UAV Drone Replenishment: Final Report

ALEC DIGIROLAMO, ADCS Dynamics, USA
DANIEL SANEI, ADCS Dynamics, USA
CARSON RAE, ADCS Dynamics, USA
SOUMI CHAKRABORTY, ADCS Dynamics, USA

The UAV Drone Replenishment project presents the development and implementation of a drone system for RoboNation's 2024 RobotX Maritime Challenge, specifically for the UAV Replenishment task. This task requires the drone to take off from the boat, identify two landing pads, transfer a tin can from one helipad to the other, and land back on the boat. During the course of this project, the team successfully assembled the drone hardware and implemented the helipad object detection model. The development of OpenCV tin can detection is still in progress, and the autonomous flight control software is a stretch goal for future development. This project is a part of Triton AI, and was conducted under the guidance of Professor Kastner and fellow staff for CSE 145(Embedded System Design Project) during the Spring 2024 quarter at the University of California, San Diego.

ACM Reference Format:
Alec DiGirolamo, Daniel Sanei, Carson Rae, and Soumi Chakraborty. 2024. UAV Drone Replenishment: Final Report. 1, 1 (June 2024), 14 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

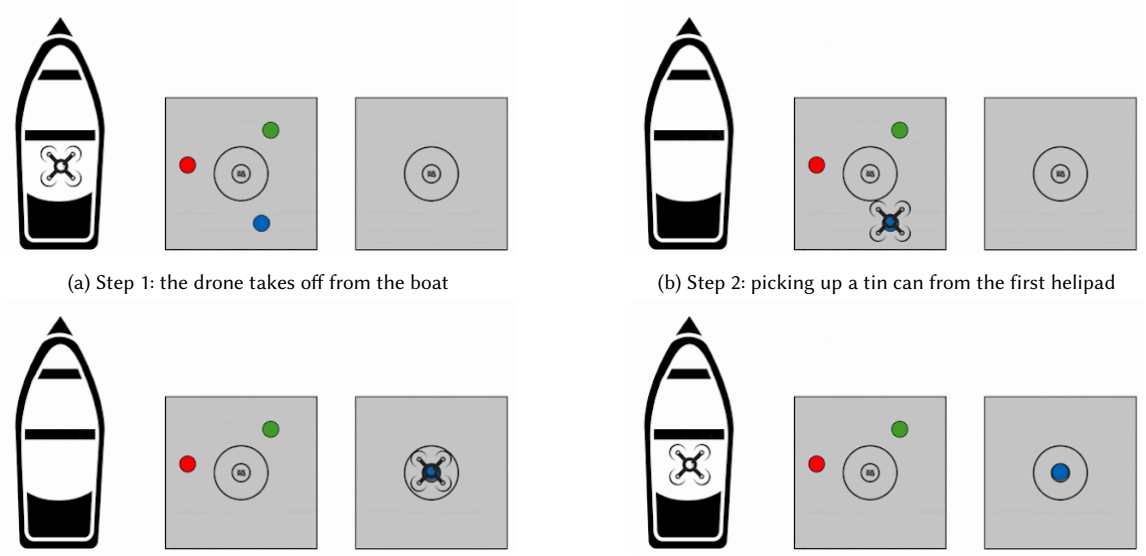
1 INTRODUCTION

Autonomous drones have seen widespread applications in delivery systems, search and rescue, and environmental monitoring. The integration of flight controllers with advanced sensors and machine-learning models has enabled drones to perform complex tasks without human intervention. One of the most difficult challenges for autonomous drones is operating in maritime environments due to changing conditions and the consequences of failure.

The Maritime RobotX Challenge offers a unique opportunity for students to hone their robot development skills in a technical environment. The challenge includes the UAV Replenishment task, where a drone needs to take off from a boat, interact with an object, and return to its starting point as shown in Figure 1. This challenge simulates delivery systems or environmental monitoring scenarios and presents unique development obstacles that enhance robotic development skills.

Authors' Contact Information: Alec DiGirolamo, ADCS Dynamics, San Diego, California, USA, adigirolamo@ucsd.edu; Daniel Sanei, ADCS Dynamics, San Diego, California, USA, dsanei@ucsd.edu; Carson Rae, ADCS Dynamics, San Diego, California, USA, carae@ucsd.edu; Soumi Chakraborty, ADCS Dynamics, San Diego, California, USA, sochakraborty@ucsd.edu.

Manuscript submitted to ACM



(a) Step 1: the drone takes off from the boat

(b) Step 2: picking up a tin can from the first helipad

(c) Step 3: delivering the tin can to the second helipad

(d) Step 4: the drone returns to the boat

Fig. 1. Visualization of the UAV replenishment task

Our project aims to develop a fully autonomous drone system capable of completing the UAV Replenishment Task for the Maritime RobotX Challenge 2024. The specific objectives of the task are to take off autonomously from a boat, identify two helipads using computer vision, fly to and pick up a tin can from the first helipad, fly to and place the tin can on the second helipad, and return to and land on the boat.

To achieve this, we have, with the help of our mentors, designed and built a drone equipped with a Pixhawk Flight Controller, Jetson Nano, GPS Module, two Arducams, a magnetic pick-up system, and a transceiver. The drone utilizes two downward-facing Arducams, one for helipad detection using YOLOv5 and one for tin can detection using OpenCV.

The contributions of this project are as follows:

(1) **Drone Build:** We have successfully built the drone as per our mentors' specifications and guidance.
(2) **System Integration:** We have successfully integrated the various hardware and software components including the Jetson Nano, Pixhawk, and ROS2 Humble.
(3) **Helipad Detection:** We have successfully trained and deployed a YOLOv5 model that can run live and offline in parallel with drone flight.
(4) **Tin Can Detection:** We have also developed a basic color and circle detector using OpenCV for this task. This task is still a work in progress.
(5) **Autonomous Control:** We are presently engaged in developing and testing autonomous control software using ROS nodes and sensor data.

2 RELATED WORKS

The development of autonomous drones for complex tasks, such as the UAV Replenishment task builds upon related research projects within the field. These projects include drone navigation, machine learning, computer vision, and maritime operations.

2.1 Machine Learning for Object Detection

Object detection is a key aspect of our UAV Replenishment task due to the need to identify helipads and tin cans. The YOLO (You Only Look Once) framework [2], introduced by Redmon in 2016, revolutionized object detection by allowing for fast and accurate identification of images. Over the last few years, improved versions of YOLO have come out, such as the YOLOv5 model [3] introduced by Jocher. Each model version has improved the models' ability to object detect accurately. Our use of YOLO in this task is key to our project, due to its speed and accuracy in offline conditions. Lee et al. proposed a hybrid approach for UAV object detection, offloading computationally intensive tasks to a computing cloud while keeping low-level object detection and short-term navigation onboard [4]. Our approach is segmented in a similar manner, with our object detection model for the landing pads being trained on RoboFlow using YOLOv5, while our tin can detection will be performed using OpenCV algorithms. Offloading computing tasks to a computing cloud would not be a necessary solution for our project, however, since our object detection model operates without overloading issues during live test flights.

2.2 Autonomous Drone Navigation

There have been many attempts at autonomous drone landing algorithms with mixed success. The paper we looked to for our initial inspiration was Autonomous Quadcopter Landing on a Moving Target [5]. In this paper, they were able to land a quadcopter on a moving target 22 out of 27 attempts, with the failure attempts never exceeding 60 cm. While initially wanting to implement a similar algorithm, we ended up designing a simpler algorithm due to time constraints.

A project by Shen et al. to develop a lightweight autonomous quadrotor used only cameras and an IMU for state estimation and control [6]. The project developed a system with the primary focus on visual-inertial state estimation and demonstrated the quadrotor's ability to autonomously fly at This research demonstrated the quadrotor's ability to autonomously travel at speeds up to 4 m/s and roll and pitch angles exceeding 20○ during indoor three-dimensional experiments [6]. This autonomous functionality is beyond what is required for the UAV Replenishment task, although there are takeaways within the object detection and autonomous navigation aspects of the quadrotor project.

2.3 Computer Vision in Maritime Environments

Zeng et al. [7] developed a computer vision system that identified buoys in a maritime environment. This paper demonstrated a robust model that operated well in moving water and varying weather conditions. Their project is particularly relevant because of its success within the maritime environment, showcasing the potential for computer vision to handle dynamic and unpredictable conditions at sea. However, our project extends their capability because we also need to manipulate objects. While Zeng et al. focused on detection and tracking our project needs to go a step forward by not only detecting helipads and tin cans, but also interacting with them.

2.4 Autonomous Object Manipulation using Machine Learning

Bohg et al. [8] reviewed many approaches to grasping and manipulating objects in autonomous settings. Their work highlights the importance of sensor fusion and confidence in the data received from sensors in object manipulation. Sensor fusion involves integrating data from multiple sensors to create a more accurate and reliable perception of the surrounding environment. In our project, many factors affect our ability to complete the task. This paper helped us to understand the key aspects that we must focus on in our development process. By leveraging these insights we worked to ensure that our drone would be able to complete the UAV Replenishment task.

2.5 Real-Time Multi-Object Tracking

Bewley et al. [9] introduced a simple, yet effective algorithm for real-time object detection and tracking called SORT (Simple Online and Real-time Tracking). Their approach combines the Hungarian Algorithm with a Kalman filter, resulting in a sturdy and efficient system with low overhang so that it can operate in real-time. This work is relevant in our project because we need to track multiple objects, including the drone, helipads, tin cans, and boat in real-time. This paper has inspired our approach to designing a control algorithm and will be crucial to future work.

2.6 Maritime Drone Operations

Operating drones in water-based environments introduces several challenges, including power, communication, and stability. Zhang et al. [10] explored the impact of the state of the sea on drone stability, they proposed that dynamic control algorithms are necessary when flying a drone in maritime conditions to maintain control of the drone. Their research is relevant to our project as it'll be crucial that we maintain stable control of the drone whilst completing the replenishment task due to the needs of computer vision and object manipulation. Additionally, Hansen et al. [11] examined the difficulties of communication in ocean environments. They concluded that hybrid communication models that use satellite and radio are crucial to ensure reliable data transmission. This is important in our project because it helps us to identify the ideal communication pattern between the QGroundControl software on our computer and the drone completing the tasks hundreds of meters away.

3 SYSTEM ARCHITECTURE

Our drone consists of two control boards and various hardware components. Specifically, a Pixhawk provides the low-level control of the drone, This takes in data from various sensors which control the output to the motor controllers. Alongside the Pixhawk is the Jetson Nano. This is what runs high-level control, such as navigation planning, as well as the computer vision software. This takes in data from two cameras mounted onto the drone and sends commands to the Pixhawk for control. Figure 2

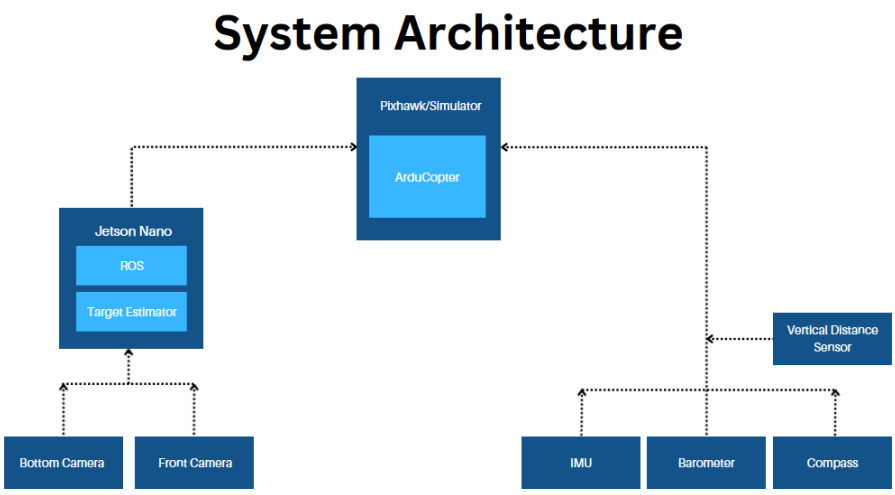


Fig. 2. A diagram of the system architecture.

4 HARDWARE COMPONENTS

The selection of the hardware components was provided by our technical leads at Triton AI. The primary components were the flight controller, a Pixhawk 2.4.8. This provides the low level control as well as integrating the majority of the sensors on the drone. These sensors include a GPS, barometer, accelerometer, and magnetometer. We used a NEO-8M based GPS module that has a positional accuracy of as small as 2 meters. The Jetson Nano mounts to the underside of the drone and takes in data from the two onboard cameras. These cameras which run our inference model are the Arducam 1080p lowlight camera. The ultrawide lens that comes with this camera allows our drone to see as much of the surrounding as possible. The Tarot TL68P07 brushless motors provide plenty of power to carry the drone's weight and perform our goals.

4.1 Hardware Challenges

Many challenges came with the assembly and first flight of our drone. None of the team members had experience building drones and only some had experience with hardware in general. This allowed each of the team members to learn skills such as soldering, wiring management, hardware assembly, and software integration. As well as building the skills to assemble the drone, most of the members had no experience flying drones. Due to our initial inexperience, we leaned on the expertise of our technical leads to provide guidance when we found ourselves stuck. One of the main issues that we faced was instability. We used QGroundControl to set the drone too fly autonomously, attempting to hold its position or altitude. When we would set the drone into these modes, the drone either shook violently or wandered from its hold position. For this reason, we would not be able to implement an autonomous control algorithm, as it was too unsafe to test the algorithm we had designed.

5 JETSON NANO

The Jetson Nano is a small computer designed for embedded applications and artificial intelligence IoT, which operates as the brains of our drone. All of our essential functionalities were implemented inside the Jetson Nano, including the code for managing the downward-facing Arducam images and videos and the ROS environment for autonomous flight control software.

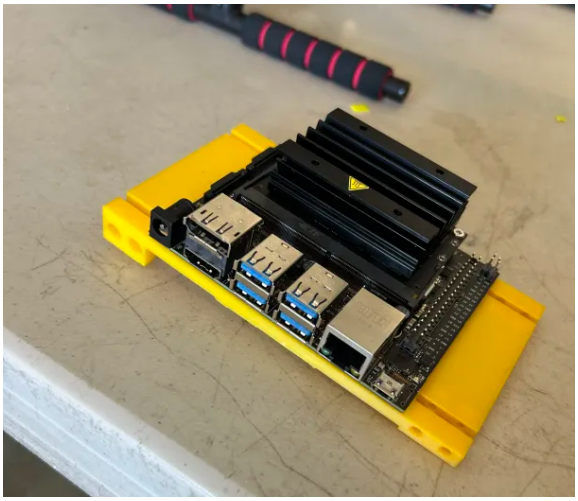


Fig. 3. Jetson Nano

5.1 Arducam Python Scripts

On the host terminal for the Jetson Nano, we had written several Python scripts for handling the image and video-taking capabilities. Having gone through various iterations, we began with a simple script that would take images from the Arducam at 1-second intervals, saving the take images to a specific directory. In the next version, we used our object detection model to run an inference on the images taken, which produced a JSON response containing details for the localization of the helipad, if it existed in the given frame. This was initially a static process, where we would collect images during a test flight, and then after safely landing and disarming the drone, running an offline inference on each frame to localize the helipad. As a visual support, we implemented a feature to draw annotations over the original image, displaying square annotations and labels for both the entire helipad and the circular target. The following iteration incorporated a new function that used the location of the helipad in reference to the image frame to determine which direction the drone should take next to adjust its positioning. While the full autonomous functionality is a stretch goal, this script successfully produces a CSV file in which directions are given to adjust the drone's position such that the helipad is at the center of the frame. The final Python script took a live video of the Arducam feed at 20 FPS.

5.2 ROS2 Humble Docker Container

While the focus of our MVP was the object detection model for the landing pad and tin can detection, we had successfully configured our Jetson Nano environment to support the Robot Operating System (ROS). We were advised to use ROS2 Humble for our development environment, however, the host machine was flashed with Ubuntu 18.04 version, whereas ROS2 Humble required Ubuntu 22.04 version. To resolve this conflict, we pulled a Docker image for ROS2 Humble and ran the container to continue our development. While autonomous functionality is beyond our MVP, we were successfully able to take an Arducam image from within the ROS2 Humble Docker container, a positive indication of the correct setup of our environment to allow for further ROS development. This achievement required the use of volume mounts, which are detailed in Challenges and Solutions (section 5.5).

5.3 Object Detection Inference

In this context, inference is the process of using pre-trained object detection models to analyze new images and identify specific objects, which in our project is the landing pad. To run inference on our Arducam images, we first needed to run the Docker container that hosts our RoboFlow model before running the Python scripts. We keep the terminal open as the inference server inside the container must be continuously running in order to process the images. Closing the terminal would stop the server and interrupt the inference process. New terminals would be opened to access and run any necessary Python scripts for both static and live tests. This is a critical step to ensure the inference for our landing pad detection can operate.

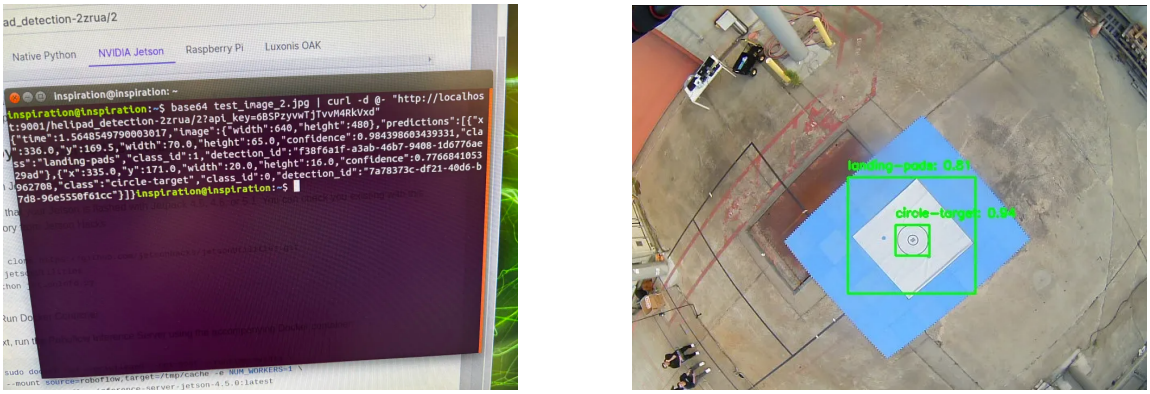


Fig. 4. Live inference during test flight

5.4 Pixhawk Integration

While the autonomous functionality of the drone is beyond our MVP, our team decided to integrate the Jetson Nano computer with the Pixhawk flight controller to successfully set up all the necessary environments to continue this project in the future. Our Jetson Nano had 4 available USB ports and our Pixhawk had an available micro-USB port, which are the ports we used to connect the two devices. This connection is possible through a serial connection as well, though we opted for the USB connection due to simplicity. We decided to integrate the Pixhawk to the Jetson Nano through the ROS2 Humble Docker container, since that would also be the environment for implementing the autonomous functionality of the drone. Within the container, we installed MAVROS, which is a ROS package that allows the user to control the drone using the MAVLink protocol. The next step was to bridge the Jetson Nano to the Pixhawk through ROS topics, and echo these topics to monitor real-time messages. After a few successful tests, we determined that we successfully achieved integration between the Jetson Nano and Pixhawk.

5.5 Challenges and Solutions

We ran into several complications with the configurations for this environment. The first was that the changes we made inside our container did not persist after the container stopped. Initially, we were running the container without memory, though even during tests where we ran the container with memory, the issue was still not resolved. Our solution was to make use of volume mounts, which ensure that code files, configurations, and data from within the container are also reflected on the host machine, and therefore remain accessible even after the container restarts.

This involves creating a new directory on the host machine and another new one inside the Docker container, and changing the file ownership so that both the host and container can access each other's files within their respective directories. This also meant that when running the container, we had to use privileged mode in order to allow full access to hardware and system resources from within the container, and we also had to use device mapping to allow specific hardware devices (i.e. Arducam camera and Pixhawk flight controller) to be accessible from within the Docker container.

The second issue stems from the Docker container's initial presets, which only contained ROS2 Humble. As a result, we had to manually install all relevant packages each time we ran the Docker container. A permanent solution to this situation would be to create a new Docker container that already comes installed with our relevant packages, though due to the interest of time and our approaching deadlines, we decided to continue with this method.

The third issue was our available storage space. The Jetson Nano had 20.1 GB free, while our RoboFlow inference server Docker container required 9 GB of space, and our ROS2 Humble Docker container required 13.3 GB of space, adding up to a total of 22.3 GB necessary. Even after performing cleaning operations through the terminal, we did not have sufficient space necessary for both Docker containers, and additionally ran into less available storage space when trying to install various packages. Once again, due to our project time constraints, we decided to continue development with a temporary solution. After setting up the ROS2 Humble Docker container and successfully accessing the Arducam to take an image, we removed the container to make room for the inference container and continue with our object detection development.

The fourth and final challenge occurred during flight tests. We were able to connect the Jetson Nano to the internet through Ethernet connection, as our hardware was not equipped with a WiFi adapter to connect wirelessly. At the offsite Inspiration lab, we performed test flights in an open space in which we did not have access to an Ethernet cable. Additionally, without a power source readily available at the test flight location, we could not use an external keyboard and monitor to manually run the Python scripts to begin taking Arducam images. Our solution was to write the command for running the script at the end of the /.bashrc file, so that the Arducam would start taking images upon the Jetson Nano startup. This resulted in many null images before and after our test flight during the walk to and from the test flight location. Due to our team not having access to the Jetson Nano while transferring the drone to the test flight location, we had no way of ensuring successful operation of the script. This resulted in instances where the Jetson Nano failed to take images during a test flight, which required us to check our environment setup and our code before performing a second attempt. This issue was resolved after we achieved offline inference, as we no longer needed the internet to utilize our object detection model for the landing pad, and after utilizing the Aerodrome facility at UC San Diego, where readily available power outlets enabled us to monitor the Jetson Nano operations directly prior to test flights.

6 OBJECT DETECTION

For this project, our drone needed to successfully identify two helipads—one for picking up a tin can and another for delivering it. Additionally, it needed to detect the tin cans themselves in order to pick them up. This requirement makes the task an object detection problem, which was the main deliverable of our project. We developed an end-to-end pipeline from collecting datasets to deploying our model on the Jetson Nano for accomplishing this task.

We separated the object detection into two distinct tasks: tin can detection and helipad detection. The detailed process for each task, from data collection to model deployment, is described below.

6.1 Helipad Detection

The RobotX competition provided the dimensions and images of the helipad to be used in the actual competition, as shown in Figure 5. Given that our drone is equipped with a Jetson Nano, we leveraged its computational capabilities to deploy a computer vision model for this task. We took inspiration from previous TritonAI teams and decided to train a YOLOv5 model from scratch.

2.5.8 Task 8 – UAV Replenishment

This task is designed to be accomplished using a UAV. The UAV launches from the USV, locates a floating helipad and collects a small colored tin (see Figure 12). The UAV delivers the tin to the circular target area on another floating helipad, then returns to the USV.

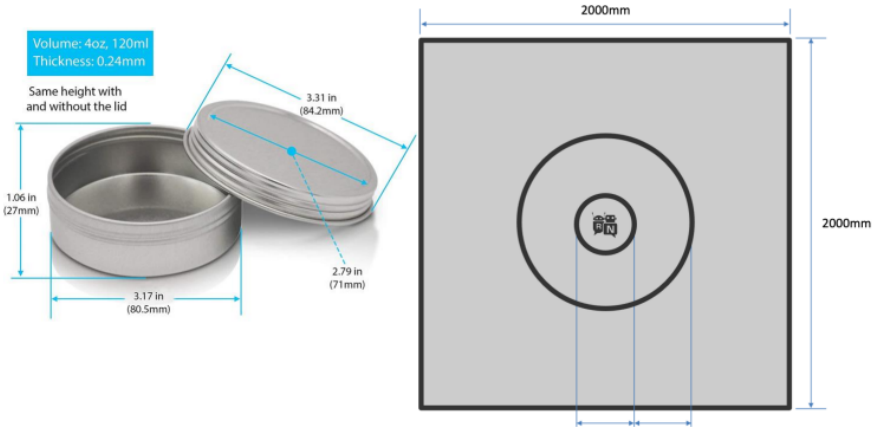


Figure 12: Colored Disk and Preliminary Helipad

Task Description

The two floating helipads are 2m x 2m and raised approximately 0.3m from the surface of the water and marked with concentric rings. The center ring is 0.24m in diameter, and the outer ring is 0.72m in diameter. The colored tin will be placed within the 0.24m ring.

The composition of the colored disks is described in Table 10. There may be multiple-colored discs on a helipad, (red, green, or blue). (Figure 12)

Task Elements

Task Element	Description	Base Dimensions
Floating Dock	Floating platform that holds helipad	Overall width: 2m x 2m
Helipad	See Figure 12. Helipad will be secured on a hard, flat, raised surface floating on the water.	2m x 2m
Colored Disks	SimbaLux Screw Top Round Steel Tin Cans 4 oz (120 ml). See Figure 12. Available for purchase: www.amazon.com	N/A
Dock units are supplied from DOCKPRO: dockpro.com.au		

Table 10: Task Elements for UAV Replenishment

Fig. 5. UAV Replenishment Task

6.1.1 Dataset. The initial step in developing a robust YOLO model involved creating a dataset for training. Due to the lack of access to footage from previous competitions, we decided to curate our own dataset. To do this, we collected

images of the helipad using the drone's downward-facing camera. The drone is equipped with an ArduCam connected to the Jetson Nano. We used Python to capture one picture per second with this ArduCam. The helipad was positioned on a grassy field, our primary test flight site, and images were collected over a 10-minute period, resulting in approximately 600 raw images (Figure 6). In addition to images of the bare helipad, we also captured scenarios with tin cans present to simulate competitive conditions. It is also worth noting that the test flights were conducted on a cloudless, sunny day, introducing glare on the helipad surface — a condition likely to mimic competition settings.

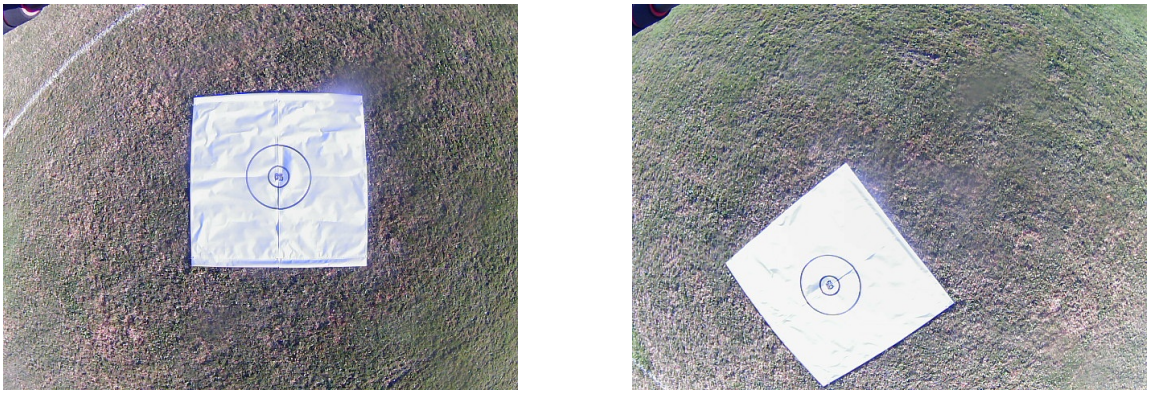


Fig. 6. Raw images of the actual helipad collected with the drone

After cleaning up the collected images, we obtained a total of 490 usable raw images for our dataset. Each of these images was annotated using Roboflow, where we identified two key elements on each helipad image: the entire helipad itself and the circles located in the middle. As for preprocessing steps, we took the computational abilities of the Jetson into consideration and resized all images to fit within 416×416. Additionally, we auto-oriented Given our limited number of raw images, we augmented the dataset by generating three new examples per image using the following transformations:

- **Rotation**: Between -15° and $+15^\circ$
- **Shear**: Horizontal shear: $\pm 10^\circ$, Vertical shear: $\pm 10^\circ$
- **Exposure**: Between -10% and $+10\%$
- **Blur**: Up to 1.6 pixels
- **Noise**: Up to 0.1% of the pixels

After augmentation, we were finally left with a dataset comprising 1500 images. These were split into training (90%), validation (7%), and testing (3%) sets.

6.1.2 YOLO Model. After curating the dataset, our next objective was to train a computer vision model capable of performing object detection specifically to detect the helipad. We drew inspiration from previous teams in TritonAI and decided to test out a YOLOv5 model. We were only able to train the model for 150 epochs due to computational constrains, but, as shown in Figure 7, the loss reduced satisfactorily over 150 epochs.

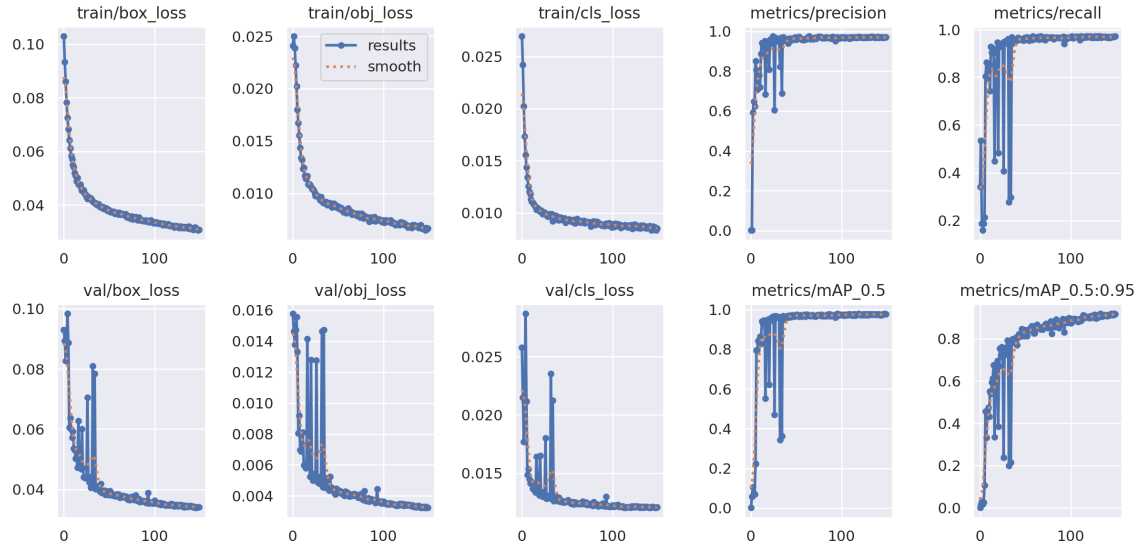


Fig. 7. Training Graphs

We also trained a YOLOv8 model, which exhibited slightly higher accuracy compared to the YOLOv5 model. However, the YOLOv8 model proved too resource-intensive for the Jetson Nano, leading to several deployment issues. Consequently, we finalized the YOLOv5 model for practical deployment. Instructions for deploying the YOLOv8 model are available in our repository for future teams interested in exploring its capabilities.

6.1.3 Testing. We evaluated our model in two primary stages: offline testing and real-time testing, both conducted on the Jetson Nano. For offline testing, we deployed our YOLOv5 model to Roboflow and ran inference on the test set from our dataset. These images were similar to the training set, as they were collected from the same test flight. This step verified that the model performed well on data similar to what it was trained on.

The real-time test was the definitive evaluation of our model's performance. For the model to be successful, it needed to run on the Jetson Nano with reasonable speed and without lag. Additionally, it had to operate without internet access, as the drone would not be connected during the competition. Therefore, we housed the YOLOv5 model on the drone inside a Docker container, ensuring all processes were executed within the Jetson Nano.

We tested the model in two different environments. First, we returned to the original test flight location where the dataset was collected. This environment was similar to the dataset, with sunlight and glare being the only variables. Second, we tested the model at UCSD's Aerodome, a significantly different setting with a concrete backdrop instead of grass and minimal direct sunlight on the helipad.

(a) Enter Caption (b) Enter Caption

Fig. 8. Overall caption describing both figures

Our model performed exceptionally well in real-time in both settings, demonstrating robustness and reliability that gives us confidence in its performance in the competition settings.

6.1.4 Challenges. Our dataset underwent several evolutions throughout the process. For most of the quarter, we were unable to obtain a full-sized print of our helipad due to complications with our printing vendor. However, not wanting to delay the development of our object detection model any further, we devised a makeshift landing pad to build our initial YOLOv5 model. We stitched together six gray boards to create a helipad measuring 60 by 60 square inches. We then drew two circles on it, matching the size of the actual helipad.

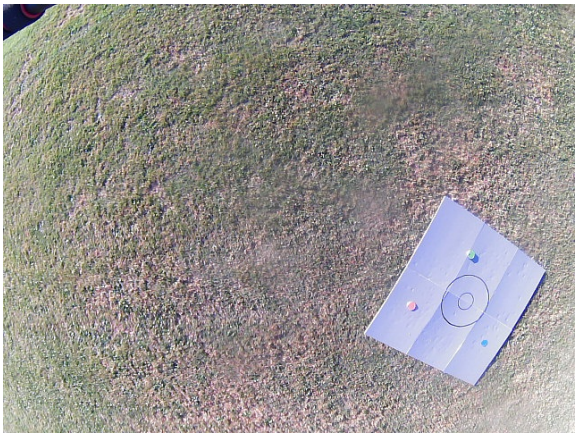


Fig. 9. Drone image of our makeshift helipad

Our first YOLOv5 model was trained on images of this makeshift helipad. Although we did not use the dataset built for this helipad or the resulting model, the process helped us establish a complete object detection pipeline from data collection to model deployment. As a result, when we finally obtained an accurate helipad, we were able to deploy our final model within three days of collecting the new dataset.

6.2 Tin Can Detection

Once the drone detects the helipad, the next step is to navigate towards it. As it approaches the helipad, it must also identify the colored tin cans to pick them up. This introduces the second computer vision task: detecting red, blue, and green colored tin cans. Initially, we trained a YOLOv5 model for this task as well. However, since the cans were distinctly colored (red, blue, or green), we opted to use OpenCV for detection. Although we did not fully complete this task, we developed a prototype that converts the image to the HSV color space and uses contour detection to identify the colored circles formed by the tin cans. We also experimented with Hough transforms to detect circles but found the previous method to work better.

7 AUTONOMOUS NAVIGATION AND CONTROL

The stretch goal we had of developing and implementing the autonomous navigation with the drone did not come to fruition. We planned on a simple algorithm to perform the maneuver of landing on the helipad. That is, assuming the drone was able to see the helipad in the downwards facing camera, the drone would rotate about its vertical axis until the helipad was in the top half of its view and centered. Once this state is achieved and is stable, the drone would then fly forward until the helipad was centered in the bottom camera's view. After maintaining stability in this position, the drone would then start its landing procedure, coming to rest on the helipad. As stated previously, the instability of the drone and the proclivity of the drone to wonder made it unsafe for us to test this method.

7.1 Challenges and Solutions

We formulated multiple solutions for the problems we encountered, but did not have the time to test them. The issues we thought may have contributed to the instability included a poorly-centered center of gravity, uncalibrated or improperly-calibrated sensors, and inaccurate GPS tracking. Solutions to these issues could be to redesign the battery mount to center the mass on the drone, calibrate the sensors more precisely, and purchase a more precise and robust GPS module. The other more complex solution that we could pursue is to use vision-based state estimation instead of GPS-based.

8 MILESTONES

8.1 Milestone 1: Drone Build and Flight

This was the first and the most crucial deliverable for the first phase of the project. While the creation of the model architecture and some of the dataset could be created without the need to fly the drone, gathering a representative dataset and testing our drone could not be done without this milestone completed.

8.2 Milestone 2: Dataset Creation

A critical component of any machine learning model is the dataset. We had multiple final target environments that we could aim for as a goal. The competition that the final drone performs in is over water. If our goal was to perform in this environment, a blue background or a background with water would be ideal. The secondary goal that we decided on was to have our model perform over grass first. The consequences of a failed flight over grass would be less severe than over water. Initially, we only had access to a concrete background to get images from high enough to be representative of the height of the drone. Training on this dataset performed alright, but once we had the drone flying, we were able to transition to datasets with grass as the background, helping improve our models.

8.3 Milestone 3: Helipad Target Detection

Our primary MVP was helipad target detection, which we successfully completed. We developed a representative dataset and trained a YOLOv5 model. Figure 8 demonstrates our model working in both test environments available to us: the offsite Inspiration Lab test flight location and UCSD's Aerodome. Our test flight development consisted of three phases; taking Arducam pictures live during a test flight, taking Arducam pictures live during a test flight and running a static inference while online, and finally, taking Arducam pictures and running an inference live while offline. The model performed exceptionally well in real-time, demonstrating robustness and reliability, which gives us confidence in its performance in competition settings.

9 CONCLUSION

Our team successfully completed the object detection MVP assigned to us by our technical leads at Triton AI for the UAV Replenishment task in the upcoming 2024 RobotX Maritime Challenge. Throughout the project, we assembled the drone hardware, performed test flights, trained our YOLO model, and conducted several flight tests. We achieved running a live and offline inference during our final test flight, with our object detection model successfully able to locate the UAV Replenishment landing pad, as well as the circular symbol in the middle of the helipad for added confidence. Our project stretch goals include tin can detection, object manipulation, and autonomous flight control software. Our OpenCV algorithm for tin can detection is currently in progress, and the mechanical hardware to pick up the tin cans is being developed by another member of Triton AI. Additionally, we successfully configured the ROS2 Humble Docker container and integrated the Jetson Nano computer with the Pixhawk flight controller, providing a basis for the stretch goals of autonomous functionality.

ACKNOWLEDGMENTS

Our team would like to acknowledge the valuable guidance and support from Professor Kastner and the staff at Triton AI. Their mentorship was crucial in helping us navigate the challenges we faced during this project.

REFERENCES

[1] RoboNation, "2022 RobotX Team Handbook," 2024. [Online]. Available: https://robonation.org/app/uploads/sites/2/2022/03/2022-RobotX_Team-Handbook_v2.pdf. [Accessed: 19-Apr-2024].

[2] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788.

[3] G. Jocher, "YOLOv5," 2020. [Online]. Available: https://github.com/ultralytics/yolov5. [Accessed: 13-Jun-2024].

[4] "Real-Time, Cloud-Based Object Detection for Unmanned Aerial Vehicles | IEEE Conference Publication | IEEE Xplore," ieeexplore.ieee.org. https://ieeexplore.ieee.org/document/7926512

[5] A. Gautam, M. Singh, P. B. Sujit, and S. Saripalli, "Autonomous Quadcopter Landing on a Moving Target," Sensors, vol. 22, p. 1116, 2022. [Online]. Available: https://doi.org/10.3390/s22031116. [Accessed: 19-Apr-2024].

[6] S. Shen, Y. Mulgaonkar, N. Michael, and V. Kumar, "Vision-Based State Estimation and Trajectory Control Towards High-Speed Flight with a Quadrotor." Available: https://www.roboticsproceedings.org/rss09/p32.pdf

[7] W. Zeng, Q. Zhang, and Y. Liu, "Vision-Based Buoy Detection and Tracking for Maritime Environments," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 1, pp. 36-46, 2019.

[8] J. Bohg, A. Morales, T. Asfour, and D. Kragic, "Data-Driven Grasp Synthesis—A Survey," *IEEE Transactions on Robotics*, vol. 30, no. 2, pp. 289-309, 2014.

[9] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple Online and Realtime Tracking," in *Proceedings of the IEEE International Conference on Image Processing (ICIP)*, 2016, pp. 3464-3468.

[10] Y. Zhang, L. Wang, and T. Xu, "Adaptive Control for UAVs Operating in Rough Sea Conditions," *Journal of Maritime Operations*, vol. 28, no. 3, pp. 54-68, 2019.

[11] M. Hansen, P. Andersen, and T. Pedersen, "Hybrid Communication Systems for Reliable Data Transmission in Maritime UAV Operations," *IEEE Journal of Oceanic Engineering*, vol. 41, no. 4, pp. 893-901, 2016.