Virtuoso
A New Architecture for RobotX 2022

Sean T. Fish, Manuel Roglan, Douglas S. Chin, Jeffrey T. Pattison, and Jorge L. Ortiz Solano

***Abstract*—This paper describes the design and development of an Autonomous Maritime System (AMS) for the 2022 RobotX Challenge by the Marine Robotics Group at the Georgia Institute of Technology. The AMS consists of a large autonomous surface vehicle (ASV) and a companion Unmanned Aerial Vehicle (UAV). The design process of the AMS was centered around accomplishing the tasks of the RobotX Challenge, with the additional consideration of creating a stable platform for experimentation. To achieve these goals, new technologies were incorporated into the AMS such as Robot Operating System 2 (ROS2) and micro-ROS, and a new propulsion system featuring greater modularity was integrated into the WAM-V.**

I. Introduction

The Marine Robotics Group (MRG) at Georgia Tech has competed in the RobotX Challenge in previous years. This year's entry into the challenge attempts to remedy issues encountered by previous years' teams, such as issues with propulsion in saltwater or difficulties in vehicle control. Additionally, in the four years between competitions, many new innovations have been developed, such as Robot Operating System 2 (ROS2) and micro-ROS, and the team has worked to incorporate these new technologies into the system. With these considerations, MRG has created a new ROS2 main software stack, *Virtuoso*, to replace the previous *Adept* software stack developed in ROS1, integrated a new holonomic propulsion system, and constructed a new Unmanned Aerial Vehicle (UAV).

II. DESIGN STRATEGY

The design strategy for this year's competition encompasses a long timespan, as MRG has been planning to compete in the next RobotX Challenge since the last occurrence in 2018. Over the duration of a typical undergraduate education, many ideas have been explored, and many members have come and gone. As a result, one of the greatest necessities for the organization was to create software which members felt a sense of ownership over, and while this resulted in redevelopment of some existing capabilities, it provided an opportunity for incorporating new developments and making improvements over the previous system. Additionally, in the interest of saving time in development, Commercial Off-The-Shelf (COTS) hardware and software were incorporated into the design of the system.

In 2018, the team had switched from an in-house software stack, ARCS, to ROS1 [3]. One of the reasons for this change was to allow for integration of externally developed libraries. With much of the robotics community switching to ROS2, and many of the developers of the ROS1 stack having graduated during the pandemic, the team made a similar shift from ROS1 to ROS2 in the past year, first testing the ROS2 stack in the Virtual RobotX (VRX) Challenge [2]. The development of the new ROS2 stack is discussed in greater detail in Section II. Some of the benefits of utilizing ROS2 include more robust support of multi-robot systems and a more robust communication layer. As early adopters of ROS2 with legacy equipment, there have been several challenges with software compatibility as well, whether that is in the VRX simulation environment or with sensor systems. However, with ROS1 support ending soon, this path will ultimately be more future-proof and enable easier development going forward.

Fig 1. The WAM-V during testing at Sweetwater Creek State Park

A. WAM-V Design

The Wave Adaptive Modular Vessel (WAM-V) that serves as the main vehicle for the competition has also been reconfigured since its appearance in 2018. The majority of efforts was concentrated on the propulsion system, as it is the most critical component across all tasks. Previous members reported encountering issues with the Torqeedo Cruise 2.0 electric outboard motors when running in saltwater. Additionally, the team wanted to switch to a holonomic motor configuration to improve control of the vehicle. Such configurations had proven to be used to great success in past competitions [4]. As the previous motors were of questionable reliability and were too few in number for the new configuration, the team opted to redevelop the propulsion system from scratch. The new system was additionally constrained by supply-chain issues. Initially, trolling motors with integrated throttle systems were chosen, but constant lack

of supply ended up holding back development by a semester. More affordable 8-speed trolling motors were chosen instead, requiring the development of a separate throttling system, culminating in a new motor controller box with a micro-ROS interface. While this deviated from the original plan, the result was a propulsion system that allowed for easily replacing the parts with the heaviest wear – the motors themselves.

The team's main campus is not located particularly close to any bodies of water, and as a result, much of the software was developed in simulation, and subsystems were developed somewhat independently. The system is composed of many individual components which are designed rather independently, with sporadic integration testing. This design strategy resulted in high coupling within individual component systems, and as a result reduced interdependency and complexity arising from this. On the other hand, this has resulted in the need to tune as systems are integrated together. For example, the simulation motors do not behave similarly to the real motors. With reduced interdependency, there has been an increase in robustness, as modules can be swapped in and out, totally replaced, without completely disabling other systems.

B. UAV Design

A few of the tasks in the 2022 RobotX competition allow or require an Unmanned Aerial Vehicle (UAV) for completion. These tasks include Follow the Path, Wildlife Encounter, UAV Replenishment, and Search and Report. Due to time and resource constraints, the only tasks planned to involve the UAV for this team were the Search and Report and UAV Replenishment tasks. This was a strategic decision to ensure adequate performance for a few tasks rather than risk inadequate performance for a larger number of tasks, and the two tasks selected for development were the only two tasks that required the assistance of the UAV. The strategy and planned solutions for each of the UAV tasks can be seen in following subsections.

Fig 2. The UAV in flight during testing

1) Search and Report

The Search and Report task is the only task that involves only the UAV. The task requires a UAV to takeoff from a platform and search a field for two letter markers, report the location of the markers, and then return and land on the launch platform identified by concentric circles. The area to search is marked by four markers, and the UAV is not allowed to go outside the markers during the search.

The plan for completing the Search and Report task was for the UAV to discretize the search area in to a list of GPS waypoints, takeoff, travel to each waypoint in the search area, and at each waypoint use the onboard camera to scan the ground below for a letter using a text detection and recognition algorithm. Once all the waypoints had been reached or both markers were found, the UAV would then return to the launch location where it would attempt to land on the launch platform using the concentric circles on the platform as a guide.

Using the onboard GPS, the UAV could record the initial position and use the relative position of the area boundaries with respect to the launching platform to approximate the GPS coordinates of the cones. With the GPS location of the boundaries known, the area was then discretized into a grid of several GPS points for the UAV to travel. This was a simple solution to stay within the boundary. Solutions that are more sophisticated could use computer vision to detect when the UAV approaches the boundary by recognizing the orange cones, but the benefits of this method were deemed too small for the complexity it would introduce to the system as an additional computer vision algorithm would strain the onboard computer.

Once the UAV has the list of GPS waypoints making up the search area, it can then takeoff and visit each location to check for the markers to find. To recognize the letters within the search area, the UAV would not only need to detect letters but also interpret the letters. For this, OpenCV and Machine Learning methods were used. OpenCV is a library used for image processing and is common in many computer vision applications. By combining the OpenCV image processing with the Python package Pytesseract, used for detecting letters, the UAV was able to detect any letters observed by the onboard camera. If any part of the video feed was detected to contain letters, a portion of the image captured in the video feed containing the letters was analyzed using the EAST deep learning model. The Pytesseract package does have the capability to determine text, but it was shown to be less reliable than the EAST deep learning model, so both were used. Implementing a deep learning model for text recognition added complexity and increased strain on the companion computer, but this was an allowable tradeoff for the increase in accuracy of detecting the letters in the search area.

Using Pytesseract, a bounding box around recognized text can be created, and the center of the bounding box can be calculated. Once the UAV had located a letter within the video feed, the next step was to center the UAV over the letter to record the location. By maneuvering the UAV so the letter was centered in the video feed, the UAV was able to get a better estimate on the GPS location of the letter. Because there is some uncertainty in the GPS location, the GPS location was recorded over a series of 3 seconds and the average location was recorded. This average location was then used as the location of

the letter.

Once the UAV had travelled to each waypoint in the search area or found both markers, the UAV would then travel to the launch location and land on the launch platform. The simplest solution would be for the UAV to rely solely on GPS location and return to the GPS location of launch. However, this method relies on GPS modules with a high level of accuracy, which increase the cost of the UAV. The GPS used on the UAV is accurate within a few meters, which is unacceptable for landing on the platform with a size of approximately 1.5m x 1.5m. A very complex solution could rely on computer vision and machine learning methods to identify the shapes on the landing platform. The decided approach relied on computer vision, but did not employ machine learning. Using the video feed during landing, OpenCV processing was used to identify shapes in the feed of the camera. By identifying the location of the center of the concentric circles in the video feed, the UAV attempted to maneuver during landing until the center of the circles was centered in the video feed. This allowed the UAV to land on the launch platform.

The plan previously described for completing the Search and Report task shapes the design of the UAV. With the goal of autonomous letter detection, a companion computer and camera were required on the UAV to process images. However, by minimizing the computation power required for image processing by forgoing a strong emphasis on machine learning methods, a smaller and cheaper companion computer can be used. Similarly, with the assistance of computer vision in the landing phase, a precise GPS is not required, allowing a cheaper model to be used instead. Both of these decisions allow for a cheaper vehicle without significantly sacrificing the ability to complete the task.

2) UAV Replenishment

The UAV Replenishment task requires a UAV to launch from the WAM-V, locate a platform containing colored tins, collect a colored tin, and then deposit the tin on another platform before returning to the WAM-V. The plan for completing this task was for the WAM-V to maneuver alongside one of the UAV landing platforms. From here, it would remain stationary until the UAV returned from the mission. The UAV would then launch and use the shape detecting computer vision algorithms to locate and land on the landing platform, where it would collect a tin using a retractable magnet. The UAV would then takeoff again and begin a search pattern looking for the other platform, where it would then land and deposit the tin by retracting the retractable magnet. Upon depositing the tin, the UAV then takes off again and returns to the location of its initial launch and lands on the WAM-V. Landing on the WAM-V is enabled through ArUco marker localization on the WAM-V landing pad using a method similar to the UAV landing on the landing platforms of the various tasks.

Based on the description of the task, it has significant overlap with the capabilities developed for the Search and Report task. Therefore, the ability to search an area, detect and recognize shapes and the ability to land on the launch platforms were taken directly from the solutions for the Search and Report task. The added capabilities that needed development were to add a communication link between the UAV and the WAM-V and to add a way to grab the tins. A communication link from the UAV to the WAM-V was established using ROS 2. When the WAM-V was in the correct position to begin the UAV Replenishment task, it would communicate this with the UAV using ROS 2. The UAV would then proceed to complete the task by taking off, searching the area for the landing platform containing the tins, landing on platform, and then taking off to find the other platform before returning to the WAM-V.

The method for obtaining the tin utilized a simple magnet that could attract the metal tin. Solutions that are more sophisticated would allow the use of computer vision to distinguish the tins by color, but due to resource constraints, the UAV was planned to just land on the platform and attract whichever tin was within range to a retractable magnet located on the landing gear. To remove the tin from the magnet, the magnet would retract and separate from the tin.

III. VEHICLE DESIGN

The vehicle design has largely revolved around two main ideas:

- Long-term robustness
- Integration of new technologies

These ideas can be seen in the design of the WAM-V's propulsion system, the new ROS2 software stack, and the overall design of the UAV.

A. WAM-V Mechanical Design

Mechanical design on the WAM-V was mainly focused on the integration of the new motor configuration, the UAV landing platform, and a sensor bar.

To implement a holonomic control system, the team opted to reconfigure the WAM-V with four total thrusters, two fore and two aft, configured with thrust vectors which form a diamond. The system was designed for typical transom-mount trolling motors with mounting clamps with minimal additional customization.

The fore motors required new motor mounts. These mounts are affixed the sides of the WAM-V pontoon skis. The initial mounts were constructed from single-bar 8020 aluminum extrusion and featured 3D-printed plastic mount points. Following initial testing at the lake, significant twisting was noticed in the motor mounts. To resolve this, the mounts were reinforced, with the mounting bars consisting of a 2-wide 8020 bar and a 1-wide 8020 bar, forming a stronger L-shaped bar. This proved to be much stronger and held up to higher motor thrusts at subsequent lake tests.

The aft motors did not require new motor mounts and were attached to the existing mount points on the stability pods where the electric outboards were located previously.

The competition also required the development of a raised UAV landing platform. Due to the main WAM-V platform holding the main electronics, the raised platform needed to sit above these components to provide a safe and flat location for the UAV to take-off and land from. The raised platform was

constructed from 8020 extrusion and features a white landing pad with 3 ArUco markers.

The system also features a sensor bar mounted to the front of the WAM-V. This sensor bar holds a camera, LIDAR, and other sensors. The purpose of this system is to enable testing of sensor systems while not onboard the WAM-V to allow for reuse of the same configuration on multiple platforms. Ideally, similar configurations can be used on vehicles for other competitions such as RoboBoat.

B. WAM-V Electrical Design

The electrical system on the WAM-V is a complete redesign from 2018. Changes were made to the motors, base power supply, and the motor controller. Sensor systems remained mostly unchanged from previous years.

Past outboard motors were reported to be unreliable in saltwater in recent years, so new saltwater trolling motors were chosen to replace them. These trolling motors operate at 12 volts as opposed to the 24 volts required by the previous outboard motors.

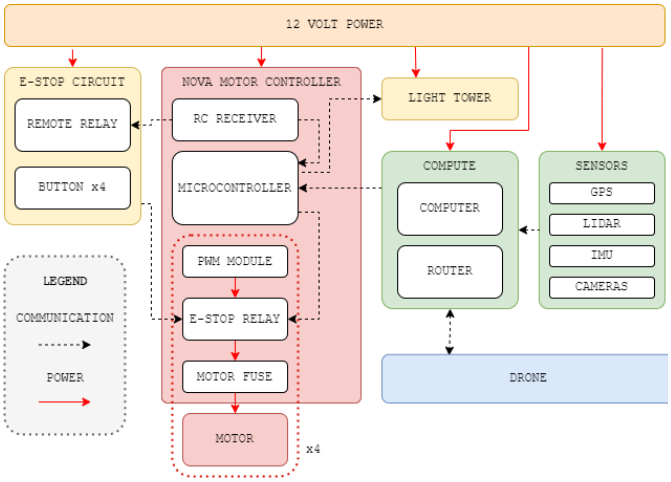

Fig 3. WAM-V Electrical System Block Diagram

Initially, the selected Minn Kota trolling motors were equipped with integrated digital speed control. However, due to supply chain issues, these motors were difficult to acquire and held up propulsion development for a semester. In retrospect, it would have been wiser to switch to an alternative after the first delays in acquiring the motors. The motors that were ultimately chosen were Newport Vessels NV-Series 8-speed 55lb trolling motors. These motors lack digital speed control, resulting in the need to implement this functionality externally. Motors of this variety tend to be more affordable and easily acquired. The switch to these motors proved useful, as the propulsion system is one of the systems subjected to the most wear. This new design allows for the propulsion system to undergo routine maintenance with less costly replacements.

The main power system is now based on a 12-volt supply, instead of 24-volt supply due to the switch from 24-volt outboard motors to 12-volt trolling motors. Additionally, 12-volt batteries tend to be more easily acquired than 24-volt batteries, especially of the Torqeedo brand. Utilizing DC/DC switching regulators, the 24-volt batteries the team already possesses were able to be reused.

In order to allow for the computer to communicate with the propulsion system, a new motor controller was necessary. This motor controller needed to be mechanically robust, support communication with ROS2, and allow for digital speed control of the motors.

Previous motor controllers built by MRG often consisted of Arduino Mega boards connected to various independent components by jumper cables. Jumper cables can often have intermittent contact and can easily come loose, costing precious time as the wiring must be reintegrated. If the cables are connected incorrectly, this can be dangerous, possibly causing unpredictable behavior. To resolve this issue, MRG manufactured a single-layer printed circuit board (PCB) using an in-house mill. In-house manufacturing allowed for working out several critical issues in the board through quick iterative design.

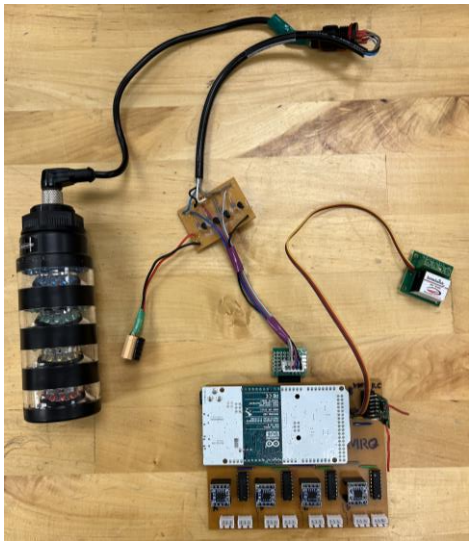

Fig 4. WAM-V Motor Controller Hardware

The motor controller board consists of multiple components, including a microcontroller and interfaces to various COTS electrical hardware components. The microcontroller is an Arduino Due, selected due to community support of micro-ROS for this board. Micro-ROS is the recommended software for enabling communication between microcontrollers and ROS2 and allows for swapping in the physical motor interfaces for the simulation interfaces without significant adjustment. Initially, a Raspberry Pi Pico was also under consideration, but since the Arduino platform was more familiar to the team, the Arduino was chosen for the first version of the motor controller.

The board also has chips providing an interface with Pulse Width Modulation (PWM) modules manufactured by Walfront for controlling individual motor speed. The modules were chosen due to reviews indicating successful use with trolling motors and support for the necessary power draw. The control interface for the modules included a 100K linear potentiometer for throttle and an ON-OFF-ON switch for direction control. The interfaces connected to the module through JST-XH connectors. To replace the interfaces, the board includes 100K

digital potentiometer boards and MUX chips. These replacements allow for the Arduino to control the motor speeds. However, the digital potentiometer tolerances are rather relaxed, and as a result, the throttle is limited by the chip with the lowest maximum resistance. Additionally, JST-XH crimps are challenging to complete properly, and as a result, some cables required rework.

The motor controller also features an OrangeRX R615X 6 channel R/C receiver. This receiver enables remote control of the motor controller, essentially allowing the WAM-V to be operated as a large R/C boat. The receiver is also connected to a BattleSwitch relay, which is incorporated into the Emergency Stop (E-Stop) system. The relay is failsafe, disabling the boat in the case of a lost R/C signal.

The light tower is also connected to the motor controller board, allowing for the motor controller to reflect the operational status of the vehicle. This includes kill status, R/C and autonomous control, and micro-ROS connection status.

The motor controller can be powered independently, allowing for the WAM-V to be operated even without a computer onboard. Powering the board properly is something that requires more consideration, as it needs to be protected from the propulsion system and cannot handle the 12-volt supply from the main power supply.

A revision of the board is in development with a Teensy 4.1 replacing the Arduino, due to full micro-ROS support and compatibility with the developed Arduino code. Future revisions will address issues in cable reliability and chip tolerances, and possibly integrate power regulation and protection circuitry. New boards will be professionally fabricated for greater longevity.

The WAM-V's autonomy system is completely independent from the motor control system, apart from a micro-USB for the motor controller. This design enables reuse of the autonomy system with minimal reconfiguration for other vehicles. Sensors include an IMU, a LIDAR, a GPS, and a camera. Most hardware has remained unchanged from 2018, except for the GPS. The GPS had proven unreliable in RoboBoat 2019, and was replaced for this reason. All other hardware was performing well. The IMU is a LORD MicroStrain 3DM-GX3-25. This IMU is convenient because it has a built-in attitude filter that estimates attitude based on magnetometer, accelerometer, and rate gyroscope data. The LIDAR is a Velodyne VLP-16, utilized in previous years. The GPS module is a ZED-F9P, which offers RTK functionality. The onboard camera is a Logitech C920 webcam. The main computer is a 2016 Intel NUC. These systems run off a 12-volt power supply, optionally provided by the same supply as the propulsion. Additionally, a small TP-Link Wi-Fi router was powered by the NUC, though this system will likely be replaced by the time of the competition to improve range. With ROS2, by simply sharing the same LAN network, it is possible to publish and subscribe to data between computers, making Wi-Fi an ideal option for remote connections.

C. WAM-V Software Design

The new ROS2 software stack builds off of many openly available packages, and was developed through the use of the Virtual RobotX [2] simulation environment. The software stack handles localization, control, and perception.

Localization utilized the GPS and IMU to estimate the attitude, position, and velocity of the vehicle. Initially, position and velocity were estimated with an Extended Kalman Filter fusing IMU acceleration, attitude, and GPS position. However, the IMU acceleration was found to be too noisy for this purpose. Instead, the inputs to the EKF were changed to be GPS position, GPS estimated velocity, and IMU attitude data. The package robot_localization was used for this [5]. The vehicle uses two main coordinate frames, wamv/base_link which is attached to the center of mass of the vehicle, and odom which has its origin at the vehicle stating position with axes corresponding to local magnetic ENU.

A controller package was created to follow paths from the navigation package. The controller consists of four main parts — an outer loop that commands a vehicle velocity, an inner loop that outputs vehicle force and torque commands, a last-meter PID controller that is intended for use once the vehicle is close to its target, and a control mixer that outputs vehicle thruster commands based on the force and torque commands. The target attitude is the direction such that the vehicle's +x axis is parallel to the desired velocity until the vehicle comes within a specified distance of its target, at which point the goal attitude specified by navigation package is set as the target. The outer loop takes the vehicle position and target path as its inputs. It calculates a desired velocity as a sum of two vectors. The first of these is a vector directly towards the line between the closest point on the path and the next point on the path. The second is the vector from the closest point on the path to the next point on the path. These vectors are scaled such that if the vehicle is further from the path, the vector towards the path is favored. In addition, the velocity magnitude is decreased if the vehicle is pointing away from the target velocity so that the vehicle can rotate itself to face in the direction of travel. This is preferred since the hydrodynamics of the vehicle favor motion in the surge direction. The main inner loop PID controller takes in the vehicle's current velocity, target velocity, and target attitude and outputs a goal X and Y force as well as torque. The last-meter PID controller takes in vehicle velocity, position, and attitude as well as target position and attitude and outputs X and Y force as well as torque commands. The last-meter PID controller is implemented so that station keeping can be tuned separately from the vehicle behavior between goals. The control mixer is what sends individual thruster commands based on target X and Y force as well as torque. The rear thruster commands are scaled down in magnitude to not induce excessive torque with Y force commands. In addition, the commands to all thrusters are scaled so that the maximum magnitude is 1.0, but the ratios of all thruster commands are kept.

PID tuning was done in simulation. When the vehicle was tested with these gains, it was found that the overall drag on the vehicle appeared greater than in simulation, so the gains were scaled up to increase the responsiveness of the vehicle.

The processing package of Virtuoso takes in raw camera and

raw LIDAR data. For the camera data, the images are downsampled to be used by the perception package. The LIDAR data is first passed through a ground filter from Autoware.auto that removes any LIDAR points on the water. Non-ground points are then passed to a self-filter that removes any LIDAR points hitting the WAM-V. Finally, the self-filtered points are passed to a shore filter. In simulation, the shore filter filters any LIDAR within a polygon created from GPS coordinates on the shore. For the competition and for practice runs, the shore filter removes any points behind the LIDAR and any points further than 15 meters to the left and right of the LIDAR.

The perception package uses LIDAR data to identify buoys. The processed LIDAR is first passed through a Euclidean clustering package developed by Autoware.auto. From the clustering, the location, width, and height of the buoys in front of the WAM-V are estimated. To ensure that a randomly incorrect clustering does not lead us to incorrectly detecting the location of a buoy, multiple consecutive clusterings identifying the buoy in the same location are detected before publishing the location of a found buoy. Upcoming developments include algorithms to scan the code using camera data and identify docking locations using camera data.

The navigation package uses Nav2, a sophisticated open-source navigation software, to create paths for the robot to follow. When the package receives a goal to navigate to (from the autonomy package), it calculates a path to that goal through Nav2's path planning action. Once Nav2 publishes the path that it has created for our controller server to use, the navigation package checks to see how we are progressing through the path, and it publishes a success message when it has successfully navigated to the goal (by using comparing our odometry to the goal pose). The navigation package can also handle navigating through multiple set waypoints. To do this, it will navigate to each waypoint in the order received, generating a path to each using Nav2 in order to avoid obstacles between waypoints. When all waypoints are completed, the server publishes a success message.

The autonomy package serves as the "brains" of the robot, coordinating all the other packages. Depending on the task we are running, the autonomy server will launch certain nodes of each package, as some may be necessary for one task but not another. It then uses information from the perception and localization servers to send goals for navigation to the navigation server. For example, for the dynamic navigation demonstration, the autonomy server uses the information on buoy locations from the perception server to send a goal waypoint to the navigation server.

D. UAV Design

Designing a UAV is an iterative process. In order to select an appropriate combination of motors, frames, propellers, and batteries, some estimation is required on the final weight of the UAV to ensure the selected combination is sufficient to lift the takeoff weight. However, this is difficult without having selected the mentioned parts because they contribute a significant portion of the weight as well. The UAV design process therefore requires a rough estimate on the final takeoff weight of the UAV. This rough estimate can then be used to select a combination of motors, propellers, and batteries sufficient to lift the takeoff weight. After finding a suitable combination, the estimated takeoff weight can then be improved based on the parts selected to confirm the chosen combination is sufficient. Even without the motors, frame, and battery selected, some of the sensors and electronics can be chosen based on what the UAV needs to accomplish during the tasks. By selecting the sensors and electronics, a better estimate on the takeoff weight can be obtained.

1) Sensors and Electronics

Depending on the UAV application, there is a variety of different sensors and electronics that can be used. Based on the tasks selected for the UAS, the vehicle needs to be able to not only record but also process images from the surroundings, pick objects up, and then land within the area of a landing platform. This means the UAV will need to carry a camera, an onboard computer for image processing, a grabbing mechanism, a GPS for localization, and a range finder to use for landing. Additionally, the vehicle also needs a flight controller to maintain steady flight.

There are several options for a UAV flight controller. Because the UAV is responsible for autonomous flight and decision making, the flight controller is required to be compatible with a companion computer that can do most of the high-level thinking. Alternative methods would be to do the flight control and the higher-level computations on the same computer, but this would introduce more complexity and would require a more powerful computer. Two popular flight controllers that are commonly used with companion computers include Navio and Pixhawk. Based on availability, the Pixhawk was selected as the flight controller, although they both are very similar. The Pixhawk is also compatible with the Raspberry Pi, a commonly used companion computer.

The Raspberry Pi was selected as the companion computer to do high level calculations like computer vision and decision making. By using the Python package Dronekit, the Raspberry Pi is able to communicate back and forth with the Pixhawk flight controller. This communication allows UAV data to be sent to the Raspberry Pi, which can then send movement commands to the Pixhawk. Alternatives to the Raspberry Pi include Beagleboards and Nvidia Jetson Nanos. The Raspberry Pi offers a larger support community than the Beagleboard, and although the Jetson Nano might be more suitable for Machine Learning applications, the Raspberry Pi offers a more compact package, making it the more suitable option.

For the UAV to detect shapes and letters, a camera is required to sense the environment. Because a Raspberry Pi was chosen as the companion computer, a Raspberry Pi camera was chosen as the onboard camera because it is compact with easy implementation into the companion computer.

The final sensor required is the range finder to assist in the precision landing on the launch platforms of various tasks. The maximum range for different range finders varies greatly, with the price increasing as the range increases. Because the range

finder assists in landing only and not in obstacle avoidance, the maximum range required could be 10m or less. Several higher-end models with longer range would be unnecessary. The selected model was the TF Mini Plus due to its budget-friendly price and maximum range of 12m, which is higher than the planned flight altitude of approximately 5m, so it is sufficient. After having selected the required electronics previously described a closer guess for the final weight of the UAV is achieved. The only components remaining are the motors, propellers, frame, and battery.

2) Mechanical Design

The UAV was designed with considerations to the competition rules as well as to the types of tasks it was expected to complete. The first design decision made for the UAV was to determine if emphasis was to be placed on maneuverability and agility or on endurance and lift capability. Because the tasks for the UAV included searching areas, assisting the WAM-V in navigation, and delivering a payload, the UAV was designed with more of an emphasis on endurance and lift capability. A UAV of this type typically has a larger frame, larger propellers, and a larger battery for increased endurance and lift at the cost of agility and speed, which was deemed as an acceptable cost. However, the competition rules and WAM-V size placed limitations of the UAV frame size and weight.. According to the rules, the UAV had to weigh under 7 kg and float in water. Additionally, the UAV would have to fit on the WAM-V. Therefore, the goal was to maximize the lift capability and flight time of the UAV while still adhering to those constraints.

To start, the frame was the first piece to be selected because there are size constraints due to the available space on the WAM-V and weight limits from the competition rules. Additionally, the frame would need to house all of the previously mentioned electronics and sensors. Because the UAV was meant for endurance and lift, a six-rotor frame was desired because a similar size UAV with four motors would not lift as much or stay airborne as long. Therefore, based on these conditions, an S550 frame was chosen as the UAV frame because it was compact enough to be under the 7 kg weight limit and fit on the WAM-V while still providing plenty of space for the electronics and sensors. Common batteries for UAVs of the size being designed include a 3 cell or 4 cell LiPo battery. While a 3 cell battery is more compact and weighs less, the 4 cell battery gives motors more lift with the additional voltage. For this reason, the goal was to use a 4 cell battery. The exact specifications for the battery would require knowledge on how many amps are required for operation, which is dependent on the onboard electronics and motors. Since the all the onboard electronics are already known, it is easy to estimate their combined required amperage. However, the type of motors will change the amperage requirement, so motor and propeller selection was next.

To select the motors and the propellers, it is important to get an estimate for how much a single motor can lift with a given battery and propeller combination. Common specifications to check for motor selection is the Kv rating, which specify how many rotations per minute a motor will spin per volt provided. The Kv ratings can range anywhere from 800 Kv up to over 2000 Kv. Typically, higher Kv motors produce less torque but spin very quickly, making them very suitable for racing UAVs with smaller propellers. Lower Kv motors have more torque for turning larger propellers, making them very suitable for the current application. The HobbyPower 920 Kv motors were selected because they are compatible with the 4 cell battery and provide enough lift when six of them are paired with 1045 propellers. The lift estimate was obtained from data sheets provided by the manufacturers.

With the motors selected, the speed controllers can then be selected based on the expected amperage requirement from the motors. The maximum amperage expected to be drawn from the motors was about 22 amps, so 30 amp ESCs were selected. The total amperage of the UAV was then calculated with the motors selected. Based on the estimate amperage requirement for the UAV to operate, a 35C 5000mAh 4S was selected.

E. Transportation Logistics

The crates used for the WAM-V in previous years are rather large, and due to the geography around the location in which the vessel is stored, they are rather difficult to move. This year, the team opted to use 3 crates from Shark Crates which are collapsible and can be assembled piece-by-piece. While disassembled, the crates are portable by a small group of people, allowing for the crates to be constructed in a more suitable location accessible by forklifts. The dimensions for the crates are as follows:

- Main crate: 96x48x54 inches
- Pontoon crates: 123x20x37 inches

Fig 5. New equipment crates that can be assembled by hand outside of the building.

IV. EXPERIMENTAL RESULTS

A. WAM-V Simulation-based Testing

The Virtual RobotX [2] simulation environment was utilized for much of the software development. Our design configuration for the WAM-V differs from the default WAM-V in the simulation, so we created configuration yaml files with our design and used them to test the appropriate WAM-V design in simulation. For this configuration, we had four motors rather than two on the WAM-V and angled in a holonomic manner. The sensor location on the WAM-V were also changed

to better match the configuration of the sensors on the physical WAM-V.

For each task we wanted to test in simulation, we first opened the VRX-provided simulation, and set up the task with buoys. Then, we launched the ROS1-ROS2 bridge and our software to see if WAM-V could complete the desired task autonomously. Additionally, we could verify certain parts of our software stack were working by using tools like Rviz to visualize the data being published to certain topics (e.g., filtered LIDAR, Euclidean clusterings, odometry, path plans, etc.). Below are examples of our task setup in simulation for certain tasks:

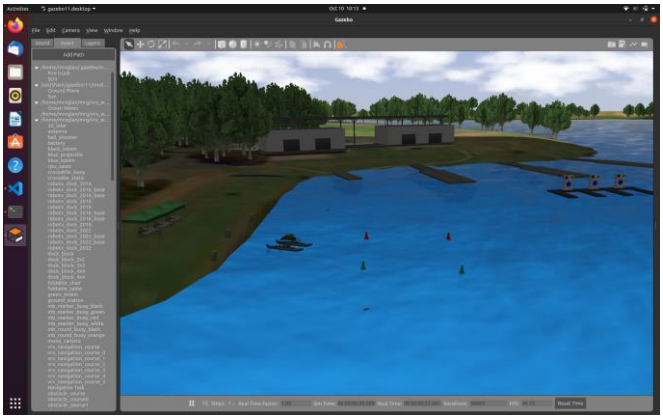

Fig 6. Dynamic navigation demonstration.

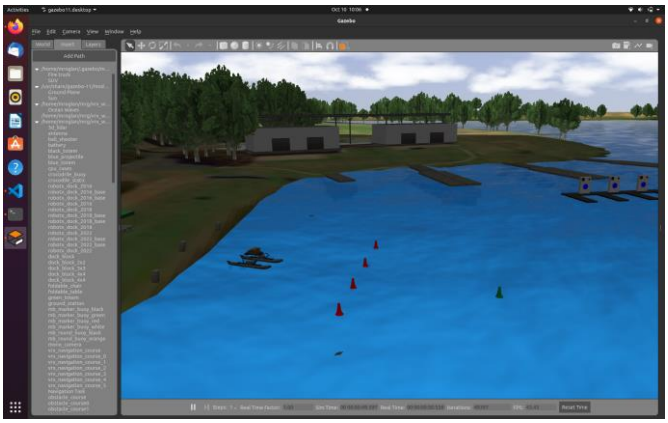

Fig 7. Enter and Exit Gates

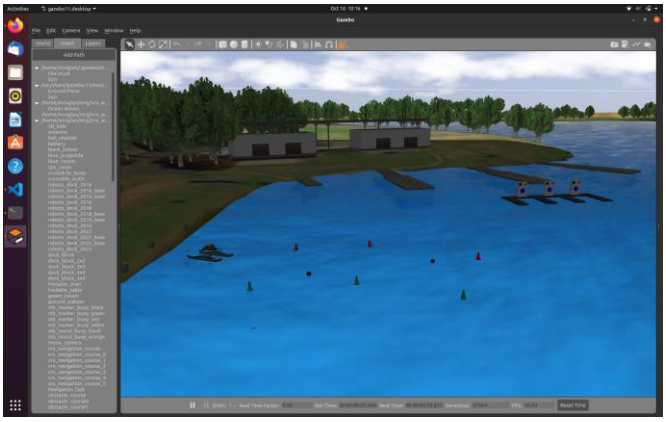

Fig 8. Follow the Path

B. WAM-V Lake Testing

The WAM-V was tested at a local reservoir three times prior to the competition.

1. The objectives of the first lake test included learning vehicle deployment strategies and testing the new propulsion system. All objectives were achieved and exceeded.
2. The objectives of the second lake test were to improve vehicle deployment strategies, test motor control over a micro-ROS interface, test remote networking and control with tmux, and test sensors, localization, and navigation. Issues were encountered with configuring auxiliary device ports on the main computer. Data was collected.
3. The objectives of the third lake test were to test improvements made to the micro-ROS interface, sensor interfaces, and navigation. Issues from the previous test were resolved, and information for debugging navigation was obtained. Data was collected.

Without any previous members present, the team relearned how to deploy the WAM-V over the several lake tests.

C. WAM-V Headless Testing

With the WAM-V heading out, testing of localization will have to be carried out without the vehicle. The sensor bar has enabled testing of localization and perception without the WAM-V platform present.

D. UAV Simulation Testing

1) Simulation Environment

To test the UAV behaviors and performance of the tasks, a simulated UAV was tested first. Using a simulated UAV to test the ability to complete tasks allowed for more rapid testing. However, a simulation environment was required for the different tasks the UAV would be completing. The simulation environment used was Gazebo, which is a popular simulation environment due to its ability to communicate with ROS, which can be used to control both a simulated and physical robot system. From [1], Gazebo is recommended to simulate robot systems with physical counterparts. To use Gazebo, a world file is required that contains models of the items in the world. To simulate the UAV, a model of the Parrot quadcopter was used. Although the physical UAV used had six rotors, the Parrot quadcopter model was sufficient for testing algorithmic performance for the tasks. With a world file specifying the objects and UAVs within the world, the next step was to simulate the UAV software for actually completing tasks.

ArduPilot is an open-source autopilot system for controlling vehicles, and it is used on the Pixhawk flight controller of the UAV. One of the benefits of using ArduPilot, is that it can be used to connect with Gazebo for Software in the Loop (SITL) simulations. By combining ArduPilot with Gazebo, a simulated UAV can behave as if it were in the real world due to the physics engine within Gazebo and the UAV can respond as it would in the physical world using the ArduPilot.

The last remaining component for testing a simulated UAV is creating scripts that can be used for completing the tasks. A combination of DroneKit and ROS were used for completing the tasks with Python scripts. DroneKit can be used in the simulation and the physical system to communicate with the ArduPilot in the flight controller. This communication could be to obtain UAV state information like position and velocity to be used in the Python scripts, but DroneKit can also send movement commands to ArduPilot to execute.

Within the model of the Parrot quadcopter was a downward facing camera on the UAV. This camera publishes video feed captured from the camera to a ROS topic. Therefore, within the Python scripts to complete the tasks, a ROS node was created that subscribed to the topics containing the captured video feed. This allowed the video feed to be analyzed within the Python scripts for various tasks.

2) Search and Report Task

As mentioned, testing the tasks within the Gazebo environment required a world task containing the items featured in the tasks. From the task description, the Search and Report task contains four cones marking the boundaries of the search area, one letter marker of an "R", one letter maker of an "N", and a landing platform with concentric circles. This description was used to create the world file. A overhead view of this world can be seen in the image below with the Parrot quadcopter on the landing platform.

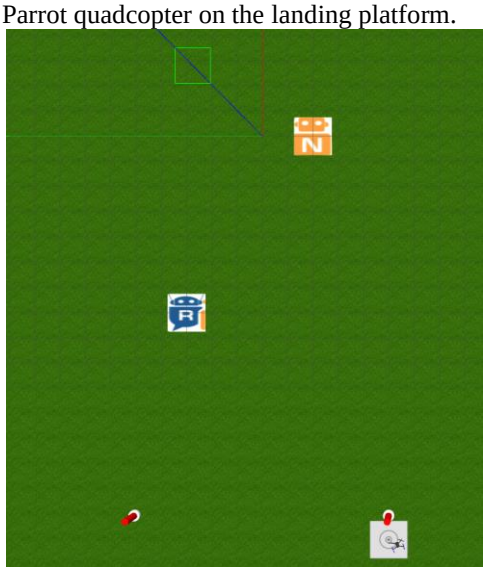

Fig 9. The Search and Report simulated world. Within the world contains the landing platform and UAV (bottom right), four cone markers (two are shown), and the "R" and "N" letters for the UAV to find.

Within the Python script for completing this task, the search area was divided into various GPS coordinates based on the initial GPS location of the UAV. Using DroneKit, these GPS coordinates were then sent to the flight controller. The flight controller then proceeded to make the UAV takeoff to an altitude of 5m, where it would then travel to the each GPS coordinate in the list sent by DroneKit. At each GPS point, the camera on the UAV would record the footage of the ground below and publish that to the ROS topic. Within the Python script for completing the task, this footage was taken and analyzed to find any letters within the feed. When a letter was detected, the script would attempt to determine which letter it was. The below pictures show some of the results of the UAV attempting to recognize the letters seen.

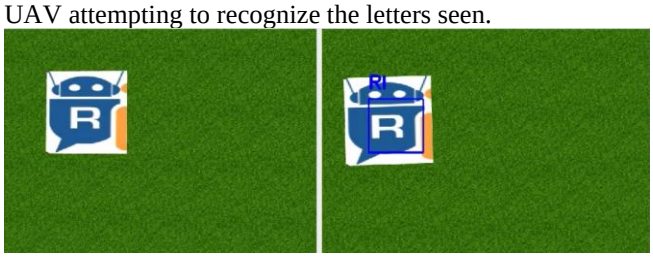

Fig 10. The UAV recognizing the text from the onboard camera. The left image is the raw footage, and the right image is the image after processing and analyzing. The UAV predicts the image contains the text "RI" instead of just the shown "R."

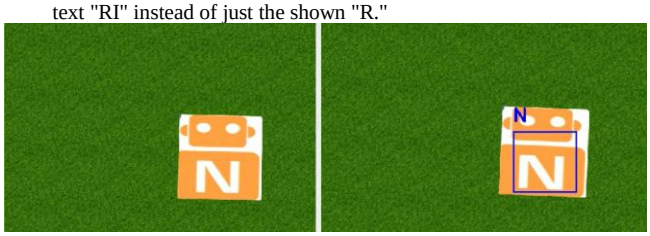

Fig 11. The UAV attempting to recognize the letter "N" shown in the image. The left is the raw footage captured while the right is the image after text detection and recognition. The UAV can correctly predict the text.

As seen in the previous photos, the UAV was able to detect and predict letters with mixed results. When attempting to find the letter "R," it was not uncommon for the UAV to predict other letters were included along with the "R." Reasons for this could be the way the "R" appears in the logo, and solutions could entail further image processing to filter out some extraneous noise. However, this is not a significant issue. If the UAV can predict an "R" is involved in the photo, then the UAV determined the location of the "R" to have been found, even if the UAV predicted other letters were with it.

The UAV was able to detect the "N" with more success than the "R." A reason for this could have to do with the difference in the colors of the logos. However, sometimes the UAV would predict other letters alongside the "N," similar to the "R." Therefore, as long as "N" was within the predicted text, then the UAV assumed the "N" was found. This approach would only be problematic if the "R" and the "N" would appear together, but this has never happened in simulation.

Once the UAV detects the "R" and the "N," it then attempts to return to the launch location to land on the landing pad. Like detecting and recognizing text, the UAV again relies on video feed from the camera, but this time it attempts to detect the circles of the landing platform to assist in the landing. A shape detector was used to find shapes within the video feed using OpenCV. Knowing the landing platform was a square, the UAV attempted to initially locate the square within the video feed and identify the center of the square. Eventually, the UAV would be too close to the ground to identify the entire landing platform, so it would switch to attempting to find the center point of the circles at about 1m. This center point was used as a guide for the UAV

to maneuver until the center point of the landing platform or circles was centered in the video feed. The image below shows the UAV attempting to land on the platform.

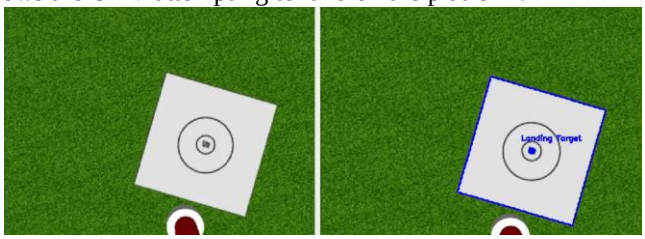

Fig 12. The UAV using shape detection to find center point of the landing platform to use for landing maneuvers. The left image is the raw footage, and the right image is the image after processing and analysis identifying the platform bounding box and center point.

Within simulation, the UAV struggled to find the circles. Part of this could be due to the low resolution of circles, resulting in a pixelated look. Therefore, the strategy was modified a bit to have the UAV just look for the square of the platform to use as a guide. This was sufficient in simulation even though as the UAV decreased in altitude, the entirety of the platform was outside the view of the camera, resulting in the UAV losing track of the center of the platform at low altitudes. Despite this, the UAV was still able to see the platform until it was just a few meters off the ground, resulting in a successful landing on the platform. However, for the physical system, the UAV will attempt to use the circles since they will be in the camera view for longer and the circles can appear less pixelated.

V. ACKNOWLEDGEMENTS

The efforts made working towards this competition extend far beyond the team attending the RobotX Challenge and beyond the current membership of the Marine Robotics Group. The team thanks all the members who worked towards this competition over the past four years, and alumni who have remained in contact to advise on development. Special thanks to Akhil Sadhu, Nicholas Graham, Nicolas Marsilio, Avery Sawyer, Bezayit Urgessa, Aaron Wu, Allison Fister, Shawn Coutinho, Ryan Otsuka, Saahas Yechuri, Yash Chitale, Tyler Campbell, Vinny Ruia, and Eric Phan.

The team also thanks Dr. Michael Steffens for advising the project, Tanya Ard-Smith for assisting logistical efforts, and Prof. Dimitri Mavris and the Aerospace Systems Design Laboratory (ASDL) for supporting the project and the group at large.

The Marine Robotics Group also thanks sponsors and other organizations who have assisted the team in preparation for this competition, as well as previous competitions which the group was unable to attend due to global circumstances. Those sponsors and supporters include Greenzie, Fischer Connectors, Altium, Connect Tech Inc., and Pattent. Additionally, MRG thanks the Georgia Tech Student Government Association, Georgia Tech Student Organization Finance Office, Georgia Tech Student Foundation, and the School of Aerospace Engineering Student Advisory Council for their support.

APPENDIX – REFERENCES

[1] M. Korber, “Comparing Popular Simulation Environments in the Scope of Robotics and Reinforcement Learning,” 2021.

[2] B. Bingham et al., "Toward Maritime Robotic Simulation in Gazebo," OCEANS 2019 MTS/IEEE SEATTLE, 2019, pp. 1-10, doi: 10.23919/OCEANS40490.2019.8962724.

[3] D. Dulle et al., “Georgia Tech Marine Robotics Maritime RobotX Challenge 2018,” Presented at RobotX 2018, 2018, [ONLINE]. Available: https://robonation.org/app/uploads/sites/2/2019/09/GT_RX18_Paper.pdf

[4] D. Frank et al., “University of Florida: Team NaviGator AMS,” Presented at RobotX 2016, 2016, [ONLINE]. Available: https://robonation.org/app/uploads/sites/2/2019/09/UF_RX16_Paper.pdf

[5] T. Moore. Robot_localization wiki. [Online] Available: http://docs.ros.org/en/noetic/api/robot_localization/html/index.html