**Department of Electrical Engineering and Computer Science**

Howard University
Washington, DC 20059

**EECE 401-01 Senior Design I**
2023-2024

Raytheon Drone Competition UGV Team

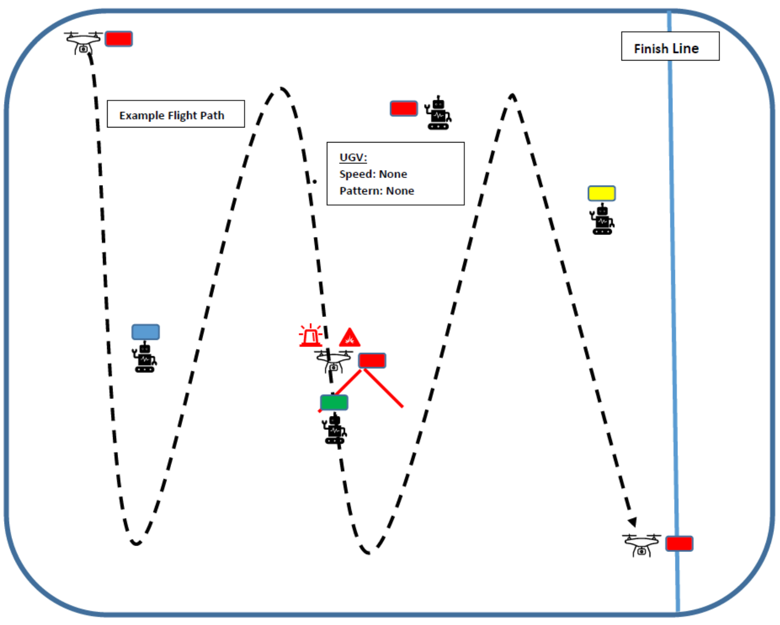
**Group Members:**
Teminijesu Oyedele
Ernest Olopoenia
Chidi Onyekwelu
Oghosa Osaghae

Advisor: Danda Rawat
Instructor: Charles Kim

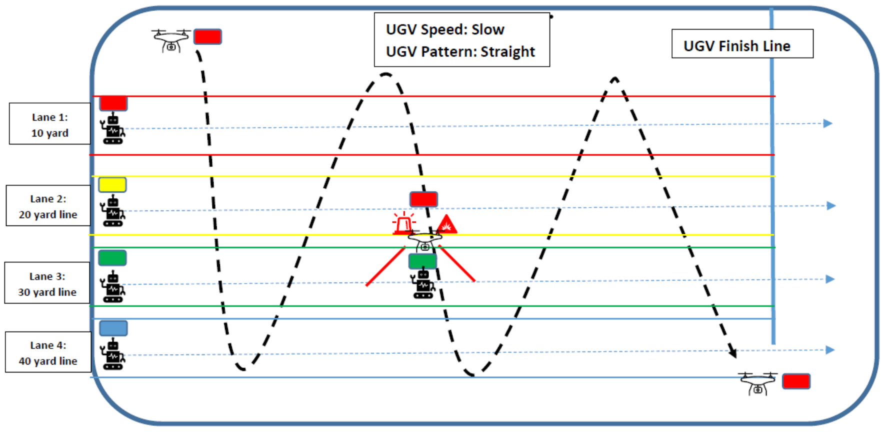
**Abstract:**
The goal of this project is to create an Unmanned Ground vehicle (UGV) that would compete as a team with an Unmanned Aerial  Vehicle (UAV), in a competition held Annually by Raytheon at Virginia Tech.
The competition involves 4 challenges and a bonus round, and our aim is to create a vehicle that meets all the competition requirements and is able to complete all challenges within the timeframe given.

The Competition Challenges are as follows:

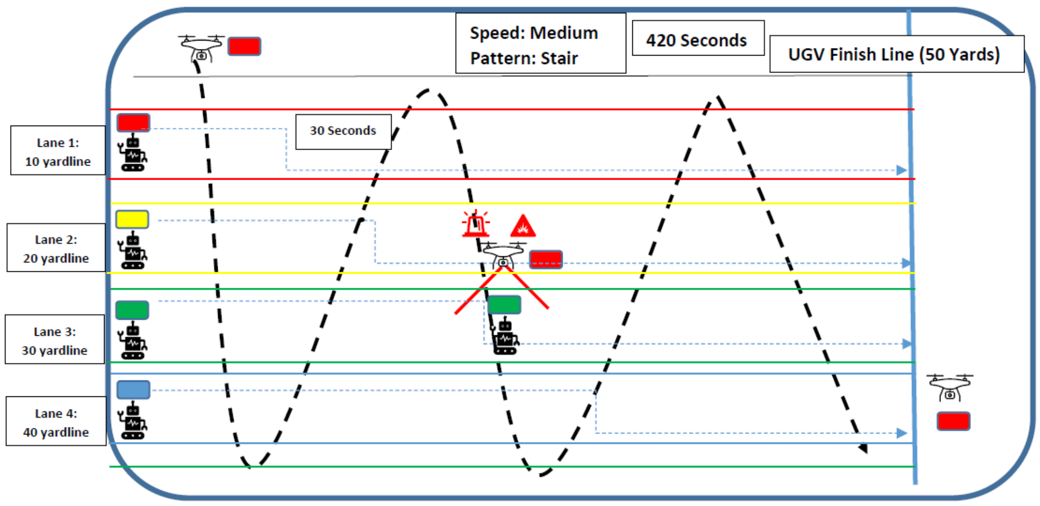
- Challenge 1: The UGVs are stationary. The UAV autonomously navigates through the defined space and drops water blasts on all enemy UGVs found within a 10 minute window, then lands in the designated landing zone.



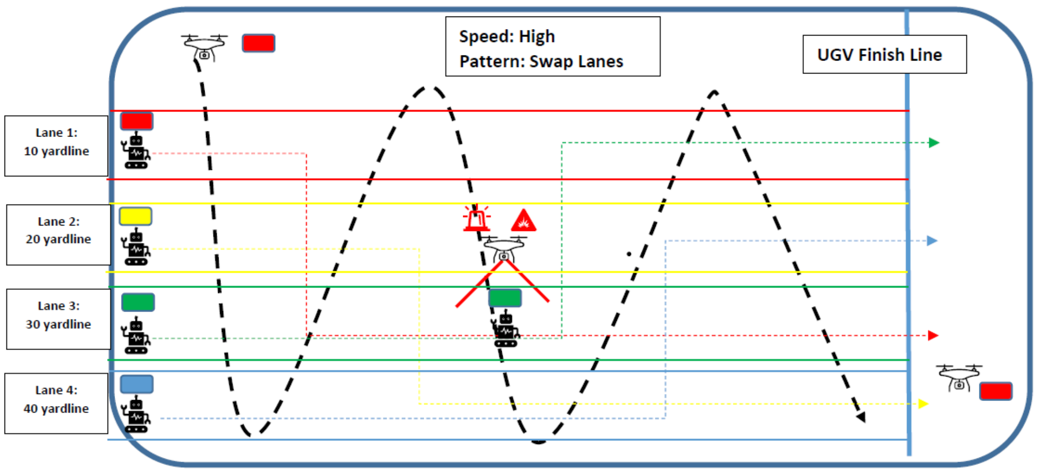
- Challenge 2: The UGVs follow a straight path on a swim lane and at a speed determined by the event judge. The UAV autonomously navigates through the defined space and drops water blasts on all enemy UGVs found within a 10 minute window, then lands in the designated landing zone.



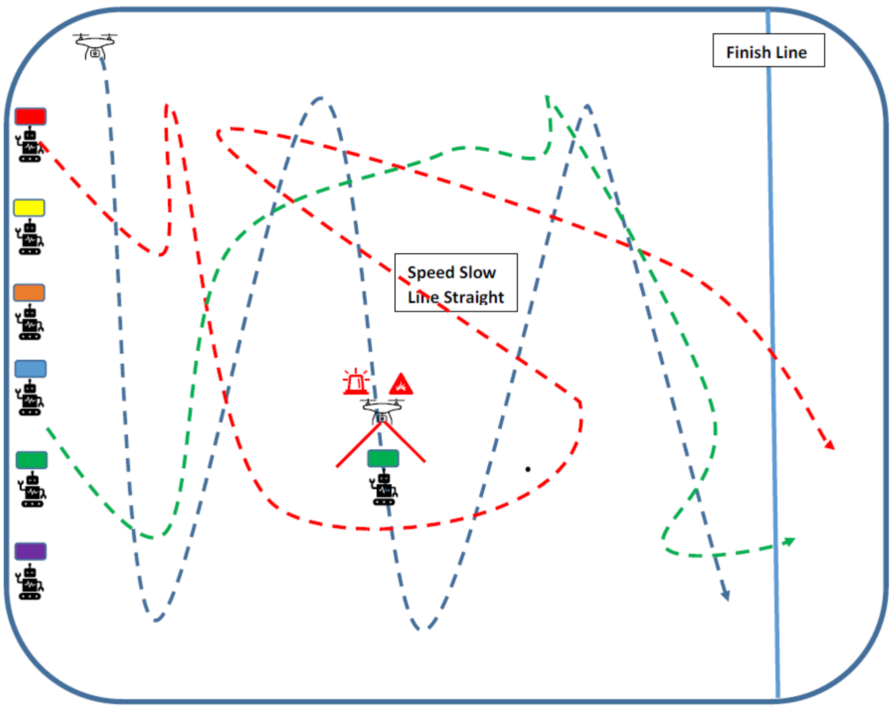
- Challenge 3: The UGVs take a right and left turn in a single swim lane and at a speed determined by the event judge. The UAV autonomously navigates through the defined space and drops water blasts on all enemy UGVs found within a 7 minute window, then lands in the designated landing zone.



- Challenge 4: The UGVs swap swim lanes with other UGVs determined by the event judge. The UAV autonomously navigates through the defined space and drops water blasts on all enemy UGVs found within a 5 minute window, then lands in the designated landing zone.

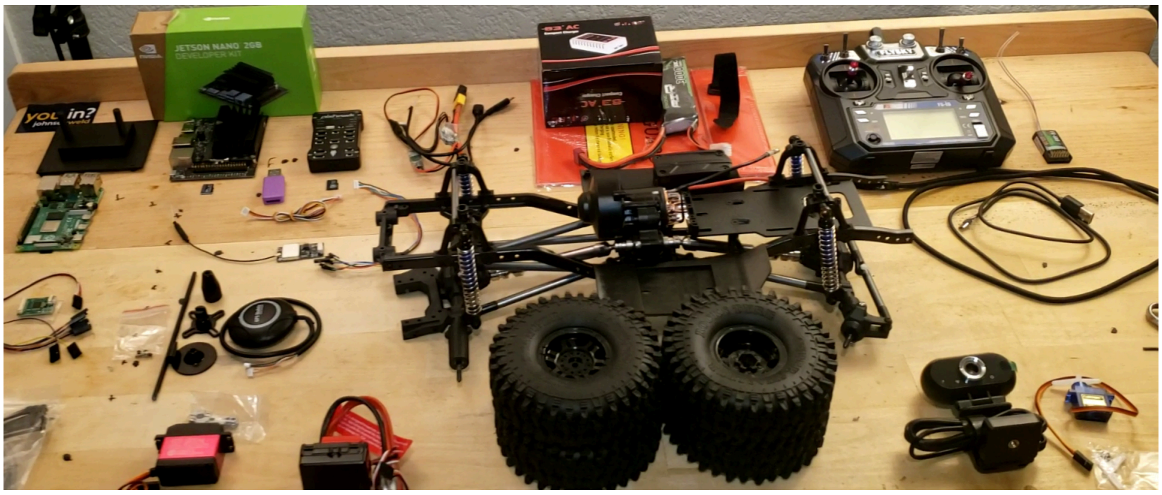


- Bonus Challenge: The UGVs follow a random path in the defined space while avoiding other UGVs. The UAV autonomously navigates through the defined space and drops water blasts on all enemy UGVs found within a 10 minute window, then lands in the designated landing zone.



The UGV is required to implement a moisture sensor that triggers upon detecting at least a 20ml volume of water. Upon triggering the moisture system, the UGV should stop, trigger a light, and make an audio alert.

To save time on the Project, given the time constraint, and to ensure a stable design, we decided to modify an existing UGV design to fit the specific challenges, by adding the necessary sensors and project specific python scripts. After some research we decided to get our UGV parts from the Drone Dojo Rover development kit.



The kit includes the majority of the components we need for the project and has a tried and tested Navigation module, it also comes with youtube videos that can help guide us with putting the UGV together.

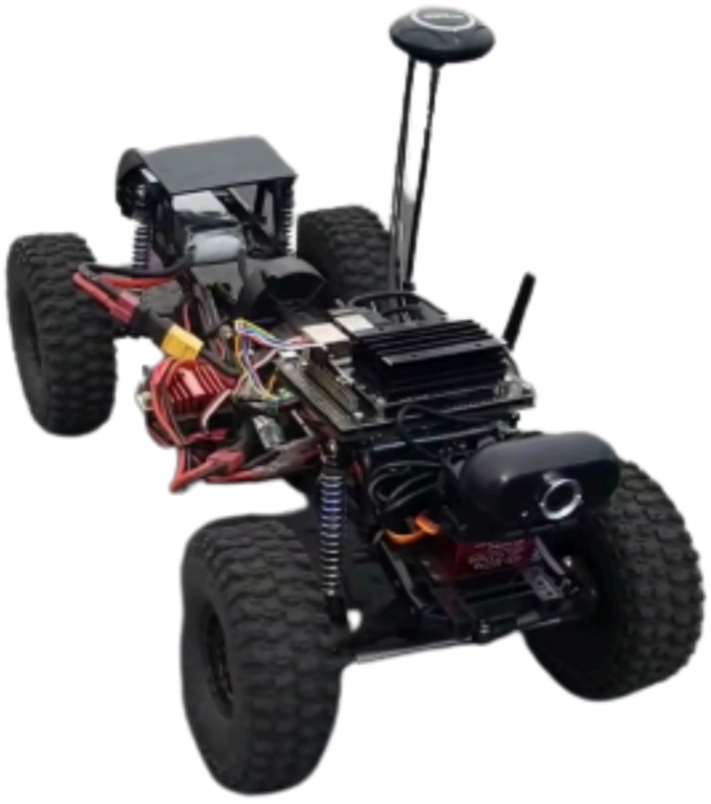
We plan on using A raspberry pi as the central computer between our sensors and the UGV. The raspberry pi would collect real-time data from the sensors then control the UGV accordingly by sending commands to the pixhawk flight controller, which handles the navigation of the UGV.

The UGV will be controlled using Python and Raspberry pi OS for the raspberry pi and Mission Planner Ardupilot software for the pixhawk flight controller.

Mission Planner

ARDUPILOT

Versatile, Trusted, Open



‹

**Problem Statement:**
The ground vehicle should be able to autonomously navigate through a predefined course, recognize and avoid obstacles in its path. We looked into adding features such as a GPS module, obstacle detecting sensors at the front of the vehicle, and wifi or bluetooth modules. These features can help ensure a vehicle that always takes optimised routes, prioritises a safe and smooth operation all while communicating and working with a drone. The vehicle uses the selected optimised routes, and data from the sensors to ensure it reaches its destination using the quickest available route, while avoiding any obstacles on the way.

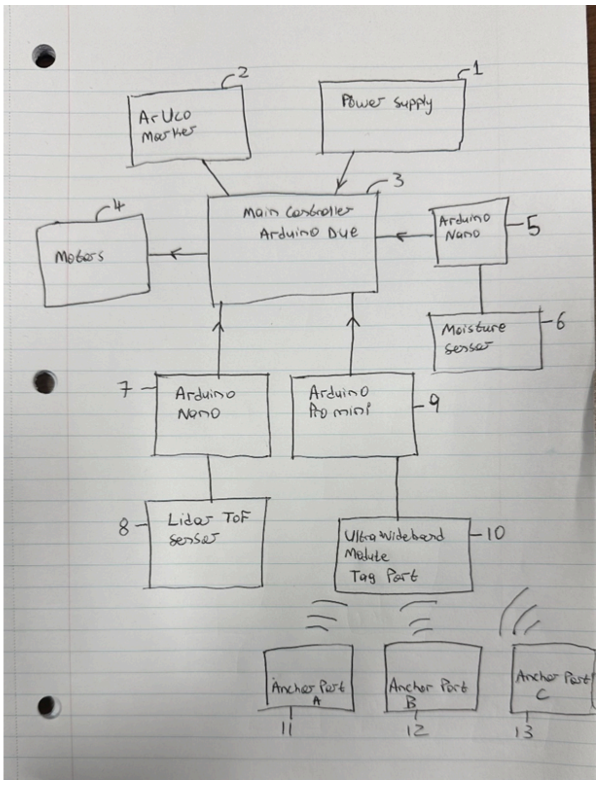
**Design Requirements:**

| **Requirements** | **Items** | **Quantity** |
|---|---|---|
| **1. Product Specification** | GPS and Compass Module | 1 |
| | On board Computer | 1 to 2 |
| | Distance sensors | 1 to 2 |
| | Rechargeable Batteries | 1 to 2 |

| | | |
|---|---|---|
| | Communication module (Wi-fi or Bluetooth) | 1 |
| | Buzzer (for audible alerts) | 1 |
| | Gear sets (for motors) and wheels | 4 |
| | Moisture Sensors | 3 to 6 |
| | ArUco marker | 1 |
| **2. Constraints** | **Environmental Constraints**<br><br>1. Energy Consumption<br><br>2. Emissions<br><br>3. Noise Pollution | |
| | **Socio-Cultural Constraints**<br><br>1. Privacy Concerns<br><br>2. Safety Concerns<br><br>3. Public Perception | |
| | **Compliance (Rules, Regulations, and Standards)**<br><br>1. Testing and Certifications<br><br>2. Regulatory Approvals<br><br>3. Data Protection<br><br>4. Liability and Insurance<br><br>5. FAA Regulations | |

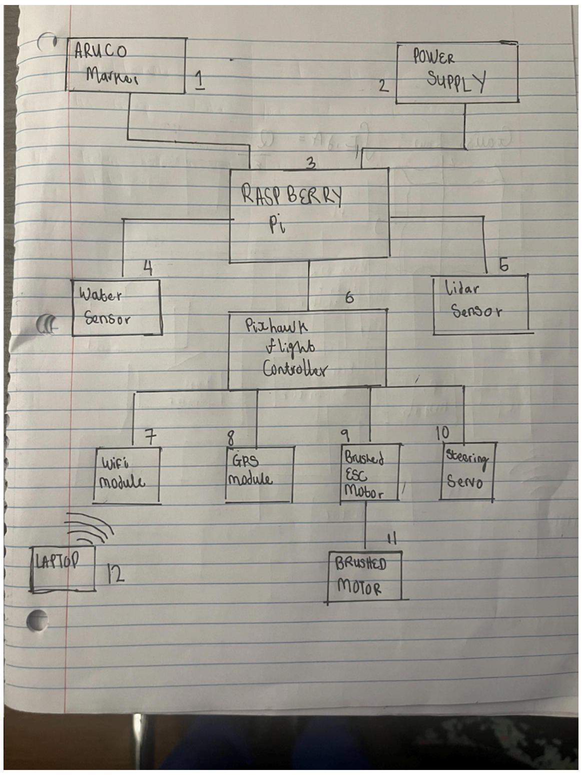
**Design Ideas:**

**Team 1** (Teminijesu Oyedele and Ernest Olopoenia)**:**



The location of the UGV is determined by triangulating using radio waves sent by ultra-wideband Anchor ports (11, 12, & 13) and received by the ultra-wideband Tag port (10) on the UGV. The location data is then sent to the main Arduino Due (3) by an Arduino Pro Mini (9) which communicates with the Tag port (10). The UGV navigates to the correct destination using the main Arduino Due (3) to control the motors (4). The UGV detects obstacles using a Lidar sensor (8) that communicates with an Arduino Nano (7) which sends data about the obstacle to the main Arduino Due (3). The UGV detects moisture using a moisture sensor (6) that communicates with an Arduino Nano (5) which sends data about the obstacle to the main Arduino Due (3). All the electronic components are powered by the power supply (1). The ArUco marker (2) is placed on the UGV facing upwards.

**Team 2**(Chidi Onyekwelu and Oghosa Osaghae):



The Pixhawk flight controller (6) serves as a primary control system for the UGV, processing real-time data from the GPS and Compass (8) to determine the UGV's position and orientation. It then controls the Steering Servo (10) and Brushed Motor ESC (9), which in turn controls the Brushed Motor (11) based on instructions sent to the Pixhawk (6) through the Wi-Fi module (7) from the ArduRover software on a Laptop (12) or through a python script on the Raspberry Pi (3). The Raspberry Pi (3) acts as an onboard computer, handling object avoidance and water detection by processing real-time data from the Lidar (5) and Water sensor (4).

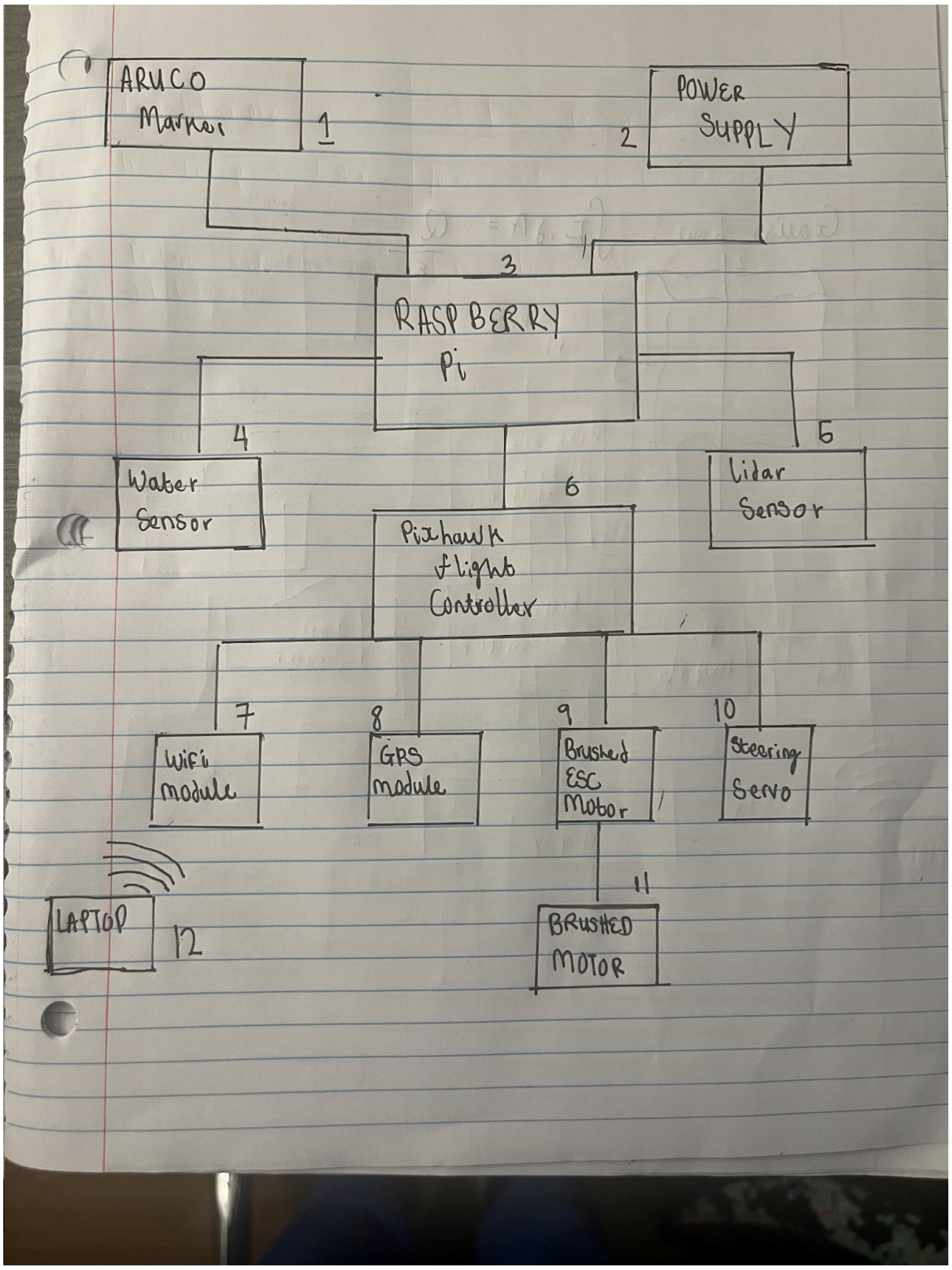
| Teams | Pros | Cons |
|---|---|---|
| Team 1 | ● Accurate locator module (to cm)<br>● Accurate obstacle detection<br>● Navigation module works indoors and outdoors | ● Uses more components.<br>● Involves complex wiring.<br>● Heavy due to the number of components. |
| Team 2 | ● Tested and tried Navigation module.<br>● Accurate obstacle detection | ● Learning curve for Raspberry Pi and ArduRover software<br>● Expensive |

| | | |
|---|---|---|
| | ● Development kit most components can be bought.<br>● Guide available for development.<br>● User Friendly | |

**Decision Matrix:**

| | Wt | Design 1 | Score | Agg. Score | Design 2 | Score | Agg. Score |
|---|---|---|---|---|---|---|---|
| Functionality | 5 | Ultra-wideband Technology (UWB) | 4 | 20 | GPS & Compass Technology | 5 | 25 |
| Connectivity | 2 | Radio Waves | 5 | 10 | WiFi | 5 | 10 |
| Weight | 4 | Approx 6-7 lbs | 3 | 12 | Approx 6-7 lbs | 3 | 12 |
| Power | 3 | More components to power | 2 | 6 | Less Components to power | 4 | 12 |
| Convenience | 1 | Learning curve to implement UWB chips | 3 | 3 | Comes with guides and videos | 5 | 5 |
| TOTAL | | | | 51 | | | 64 |

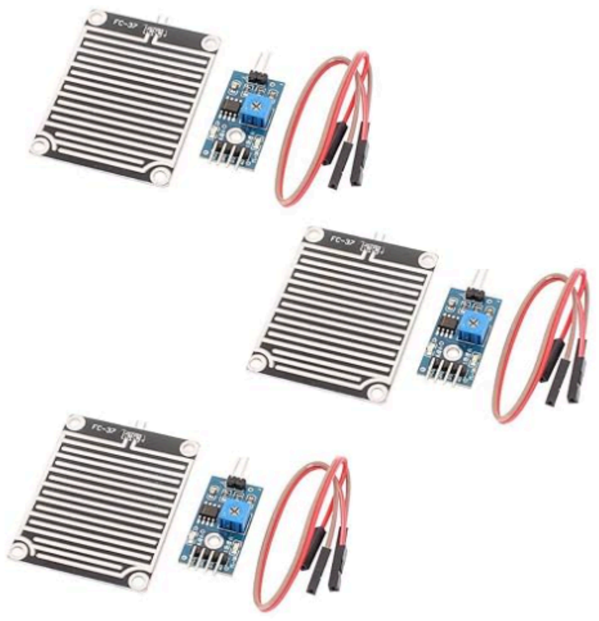
**Top Solution:**



The Pixhawk flight controller (6) serves as a primary control system for the UGV, processing real-time data from the GPS and Compass (8) to determine the UGV's position and orientation. It then controls the Steering Servo (10) and Brushed Motor ESC (9), which in turn controls the Brushed Motor (11) based on instructions sent to the Pixhawk (6) through the Wi-Fi module (7) from the ArduRover software on a Laptop (12) or through a python script on the Raspberry Pi (3). The Raspberry Pi (3) acts as an onboard computer, handling object avoidance and water detection by processing real-time data from the Lidar (5) and Water sensor (4).

**Components and Prices:**

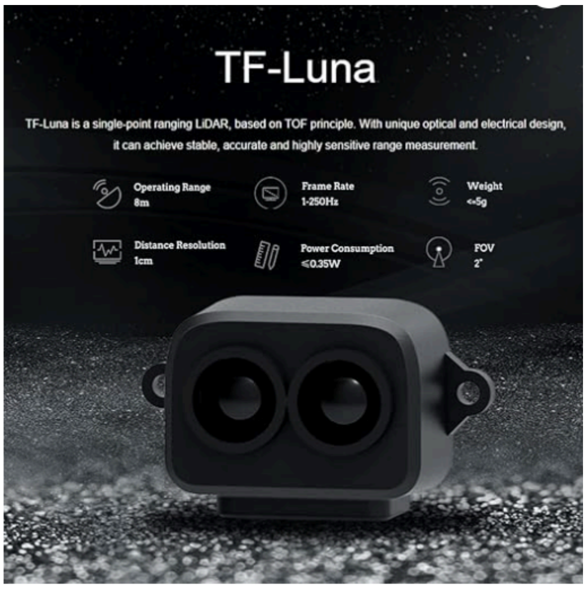
1. **Water Sensor($6)**:
   LM393 rain drop sensor link:
   https://www.amazon.com/HiLetgo-Moisture-Humidity-Sensitivity-Nickeled/dp/B01DK29K28/ref=sr_1_3?keywords=rain+sensor+arduino&qid=1699226521&sr=8-3



- The sensor uses high quality FR-04 double-sided materials, large area of 5.0 * 4.0CM, and nickel-plated surface, with oxidation resistance, conductivity, and life expectancy superior performance
- Comparator output, the signal clean, good waveform, strong driving ability, more than 15mA
- The raindrop board and the control board are separate, easy to lead the line
- TTL level output, TTL output valid signal is low. Drive capacity of about 100MA, direct drive relays, buzzers, small fans, and so on.
- Adjust sensitivity by potentiometer

2. **Lidar ToF Sensor($26)**:

TF-Luna Lidar:
https://www.amazon.com/Benewake-TF-Luna-Single-Point-Ranging-Interface/dp/B086MJQSLR


TF-Luna
TF-Luna is a single-point ranging LiDAR, based on TOF principle. With unique optical and electrical design, it can achieve stable, accurate and highly sensitive range measurement.
Operating Range 8m
Frame Rate 1-250Hz
Weight <=5g
Distance Resolution 1cm
Power Consumption ≤0.35W
FOV 2°

- 5V input voltage
- Communication level: LVTTL(3.3V), Communication interface: UART/IIC (the default is UART, you can send comment to set it to IIC ), Default baud rate: 115200
- Low-cost ranging LiDAR module with highly stable, accurate, sensitive range detection. Operating range: 0.2-8m
- Application: Traffic Monitoring, Obstacle detection, Level measurement, Smart device, Security and obstacle avoidance, Drone altitude holding and terrain following

3. **Raspberry Pi ($61)**
   Raspberry Pi 4 Model B 2019 Quad Core 64 Bit WiFi Bluetooth (4GB) :
   https://www.amazon.com/Raspberry-Model-2019-Quad-Bluetooth/dp/B07TC2BK1X/ref=sr_1_3?crid=18T4JG565CVOZ&keywords=Raspberry+Pi&qid=1699810273&sprefix=raspberry+pi%2Caps%2C105&sr=8-3&ufe=app_do%3Aamzn1.fos.006c50ae-5d4c-4777-9bc0-4513d670b6bc

\

- Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1. 5GHz---4GB LPDDR4-2400 SDRAM
- 2. 4 GHz and 5. 0 GHz IEEE 802. 11B/g/n/ac Wireless LAN, Bluetooth 5. 0, double-true Gigabit Ethernet
- 2 × USB 3. 0 ports, 2 x USB 2. 0 Ports---2 × micro HDMI ports supporting up to 4Kp60 video resolution
- 2-lane MIPI DSI/CSI ports for camera and display--4-pole stereo audio and composite video port--Micro SD card slot for loading operating system and data storage
- Requires 5. 1V, 3a power via USB Type C or gpio-poep (power over Ethernet) enabled (requires PoE hat-not included)

4. **64GB SD Card ($19):**
   https://www.amazon.com/SanDisk-Extreme-UHS-I-Memory-SDSDXXU-064G-GN4IN/dp/B09X7C7NCZ/ref=sr_1_3?crid=2G5Z7HV1STJAO&keywords=64%2Bgb%2Bsd%2Bcard&qid=1700102936&sprefix=64%2BGb%2B%2Caps%2C117&sr=8-3&th=1

- Save time with card offload speeds of up to 200MB/s powered by SanDisk QuickFlow Technology
- Item weight: 0.071 ounces, Dimensions: 0.09 x 0.94 x 1.26 inches

5. **Pixhawk flight controller ($190) :**
   https://www.amazon.com/sspa/click?ie=UTF8&spc=MTo4Nzc4NzE2MTYyNzMzNjgwOjE2OTk4MTAzMDk6c3BfYXRmOjIwMDEwNzEyMzY3OTc5ODo6MDo6&url=%2FR

eadytosky-Pixhawk-Controller-Autopilot-Splitter%2Fdp%2FB07CHQ7SZ4%2Fref%3Dsr_1_1_sspa%3Fcrid%3D23ANJVWM6PIYK%26keywords%3DPixhawk%2BFlight%2BController%26qid%3D1699810309%26sprefix%3Dpixhawk%2Bflight%2Bcontroller%252Caps%252C85%26sr%3D8-1-spons%26ufe%3Dapp_do%253Aamzn1.fos.f5122f16-c3e8-4386-bf32-63e904010ad0%26sp_csd%3Dd2lkZ2V0TmFtZT1zcF9hdGY%26psc%3D1

- The advanced 32-bit ARM CortexM4 high-performance processors, can run NuttX RTOS real-time operating system;
- We have M8N GPS for the Pixhawk flight controller,ASIN : B01KK9A8QG
- And the 3DR Radio Telemetry Kit for this flight controller,ASIN: B01DHV4DVA
- Item weight: 3.13 ounces, Dimensions: 4.49 x 4.09 x 1.46 inches

6. **1/10th Crawler Chassis ($97) :** https://www.amazon.com/INJORA-Wheelbase-Chassis-Crawler-Prefixal/dp/B083ZSPN8V/ref=sr_1_5?crid=326GX7IB06M9Q&keywords=Crawler+Chassis&qid=1699810594&sprefix=crawler+chassis%2Caps%2C65&sr=8-5&ufe=app_do%3Aamzn1.fos.006c50ae-5d4c-4777-9bc0-4513d670b6bc

- Made of high quality metal, strong and durable.
- Novel design, makes your car more eye-catching.
- Comes with Prefixal Gearbox, easy and convenient to install.
- 313mm Wheelbase, suits for 1:10 RC crawler Axial SCX10 SCX10 II 90046.
- Prefixal gearbox design, best choice for RC Car fans.
- Weight: 1.3 pounds, Dimensions: 19.13 x 3.94 x 3.62 inches

7. **Brushed Motor and Brushed Motor ESC combo ($32) :** https://www.amazon.com/sspa/click?ie=UTF8&spc=MTo0NTg5MTk1ODgyNTU5NjgwOjE2OTk4MTEwMTY6c3BfYXRmOjIwMDEwMzMwNzYwMDcyMTo6MDo6&url=%2FGLOBACT-Brushed-Crawler-Traxxas-Bronco%2Fdp%2FB09GF4HKKB%2Fref%3Dsr_1_2_sspa%3Fcrid%3D1UFTBXAEB3HWL%26keywords%3Dbrushed%2Bmotor%2Besc%26qid%3D1699811016%26sprefix%3Dbrushed%2Bmotor%2B%252Caps%252C114%26sr%3D8-2-spons%26sp_csd%3Dd2lkZ2V0TmFtZT1zcF9hdGY%26psc%3D1

Roll over image to zoom in

- Applicable to All Weather Conditions, The waterproof and dust-proof design allows the Globact 21T 550 motor 80A esc Brushed to be used in all weather conditions without any issue of damage caused to the ESC from water or dust.
- Suit for 1/10 Scale Electric Rc crawler car Trx4/1979&2021/Trx6/Replaces Titan 550 21 Turn Motor (Model #3975)/Axial Scx10 II III/Redcat Gen7 Gen8 V2/RC4WD D90.
- Low voltage cut-off protection for lithium or nickel battery / Over-heat protection / Throttle signal loss protection. Note: The ESC signal cable has reverse connection protection, The power cord has no reverse connection protection.
- ESC supports BEC high voltage & low voltage switching (5A 7.4V/6.4V), Crawlers & High Speed Dual Mode switching, Input voltage:7.4V (2S Lipo) / 6V-8.4V (5-7 cells NiMH).
- Weight: 50g Dimensions: 42x35x27mm

8. **RC Transmitter and Receiver ($159):**
https://www.amazon.com/sspa/click?ie=UTF8&spc=MTo4NTg1OTA3MTE4MjE2NjgwOjE2OTk4MTExMzU6c3BfYXRmOjIwMDA3NDI2MDYyNTA5ODo6MDo6&url=%2FRadiolink-Transmitter-Controller-Multicopters-Helicopter%2Fdp%2FB07FPF2HQR%2Fref%3Dsr_1_1_sspa%3Fcrid%3DGIPZW76GG0E%26keywords%3DRC%2BTransmitter%2Band%2BReceiver%26qid%3D1699811135%26sprefix%3Drc%2Btransmitter%2Band%2Breceiver%252Caps%252C97%26sr%3D8-1-spons%26ufe%3Dapp_do%253Aamzn1.fos.f5122f16-c3e8-4386-bf32-63e904010ad0%26sp_csd%3Dd2lkZ2V0TmFtZT1zcF9hdGY%26psc%3D1

- Excellent Anti-interference--DSSS&FHSS communication technology and 7dBi high gain antenna make AT10II transmitter with stable transmission signal and the control range is up to 2.5 miles(4km) in the air.
- Key Features--12 channels, Battery Voltage Telemetry, RSSI telemetry, Dual/Triple Rates, Throttle Lock, Fail-safe Setting, End Point Adjustment(EPA), Switch Customized, Channels Mix Control, etc.
- User-Friendly Interface: support basic menu and advanced menu for different user needs and the user interface is quite intuitive and easy to navigate. No need for any radio setting to bind the transmitter with receivers.
- Safety Signal Strength Indicator (RSSI) Alerts, low voltage alarm, failsafe protection. DD sounds and words alarming display on the 3.5" LCD screen to remind you when you are in noisy surroundings.
- A large 3.5-inch 320*480 resolution color display is easy to read. Background color can switch from black to white.
- Weight: 2.09 pounds, Dimensions: 7.1 x 3.7 x 8.7 inches

9. **Webcam ($35):**
https://www.amazon.com/Webcam-Microphone-Streaming-Privacy-Computer/dp/B09MJ642ZY/ref=sr_1_16?crid=2AK0CF2DT48ML&keywords=Webcam+for+OpenCV+Scripting&qid=1699811414&sprefix=webcam+for+opencv+scripting%2Caps%2C97&sr=8-16

- Autofocus Streaming Webcam with Privacy Cover - With the EMEET C965 USB webcam, the autofocus function can capture every moment you move without worrying about losing focus, giving you a better video experience
- 1080P Webcam with Auto-Low Light Correction - EMEET C965 PC webcam is a 30fps HD webcam, which has 2 million pixels to provide clear image quality.
- 2 Built-in Noise Cancellation Microphones - EMEET C965 webcam 1080P has 2 noise reduction mics inside. Webcam with microphone can greatly improve the sound quality of live video or online calls, filter ambient noise, and optimize the call effect.
- Weight: 5.8 ounces, Dimensions: 3.94 x 2.32 x 2.13 inches

**10. Lipo Battery Charger ($10):**
https://www.amazon.com/HTRC-Battery-Balancer-Charger-7-4-11-1V/dp/B073WSDCZM/ref=sr_1_15?crid=3I5H0T9NIBIM8&keywords=lipo%2Bbattery%2Bcharger&qid=1699896987&sprefix=lipo%2Bbattery%2Bcharger%2Caps%2C794&sr=8-15&th=1

B3 AC
Compact Charger
HTRC
Input voltage:100-240V AC
Max charge current:3*1000mA
cell1
cell2
cell3
Roll over image to zoom in

- Parameter: 2S & 3S charger, AC100-240V input, max charge current: 3*1000mA. Our charger's shell is made of fire-proof material.
- Each cell is charged by an independent line, which makes sure each battery's cell is always fully charged, extending the battery's life. Note: This balanced compact charger is suitable for LiPo Battery ONLY.
- LED to indicate: Double color LED can display the process of charging (RED: charging; GREEN: full).
- Weight: 6 ounces, Dimensions: 3.54 x 2.17 x 1.3 inches

11. **Lipo Battery($45):** https://www.amazon.com/OVONIC-Dean-Style-Connector-Airplane-Helicopter/dp/B07CVDP9DB/ref=sr_1_4?crid=HLVP129YXOOC&keywords=lipo%2Bbattery%2BCAR%2B3000&qid=1700102536&sprefix=lipo%2Bbattery%2Bcar%2B3000%2Caps%2C70&sr=8-4&th=1

- Strong Compatibility - Compatible with RC Evader BX Car Truck Truggy Airplane UAV Drone FPV Helicopter 1/10 scale Crawler truck.
- 2x OVONIC 3s Lipo battery
- Multiple Advantages - ISO international quality certification(safer), premium materials(LiCoO2), longer cycle life(350 times), longer running time(+12%), faster charging time(-6%), lighter weight( -5%), lower Resistance(-4%)
- Specification - Ovonic 3000mAh 11.1V Lipo Battery Dimension(±3mm): 105*32*28mm /4.13*1.26*1.1inch(L*W*H), Approx Weight(±5g): 193g /0.42lb
- Parameter - Material: Lithium polymer, Voltage: 11.1V, Cell: 3S, Capacity: 3000mAh, Discharge: 50C, Charging plug: JST-XHR-4P, Discharging plug: Dean-Style T connector, Pack: Soft case
- Weight: 6.7 ounces, Dimensions: 6.42 x 2.52 x 2.05 inches

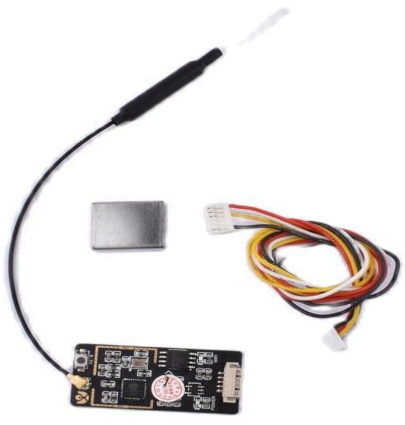
12. **Lipo Battery Fireproof case($15):**

https://www.amazon.com/Battery-Fireproof-Explosion-Proof-Storage/dp/B078Y723Z3/ref=sr_1_35?crid=2G90GM6MC4ODY&keywords=lipo+battery+fireproof+case&qid=1699896758&sprefix=lipo+battery+fireproof+cas%2Caps%2C191&sr=8-35

- Use for safer charging and storage of batteries
- Made from fire-proof and explosion-proof Material
- Velcro closure, easy to open and close
- Stylish and artistic appearance, convenient to carry
- Single Dimensions (LxWxD): 125 x 64 x 50mm
- Weight: 7ounces , Dimension : 11.02 x 4.96 x 0.75 inches

**13. Wifi Telemetry Module($31)**

[https://www.amazon.com/PONPED-Telemetry-Compatible-Controller-Smartphone/dp/B0CLHXLY2R/ref=sr_1_4?crid=2HT7714ARAFSK&keywords=Wifi%2BTelemetry%2BModule&qid=1699819479&sprefix=wifi%2Btelemetry%2Bmodule%2Caps%2C107&sr=8-4&th=1](https://www.amazon.com/PONPED-Telemetry-Compatible-Controller-Smartphone/dp/B0CLHXLY2R/ref=sr_1_4?crid=2HT7714ARAFSK&keywords=Wifi%2BTelemetry%2BModule&qid=1699819479&sprefix=wifi%2Btelemetry%2Bmodule%2Caps%2C107&sr=8-4&th=1)



- High-quality materials, environmentally friendly and durable
- Damaged parts can be completely replaced

- Weight: 12.2 ounces, Dimensions: 1.18 x 0.79 x 0.39 inches

**14. GPS, Compass Module and Mount ($35):**
[https://www.amazon.com/FPVDrone-Compass-Protective-Anti-Interference-Controller/dp/B06XR6JHDW/ref=sr_1_2_sspa?crid=3IZA9V1SPAEJA&keywords=GPS+and+Compass+Module&qid=1699819529&sprefix=gps+and+compass+module%2Caps%2C63&sr=8-2-spons&sp_csd=d2lkZ2V0TmFtZT1zcF9hdGY&psc=1](https://www.amazon.com/FPVDrone-Compass-Protective-Anti-Interference-Controller/dp/B06XR6JHDW/ref=sr_1_2_sspa?crid=3IZA9V1SPAEJA&keywords=GPS+and+Compass+Module&qid=1699819529&sprefix=gps+and+compass+module%2Caps%2C63&sr=8-2-spons&sp_csd=d2lkZ2V0TmFtZT1zcF9hdGY&psc=1)

- M8N GPS Module for APM2.6 APM2.8 Flight Controller.
- A new generation M8N GPS Module,with low power consumption and high precision.
- The M8N and ceramic antenna make this a very accurate receiver,fast locks & lots of satellites.
- Note:it is the default for the APM flight controller,if you use the Pixhawk flight controller,you need to change the connector by yourself.
- Weight: 3.2 ounces, Dimensions: 4.8 x 4.09 x 0.2 inches

**15. Steering Servo ($30)**
[https://www.amazon.com/FEETECH-Coreless-Stainless-Waterproof-Aluminium/dp/B0BV9YT75Z/ref=sr_1_2_sspa?crid=2AUBFGCFTAYOT&keywords=Steering%2BServo&qid=1699819688&sprefix=steering%2Bservo%2Caps%2C83&sr=8-2-spons&sp_csd=d2lkZ2V0TmFtZT1zcF9hdGY&th=1](https://www.amazon.com/FEETECH-Coreless-Stainless-Waterproof-Aluminium/dp/B0BV9YT75Z/ref=sr_1_2_sspa?crid=2AUBFGCFTAYOT&keywords=Steering%2BServo&qid=1699819688&sprefix=steering%2Bservo%2Caps%2C83&sr=8-2-spons&sp_csd=d2lkZ2V0TmFtZT1zcF9hdGY&th=1)

- IGH QUALITY RC SERVO: This coreless servo is composed of a sturdy PA aluminum middle shell, a strong steel metal gear, and a hard iron shaft, provide higher stability and provide longer service life.
- HIGH TORQUE SERVO: The operating voltage range is 4V-8.4V, Stall Torque (at locked):(6V) 349.18oz/in(25.1kg/cm), (7.4V) 429.87oz/in(30.9kg/cm),(8.4V) 475.78oz/in(34.2kg/cm). Size is 40.17mm*20.16mm*36.5mm(1.58*0.79*1.44in).
- FULL WATERPROOF SERVO: IP54 waterproof function to maintain good water tightness in wet conditions. Will get amazing experience all the time.
- HIGH-PRECISION STEERING SERVO: High precision potentiometers and High-resolution digital chip, stable performance, strong anti-interference ability, fast response, high precision, control is more delicate and smooth operation.
- HIGH COMPATIBLE SERVOS: High performance and high-precision digital standard servo system is controlled by pulse, compatible with 1/8 and 1/10 scale RC robots, trucks, off-road vehicles, robotic arms, and any application that uses a standard servo system.
- Weight: 2.39 ounces , Dimensions: 1.58 x 0.79 x 1.44 inches

**16. Screws ($7):**

https://www.amazon.com/Mr-Screws-Assortment-Phillips-Assorted/dp/B092LP684Q/ref=sr_1_3?crid=3OSP7GMVWJ9GZ&keywords=screws&qid=1699819955&sprefix=screws%2B%2Caps%2C69&sr=8-3&th=1

- Total package quantity of 152 pcs consist 50 pcs x 20mm, 30 pcs x 25mm, 24 pcs x 30mm, 20 pcs x 35mm, 16 pcs x 40mm and 12 pcs x 50mm.
- Made of galvanized iron, corrosion-resistant, durable and steady.
- Don't easily slip, provides good holding power in different woods and man-made materials
- Weight: 1.3 pounds, Dimensions: 19.13 x 3.94 x 3.62 inches

**17. 20ml Cup for Testing ($6):**
https://www.amazon.com/Disposable-Graduated-Plastic-Medicine-Measuring/dp/B08JHB4RHY/ref=sr_1_6?crid=3NB3CLJ07407I&keywords=20+ml+cup+for+medicine&qid=1700010527&s=industrial&sprefix=20+ml+cup+%2Cindustrial%2C89&sr=1-6
- Manufactured with easy-to-read graduations on all sides
- Perfect for dispensing both liquids and medications
- cc or mL: (2.5, 5, 7.5, 10, 15, 20, 25, 30)

| **Component** | **Price** |
|---|---|
| Water Sensor | $6 X 10 |
| Lidar ToF Sensor | $26 X 4 |
| 64GB SD Card | $19 X 5 |
| Raspberry Pi | $61 X 4 |
| 20ml Measuring Cups | $6 |
| **Components In Drone Dojo UGV Kit** | |
| Raspberry Pi | $61 |
| Pixhawk Flight Controller | $190 |
| Brushed Motor & Motor ESC | $32 |
| Lipo Battery | $45 |
| Lipo Battery Charger | $10 |
| Lipo Battery Fireproof Case | $15 |
| GPS, Compass Module & Mount | $35 |

| | |
|---|---|
| Wifi Telemetry Module | $31 |
| Steering Servo | $30 |
| RC Transmitter and Receiver | $159 |
| 1/10th Crawler Chassis | $97 |
| Screws | $7 |

## Agile Project Implementation

| Starting Date of Week (M) | Sprint # | Increment (or intermediate working component) | Weekly development tasks |
|---|---|---|---|
| 1/30/2024 | 1 | REMOTE CONTROLLED UGV | ORDER PARTS |
| 2/5/2024 | | | ASSEMBLE PARTS TO FORM UGV |
| 2/12/2024 | | | TEST REMOTE CONTROLLED UGV |
| 2/19/2024 | 2 | DEVELOP PYTHON SCRIPTS AND SETUP SOFTWARE KITS | DEVELOP PYTHON SCRIPTS TO COMPLETE CHALLENGES |
| 2/26/2024 | | | TEST AND DEBUG PYTHON SCRIPTS |
| 3/4/2024 | | | INSTALL AND SET UP VM VIRTUALBOX |
| 3/11/2024 | | | INSTALL DEPENDENCIES NEED FOR SIMULATION |
| 3/18/2024 | 3 | SIMULATE AUTONOMOUS VEHICLE CHALLENGES | TEST PYTHON SCRIPTS ON VIRTUAL ENVIRONMENT |
| 3/25/2024 | | | DEBUG AND UPDATE SCRIPTS |
| 4/1/2024 | | | RUN SIMULATIONS |

## Agile Project Implementation Process

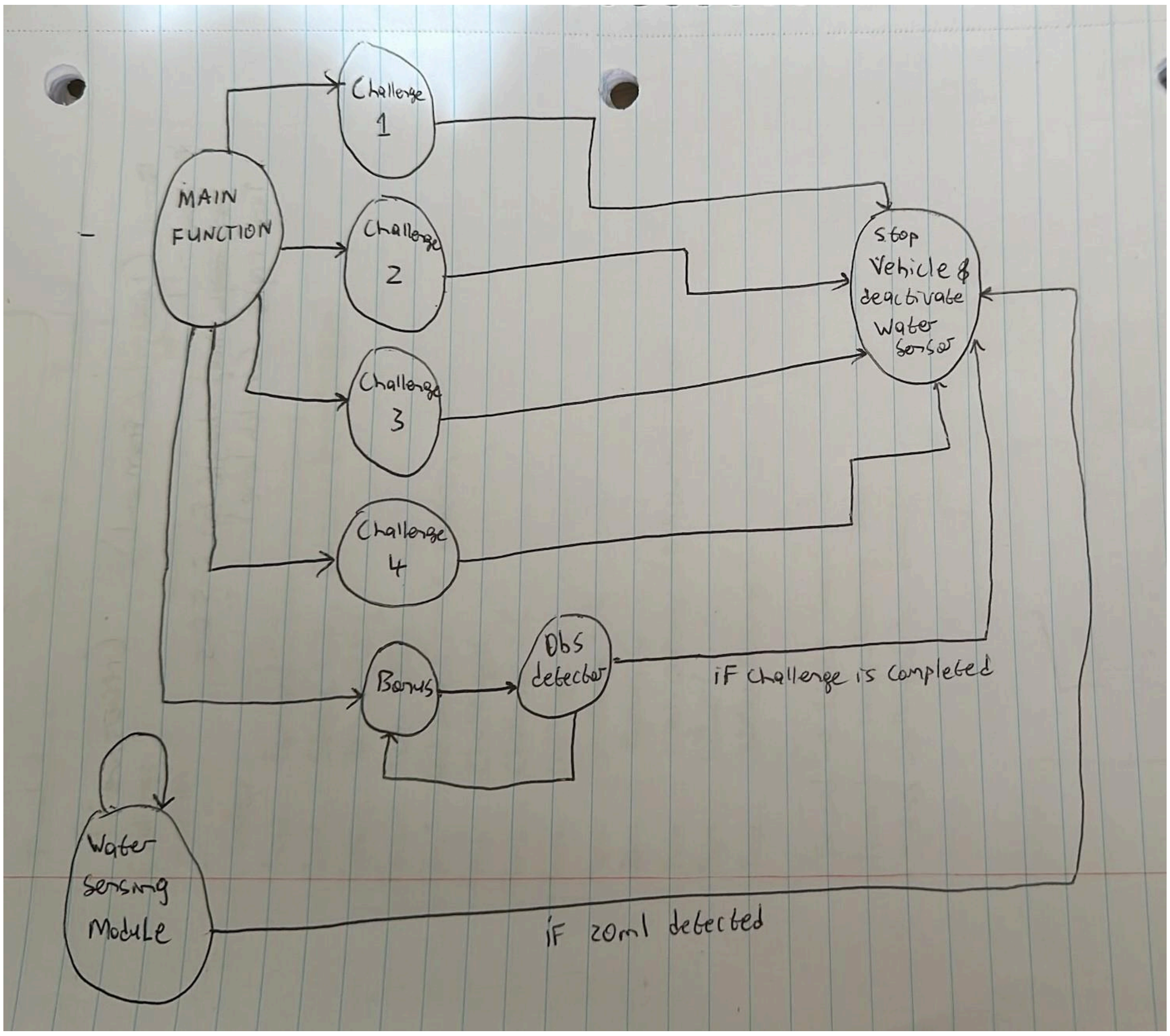
## Sprint 1:

The UGV team spent the first few weeks communicating with Dr Rawat and representatives from RTX, to sort the funding and ordering of the equipment we need. By the third week in sprint 1, the team had yet to order our parts due to some complications with school releasing the funds, so the team started to look at alternatives to progress with the project. The team decided to take a software approach and simulate the UGV in a virtual environment.

## Sprint 2 and 3:

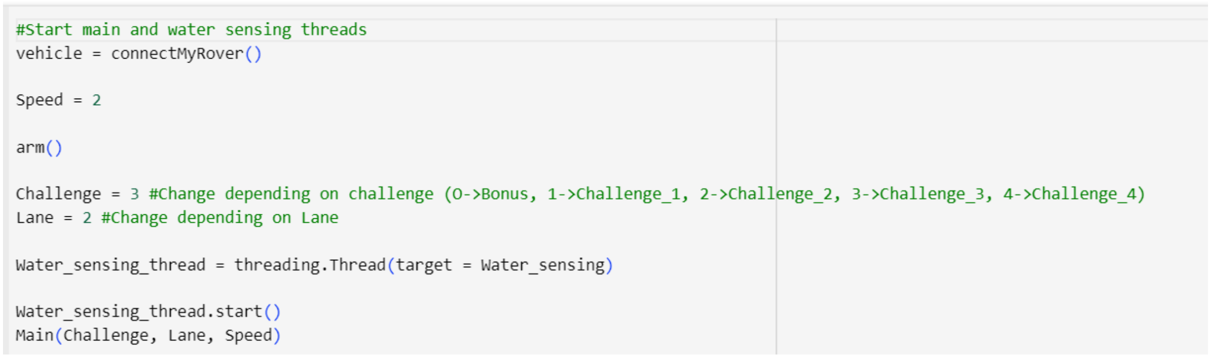
The UGV team spent the following three weeks on developing python scripts that could control both the physical and simulated UGV. We also worked on setting up a virtual environment that could run the simulations using Oracle VM VirtualBox. After setting up the software and installing the dependencies, we ran the simulation using Oracle VirtualBox.

### UGV Script Overview:



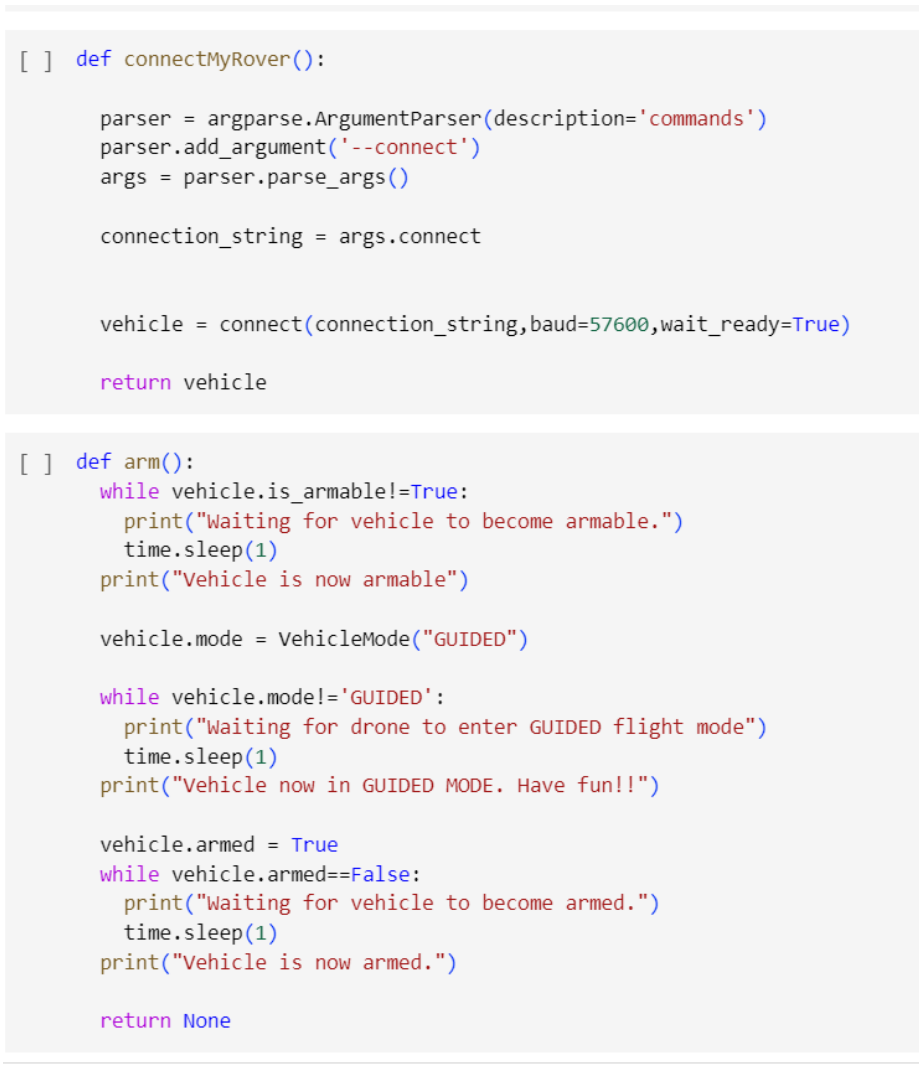
The script processes two threads simultaneously, one main thread responsible for the bulk of the work including identifying what challenge the UGV should complete and how it should complete it. The second thread is responsible for  detecting water dropped on the UGV and stopping the UGV when 20 ml of water is detected.

**<u>Script Details:</u>**



```
#Start main and water sensing threads
vehicle = connectMyRover()

Speed = 2

arm()

Challenge = 3 #Change depending on challenge (0->Bonus, 1->Challenge_1, 2->Challenge_2, 3->Challenge_3, 4->Challenge_4)
Lane = 2 #Change depending on Lane

Water_sensing_thread = threading.Thread(target = Water_sensing)

Water_sensing_thread.start()
Main(Challenge, Lane, Speed)
```

First we connect the script to the UGV using the “connectMyRover()” function, then we arm the UGV by setting it into “GUIDED” mode. Setting the UGV to “GUIDED” mode lets us send commands to the UGV using the script in place of the remote controller.

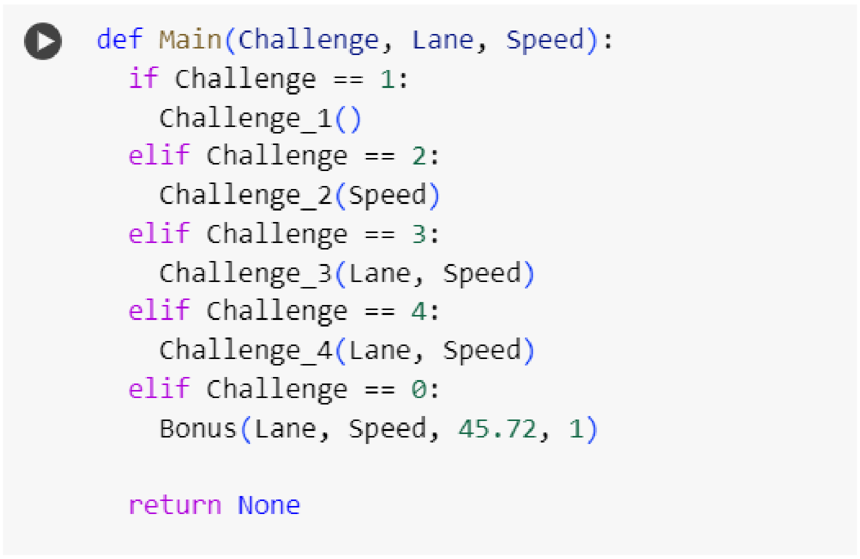


```
[ ]  def connectMyRover():

       parser = argparse.ArgumentParser(description='commands')
       parser.add_argument('--connect')
       args = parser.parse_args()

       connection_string = args.connect


       vehicle = connect(connection_string,baud=57600,wait_ready=True)

       return vehicle
```

```
[ ]  def arm():
       while vehicle.is_armable!=True:
         print("Waiting for vehicle to become armable.")
         time.sleep(1)
       print("Vehicle is now armable")

       vehicle.mode = VehicleMode("GUIDED")

       while vehicle.mode!='GUIDED':
         print("Waiting for drone to enter GUIDED flight mode")
         time.sleep(1)
       print("Vehicle now in GUIDED MODE. Have fun!!")

       vehicle.armed = True
       while vehicle.armed==False:
         print("Waiting for vehicle to become armed.")
         time.sleep(1)
       print("Vehicle is now armed.")

       return None
```

We set parameters like “Speed”, “Challenge”, and “Lane”. We use these parameters to determine what challenge the UGV would be completing, the lane it will be completing the challenge on and the speed it will complete the challenge at. We then start both threads mentioned earlier.

We start the main thread by calling the function “Main” using Speed, Challenge, and Lane as parameters.

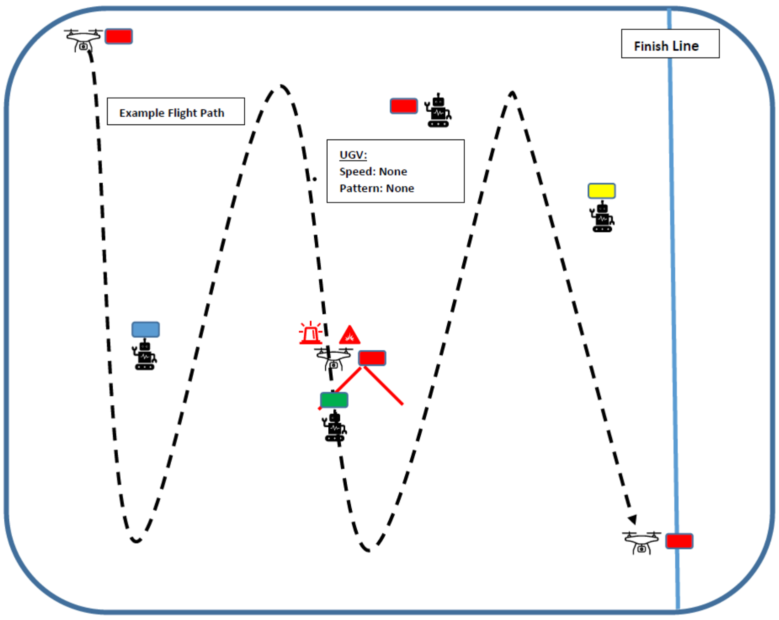


```
def Main(Challenge, Lane, Speed):
  if Challenge == 1:
    Challenge_1()
  elif Challenge == 2:
    Challenge_2(Speed)
  elif Challenge == 3:
    Challenge_3(Lane, Speed)
  elif Challenge == 4:
    Challenge_4(Lane, Speed)
  elif Challenge == 0:
    Bonus(Lane, Speed, 45.72, 1)

  return None
```

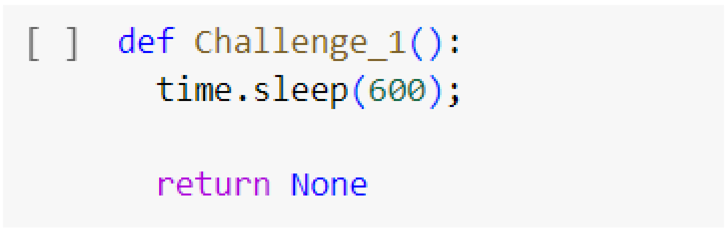
The function uses its parameters to determine what challenge the UGV should run and calls the appropriate function.

**Challenge 1 :**

The UGVs are stationary. The UAV autonomously navigates through the defined space and drops water blasts on all enemy UGVs found within a 10 minute window, then lands in the designated landing zone.



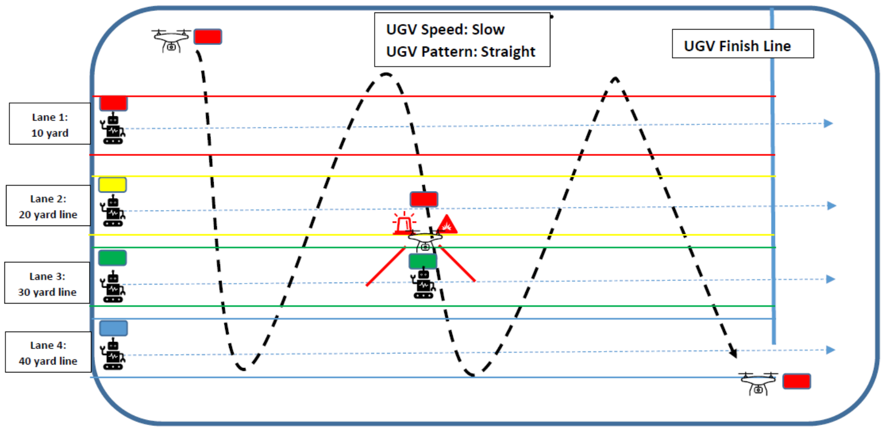
The “Challenge_1” function simply delays for the duration of the test while the water sensing thread checks if 20 ml of water is detected.



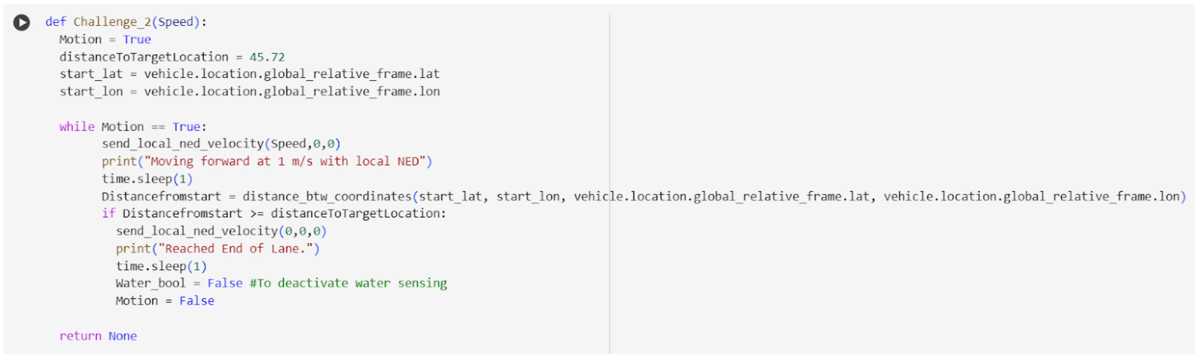
```
[ ]  def Challenge_1():
       time.sleep(600);

       return None
```

## Challenge 2:

The UGVs follow a straight path on a swim lane and at a speed determined by the event judge. The UAV autonomously navigates through the defined space and drops water blasts on all enemy UGVs found within a 10 minute window, then lands in the designated landing zone.

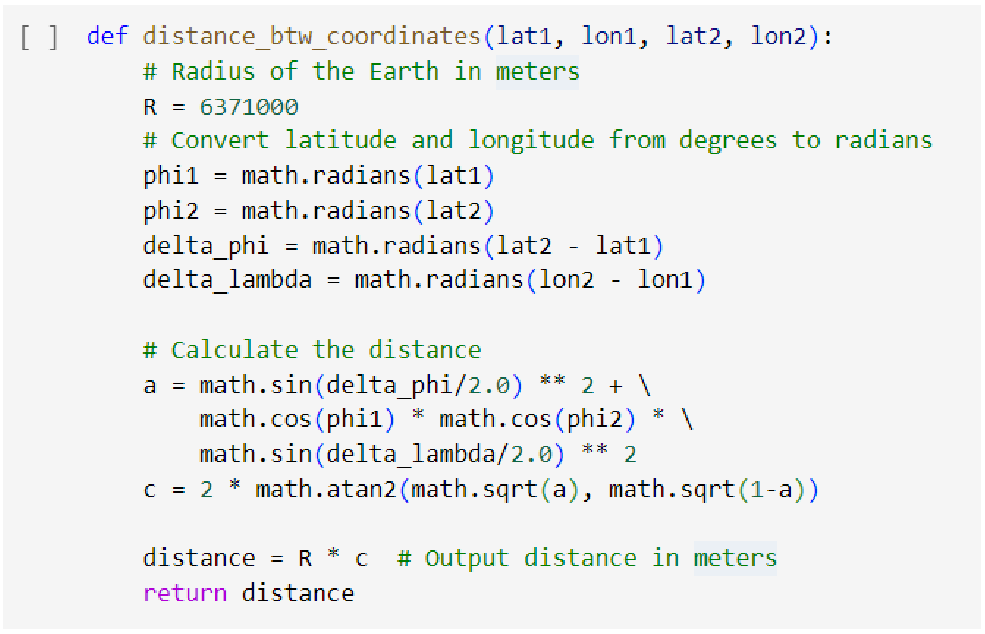


We control the UGV by sending continuous velocity based commands of the form "send_local_ned_velocity(speed,turn_speed,0)". The value of "speed" determines how fast the UGV moves forward, and the value of "turn_speed" determines how fast the UGV turns. The UGV turns right when "turn_speed" is positive but turns left when it is negative. To keep the UGV moving we need to send the velocity based command every second.
In the "Challenge_2 " function we use a while loop to send the command to move the UGV, every one second, on the condition that the boolean variable "Motion" remains true.



```
def Challenge_2(Speed):
  Motion = True
  distanceToTargetLocation = 45.72
  start_lat = vehicle.location.global_relative_frame.lat
  start_lon = vehicle.location.global_relative_frame.lon

  while Motion == True:
        send_local_ned_velocity(Speed,0,0)
        print("Moving forward at 1 m/s with local NED")
        time.sleep(1)
        Distancefromstart = distance_btw_coordinates(start_lat, start_lon, vehicle.location.global_relative_frame.lat, vehicle.location.global_relative_frame.lon)
        if Distancefromstart >= distanceToTargetLocation:
          send_local_ned_velocity(0,0,0)
          print("Reached End of Lane.")
          time.sleep(1)
          Water_bool = False #To deactivate water sensing
          Motion = False

  return None
```

Before the loop begins we store the coordinates of the UGV at the start of the challenge. In each iteration of the loop we use the function "distance_btw_coordinates()" to calculate the distance between the starting coordinates and the current coordinates.

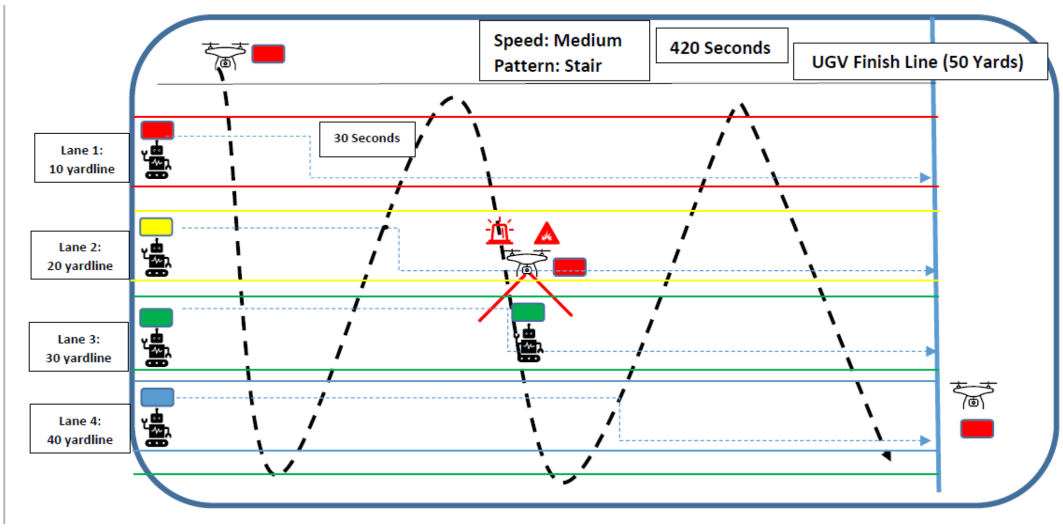


```
[ ]  def distance_btw_coordinates(lat1, lon1, lat2, lon2):
         # Radius of the Earth in meters
         R = 6371000
         # Convert latitude and longitude from degrees to radians
         phi1 = math.radians(lat1)
         phi2 = math.radians(lat2)
         delta_phi = math.radians(lat2 - lat1)
         delta_lambda = math.radians(lon2 - lon1)

         # Calculate the distance
         a = math.sin(delta_phi/2.0) ** 2 + \
             math.cos(phi1) * math.cos(phi2) * \
             math.sin(delta_lambda/2.0) ** 2
         c = 2 * math.atan2(math.sqrt(a), math.sqrt(1-a))

         distance = R * c  # Output distance in meters
         return distance
```

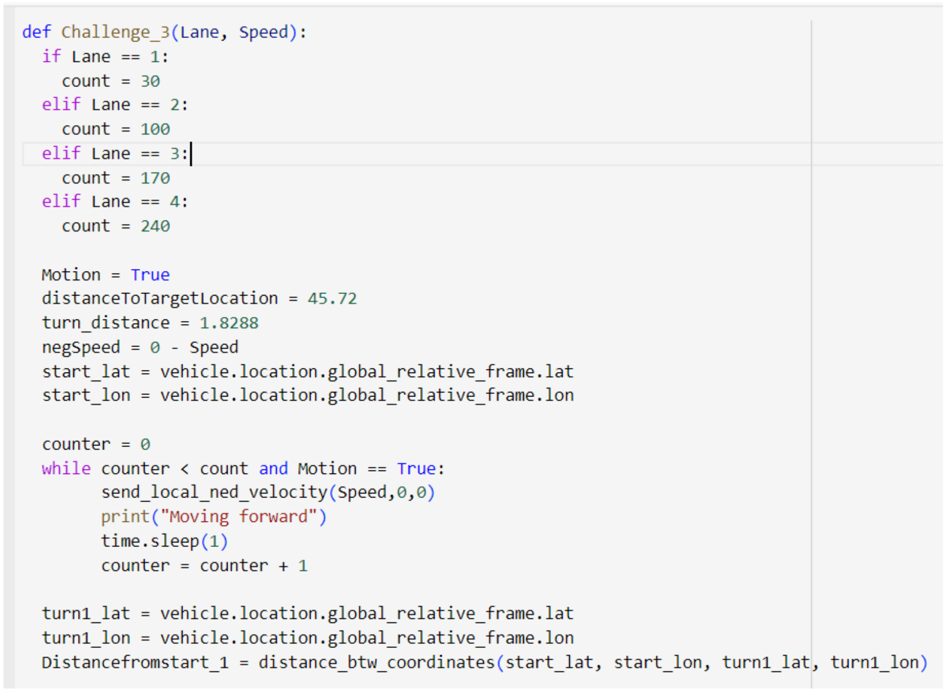
If the distance is equal to or greater than the length of the loans (50 yards), then the challenge is complete and we stop the UGV and set “Motion” and “Water_bool” to false to end the loop and deactivate the water sensing module.
**Note:** If the water sensing thread detects 20 ml of water before the challenge ends, it sets “Motion” to false and stops the UGV.

## Challenge 3:

The UGVs take a right and left turn in a single swim lane and at a speed determined by the event judge. The UAV autonomously navigates through the defined space and drops water blasts on all enemy UGVs found within a 7 minute window, then lands in the designated landing zone.

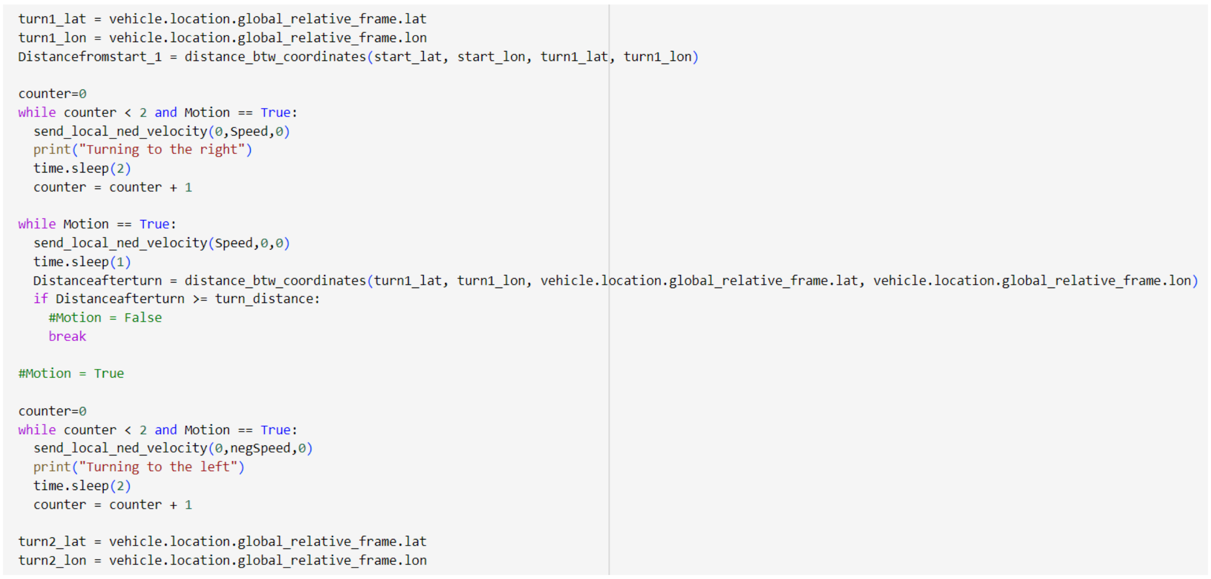


For challenge 3, using the same commands for motion as in challenge 2, we command the UGV to move forward for a certain amount of time. We use an if statement to set a duration for the first part of the challenge depending on the UGV’s lane.



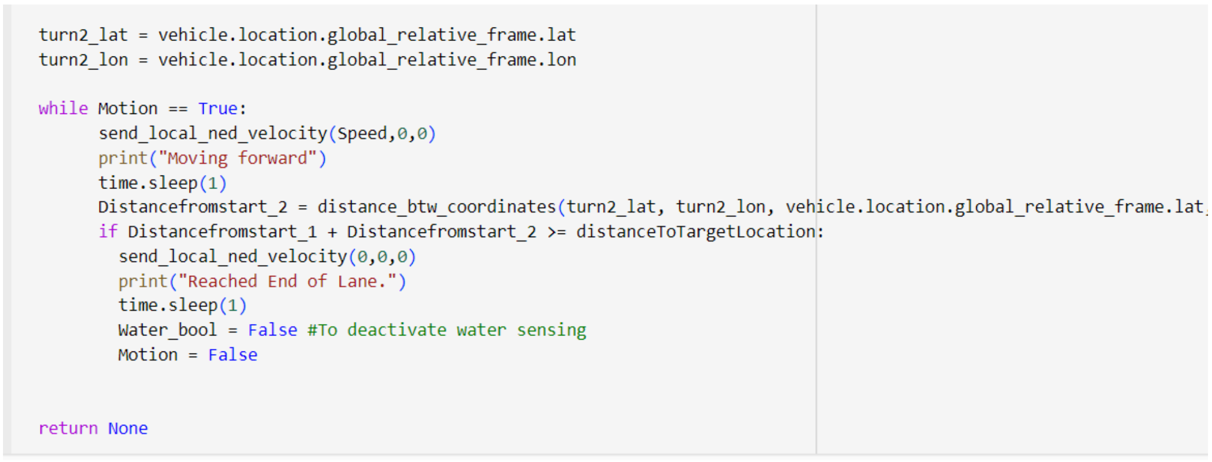
```python
def Challenge_3(Lane, Speed):
  if Lane == 1:
    count = 30
  elif Lane == 2:
    count = 100
  elif Lane == 3:
    count = 170
  elif Lane == 4:
    count = 240

  Motion = True
  distanceToTargetLocation = 45.72
  turn_distance = 1.8288
  negSpeed = 0 - Speed
  start_lat = vehicle.location.global_relative_frame.lat
  start_lon = vehicle.location.global_relative_frame.lon

  counter = 0
  while counter < count and Motion == True:
        send_local_ned_velocity(Speed,0,0)
        print("Moving forward")
        time.sleep(1)
        counter = counter + 1

  turn1_lat = vehicle.location.global_relative_frame.lat
  turn1_lon = vehicle.location.global_relative_frame.lon
  Distancefromstart_1 = distance_btw_coordinates(start_lat, start_lon, turn1_lat, turn1_lon)
```

We calculate how much distance the UGV covered in the first part of the challenge using the “distance_btw_coordinates” function and store it in the variable “Distancefromstart_1”. We then send commands to turn the UGV right, move forward for 2 yards then turn left.



```python
turn1_lat = vehicle.location.global_relative_frame.lat
turn1_lon = vehicle.location.global_relative_frame.lon
Distancefromstart_1 = distance_btw_coordinates(start_lat, start_lon, turn1_lat, turn1_lon)

counter=0
while counter < 2 and Motion == True:
  send_local_ned_velocity(0,Speed,0)
  print("Turning to the right")
  time.sleep(2)
  counter = counter + 1

while Motion == True:
  send_local_ned_velocity(Speed,0,0)
  time.sleep(1)
  Distanceafterturn = distance_btw_coordinates(turn1_lat, turn1_lon, vehicle.location.global_relative_frame.lat, vehicle.location.global_relative_frame.lon)
  if Distanceafterturn >= turn_distance:
    #Motion = False
    break

#Motion = True

counter=0
while counter < 2 and Motion == True:
  send_local_ned_velocity(0,negSpeed,0)
  print("Turning to the left")
  time.sleep(2)
  counter = counter + 1

turn2_lat = vehicle.location.global_relative_frame.lat
turn2_lon = vehicle.location.global_relative_frame.lon
```

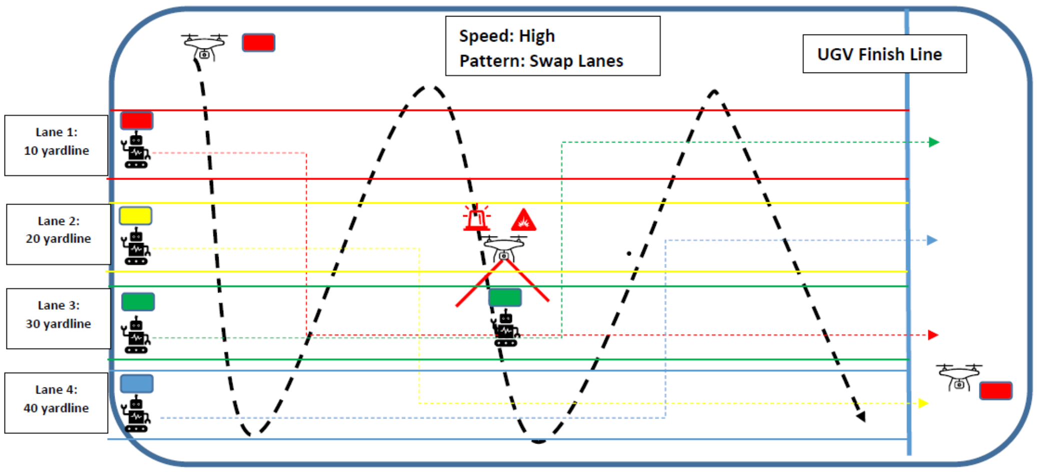
We store the coordinates of the UGV after it turns left, then send commands every second to move forward until the distance between the turn coordinates and current coordinates added to “Distancefromstart_1” is equal to or greater than the length of the lane (50 yards). The challenge is complete and we stop the UGV and set “Motion” and “Water_bool” to false to end the loop and deactivate the water sensing module.



```
turn2_lat = vehicle.location.global_relative_frame.lat
turn2_lon = vehicle.location.global_relative_frame.lon

while Motion == True:
      send_local_ned_velocity(Speed,0,0)
      print("Moving forward")
      time.sleep(1)
      Distancefromstart_2 = distance_btw_coordinates(turn2_lat, turn2_lon, vehicle.location.global_relative_frame.lat,
      if Distancefromstart_1 + Distancefromstart_2 >= distanceToTargetLocation:
        send_local_ned_velocity(0,0,0)
        print("Reached End of Lane.")
        time.sleep(1)
        Water_bool = False #To deactivate water sensing
        Motion = False


return None
```

## Challenge 4:

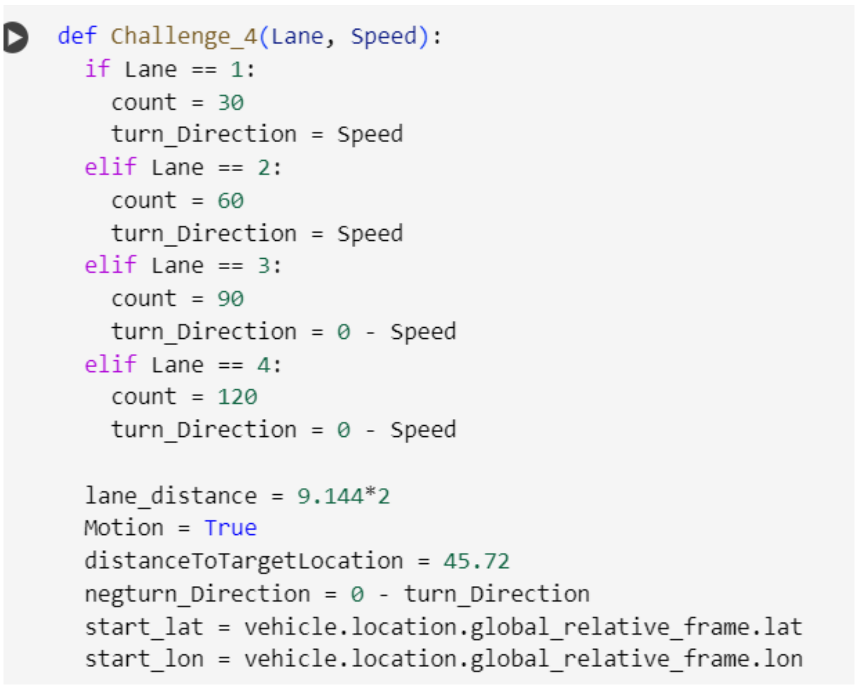
The UGVs swap swim lanes with other UGVs determined by the event judge. The UAV autonomously navigates through the defined space and drops water blasts on all enemy UGVs found within a 5 minute window, then lands in the designated landing zone.



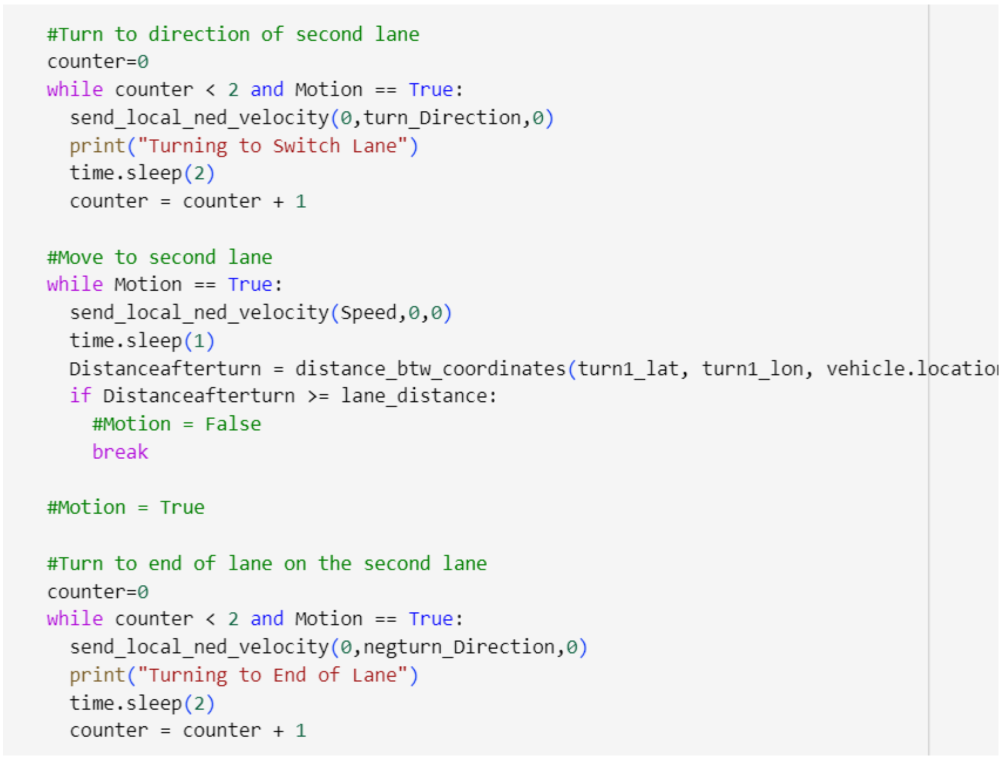
Our approach to challenge 4 is very similar to that of challenge 3, the major difference being that after the first turn, the UGV switches lanes “travels 20 yards” instead of traveling 2 yards before turning again. Another noteworthy difference between challenge 4 and 3 is that for challenge 3 the UGV always turns right then turns left independent of its lane but for challenge 4, the UGV turns right then left if it's on lane 1 or 2 and turns left then right if it's on lane 3 or 4.

We go about this difference by adding a new variable “turn_direction” to the if statement. We store “Speed” in the variable for lanes 1 & 2 and store the negative value of “Speed” in the variables for lanes 3 & 4.

Recall that for the command “send_local_ned_velocity(0,turn_speed,0)”, the UGV turns right when “turn_speed” is positive but turns left when it is negative.



```
def Challenge_4(Lane, Speed):
  if Lane == 1:
    count = 30
    turn_Direction = Speed
  elif Lane == 2:
    count = 60
    turn_Direction = Speed
  elif Lane == 3:
    count = 90
    turn_Direction = 0 - Speed
  elif Lane == 4:
    count = 120
    turn_Direction = 0 - Speed

  lane_distance = 9.144*2
  Motion = True
  distanceToTargetLocation = 45.72
  negturn_Direction = 0 - turn_Direction
  start_lat = vehicle.location.global_relative_frame.lat
  start_lon = vehicle.location.global_relative_frame.lon
```

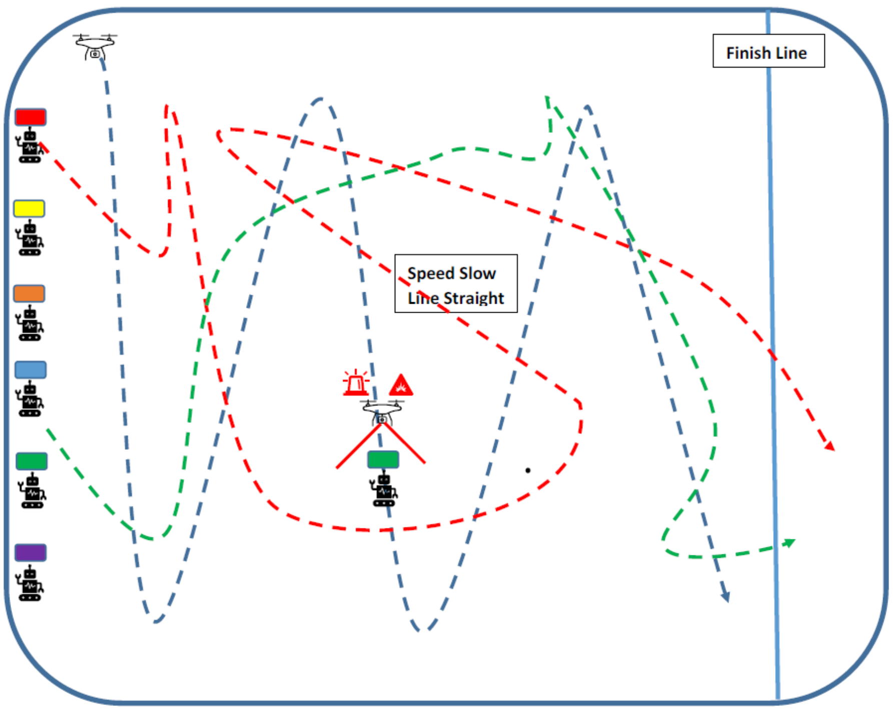


```
#Turn to direction of second lane
counter=0
while counter < 2 and Motion == True:
  send_local_ned_velocity(0,turn_Direction,0)
  print("Turning to Switch Lane")
  time.sleep(2)
  counter = counter + 1

#Move to second lane
while Motion == True:
  send_local_ned_velocity(Speed,0,0)
  time.sleep(1)
  Distanceafterturn = distance_btw_coordinates(turn1_lat, turn1_lon, vehicle.locatio
  if Distanceafterturn >= lane_distance:
    #Motion = False
    break

#Motion = True

#Turn to end of lane on the second lane
counter=0
while counter < 2 and Motion == True:
  send_local_ned_velocity(0,negturn_Direction,0)
  print("Turning to End of Lane")
  time.sleep(2)
  counter = counter + 1
```

Aside from these minor changes, challenge 4 uses the same logic as challenge 3.

**Bonus:**

The UGVs follow a random path in the defined space while avoiding other UGVs. The UAV autonomously navigates through the defined space and drops water blasts on all enemy UGVs found within a 10 minute window, then lands in the designated landing zone.



To complete the bonus challenge, we considered an approach where the UGV periodically switches lanes so it better avoids water from the UAVs.

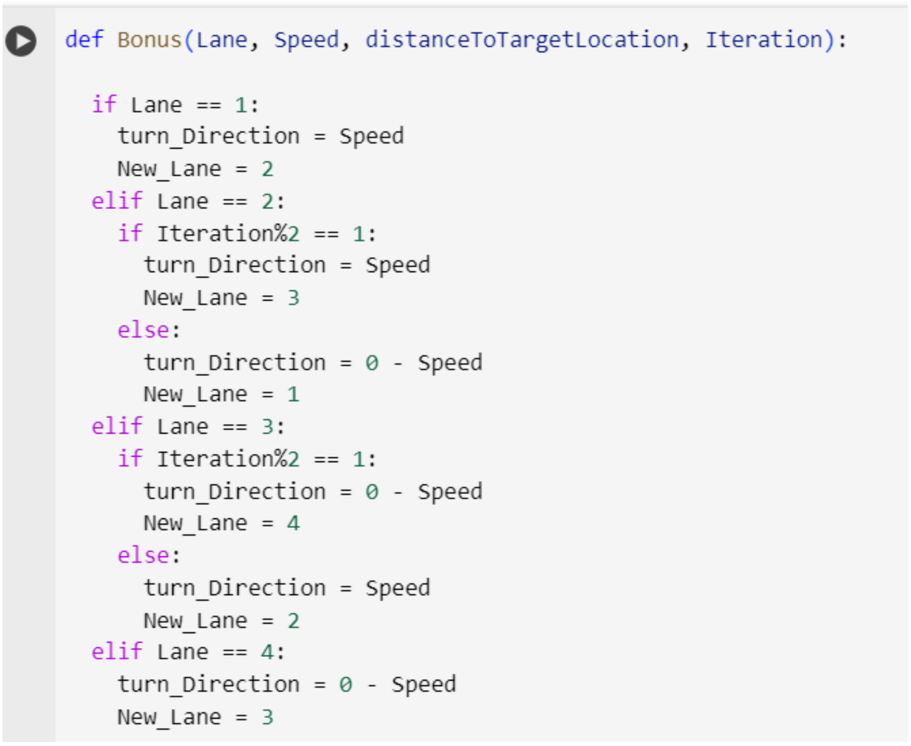
Our approach involves using a recursive function that makes the UGV move forward for some time (10 seconds) on its current lane then switch to lane next to it and move forward again for some time (30 seconds) before the function calls itself again. The function repeats itself until the UGV reaches the end of one of the lanes.

Important problems to note for this approach:

- Unlike for lanes 1 & 4 that have only one option for lanes to switch to, lanes 2 & 3 have 2 options, we had to develop a system for the UGV to decide what lane to switch to.
- Other UGVs would also be simultaneously attempting the challenge, which adds a risk of collusion. We had to develop a way to detect and avoid obstacles, in this case other UGVs.

- We need a way to determine if the UGV has reached the end of any lane and end the recursive function.

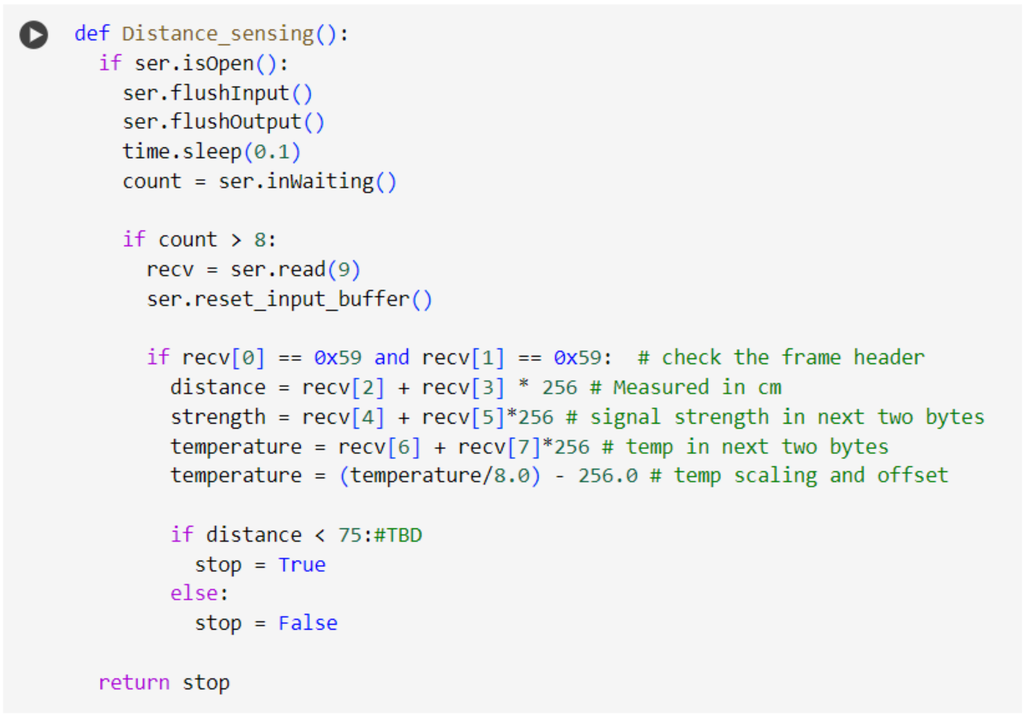
➔ We solved the first problem by adding a new parameter “iteration” to the “Bonus()” function that keeps track of how many times the function has been called.



```
def Bonus(Lane, Speed, distanceToTargetLocation, Iteration):

  if Lane == 1:
    turn_Direction = Speed
    New_Lane = 2
  elif Lane == 2:
    if Iteration%2 == 1:
      turn_Direction = Speed
      New_Lane = 3
    else:
      turn_Direction = 0 - Speed
      New_Lane = 1
  elif Lane == 3:
    if Iteration%2 == 1:
      turn_Direction = 0 - Speed
      New_Lane = 4
    else:
      turn_Direction = Speed
      New_Lane = 2
  elif Lane == 4:
    turn_Direction = 0 - Speed
    New_Lane = 3
```

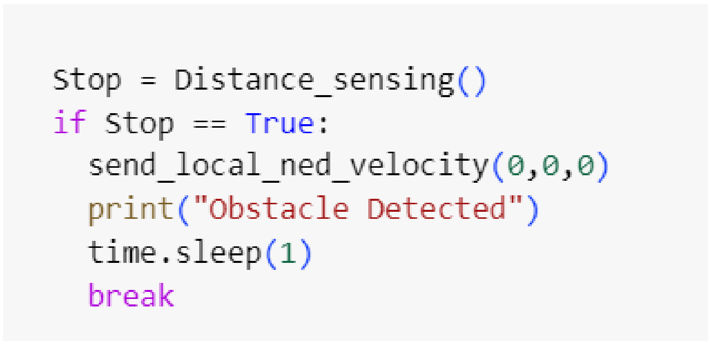
We then use an if statement, the “iteration” parameter, and the UGVs current lane to determine what lane the UGV will switch. When the UGVs current lane is 1 or 4 the only possible lanes the UGV can switch to are 2 and 3 respectively. When the UGVs current lane is 2, we determine the new lane to switch to depending on whether the value in “iteration” is even or odd, as shown in the picture above.

➔ We detect obstacles on the UGVs path by using the function “Distance_sensing()”. The function uses the LiDAR sensor to check if there is an obstacle within a certain range of the UGV (75 cm in front of the UGV). The function returns a boolean value, returning true if it detects an obstacle, and false otherwise.



```
def Distance_sensing():
  if ser.isOpen():
    ser.flushInput()
    ser.flushOutput()
    time.sleep(0.1)
    count = ser.inWaiting()

    if count > 8:
      recv = ser.read(9)
      ser.reset_input_buffer()

      if recv[0] == 0x59 and recv[1] == 0x59:  # check the frame header
        distance = recv[2] + recv[3] * 256 # Measured in cm
        strength = recv[4] + recv[5]*256 # signal strength in next two bytes
        temperature = recv[6] + recv[7]*256 # temp in next two bytes
        temperature = (temperature/8.0) - 256.0 # temp scaling and offset

        if distance < 75:#TBD
          stop = True
        else:
          stop = False

  return stop
```

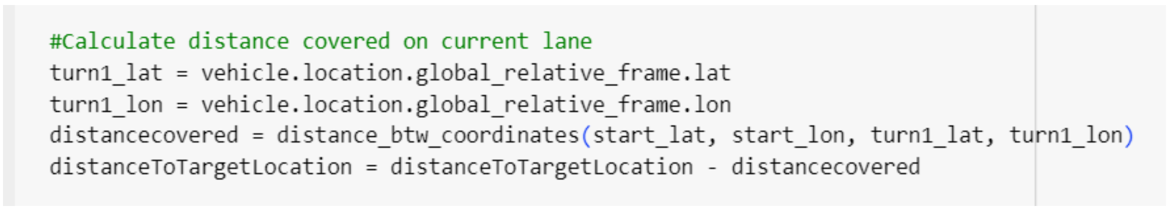
With regards to obstacle detection, the “Bonus()” function is functionally split into 3 parts. First the loop handling navigation on the first lane, then the loop handling the UGVs switch to a new lane, and finally the loop handling the navigation on the new lane. The UGV needs to detect any obstacle in its path in all the aforementioned parts of the function. We do this by calling the “Distance_sensing()” function in all 3 loops and using an if statement to break the loop when an obstacle is detected.



```
Stop = Distance_sensing()
if Stop == True:
  send_local_ned_velocity(0,0,0)
  print("Obstacle Detected")
  time.sleep(1)
  break
```

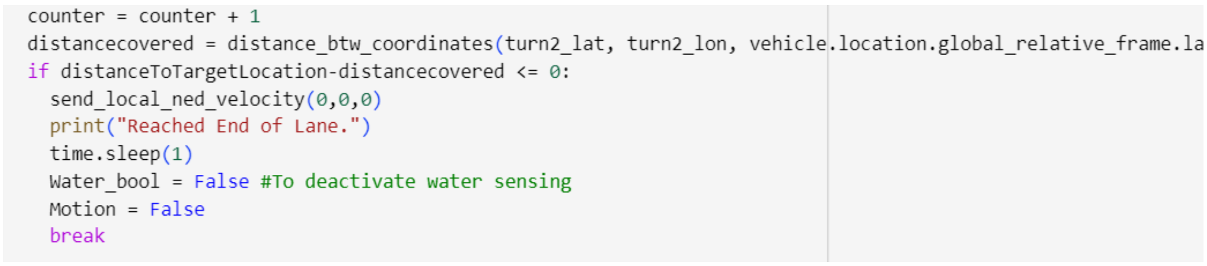
Using this approach, if the UGV detects an obstacle while it's moving forward on its first lane, it stops, switches lanes and continues on the path determined by the function. If the UGV detects an obstacle while it is switching lanes, it stops and turns to the end of whatever lane it is on and continues on the path determined by the function. If the UGV detects an obstacle while it's moving forward on its new lane it stops and recursively calls the “Bonus()” function.

➔ We determine if the bonus challenge has been completed by adding a new parameter “distanceToTargetLocation” to the “Bonus()” function. The first time the function is called, we set the variable to the length of a lane (50 yards). Everytime the UGV moves closer to the end of a lane, we calculate the distance the UGV travels using the “distance_btw_coordinates()” function, and subtract that distance from the “distanceToTargetLocation” variable.



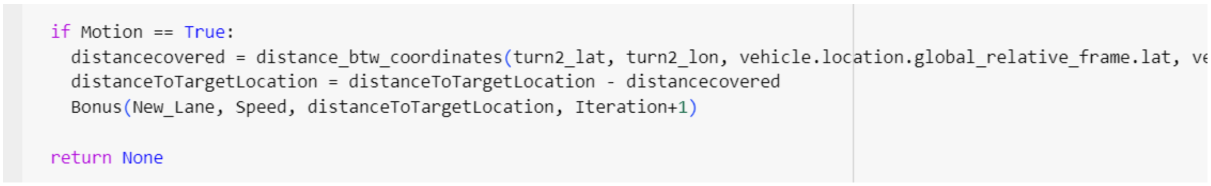
```
#Calculate distance covered on current lane
turn1_lat = vehicle.location.global_relative_frame.lat
turn1_lon = vehicle.location.global_relative_frame.lon
distancecovered = distance_btw_coordinates(start_lat, start_lon, turn1_lat, turn1_lon)
distanceToTargetLocation = distanceToTargetLocation - distancecovered
```

We check if “distanceToTargetLocation” is equal to or less than zero every time the UGV moves closer to the end of the lane. If it is equal to less than zero, i.e if the UGV has crossed the finish lane, we deactivate the water sensing function, set the boolean variable “Motion” to false and break the loop we are in.



```
counter = counter + 1
distancecovered = distance_btw_coordinates(turn2_lat, turn2_lon, vehicle.location.global_relative_frame.la
if distanceToTargetLocation-distancecovered <= 0:
  send_local_ned_velocity(0,0,0)
  print("Reached End of Lane.")
  time.sleep(1)
  Water_bool = False #To deactivate water sensing
  Motion = False
  break
```

We then recursively call the “Bonus()” function if the challenge has not been completed, i.e if “Motion” is true. We call the function with the new lane, the speed,  the new value of “distanceToTargetLocation”, and “iteration” incremented by one, as its new parameters.

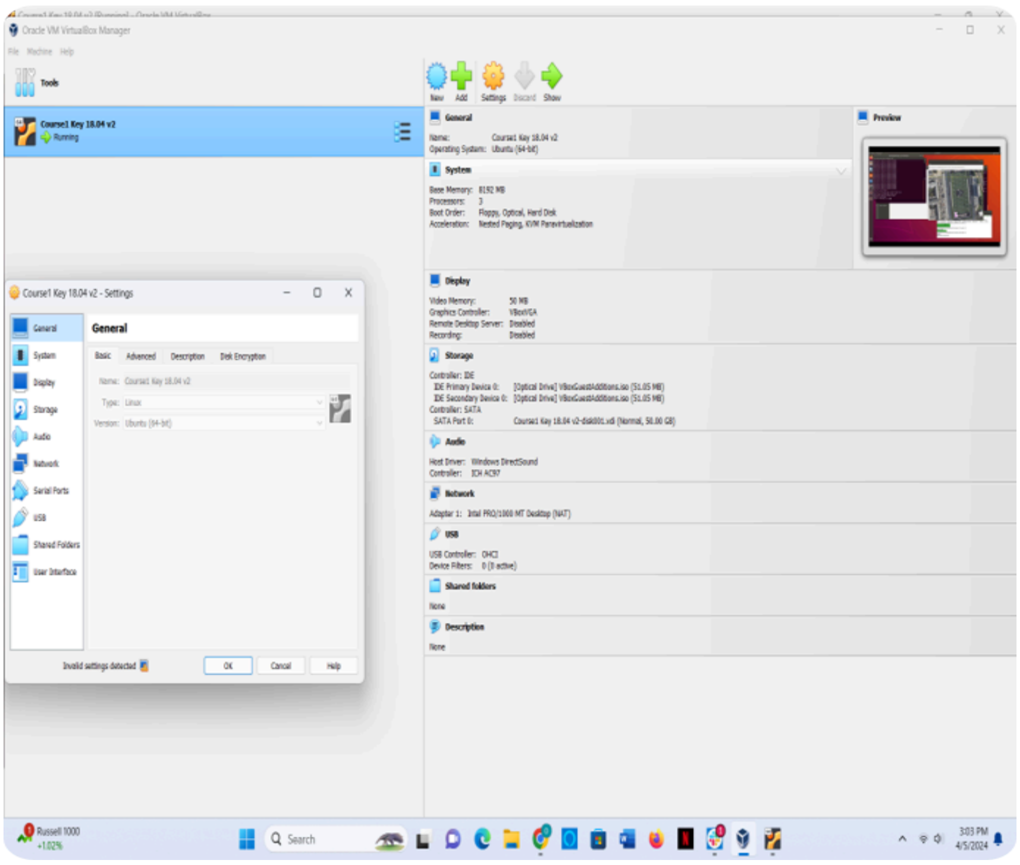


```
if Motion == True:
  distancecovered = distance_btw_coordinates(turn2_lat, turn2_lon, vehicle.location.global_relative_frame.lat, ve
  distanceToTargetLocation = distanceToTargetLocation - distancecovered
  Bonus(New_Lane, Speed, distanceToTargetLocation, Iteration+1)

return None
```

## Sprint 3:

In the final sprint, we focused on creating a virtual environment where we could test and improve our code and algorithms.

Using VirtualBox, we setup the following virtual environment:



Course1 Key 18.04 v2
Running

Course1 Key 18.04 v2 - Settings

General

Basic | Advanced | Description | Disk Encryption

Name: Course1 Key 18.04 v2
Type: Linux
Version: Ubuntu (64-bit)

Invalid settings detected

General
Name: Course1 Key 18.04 v2
Operating System: Ubuntu (64-bit)

System
Base Memory: 8192 MB
Processors: 3
Boot Order: Floppy, Optical, Hard Disk
Acceleration: Nested Paging, KVM Paravirtualization

Display
Video Memory: 50 MB
Graphics Controller: VBoxVGA
Remote Desktop Server: Disabled
Recording: Disabled

Storage
Controller: IDE
IDE Primary Device 0: [Optical Drive] VBoxGuestAdditions.iso (51.05 MB)
IDE Secondary Device 0: [Optical Drive] VBoxGuestAdditions.iso (51.05 MB)
Controller: SATA
SATA Port 0: Course1 Key 18.04 v2-disk001.vdi (Normal, 50.00 GB)

Audio
Host Driver: Windows DirectSound
Controller: ICH AC97

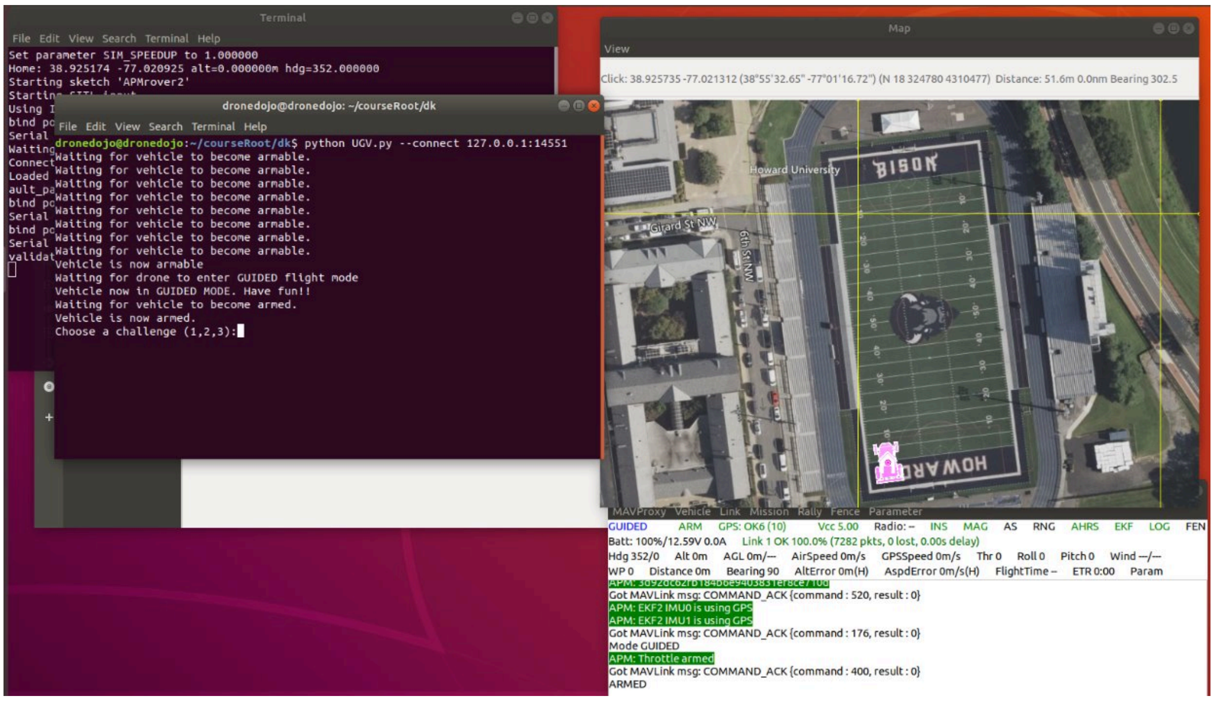
Network
Adapter 1: Intel PRO/1000 MT Desktop (NAT)

USB
USB Controller: OHCI
Device Filters: 0 (0 active)

Shared folders
None

Description
None

On the virtual environment, we were able to successfully simulate the UGV at any given location, and control the UGV using commands from the scripts.


Terminal
File Edit View Search Terminal Help
Map
View
Click: 38.925735 -77.021312 (38°55'32.65" -77°01'16.72") (N 18 324780 4310477) Distance: 51.6m 0.0nm Bearing 302.5
Howard University
Girard St NW
6th St NW
BISON
HOWARD
MAVProxy Vehicle Link Mission Rally Fence Parameter

```
Set parameter SIM_SPEEDUP to 1.000000
Home: 38.925174 -77.020925 alt=0.000000m hdg=352.000000
Starting sketch 'APMrover2'
```

```
dronedojo@dronedojo:~/courseRoot/dk$ python UGV.py --connect 127.0.0.1:14551
Waiting for vehicle to become armable.
Waiting for vehicle to become armable.
Waiting for vehicle to become armable.
Waiting for vehicle to become armable.
Waiting for vehicle to become armable.
Waiting for vehicle to become armable.
Waiting for vehicle to become armable.
Waiting for vehicle to become armable.
Vehicle is now armable
Waiting for drone to enter GUIDED flight mode
Vehicle now in GUIDED MODE. Have fun!!
Waiting for vehicle to become armed.
Vehicle is now armed.
Choose a challenge (1,2,3):
```

```
GUIDED      ARM    GPS: OK6 (10)      Vcc 5.00    Radio: --    INS    MAG    AS    RNG    AHRS    EKF    LOG    FEN
Batt: 100%/12.59V 0.0A    Link 1 OK 100.0% (7282 pkts, 0 lost, 0.00s delay)
Hdg 352/0    Alt 0m    AGL 0m/---    AirSpeed 0m/s    GPSSpeed 0m/s    Thr 0    Roll 0    Pitch 0    Wind ---/---
WP 0    Distance 0m    Bearing 90    AltError 0m(H)    AspdError 0m/s(H)    FlightTime --    ETR 0:00    Param
Got MAVLink msg: COMMAND_ACK {command : 520, result : 0}
APM: EKF2 IMU0 is using GPS
APM: EKF2 IMU1 is using GPS
Got MAVLink msg: COMMAND_ACK {command : 176, result : 0}
Mode GUIDED
APM: Throttle armed
Got MAVLink msg: COMMAND_ACK {command : 400, result : 0}
ARMED
```

## Conclusion

This project began with the objective of creating an Unmanned Ground Vehicle (UGV) to participate in the annual Autonomous Vehicle RTX competition at Virginia Tech, collaborating with an Unmanned Aerial Vehicle (UAV) to complete a series of intricate challenges. Unfortunately we were unable to complete our aim of building a physical UGV, due to administrative complications with Howard University allocating funds to buy the equipment we needed. Despite the absence of physical hardware components, we were able to improvise by simulating the UGV completing the challenges from the autonomous vehicle challenge , resulting in valuable experiences, and insights into the potential capabilities and challenges of building a fully autonomous vehicle.

## References

https://dojofordrones.com/pihawk-rover-kit/