SCOUT: A Collaboration of Ground and Aerial Based Robots Via An Independent, Modular System

Lauren R. Castellon
Department of Mechatronics Engineering
Kennesaw State University
Marietta, Georgia
lcastel1@students.kennesaw.edu

Ruzan Daruwalla
Department of Mechatronics Engineering
Kennesaw State University
Marietta, Georgia
rdaruwal@students.kennesaw.edu

Jarrett-Scott K. Jenny
Department of Mechatronics Engineering
Kennesaw State University
Marietta, Georgia
jjenny@students.kennesaw.edu

Levi Joseph Vande Kerkhoff
Department of Mechatronics Engineering
Kennesaw State University
Marietta, Georgia
lvandeke@students.kennesaw.edu

Charles Koduru
Department of Mechatronics Engineering
Kennesaw State University
Marietta, Georgia
ckoduru@students.kennesaw.edu

***Abstract*—This project addresses a current limitation found in robotics hindering the collaboration between terrestrial and aerial robotic systems. What is being proposed is a modular attachment that allows a terrestrial based robot to transport, charge, deploy, and receive a quad-rotor drone. This specific system combines the Unitree Go1, a quadrupedal robot, with a Parrot Anafi, a quad-rotor UAV, providing a transformative solution to challenges related to coverage, efficiency, and safety. The system also acts as a rechargeable, external power supply for its own components as well as for a Velodyne VLP-16 LiDAR. The project aims to pioneer advancements in robotic collaboration with a focus on modularity. Key objectives included achieving the deployment and recapture of a quad-rotor drone, the integration of a drone charging circuit, and the design of an autonomous drone controller program.**

Index Terms*—*quadrupedal robot, quadrotor UAV, autonomous flight controller, master-slave communication, drone charging cradle, modular

I. Introduction

Collaborative robotics between terrestrial and aerial robots represents an area of science where current solutions fall short of providing complete and efficient outcomes. The motivation for this project was to contribute to this frontier by combining common techniques found in robotics and academia. In this paper, discussion is made on the integration of terrestrial mobility with aerial surveillance; this is accomplished through the design of a modular and reconfigurable system which facilitates the charging, transportation, and landing of a quad-rotor drone.

The chosen terrestrial based robot used in the project is the Unitree Go1 quadrupedal robot. This robot was chosen based upon criteria such as payload capability, programmability, and its battery life. The aerial based robot decided upon for this project was the Parrot Anafi quadrotor drone; the default and robust internal firmware, intelligent "hover" state, and well developed controller software development kit made this drone an easy platform to begin designing algorithms for control. This drone was also easily augmented by external attachments without hindering flight; this was necessary for creating a form factor compatible with a charging cradle.

The goal was to design a multi-robot system that employs concepts in autonomous mapping and robotic collaboration. The project's design objectives include achieving high-level mapping; design of an autonomous drone controller for the Parrot Anafi drone; and integration of a modular drone storage, deployment, and battery charging mechanism. The drone storage and charging device will be referred to as the "drone dock". This integration aims to enhance coverage, efficiency, and safety in a transformative manner.

II. Related Works and Background

A. *Compilation of Existing Concepts*

Many of the constituent concepts of this project are well established in academia and in industry but as previously stated, the goal of this project is to consolidate those to create a novel system. Some examples are autonomous quadrotor drone landing using fiducial markers [1], the charging of drones using specialized modular attachments and landing stations [2-3], vision-based navigation and gait planning for indoor quadrupedal locomotion [4], master slave communication protocols between autonomous parent vehicle and child vehicle(s) [5-6], and simultaneous localization and mapping system via LiDAR [7].

B. *Relating Major Components to Other Works*

The unification of the major parts (the quadrupedal robot, the quadrotor drone, and the microcontroller managed drone charging cradle) was facilitated by the operating system, Robot Operating System (ROS), of the quadrupedal Unitree Go1 robot. The quadrupedal robot acts as the master node which communicates with the drone and the drone cradle. An example of an industrial application of tandem robots can be seen in [8] in which one robot of the pair was used as a master and the other as the slave. In comparison to this application, the drone accomplishes a similar task. In [8], one robot is tasked with keeping a specified distance from the master robot using data acquired from sonar sensors. In our application, the drone is tasked with the alignment of itself with the drone cradle via vision-based camera localization and pose estimation with ArUco marker detection.

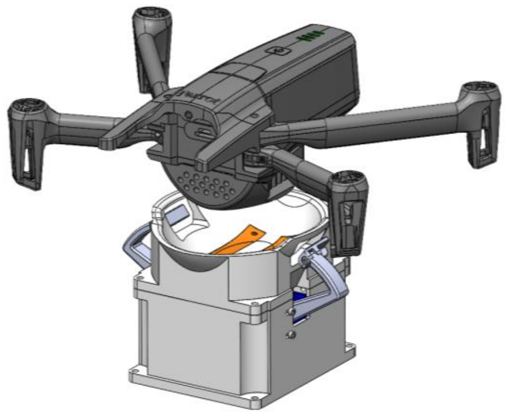


Fig. 1. Drone Charging, Deployment, and Recovery Cradle

An example of exploration and mapping using ground and aerial based robots can be seen in [9]. In this case, the drone used provides constant assistance with mapping and locational positioning within the context of a larger frame. The SCOUT project will use the assistance of a drone in a similar manner for aerial observation and target identification, but in this case the drone is only deployed at the operator’s discretion. It is not used for the pose estimation of the master robot as in [9], but rather for what can be called a "periscope function" as the drone acquires video footage of the surroundings from a higher elevation.

The management of the drone by quadrupedal robot is assisted by a modular addition to the frame of the quadrupedal robot, as shown in Fig. 1. A notable difference can be seen between the charging stations for quadrotor UAVs in [2-3] and the design employed in SCOUT. The former is a passive mechanical format using conical depressions to guide the legs of the drone. The latter is a mechanically actuated design that uses a similarly lofted shape to catch the drone; a key difference lies in the mechanical components which secure the drone via small alignment clamps and embedded neodymium magnets to ensure proper charge and safe transport. This was necessary due to the mobile nature of the project.

Autonomous mapping is also a component of this project to show the successful integration of the Velodyne LiDAR power supply. The data used for Simultaneous Localization and Mapping (SLAM), will be attained using a full 360-degree LiDAR sensor, the Velodyne VLP-16, to sense the quadrupedal robot’s surroundings and determine its position relative to the environment it is travelling in. One SLAM method that is used is called feature extraction, which comprises taking multiple scans while the robot is in motion. After feature extraction, the next steps are LiDAR odometry and LiDAR mapping, which estimate the motion between two scans and create a transformation of select points that will be used to estimate the pose and position. Similar methods are described here, [10].

One of the most critical aspects of autonomous robot collaboration is communication protocols. SSH Port Forwarding is widely adopted for accessing ports and redirecting IP addresses within a local area network (LAN). N. Verma, M. Kashyap and A. Jha established a network utilizing two ports through a router to demonstrate how accessibility can be attained by server and client protocols via TCP connections [6]. Additionally, the paper explains the process of connecting two private LANs to access various ports in each network. The communication between the quadrupedal robot and the drone will adopt similar methodologies outlined in this study, as both robots possess their own private LAN connections. However, the data transfer between the quadrupedal robot and the drone will differ because there will be no external router used to bridge connections between the devices. Instead, we will utilize direct port forwarding to combine these private LANs, maintaining a simple server-client relationship for data transmission from the drone to the quadrupedal robot and vice versa, all within the singular LAN of the quadrupedal robot.

One essential aspect of autonomous robot collaboration is communication and data transfer. SSH Port Forwarding is widely adopted for accessing ports and redirecting IP addresses within a local area network (LAN) or via Wireless Internet (Wi-Fi). Naoto Mizumura and et.al. Found ways to optimize and Time of File Transfer Among Multiple Smartphones using Wi-Fi Direct [6]. Additionally, the paper explains the process of connecting two devices to a central Wi-Fi network to access various ports in each network. Since we are using two robotic devices, the communication between the quadrupedal robot and the drone will adopt similar methodologies outlined in this study, as both devices possess their own private network. However, the data transfer between the quadrupedal robot and the drone will differ because there will be no external router used to bridge connections between the devices. Instead, we will utilize direct port forwarding to combine these private networks, maintaining a simple server-client relationship for data transmission from the drone to the quadrupedal robot and vice versa, all within the singular connection of the quadrupedal robot.

III. Drone Docking System

The docking system, integral to both the drone and the quadrupedal robot, includes multiple components with separate power sources for the Light Detection and Ranging (LiDAR) unit and the drone docking mechanism itself. This system is designed to ensure operational continuity and efficiency during deployment. The entire system includes the LiDAR battery, the dock battery, a power converter, and a relay for controlling the flow of power to the charging system.

The power system starts with power regulation through a DC-DC step-down module, as seen in Fig. 2, which stabilizes the input voltage to a consistent 5 volts. This regulated power is directed to a 5 volts relay system and concurrently branched out to a 16-Channel 12-bit Pulse Width Modulation (PWM) servo driver. The relay system facilitates controlled power delivery to the charging cradle, while the PWM/Servo Driver manages the operation of the servos at the controller’s call. Control and timing of these components, including the activation of charging strips, are orchestrated by an Arduino Uno 8-bit AVR microprocessor via Inter-Integrated Circuit (I^2C) protocol based on the operational demands of the quadrupedal robot. This integrated approach ensures precise management of power distribution and mechanical movement, critical for the system's functionality. If the microcontroller loses power, such as the quadrupedal losing power, the system defaults to an inert state for safety.

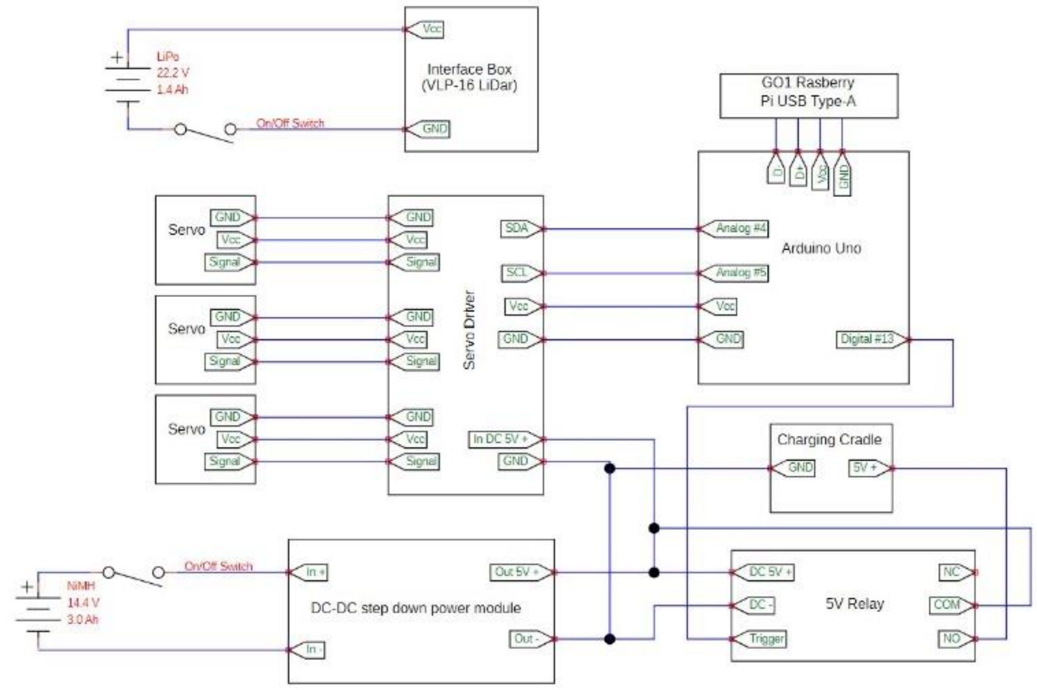


Fig. 2. Modular System Circuit Diagram

The LiDAR's power supply was designed to have a small footprint and sustain operations for ninety minutes, matching two battery cycles of the quadrupedal robot. It requires 1.33 amp-hours based upon an 8 watts average power consumption and a 2 amps start-up current. To meet these needs, we selected Lithium Polymer (Li-Po) batteries due to their higher energy density and flexibility in charging without memory effect. The chosen Li-Po offers 1.4 amp-hours at 22.2 volts with a 130 C discharge capability, supporting up to a 182 amps continuous current output. This current output can be calculated using methods in (1).

$$A = C \times C_{rating} \rightarrow A = 1.4\ \text{Ah} \times 130\ \text{C} = 182\ \text{A} \tag{1}$$

For the drone charging cradle, Nickel-Metal Hydride (NiMH) batteries were chosen for their adaptability, safety, and environmental benefits. They power the dock's servos, the drone's charging system, after regulation, and provide a 5 volts, 5 amps USB-C power output for the drone, requiring 27.78 Watts, as ascertained in (2), after accounting for a 10% efficiency loss from the step-down module.

$$P_{in} = \frac{P_{out}}{\eta} \rightarrow P_{in} = \frac{25\ \text{W}}{0.9} = 27.78\ \text{W} \tag{2}$$

Mechanically, the cradle features a bowl-type structure with magnetic and mechanical clamping systems to assist in precise drone docking and charging. Magnets aligned via servo mechanisms ensure a secure hold, with a 90-degree rotation allowing for quick release. The cradle's magnetic field strength was calculated to avoid interference with electronic components, ensuring safety and reliability.

The docking cradle is designed with an elongated paraboloid shape and a modular component (the jacket) that inversely mimics this geometry, optimizing the alignment over charging contacts and facilitating precise drone placement, as shown in Fig. 3. High-impact resin was selected for rapid prototyping of these components due to its strength and resolution capabilities.

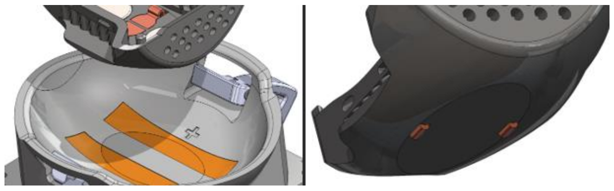


Fig. 3. Closeup Dock Geometry (left), Inversly Matching Jacket (right)

A frictional analysis was conducted to determine the energy required to overcome friction as the drone moves across the cradle's surface. The normal force was calculated at 3.52 Newton, with a friction coefficient of 0.4, resulting in a frictional force of 1.41 Newton, as per (3).

$$F_{friction} = \mu \times N$$

$$N = m \times g \rightarrow N = 0.359\ \text{kg} \times 9.81\ \frac{\text{m}}{\text{s}^2} = 3.52\ \text{N}$$

$$F = \mu \times m \times g \rightarrow$$
$$\rightarrow F = 0.4 \times 0.359\ \text{kg} \times 9.81\ \frac{\text{m}}{\text{s}^2} = 1.41\ \text{N} \tag{3}$$

Thermal management was considered in the design of the drone's jacket, which includes a system to optimize airflow and maintain operating temperatures within safe limits. The jacket allows the drone to cool efficiently, despite an observed temperature increase of 8 to 9 degrees Celsius with the jacket attached. Fig. 4 shows thermal imaging comparing the drone temperature with and without the charging attachment.

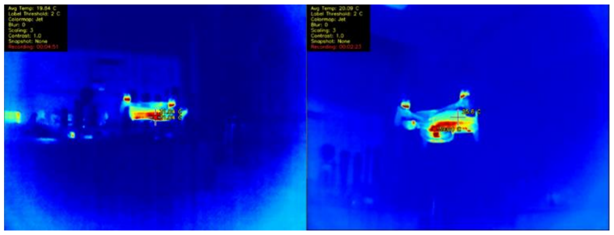


Fig. 4. Thermal Comparison Without Jacket (left) and With (right)

The clamping mechanism features dual servo-driven levers that engage with the drone jacket once positioned, securing the drone for transport. Early tests revealed that magnetic forces alone were insufficient for holding the drone during movement; however, the addition of mechanical clamps ensured stable charging contact and security. Stress and deformation analyses were performed to ensure the integrity of the clamping system under load, confirming the system's robustness and reliability.

Each design aspect, from magnetic alignment to mechanical clamping and thermal management, was optimized to enhance functionality and ensure safety and efficiency in the drone docking process.

To communicate with the drone cradle the quadrupedal utilized a serial connection between the onboard Raspberry Pi and an Arduino controller in the drone cradle setup. The system uses a simple communication protocol to allow the quadrupedal to cycle through the stages of the dock and charging system. The system's operational logic and communication protocols are programmed using Arduino for embedded control and Python 3 for higher-level interfacing.

IV. Drone Control Algorithms

A. Overview

The drone control program consists of multiple threads that manage the following:

1. Requesting and queuing camera frames from the drone gimbal camera
2. Image processing via the OpenCV library to calculate pose estimation relative to an ArUco marker
3. Sending high level movement commands to reposition the drone as necessary

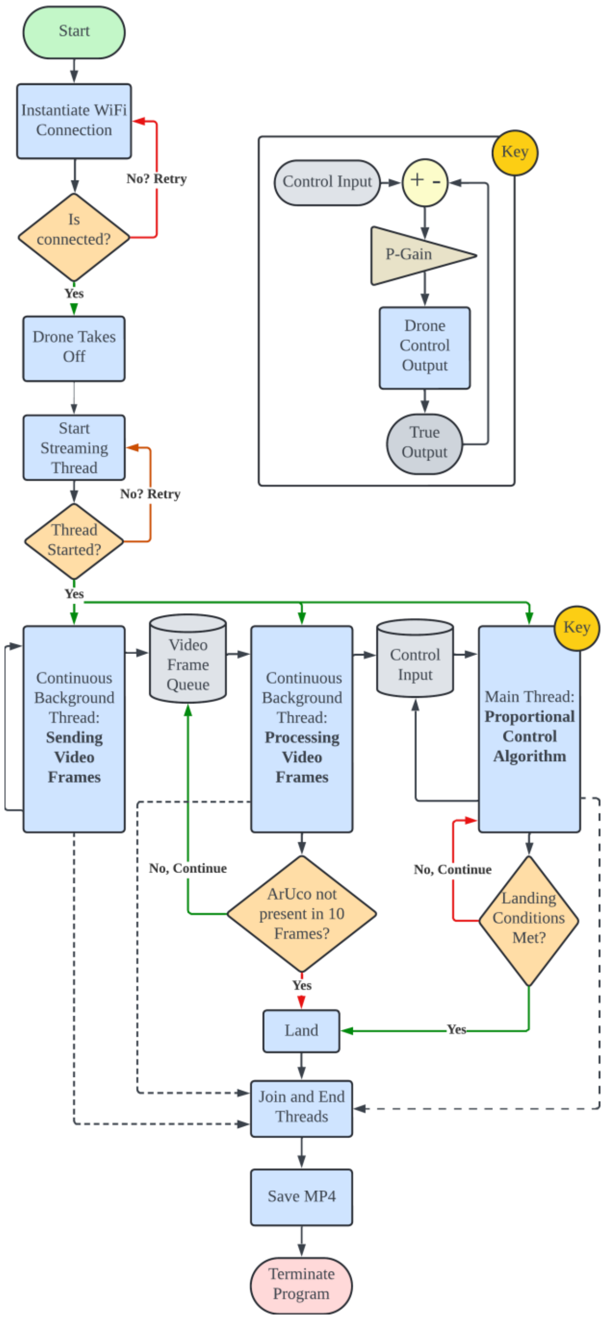


Fig. 5. Autonomous Drone Controller Program

A high-level flowchart of the autonomous drone controller program can be seen in Fig. 5. This program was developed using the Parrot Olympe SDK which provided the programming interface for creating an autonomous controller [11]; this was helpful specifically for connecting by means of Wi-Fi with Parrot Anafi drones. This controller was originally intended to operate on the Raspberry Pi 4B within the quadrupedal robot, but in testing the device was found to lack the processing power necessary for live image processing. No matter how simplified the program was, over-current warnings on the Raspberry Pi kept the program from being completed. The solution to this problem was for the quadrupedal robot to instead outsource the necessary processing power to a host PC via secure shell protocols.

To land the drone accurately within the charging cradle, ArUco pose estimation was used to measure the relative position of the drone to the cradle. The OpenCV Python library was employed to not only measure the relative position, but also to calibrate the drone gimbal camera [12]; without some intrinsic parameters, accurate pose estimation cannot be achieved. For brevity, those parameters will not be discussed in this paper. Fig. 6 shows the checkerboard used for calibration.

Fig. 6. Autonomous Drone Controller Program

B. *Drone Control Algorithm*

In order to accomplish vision-based camera localization with ArUco markers, the OpenCV ArUco library function estimatePoseSingleMarkers() was used for each frame to gain the translational vector and the rotation matrix, *R*, that described the current position relative to the marker [12]. The translational vector contained the x, y, and z position of the center of the marker relative to the center of camera frame in meters. The rotation matrix was used to measure the current yaw value from the marker x-axis and was calculated as follows in (4):

$$\text{yaw} = \begin{cases} 0, & if \sqrt{{R_{11}}^2 + {R_{21}}^2} = 0 \\ \arctan\left(\frac{R_{21}}{R_{11}}\right), & Otherwise \end{cases} \tag{4}$$

The yaw value measures the angle between the drone camera orientation and the ArUco marker orientation within the common horizontal plane. The x, y, z, and yaw values are used to correct the pose of the drone relative to the drone dock and are passed through a control algorithm that can be seen in Fig. 7. The x, y, z, and yaw values returned by the optical pose estimation are used to generate control signals for roll, pitch, yaw and thrust that are then passed into the control algorithm. Because the commands in the Olympe SDK are at a high level, the control algorithm is somewhat simple and is based primarily upon conditionals. The hierarchal P-Control program is outlined in a flow chart in Fig. 8.

The control algorithm starts with broad tolerances that control large movements and eventually focuses on small errors and returns finer control commands. The program first evaluates the x and y error and corrects these errors as necessary. If the larger x and y tolerances are satisfied, the next criterion evaluated is yaw. After yaw, the height of the drone is evaluated. This is assessed after the larger x and y bounds to ensure that as the drone lowers, the ArUco marker will not pass out of the frame. Once the z error is satisfactory, the finer x and y tolerance is evaluated and corrected. Once all of the criteria are met, these criteria must remain satisfied for 0.75 seconds to ensure that the drone is not passing over the target location while undergoing overshoot. If this were the case, once the land command was called, the drone would maintain its horizontal velocity and shoot past the target landing location during descent. Each movement command input is adjusted by a proportional gain value to assist in reducing chances of overshoot.

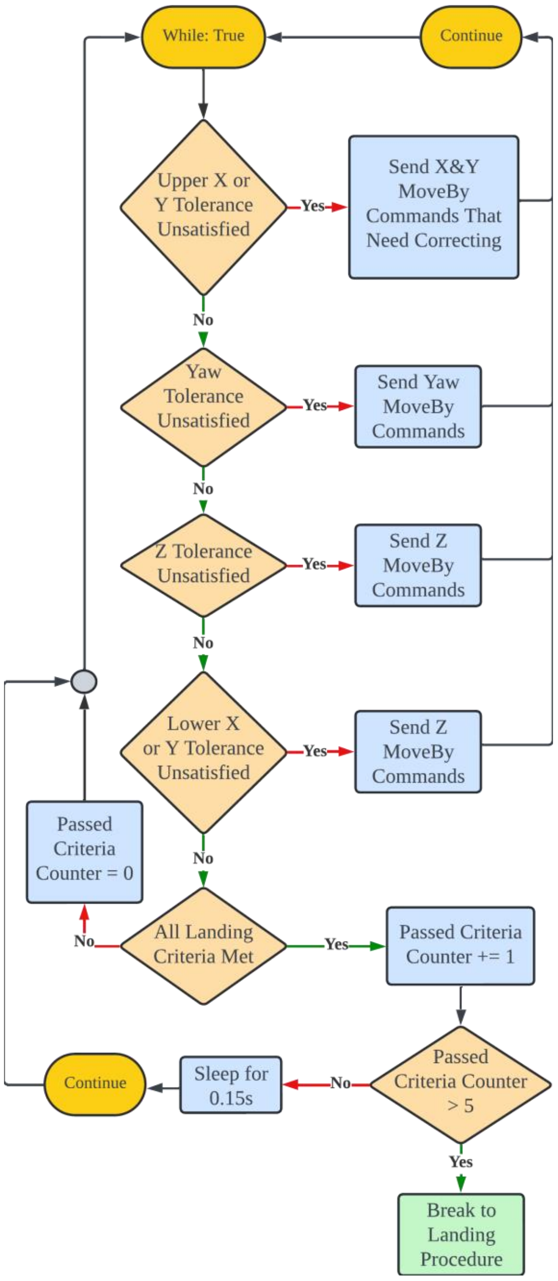

While: True
Continue
Upper X or Y Tolerance Unsatisfied
Yes
Send X&Y MoveBy Commands That Need Correcting
No
Yaw Tolerance Unsatisfied
Send Yaw MoveBy Commands
Z Tolerance Unsatisfied
Send Z MoveBy Commands
Lower X or Y Tolerance Unsatisfied
Send Z MoveBy Commands
Passed Criteria Counter = 0
All Landing Criteria Met
Passed Criteria Counter += 1
Passed Criteria Counter > 5
Sleep for 0.15s
Continue
Break to Landing Procedure

Fig. 7. P-Control Conditionals

Although the average flight duration was suboptimal, the flight controller yielded approximately 95% successful landings in the charging cradle. The geometry of the charging cradle allowed for error of ±2 centimeters in the x direction, longitudinally, and ±1 centimeter in the y direction, laterally. A graph showing the average flight path can be seen in Fig. 8.

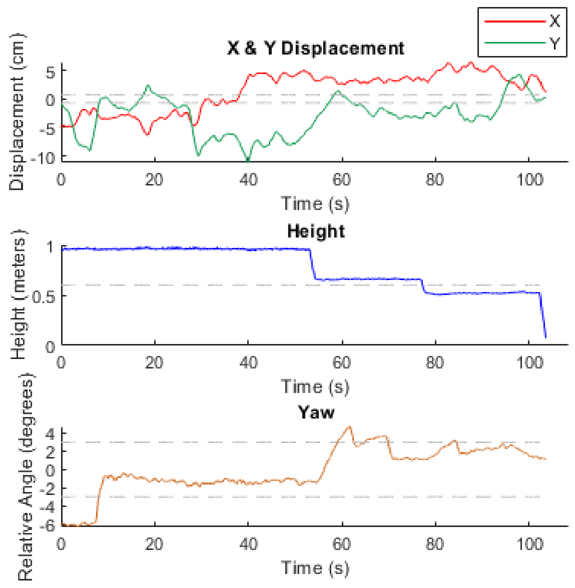


Fig. 8. Flight Path Example (Dashed Lines Show Criteria For Landing)

C. *Limitations and Further Improvements*

The high-level Olympe landing command must be called due to unchangeable safety parameters within the firmware of the Anafi drone. Once this command is called, the drone enters a "landing state" wherein it begins an autonomously controlled descent towards the ground and judges current progress via a downward facing ultrasonic sensor. While in the normal "flight state", the height of the drone cannot pass below 35 centimeters; if the drone does pass beneath this threshold, the internal firmware increases the thrust of the drone to return it to its minimum height. This prohibited the development of a controller that landed the drone by reducing thrust while adjusting x, y, and yaw. While this methodology would have been preferred to attain even higher landing accuracies, all landings needed to be completed through the existing high-level command.

Other constraints experienced in programming the autonomous drone controller included a lack of precision with the high-level movement commands for the x and y displacements. A cause for this could be linked to noisy accelerometer readings due to the vibrations of the drone in the air and the unavoidable horizontal drift that drones experience while in flight. Another possible reason could be linked to inaccuracies of the optical flow algorithm used by the internal flight controller. When making precision horizontal adjustments at the order of 1 centimeter, the average velocity of the drone is low, less than 0.1 meters per second; this would cause the differences between frames to be quite insignificant leading to inaccurate results. In contrast, the z and yaw displacements could be accurately measured by the downward facing ultrasonic sensor and the internal gyroscope, respectively. For this reason, it was found in experimentation that the drone was better suited for outdoor applications where fine movements are not necessary. For example, if a command to move +1 meter in the X direction is sent, the final location will be accurate ±5 centimeters. As the movement command parameters become smaller (i.e., 5 centimeters), the outputs of these parameters become smaller also at a nonlinear rate. This lack of precision caused each landing to last roughly 6 minutes on average, which is far from optimal.

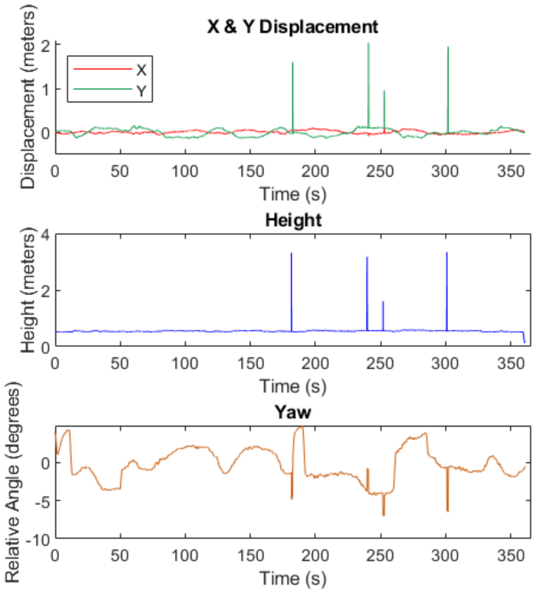
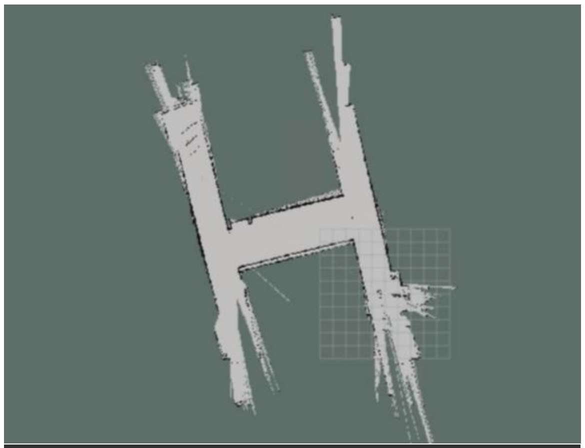


Fig. 9. Noise From ArUco Pose Estimation

The long duration of the landing procedure is also caused by noisy data returned by the ArUco pose estimation. The noise is characterized by random spikes which can be seen in Fig. 9. These erroneous values in the data cause the drone to move unpredictably and occasionally caused landing procedures to fail. To solve this issue, a rudimentary saturation filter was applied to the X and Y inputs; this made it possible for the X and Y positions returned by the pose estimation to be used as control inputs. The saturation limits were determined by the maximum distance that the drone could travel while retaining the ArUco marker in the camera frame at the maximum allowed height of flight indoors of 1.5 meters; this maximum indoor flight height was a parameter determined in the autonomous controller. This saturation filter improved the flight performance greatly.

V. Integration of Major Parts and the Inclusion of Hector SLAM

To show the successful integration of the LiDAR power supply and the modular structure that supports the Velodyne VLP-16 on the chassis of the Unitree Go1, the ROS Melodic Hector SLAM package was installed in the Unitree Go1 ROS environment. This is the same package discussed in [13]. Fig. 10 shows a hallway map that was generated using the Hector SLAM package and Fig. 11 shows the 3D point cloud generated by the LiDAR sensor in Rviz.



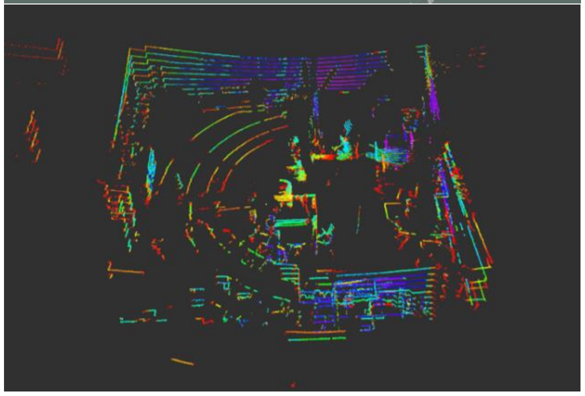


Fig. 10. Implementation of Hector SLAM (Above)
Fig. 11. 3D Point Cloud (Below)

To show the integration of all the major components within the system, a simple obstacle avoidance algorithm was written that autonomously controlled the motion of the Go1 in the forward direction. This algorithm was run on the same host PC that the drone controller software was outsourced to and used the same SSH protocols. It acted in a similar manner as a controller to the Go1. A program was also run on the Go1 that wrote conditions based on the depth of objects from the sensor to a text file on the host PC; the file on the host PC acted as a discussion space between the two programs where a Boolean value based on the forward scans was published. The program on the Go1 only accesses a 2D-scan topic published within the Go1 ROS environment and those values were evaluated natively. At the start of the program, the Go1 walks forward until an obstacle is met. When the Go1 encounters an obstacle in its path, the commands to move forward sent from the host PC cease, the Go1 stops, and then launches the drone from the drone dock. The release of the drone from the drone dock was accomplished using serial communication with the Arduino Uno by the quadrupedal robot as aforementioned. The command to lock the cradle is sent from the host PC to the quadruped's microcontroller to trigger the drone launch. At the completion of the drone's actions and upon successful landing, the drone dock is locked and charging resumes.

The control of the quadruped, particularly its walking patterns and the decision to launch the drone, hinges on clear and blocked conditions. To address challenges posed using different Python versions across the system's components, a unique solution involving a temporary file on the host PC was implemented. This file allows one of the quadruped's onboard microcontrollers to read the navigation conditions from a filesystem that is external to ROS, thereby facilitating communication between components operating on different and slightly incompatible versions of ROS. The coordination between the quadruped and the drone is managed through high-level control software that operates over the ROS network. This software includes a custom node designed to synchronize commands between the systems running Python 2.7 and Python 3.7, ensuring seamless operation despite the disparities in software environments. The use of environment variables plays a critical role in bridging the gap between these different systems. For direct interaction and control, a Python script runs on a Raspberry Pi aboard the quadruped. This script is responsible for processing navigation and LiDAR data written by ROS and triggers the drone's launch sequence when an unnavigable obstacle is detected. To maintain system compatibility, SSH is used to connect to a secondary Raspberry Pi within the Go1 configured with Python 2.7, thus creating a coherent environment while the master control operates on Python 3.7.

The scripts demonstrate a complex method for managing an integrated robotic system that includes both a drone and a LiDAR-equipped quadrupedal robot. It employs a variety of libraries and technologies such as SSH for executing commands remotely, ROS for overall robotics management, and a custom module (robot_interface), which served to facilitate specific interactions with the hardware.

At the core of the script is the robot_interface designed to interface directly with the hardware components of the system. A custom module was implemented as a critical component in controlling drone functionalities, representing a customized strategy for managing aerial operations. The script utilizes the Python subprocess module to execute shell commands, enabling the remote execution of scripts across different machines. This was a critical feature for synchronizing actions between the ground-based robot and the aerial drone. Additionally, rospy, a Python client library for ROS, allows the script to integrate seamlessly with ROS functionalities, enhancing its capability to manage complex robotic operations.

Functionally, the script is divided into several key operations. Functions such as takeOff, waitLand, and land; all of which are managed through SSH, sending instructions remotely to control the drone's flight to the host PC. The LiDAR function was pivotal for environment sensing, executing a remote script that checks LiDAR readings and adjusts operational flow based on the detection of obstacles, thereby implementing real-time environmental mapping and obstacle avoidance. A script was also written to actively monitor sensor inputs, such as LiDAR data, to dynamically adjust the movement commands of the quadrupedal robot, ensuring it navigates safely by responding quickly to the proximity of obstacles. The networking script employs User Datagram Protocol (UDP) for fast and efficient packet transmission between the controller and the robot, which is essential for maintaining real-time communication across the components.

Additionally, a robust error management script is embedded within the script to handle potential issues during subprocess executions, ensuring reliable delivery of commands, even in complex nested SSH sessions.

The integration strategy of this system was designed to synchronize the functionalities of the drone and the quadrupedal, enabling them to perform coordinated tasks that are crucial in scenarios like surveillance or search and rescue operations. The real-time data from the LiDAR sensors directly informs and adjusts the quadrupedal robot's navigation paths, facilitating autonomous obstacle navigation. These scripts demonstrate a high-level application of multiple advanced technologies, creating a responsive and integrated robotic system capable of conducting complex operations such as simultaneous aerial and ground surveillance, dynamic environmental mapping, and adept obstacle navigation. This elaborate use of ROS, coupled with custom hardware interface libraries and direct hardware control commands via UDP, underscores the script's potential in cutting-edge robotics projects or advanced research endeavors.

VI. Conclusion

This prototype system demonstrates an effective use of different environments to manage a complex interaction between a quadrupedal robot and a drone, achieving a high degree of automation and coordination. The project successfully integrates advanced robotics, autonomous navigation, operational synchronization, and innovative scripting solutions to overcome technical challenges, setting a strong foundation for future enhancements and applications.

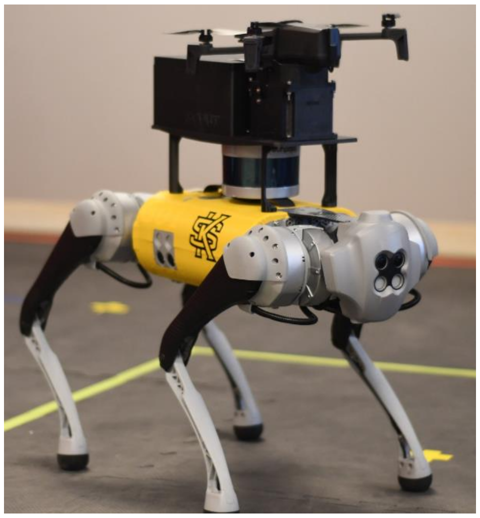


Fig. 12. Complete System

References

[1] M. F. Sani and G. Karimian, "Automatic navigation and landing of an indoor AR. drone quadrotor using ArUco marker and inertial sensors," 2017 International Conference on Computer and Drone Applications (IConDA), Kuching, Malaysia, 2017, pp. 102-107.

[2] F. Cocchioni, E. Frontoni, G. Ippoliti, S. Longhi, A. Mancini, and P. Zingaretti, "Visual based landing for an unmanned quadrotor," Journal of Intelligent & Robotic Systems, vol. 84, pp. 511-528, 2016.

[3] F. Cocchioni, V. Pierfelice, A. Benini, A. Mancini, E. Frontoni, P. Zingaretti, et al., "Unmanned ground and aerial vehicles in extended range indoor and outdoor missions," in Unmanned Aircraft Systems (ICUAS), 2014 International Conference on, 2014, pp. 374-382.

[4] Narayanaswamy, Nara & Kanehiro, Fumio. (2024). Vision-Based Software System for Indoor Quadrupedal Locomotion: Integrated with SLAM, Foothold Planning, and Multimodal Gait. 1330-1335. 10.1109/SII58957.2024.10417432.

[5] Aswin, Lakshmi & Rajasekaran, Prasanth & Radhakrishnan, Santhosh & Kumar, Shivaramakrishnan. (2013). Design and Structural Analysis for an Autonomous UAV System Consisting of Slave MAVs with Obstacle Detection Capability Guided by a Master UAV Using Swarm Control. International Journal of Scientific and Engineering Research. 4. 1. 10.6084/m9.figshare.1132189.

[6] N. Verma, M. kashyap and A. Jha, "Extending Port Forwarding Concept to IOT," 2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN), Greater Noida, India, 2018, pp. 37-42, doi: 10.1109/ICACCCN.2018.8748430.

[7] M. Zeybek, "Indoor Mapping and Positioning Applications of Hand-Held LiDAR Simultaneous Localization and Mapping (SLAM) Systems," Turkish Journal of LiDar, vol. 3, pp. 7-16, 2021.

[8] G. Li, R. Lin, M. Li, R. Sun, and S. Piao, "A Master-Slave Separate Parallel Intelligent Mobile Robot Used for Autonomous Pallet Transportation," Applied Sciences, vol. 9, no. 3, pp. 368. January, 2019.

[9] D. Chatziparaschis, M.G. Lagoudakis; P. Partsinevelos, "Aerial and Ground Robot Collaboration for Autonomous Mapping in Search and Rescue Missions," Drones, vol. 4, pp. 79, 2020, https://doi.org/10.3390/drones4040079

[10] L. Huang, "Review on LiDAR-based SLAM Techniques," 2021 International Conference on Signal Processing and Machine Learning (CONF-SPML), Stanford, CA, USA, 2021, pp. 163-168, doi: 10.1109/CONF-SPML54095.2021.00040.

[11] *Olympe SDK*. (7.7), Parrot. Accessed: April, 2024. Available: https://developer.parrot.com/docs/olympe/index.html

[12] *Open Source Computer Vision, OpenCV-ArUco*. (4.10.0), OpenCV. Accessed: April, 2024. Available:https://docs.opencv.org/4.x/index.html

[13] S. Kohlbrecher, O. von Stryk, J. Meyer and U. Klingauf, "A flexible and scalable SLAM system with full 3D motion estimation," 2011 IEEE International Symposium on Safety, Security, and Rescue Robotics, Kyoto, Japan, 2011, pp. 155-160, doi: 10.1109/SSRR.2011.6106777.