Autonomous Follower Drone

Andy V, Hugh F, David P, Lawrence D, and Mahdi Maaref

***Abstract* - Our project, Follow the Leader aims to allow a follower drone to follow a leader drone autonomously. The idea behind this is to learn more about the methods used to interact between a leader and follower. The way that we are implementing this is via a flight control program written in Node.js that utilizes PID controllers, GPS modules on both the leader and follower drone. Furthermore, we used HTML, javascript, and Express to implement a GUI that displays real time graphs of the x,y and z coordinates of the drones and allows the user to modify certain parameters of the flight. We have successfully made the Parrot AR Drone 2.0 follow a RaspberryPi that is attached to the Leader drone. We were able to implement a Node.js script that would calculate the location of the RaspberryPi, and make the Follower drone spin until it faces it. It would then move forward towards the Leader drone until it reaches the distance threshold set by the user.**

I. INTRODUCTION

Follow the Leader involves creating a drone that is able to autonomously follow another drone. The drone must stay within a set threshold distance from the leader drone.

We used the Parrot AR Drone 2.0 to act as our Follower drone. We also used a RaspberryPi with a GPS module; this RaspberryPi is attached to the Leader drone to collect GPS data. The RaspberryPi also runs the PID controller that will autonomously control the Follower Drone. We adjusted the drone's elevation and location in terms of the RaspberryPi elevation and location. The RaspberryPi first calculates the Follower Drone's heading (direction of the Leader Drone). After calculating the correct heading, the RaspberryPi will send commands so that the Follower drone will spin towards the Leader drone. After it faces the Leader drone, the RaspberryPi made the drone move forward until it was within a certain distance from the Leader drone.

We conducted tests by having someone carry the RaspberryPi and then running our PID controller. We would wait for the Follower drone to calibrate, and calculate the heading. Once it calculated the heading, the drone would spin towards the RaspberryPi and adjust its elevation. After facing the RaspberryPi, the Follower drone would begin flying towards the RaspberryPi. From these results, we were able to conclude that our Follower drone was able to successfully follow the Leader drone. Some of the tests showed inaccuracies; these inaccuracies are due to the Follower drone's compass, as at times, it would have an incorrect measuring. when compared to an actual compass. Some of the inaccuracies is also due to strong winds, as it would alter the flight path of the Follower drone.

We conducted further tests, but instead of holding the RaspberryPi, we attached the RaspberryPi to another drone. The test would be the same, with the exception of the use of a drone to move the RaspberryPi. These tests showed great inaccuracies, as the drone carrying the RaspberryPi emitted another Wifi connection. This Wifi connection would interrupt the RaspberryPi's connection with our base station, causing the flight controller to be severely inaccurate.

By using GPS data from both the RaspberryPi and the Follower drone, and a PID controller, the Follower drone was able to successfully follow the Leader drone with only slight inaccuracies due to faulty compass data. These inaccuracies can be solved with a better compass for our Follower drone.

II. SOFTWARE

A. GUI

The GUI is implemented in HTML, Node.js, and Express. As seen in Figure 1, Real-time graphs were made with Smoothie.js and represent the elevation (Z-coordinate), X-coordinate, and Y-coordinate respectively. Two lines are shown in each graph to show the position (elevation, X-coordinate, and Y-coordinate) of the Follower drone and the Leader drone. The distance between the lines represent the recorded distance between the Follower drone and the Leader drone.

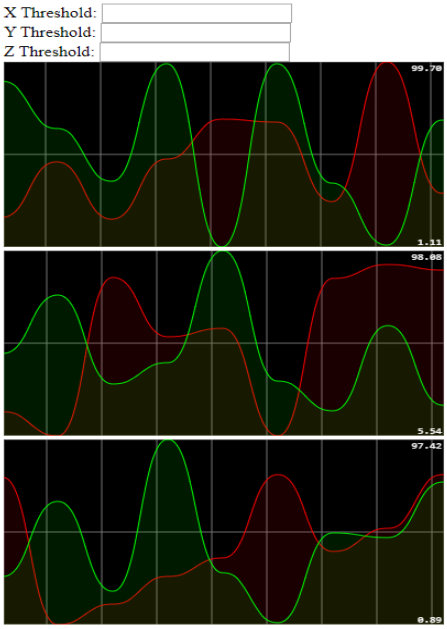

Fig. 1. GUI that contains real-time graphs and X, Y, Z distance thresholds. The graphs represent elevation, X-coordinate, and Y-coordinate, from top to bottom.

Above the graphs are text boxes where the user can input a numeric value. These values represent the distance threshold in between the Leader Drone and the Follower drone. After inputting the values, the distance threshold for the drones change.

III. HARDWARE

A. Connection between drone, laptop, and RaspberryPi

In order for the drone to communicate with the laptop, we have a laptop and the RaspberryPi connected to the drone's wifi signal. From there, we are able to ssh the laptop to the RaspberryPi and send Node.js programs to our drone. Once connected to the drone, we simply run the code on the RaspberryPi code with our flight controller program and it will run it on the drone. We are using a library that allows the drone to do simple commands such as takeoff(), forward(), clockwise(),and counterclockwise() in order to have the drone fly forwards and rotate.

The RaspberryPi has a GPS module, wifi antenna, and ultrasonic sensor. The RaspberryPi will record the Leader drone's GPS coordinates through the GPS module. With this GPS data, the RaspberryPi will connect to the Follower drone's wifi via the wifi antenna and run Node.js scripts that will calculate the correct heading for the Follower drone. In order to adjust elevation, the ultrasonic sensor will record the Leader drone's altitude and compare it to the altitude data from the Follower drone. It will again run Node.js scripts to adjust the height of the Follower drone.

B. Flight Command

We are able to send basic commands to our drone such as forwards, backwards, spin right, and spin left. On top of that we are able to increase and decrease altitude, and change altitude. From these commands, it works in tandem with the GPS module that we have attached to the drone and we can send the drone to a certain elevation based on constantly checking the elevation from the GPS module.

C. Approach

Our method for implementing the autonomous portion of the follower drone was to calculate angles and distance based on GPS coordinates of each respective drone. This is possible using the drone seen in Figure 2. We would then use Node.js scripts in order to move the Follower drone so that it would go towards the Leader drone.

Fig. 2. Parrot AR Drone 2.0 used as the Follower drone. This drone has a GPS module attached and provided a wifi connection for the RaspberryPi and laptop.

Our program continuously feeds elevation, latitude and longitude of both the follower and leader drone. The cardinal direction (represented as an angle) from the front of the Follower drone to Leader drone is then calculated using the GPS coordinates. With this calculation, the Follower drone continuously compares this angle with its own heading. The Follower drone spins right or left depending on which way will faster turn towards the Leader drone. Once the Follower drone's heading is similar to the cardinal direction of the Leader drone (within 3 degrees), it stops turning. The Follower drone then moves forward and adjusts its elevation until it reaches the distance threshold. By implementing it this way, the drone can accurately follow the Leader drone.

IV. SUMMARY AND CONCLUSIONS

We were successfully able to make a Follower drone, the Parrot AR Drone 2.0, follow a Leader drone. Doing so involved attaching a RaspberryPi to our Leader drone. This

RaspberryPi has a GPS module to track the Leader drone's GPS coordinates, an ultrasonic sensor to record the Leader drone's altitude, and a wifi antenna to connect to the Follower drone's wifi. The RaspberryPi also has the Node.js script that controls the Follower drone's movement. By connecting a laptop to the Follower drone's wifi, we are able to ssh into the RaspberryPi and run the Node.js script. The RaspberryPi will then use the GPS data gathered from the Follower drone, as well as the GPS data from its GPS module to calculate the direction the Follower drone should face. It then calls commands to spin the Follower drone towards the Leader drone. The RaspberryPi then moves the Follower drone forward until it reaches a distance threshold.

The results of our tests showed that the Follower drone was capable of following the Leader drone with only slight inaccuracies. These inaccuracies were apparent, as sometimes the Follower drone would not be facing directly at the Leader drone, so when it moved forward, it was not meeting the distance threshold. This inaccuracy is due to the compass in the Follower drone; a more accurate compass would solve this inaccuracy. Further improvements for the project would be to implement object detection so that the Follower drone will be able to move around obstacles between it and the Leader drone. Modifying the Node.js scripts to stabilize the drone's flying against strong winds would also improve the project, as the wind was also a factor in creating inaccuracies with our tests.

V. ACKNOWLEDGMENTS

Andy Vo, Hugh Fong, David Phan, and Lawrence Dizon thank Mahdi Maaref being our advisor and leading our team to making substantial progress in the Follower the Leader project. We thank the U.S Navy for supplying us with a drone as well as additional funding to purchase an additional drone and GPS modules.

REFERENCES

[1] IEEE Standard for Information technology-- Local and metropolitan area networks-- Specific requirements-- Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications , IEEE Standard 802.11n, 2009.

[2] J. Li, Y. Li, "Dynamic Analysis and PID Control for a Quadrotor", *IEEE International Conference on Mechatronics and Automation*, pp. 573-578, 2011.

[3] C. Maple, "Security and privacy in the internet of things," Journal of Cyber Policy, vol. 2, no. 2, pp. 155–184, Apr. 2017.

[4] V. M. Babu, K. Das and S. Kumar, "Designing of self tuning PID controller for AR drone quadrotor," *2017 18th International Conference on Advanced Robotics (ICAR)*, Hong Kong, 2017, pp. 167-172.

Andy Vo was born in San Diego, California in 1998. He received the B.S. degree in computer science and engineering from the University of California, Irvine in 2020. In the summer of 2019, he was a software engineer intern in the Naval Research Enterprise Internship Program at NIWC.

Hugh Fong received the B.S degree in computer engineering from the University of California Irvine, California in 2020.

David Phan was born in San Diego, California in 1998. He received the B.S degree in computer engineering from the University of California Irvine, California in 2020. He has an interest in going into software engineering in the future, specifically dealing with databases or mobile development.

Lawrence Dizon was born in San Diego, California in 1995. He received the B.S degree in computer science and engineering from the University of California, Irvine, California in 2020. His interests are in the software development side, more focused toward mobile development.