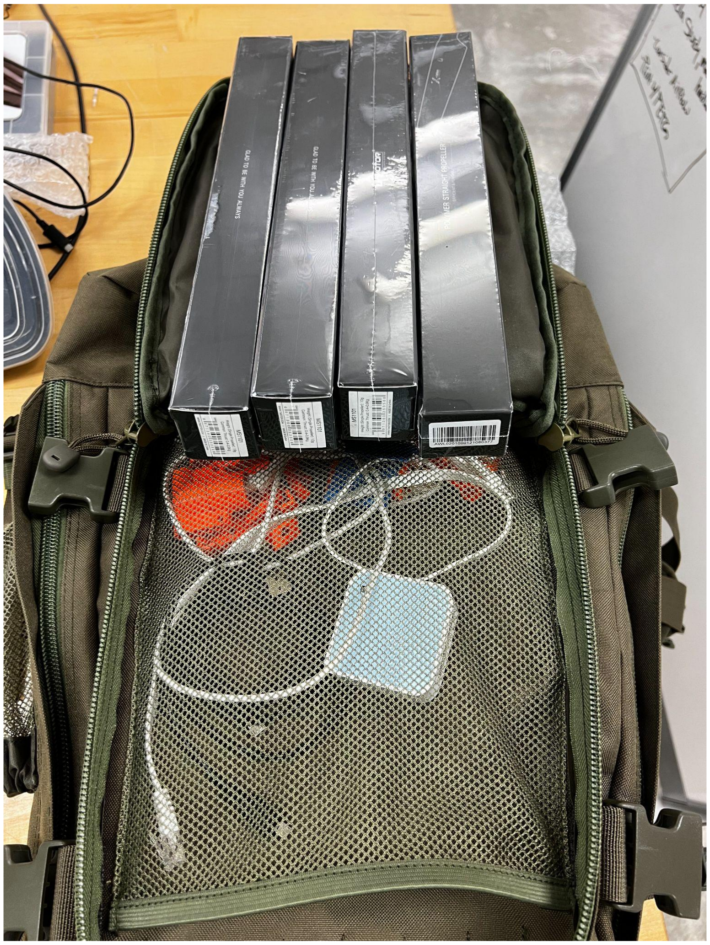


Figure 28: Packed Middle Pocket

- □ Front pocket:
  - □ Small net zip pocket: Spare GPS's and allen keys for taking props off, HDMI 1080p plug
  - □ Other pockets contain: USB2.0 hub, Spare ESC's, Spare BEC's, Spare Velcro, Spare heat shrink tubing (ESC)
  - □ Pen holder: Telemetry radio for Hex and micro-USB cable
  - □ Drone remote lanyard

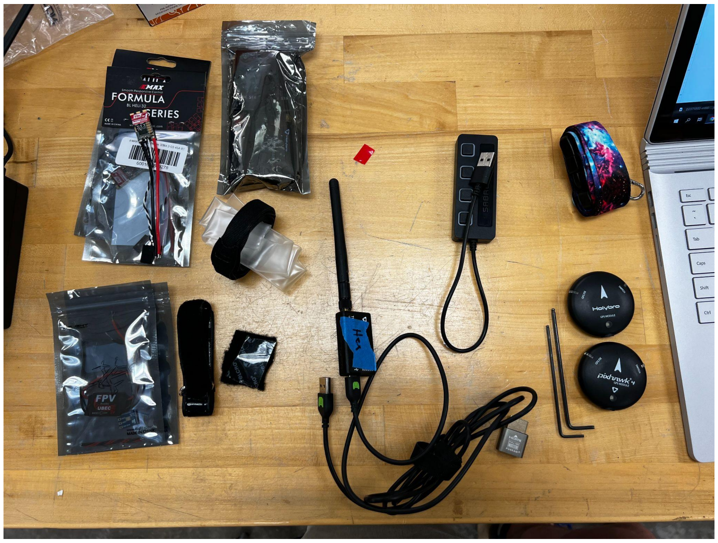


Figure 29: Everything inside of the front pocket

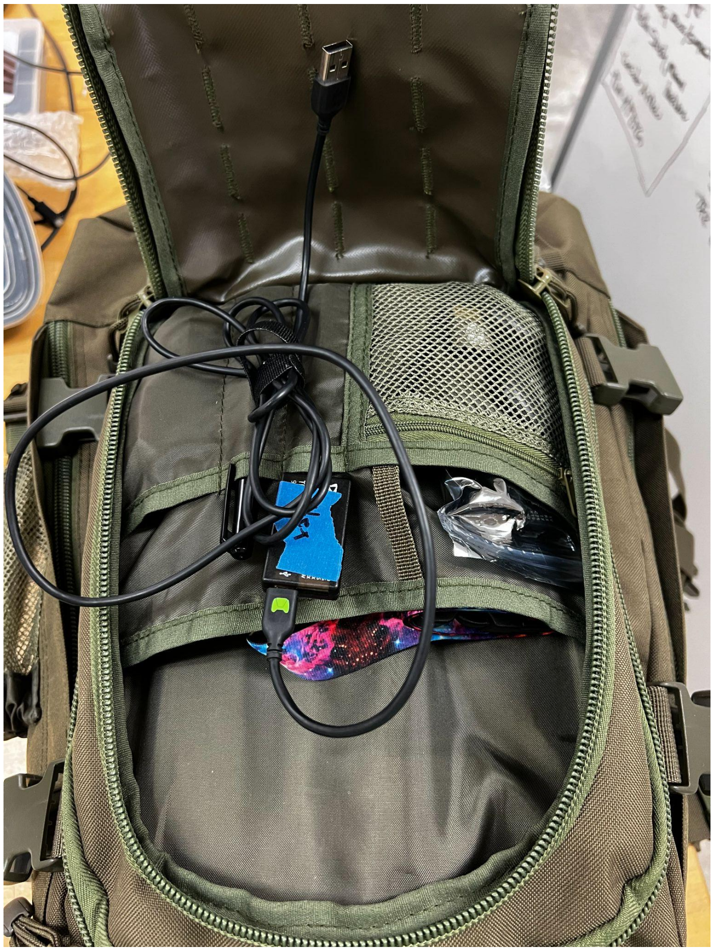


Figure 30: Packed Front Pocket

    - □ Outside the bag:
        - □ Taranis X7 Radio Transmitter clipped to carabiner and under the front elastic
        - □ RTK GPS tripod strapped to one side of the backpack with USB-C & extension on it
- □ Pack LiPo bags
    - □ LiPo bag 1: All three pairs of hex batteries and the small smart charger
    - □ LiPo bag 2: Blue and White batteries and big black battery

13.2.3 Field Setup

- □ RTK setup
    - □ This needs to happen immediately on arrival, locate the spot on the field where the GCS table will be located, set up and connect the RTK GPS tripod, and start RTK Survey in.
        - □ In mission planner open `Setup -> Optional Hardware -> RTK/GPS Inject`
        - □ Set to correct COM port and BAUD rate
        - □ Check all boxes, including `UBlox M8P/F9P autoconfig` and `M8P fw 130+/F9p`
        - □ Set SurveyIn Acc(m) to .5 and Time(s) to 60
        - □ Press "Restart". Mission planner should now show the GPS triangulating the RTK station with everything but "Base" having a green indicator.
        - □ Once the Survey In is complete the "Base" indicator should be green.
        - □ Click "Save Current Position" so we can later reuse this RTK position without a Survey In.
        - □ Click "Use" on the GPS set for our location we just Surveyed In.
    - □ Drone Calibration
        - □ Attach drone batteries
        - □ Remove props using allen key or screwdriver kit
        - □ Connect drone batteries
        - □ Connect GCS to drone via USB telemetry dongle, then attach dongle to velcro on laptop. (Note: It only works on the USB ports on the right side of the lenovo laptop)
            - □ In mission planner on the top right select the COM port (Usually COM3, can change) of the telemetry module, and BAUD rate of 57600. Click Connect.
            - □ Navigate to `Setup -> Mandatory Hardware-> Accel Calibration`
                - □ Click Calibrate Accel and follow the steps on screen
                - □ Click Calibrate Level and follow the steps on screen

- □ Navigate to `Compass` below `Accel Calibration`
    - □ In `Onboard Mag Calibration` click Start
    - □ Move the drone along all 3 axes similar to accel calibration, the goal is for the compass to move in every direction. Once both Mag 1 & 2 bars fill up, the calibration is complete.
    - □ If the bars keep filling up and do not complete, power cycle the drone then try again.
        - □ Disabling the onboard FMU compass may be required if there is too much interference around the flight controller.

- □ Test takeoff
    - □ Once calibration is complete, reattach all props.

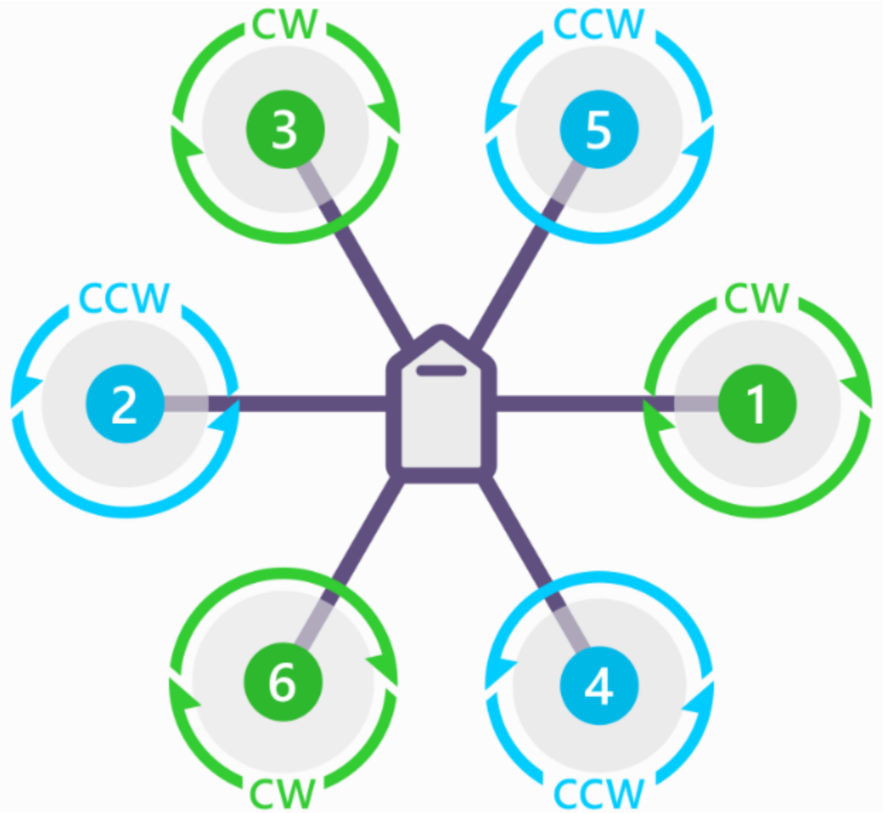


Figure 31: Drone Propeller Orientations

- □ Perform a test flight in PosHold mode. A low hover is ideal.
- □ The things you are looking for are:
    - □ All motors arm
    - □ Drone holds GPS position hovering

- ☐ Drone is in the correct position and heading on the map
- ☐ No erratic behaviors
- ☐ No error messages in mission planner `Messages` Tab

13.2.4 Taranis X7 Switches Setup

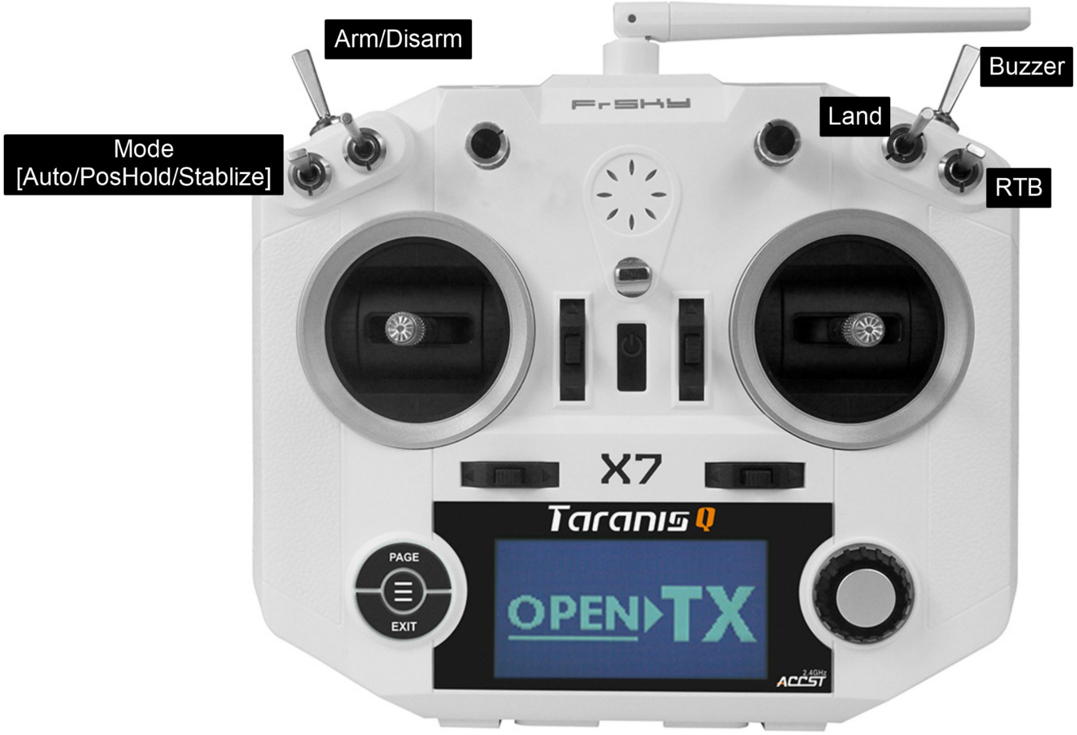


Figure 32: Taranis Switch Setup

13.3 TX2 Restore Guides

13.3.1 TX2 Image Backup & Restore

Source: https://elinux.org/Jetson/TX2_Cloning

Requirements:

- Ubuntu 18.04 Host Machine, containing Jetpack install
- TX2 on either the development carrier or orbitty carrier
- Micro-USB Cable

TX2 Image Backup

1. Boot the TX2 into recovery mode
   a. On the Orbitty carrier (blue board) hold recovery and push reset, do not let go of recovery for two seconds

2. Connect the board to the host Ubuntu PC that flashed it using a Micro-USB cable
3. Navigate to the folder of the L4T installation (Typically something like cd ~/nvidia/nvidia_sdk/JetPack_{Version}_Linux_JETSON_TX2_TARGETS/Linux_for_Tegra)
4. Check the TX2 is connected with `lsusb` you should see an Nvidia Corp device listed. Our's hardware ID is `0955:7c18` if you want to make sure it's correct.
5. Use sudo ./flash.sh -r -k APP -G backup.img jetson-tx2 mmcblk0p1 to backup the image of the TX2.

   Note: This process takes around an hour so make sure you have enough power to supply the TX2 during that time.

TX2 Image Restore

Note: This process takes around an hour so make sure you have enough power to supply the TX2 during that time.

1. Boot the TX2 into recovery mode
   a. On the Orbitty carrier (blue board) hold recovery and push reset, do not let go of recovery for two seconds
2. Connect the board to the host Ubuntu PC that flashed it using a Micro-USB cable
3. Navigate to the folder of the L4T installation (Typically something like cd ~/nvidia/nvidia_sdk/JetPack_{Version}_Linux_JETSON_TX2_TARGETS/Linux_for_Tegra)
4. Check the TX2 is connected with lsusb you should see an Nvidia Corp device listed. Our's hardware ID is **0955:7c18** if you want to make sure it's correct.
5. Copy the raw image we made before into the bootloader image sudo cp backup.img.raw bootloader/system.img
6. Flash the TX2 sudo ./flash.sh -r -k APP jetson-tx2 mmcblk0p1

13.3.2 Post Flash TX2 Steps

1. Copy and paste these files from this zip onto the TX2's desktop (I recommend a flash drive from your pc onto the TX2): https://drive.google.com/file/d/1iSQdCkrk7WViEbgh5LRhQrkgvn-pzu1d/view?usp=sharing
2. Install python 3.9 (Google Python3.9 install ubuntu/linux), reboot after this step
3. Install setuptools, python3-dev, and distutils `sudo apt-get install python3.9-dev python3.9-setuptools python3.9-distutils`
4. Install pip for our python3.9 install `sudo apt install python3-pip`
5. Upgrade a few packages for building modules

   ```
   Python3.9 -m pip install --upgrade pip
   Python3.9 -m pip install --upgrade distlib
   Python3.9 -m pip install --upgrade setuptools
   Python3.9 -m pip install --upgrade wheel
   ```
6. Install cython and numpy (reference challenge2_requirements for versions) python3.9 -m pip install {package}=={version}

7. Install ZED SDK [Jetpack 4.6 NOT 4.6.1] (Yes to all optional things, `python3.9` for our python install when it asks)
8. Open the folder Temoc-Air and right click -> open in terminal
9. git status to make sure it is getting git stuff, git branch to make sure its on the right branch
10. git pull to update it to the most recent commit
11. python3.9 -m pip install -r requirements_challenge_2.txt If something fails, google it and note down the package it was missing
12. add linux username to dialout group (serial access) - sudo gpasswd --add ${USER} dialout
13. Restart sudo reboot
14. Try to run challenge2 main or challenge 1 and see if it errors [DO NOT RUN WHILE CONNECTED TO THE DRONE WITH PROPS ON]

13.4 Simulator Setup

13.4.1 Gazebo Simulator setup

Requirements:

- Ubuntu 18.04 desktop install
- Graphics card
- Git (sudo apt-get install git)
- Blender (https://www.blender.org/download/)

Steps to install helium (Ardupilot, Ardupilot Gazebo plugin, Gazebo 9, ROS Melodic, MAVROS) [https://github.com/yanhwee/helium]

1. Open Gnome Terminal
2. Clone the helium repository (git clone https://github.com/yanhwee/helium.git)
3. Enter helium (cd helium)
4. Run the installation script (wget -O - https://raw.githubusercontent.com/yanhwee/helium/master/docs/install.bash | bash)
5. Test that helium installed correctly by launching gazebo, SITL, ROS, and QGroundControl
    a. Open four terminals in the home directory (Tip: Ctrl-Alt-T + Ctrl-Shift-T x4)
    b. In each:
        i. roslaunch Gazebo world
            1. roslaunch helium flat_pillars.launch
        ii. Start ArduPilot SITL
            1. sim_vehicle.py -v ArduCopter -f gazebo-iris
                a. Will need to compile (for first time)
                b. Wait for APM: EKF2 IMU1 tilt alignment complete before continuing to the next step
                c. Initialisation is completed when APM: EKF2 IMU1 is using GPS

            i. Then you can start sending MAVLink commands & interface with MAVProxy

    iii. Start MAVROS
        1. roslaunch helium apm.launch
    iv. Start QGroundControl
        1. ./QGroundControl.AppImage

Steps to install blender_gazebo and import our challenge worlds (Required to open our custom worlds)

1. Navigate to where our active ROS package is (cd /opt/ros/melodic/share)
2. Git clone the blender_gazebo repository (git clone https://github.com/david0429/blender_gazebo.git)
3. Rename “blender_gazebo-master” to “blender_gazebo” (mv blender_gazebo-master blender_gazebo)
4. Run the installation script (rosrun blender_gazebo install)
5. Open Blender and navigate to addons and enable blender_gazebo (Edit -> Preferences -> Addons -> Check “Import-Export: ROS Gazebo Exporter”

Steps to export our blender worlds to Gazebo

1. Open terminal and open nautilus file viewer in sudo (sudo nautilus)
2. Navigate to where we put blender_gazebo worlds (Other Locations -> Computer -> opt -> ros -> melodic -> share -> blender_gazebo -> worlds)
3. Open “actually_empty_world.world” in the text editor
4. Copy and paste the contents of the “actually_empty_world.world” on our google drive to replace the one open in our text editor (Located in Google Drive General Subteam -> Simulator -> Gazebo Files -> “actually_empty_world.world”)
    a. This will set our spawn world to have our gazebo-iris drone with LiDAR to spawn.
5. Open the world you want to import to Gazebo in Blender, in this tutorial we will use the “stadium-empty.world” (Located in Google Drive General Subteam -> Simulator -> Blender files -> “stadium-empty.world”)
6. In blender go to File -> Export -> Gazebo Launch (.launch) and export to the following directory (/home/{linux username}/catkin_ws/src/helium/launch)
7. You should now be able to launch that gazebo world by using (roslaunch helium stadium-empty.launch)
    a. The world should contain the stadium and the gazebo-iris.

Notes: Objects exported from blender must have no modifiers attached to them for Gazebo to import them correctly.

Steps to modify obstacle placement for Challenge 3 world

1. Download and open “stadium-obstacle.blend” (Located in Google Drive General Subteam -> Simulator -> Blender files -> “stadium-obstacle.world)
2. Select the “Obstacle emitter” object on the right panel (left click)
3. In the panel below, select the particle system (4 dots connected below the wrench icon)
4. Open the Emission tab and modify
    a. Number - How many obstacles total
    b. Seed - Seed for randomization of placement
    c. Hair Length - Do not modify, height of obstacles
    d. Segments - Do not modify, outside of scope of this project
5. Once satisfied select the wrench icon then click “Make Instances Real”
    a. This will set our “particle” obstacles to become real meshes gazebo can use
6. Select all objects on the top right panel with a wrench icon using the control key, then select “field” last. (All objects with a wrench should be dark orange, while the last selected object with no wrench icon should be light orange)
7. Push “CTRL+L” with the mouse over the 3D viewer and select “Copy Modifiers”
8. If this was successful there should not be any objects with wrench icons (modifiers)
9. In blender go to File -> Export -> Gazebo Launch (.launch) and export to the following directory (/home/{linux username}/catkin_ws/src/helium/launch)
10. You should now be able to launch that gazebo world by using (`roslaunch helium {export name}.launch`)

Steps to run gazebo simulations

1. Open four terminals in the home directory (Tip: Ctrl-Alt-T + Ctrl-Shift-T x4)
2. In each:
    a. roslaunch Gazebo world
        i. `roslaunch helium flat_pillars.launch`
    b. Start ArduPilot SITL
        i. `sim_vehicle.py -v ArduCopter -f gazebo-iris`
            1. Will need to compile (for first time)
            2. Wait for APM: EKF2 IMU1 tilt alignment complete beforecontinuing to the next step
            3. Initilisation is completed when APM: EKF2 IMU1 is using GPS
                a. Then you can start sending MAVLink commands & interface with MAVProxy
    c. Start MAVROS
    d. Start QGroundControl - from here you can modify drone parameters and plan ardupilot missions
    e. (Optional) Open another terminal (`Ctrl-Shift-T`) and run your dronekit code

        i. Python{name of file}.py
            1. Note: It is known that dronekit may disconnect QGroundControl.

13.4.2 SITL Setup

Requirements

- Ubuntu 18.04 desktop install
- Git (sudo apt-get install git)

Clone Ardupilot repository

1. open terminal
2. input the following :
    - git clone https://github.com/your-github-userid/ardupilot
    - cd ardupilot
    - git submodule update --init --recursive

Install Additional Packages

1. from the Ardupilot directory input"
    - Tools/environment_install/install-prereqs-ubuntu.sh -y
2. Reload the path (log-out and log-in to make permanent) by inputting:
    - . ~/.profile

Start SITL

1. Change directory to "ardupilot/Arducopter"
    - cd ardupilot/ArduCopter
2. start the sim vehicle to load the default parameters by entering:
    - sim_vehicle.py -w
3. After the default parameters are loaded end the sim vehicle script by pressing:
    - Ctrl + C
4. The simulator may now be started normally by inputting:
    - sim_vehicle.py –console –map
5. Arducopter, Console, and Map windows will open

Control SITL Drone

1. Within the terminal used to run sim vehicle script hit enter until "STABILIZE>" is displayed as the prompt. The terminal window is now running MAVProxy.
2. Within the terminal type
    - mode GUIDED
3. Hit enter and "GUIDED>" should be displayed as the prompt
4. to arm the drone type:
    - "arm throttle"
5. The status within the Console window should change to armed
6. to take off input:
    - takeoff <altitude in meters>

7. Left click a point on the map to place place a blue “X”
8. Right click on “X” and select “Fly To” from dropdown

Connect Mission Planner to SITL

1. open mission planner
2. on the top left dropdown where the connection protocol is selected, select “UDP”
3. Within the MAVProxy terminal input:
    - “output add <Your IP address>:14550”
4. Open mission planner and input port: “14550” and click connect
5. Mission planner should now be connected to the SITL drone

Install dronekit

1. open terminal and input:
    - sudo -H pip install dronekit
2. Next input:
    - sudo -H pip install dronekit-sitl

13.5 Programming Guide

13.5.1 Installing & Setting Up Python

(Note: use “py” for Unix/Mac and “python” for Windows when running python commands other than version checking)

1. Go to this website and download the latest 64-bit python executable for your OS.
2. Verify that Python was installed correctly by typing python into the cmd prompt and it should display the version you downloaded > Python #.#.#.
3. The Pip package manager should be installed by default with the Python installation so type in pip -V into the cmd prompt to ensure that it’s installed.
4. Then type py -m pip install --upgrade pip to ensure Pip is up to date.
5. The commands to install the DroneKit library from the cmd line can be found here
6. Install a virtual env to host Python 2.7 by typing pip install virtualenv into the cmd line
    a. Create a Directory (Optional): mkdir dir_name then cd dir_name, this created a directory with dir_name and moves you to that new directory
7. Lastly create the python 2.7 venv by typing virtualenv -p

13.5.2 Nvidia Jetson Setup

- Two ways to do this:
    - Via the built-in docker
    - Building project from source

13.5.3 OpenCV Setup

1. Download the OpenCV library for Python by typing pip install opencv-python into the cmd line.

    a. Import = import cv2

2. Here is the documentation for image processing in OpenCV ([Link](#))

13.5.4 Basic Push Flow Git Guide

1. git checkout <branch_name> make sure you are in the correct branch
    a. Or git checkout -b <branch_name> to checkout a non-existent branch
2. git status ensure that your current working tree is correct before adding files to commit
3. git add . (adds all files that have changes and new files) or git add <filepath> (can add all edited files in a specific directory or singular file) to the working tree for checkout
    a. Use git remove <filepath> to remove a file that u added to the working tree from being pushed
4. git status to ensure the working tree has the correct files to be staged
5. git commit -m <title> -m <desc> stage the changes to be pushed (only saved in local repo)
6. git push -u origin <branch_name> push the files to the wanted branch
    a. If u need to revert a push or pull, dm Sean or Alex
7. Open github and click Compare & pull request
8. Have your code reviewed by at least one other team member before pushing it to main.

13.5.5 VSCode SSH Into TX2 Setup Guide

Requirements

- Personal Computer
- VSCode

Steps to setup and connect to the TX2 VSCode instance

1. Install the "Remote - SSH" extension for VSCode.
    a. Extensions are located in the 4 cube tab on the left side of the screen
2. Open the Remote Explorer tab, it will look like a computer with a circle infront of it
3. Make sure you are connected to the same WiFi as the TX2
4. Click the + in SSH targets and type `ssh tx2@tx2.local`
5. Push enter to save it into your SSH config
6. Right click the new computer named "tx2.local" and click "connect to Host in current window"
7. Allow network access to VSCode
8. Set the environment to Linux when asked, accept the key with "continue" and type in the password.
9. Wait a moment for the connection to complete
10. Navigate to the file explorer, the first tab on the left
11. Select open folder and enter "/home/tx2/" or just "/" if you would like access to all files.
12. VSCode will reconnect, re-enter the password, and click "Yes, I trust the authors"\
13. Next time you connect to the TX2 click the file name instead of tx2.local in the SSH tab.

13.6 MAVROS/ROS Installation

Requirements

- Bionic Ubuntu 18.04
- Python3
- ZED Camera and SDK

13.6.1 Install ROS Core on TX2

1. Setup sources.list:
   ```
   sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
   ```
2. Set up your keys: (must have curl already installed)
   ```
   curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add -
   ```
3. sudo apt update
4. Install ROS desktop-full package:
   ```
   sudo apt install ros-melodic-desktop-full
   ```
5. Setup the environment so that ROS environment variables are automatically added to bash session every time a new sell is launched:
   ```
   echo “source /opt/ros/melodic/setup.bash” >> ~/.bashrc
   source ~/.bashrc
   ```
6. Setup dependencies for building packages:
   ```
   sudo -H pip3 install rosdep rospkg rosinstall_generator rosinstall wstool vcstools catkin_tools catkin_pkg
   ```
7. Force rosdep to build with python3 instead of python2 by changing environment variable:
   ```
   export ROS_PYTHON_VERSION=3
   ```
8. Initialize rosdep:
   ```
   sudo rosdep init
   rosdep update
   ```

The above steps can be found on the ROS wiki (http://wiki.ros.org/melodic/Installation/Ubuntu)

13.6.2 Setup ROS Workspace

1. Create a catkin workspace:
   ```
   mkdir -p ~/catkin_ws/src
   cd ~/catkin_ws/
   ```

2. Since we are using python3 we need to make catkin with python3:
   ```
   catkin_make -DPYTHON_EXECUTABLE=/usr/bin/python3
   ```
3. We now have new source files that we need to source:
   ```
   source devel/setup.bash
   ```

4. To check if the workspace is setup properly, make sure the ROS_PACKAGE_PATH environment variable includes your workspace directory.
   ```
   echo $ROS_PACKAGE_PATH
   Expected output: “/home/youruser/catkin_ws/src:/opt/ros/melodic/share”
   ```

The above steps can be found on the ROS wiki (http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment)

13.6.3 Install ZED ROS Wrapper & MAVROS

1. Clone the ZED wrapper and ZED Examples repository:
   ```
   cd ~/catkin_ws/src
   git clone --recursive https://github.com/stereolabs/zed-ros-wrapper.git
   git clone https://github.com/stereolabs/zed-ros-examples.git
   cd ../
   ```
2. Setup the workspace for MAVROS:
   ```
   cd ~/catkin_ws
   catkin init
   wstool init src
   ```
3. Install MAVLink:
   ```
   rosinstall_generator --rosdistro melodic mavlink | tee /tmp/mavros.rosinstall
   ```
4. Install MAVROS:
   ```
   rosinstall_generator --upstream mavros | tee -a /tmp/mavros.rosinstall
   ```
5. Create workspace & dependencies:
   ```
   wstool merge -t src /tmp/mavros.rosinstall
   wstool update -t src -j4
   rosdep install --from-paths src --ignore-src -y
   ```
   (if rosdep step fails, try copying the above rosinstall_generator commands directly from mavros github. Google docs likes to format characters weird)
6. Install GeographicLib datasets for MAVROS:
   ```
   sudo ./src/mavros/mavros/scripts/install_geographiclib_datasets.sh
   ```
7. Build source:
   ```
   Catkin build
   Source devel/setup.bash
   ```
8. If the previous “catkin build” did not work, delete the “build” and “devel” folders and try again.

The above steps can be found on the ZED Github (https://github.com/stereolabs/zed-ros-wrapper) and MAVROS Github (https://github.com/mavlink/mavros/tree/master/mavros#installation)

13.6.4 Testing ROS Install

1. To test the ZED ROS install try running:
   ```
   roslaunch zed_display_rviz display_zed2.launch
   roslaunch zed_wrapper zed2.launch
   ```
2. To test the MAVROS install try running:
   ```
   roscore
   ```
   Open a new terminal and run
   ```
   roslaunch mavros apm.launch fcu_url:=udp://:14855@
   ```

   And then you should be able to send mavros commands like
   ```
   rosrun mavros mavsys mode -c 0 (sets the vehicle to mode "0")
   rosrun mavros mavsafety arm (to arm the vehicle)
   ```

13.6.5 Creating Custom ROS Package

1. Use catkin_create_pkg <name> to create a package in the workspace, we named our package “challenge4”:
   ```
   cd ~/catkin_ws/src
   catkin_create_pkg challenge4
   cd challenge4
   mkdir scripts
   ```
2. Create your python or other programs here, ours was named c4_mavlink.py, Include this as the FIRST line of your program to let it know its a script:
   ```
   #!/usr/bin/env python
   ```
3. Make the program executable:
   ```
   chmod +x c4_mavlink.py
   ```
4. Edit the CMakeLists.txt line “catkin_install_python”, it should look something like this:
   ```
   catkin_install_python(PROGRAMS scripts/c4_mavlink.py scripts/<additional script>
   scripts/<additional script>
     DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION})
   ```
5. Finally rebuild the catkin workspace or build the custom package only:
   ```
   catkin build
   ```
   Or
   ```
   catkin build challenge4
   ```

Much of the above steps can be found on the ROS wiki (http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29)

13.6.6 How To Run Custom Package from 13.6.5

```
roscore
```

In a new terminal:

```
rosrun <package name> <program name>
rosrun challenge4 c4_mavlink.py
```