CS Capstone Winter Progress Report

March 22, 2019

American Helicopter Society Micro Air Vehicle Competition

Prepared for

Potentially Columbia Helicopters

Nancy Squires _____ Signature _____ Date

Prepared by

Group 14

Beaver Hawks

Anton Synytsia _____ Signature _____ Date

Matthew Phillips _____ Signature _____ Date

Shanmukh Challa _____ Signature _____ Date

Nathan Tan _____ Signature _____ Date

Abstract

This progress report outlines the overall functionalities of OSU's Micro-Air Vehicle (MAV) system. It outlines the devices that are installed on the MAV, and how the data is received, organized, and displayed on the user interface built by our team. Each component for the project is explained in depth, starting with the Graphical User Interface (GUI), WiFi data transmitter, camera feeds, and sensor data. The web application currently contains all necessary functionality, and the product is ready for pilot consumption. Each facet of the MAV's software is covered in this document.

Contents

List of Figures

1 Project Purpose and Goals

The Vertical Flight Society (VFS) is the world's premiere international technical society for engineers, scientists and others working to advance vertical flight. VFS brings together industry, academia and government to face difficult challenges in vertical flight. Each year, VFS hosts the Micro Air Vehicle challenge to help prepare and attract the next generation of scientist and engineers to work in the field of vertical flight.

The goal of this project is to collaborate closely with the mechanical engineering and electrical engineering subteams in building a micro air vehicle to accomplish the mission presented by the Vertical Flight Society.

2 Current Progress

2.1 Overview

Our team has been meeting consistently throughout the term and made great progress in completing the project. The Electrical Engineering team assisted in installing the cameras and sensors on the helicopter, and the computer science team developed a web application which displays real-time data transmitted from the helicopter, such as the front-facing camera, bottom-facing camera, depth map, attitude, and sensor data.

All the core functionality is available on our user interface. The Electrical Engineering team has installed our bottom facing camera and we are able to display live video feeds through our application. While the web application is ready for effective use, there is some delay from the Electrical Engineering team with installing all the sensors. However, they have installed all necessary sensors to complete the obstacle course in the helicopter's present state.

The helicopter has gone through a couple iterations on its frame, but now it has been altered into a triangular prism shape to help reduce the weight (as shown in figure). As far as the way this affects the CS team, our biggest concern has been solved. The angling of the ultrasonic sensors for our calculations is optimized if we decide to pursue our stretch goal of implementing collision avoidance.

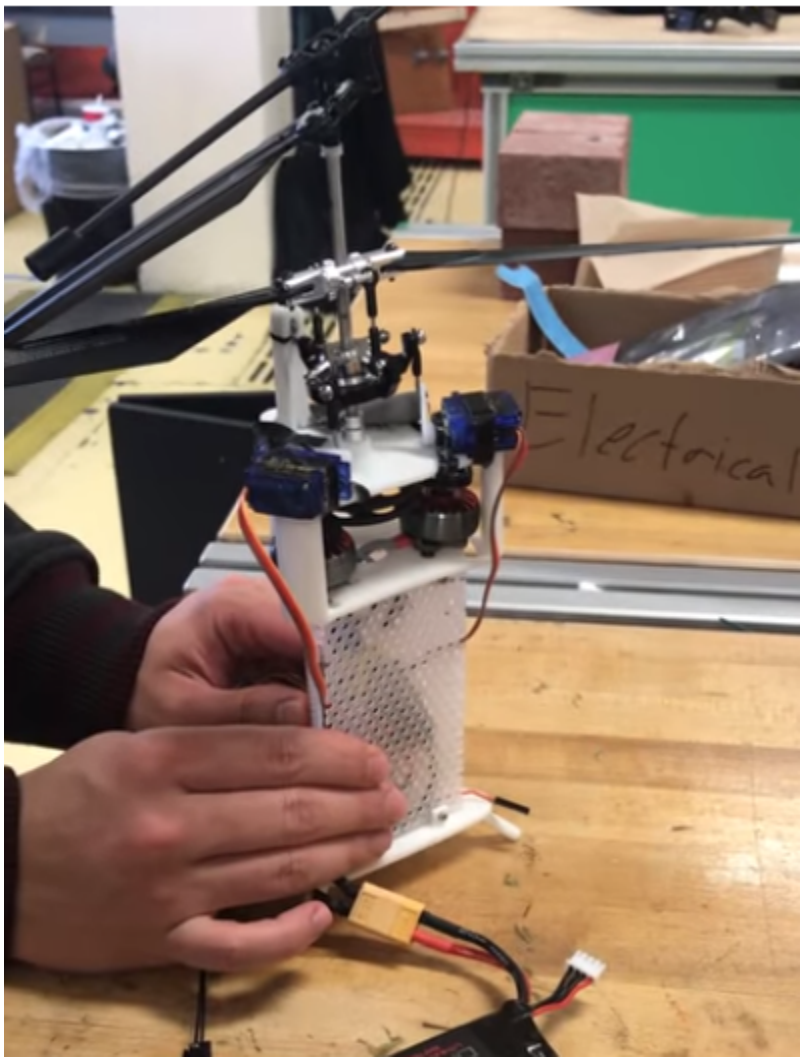

Figure 1: Helicopter Body

2.2 Design Summary

Reflecting on the design, we have all necessary portions of the user interface completed. We initially setup a NodeJS server for the user interface. The server is a prerequisite for all the design requirements. We then created a very low fidelity layout for the main page, consisting of HTML and CSS. Having the layout, we then focused on developing GUI elements for the flight instruments, porting the camera feed, creating a data recording mechanism, and developing a portion of the code on the Raspberry Pi.

2.3 Collision Warning GUI

There are two flight instruments that involve GUI. They are collision warning and attitude indicator. The collision warning instrument displays three sensor ranges in a color-coded format. The floating point sensor, which range from 6 to 254 inches, are converted to their appropriate colors and applied to their associated collision warning GUI elements. The colors transition from green being the farthest, to yellow being the intermediate, to red being the closest, with example states displayed in figures 2, 3, and 4. The three ranges are fed in by live data transmission coming from the sensors on the

helicopter. While not all sensors are installed by the Electrical Engineering team, the infrastructure of our network is set up to include more sensors with ease. For now, we are transmitting data from one sensor.

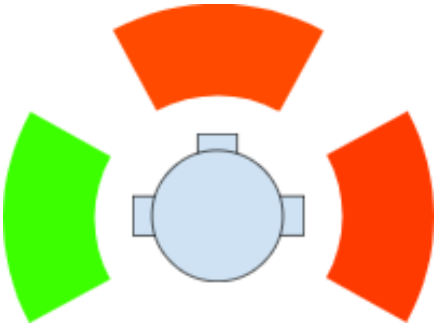

Figure 2: Collision Warning A

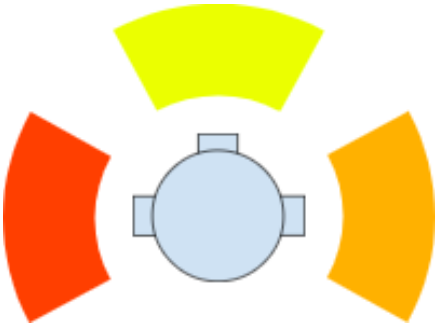

Figure 3: Collision Warning B

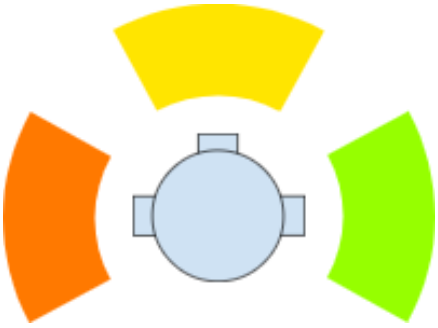

Figure 4: Collision Warning C

2.4 Wifi Data Transmitter

We developed a wireless data transmitter to communicate information between the Raspberry Pi and the server. Our wireless data transmitter is based on the User Datagram Protocol (UDP). UDP is used for sending information from the Raspbian, which is based on the the Raspberry Pi, and to the server, which is based on the pilot's laptop. The information includes all the ultrasonic sensor data and gyroscope angles.

Getting into the details of our wireless data transmitter, the way it works is that the IP addresses, for transmitting the packets across, are not hard-coded. We actually developed a functionality that allows the server and the Raspberry Pi to familiarize with each others IP addresses. When the server is started, it multi-casts a signal to a specific multi-cast address and port. At the same time, the server also starts a listener coming to any of its interfaces at the pre-defined port. When the Raspberry Pi is turned on, a boot-up script launches a multi-cast receiver, which listens for the messages coming to the pre-defined multi-cast address and port. Upon receiving a signal at the pre-defined multi-cast port and address, the Raspberry Pi extracts the IP address of the interface that sent the packet, which presumably is the IP address of the server, and then sends a response to the extracted IP address, at a pre-defined port number. Upon receiving a response at the pre-defined, the server extracts the sender IP address, which is presumably the IP address of the Raspberry Pi, and establishes an IP-based communication, for sending data back and forth. When we say, "presumably", we actually have two predefined messages: "BeaverHawks-1", and "BeaverHawks-2" that we check the received messages against, just as a weak barrier against a potential hacker. We call the set of steps for identifying the IP addresses a handshake and for the handshake we use UDP. This all works provided the Raspberry Pi and the server are connected to the same Wifi network.

Speaking of networks, the school's routers do not forward our multi-cast signals, even though we are able to connect both, the server and the Raspberry Pi, to the same Wifi network. For that reason, and for portability, in general, we got an ASUS router, which provides us with our own network. We also got a wireless access point to potentially extend the range of our network. For the compatibility of our handshake system, we attempted to configure the ASUS router to permit multi-cast signals; however, we were not successful. To resolve the issue, we simply updated our handshake system to send and listen for broadcast signals. Broadcasting is very similar to multi-casting but a less accepted way of networking.

Communicating data between the Raspberry Pi and the server was actually a challenge. We first had to determine how to send and receive data. We figured using UDP would be an appropriate way to go, due to its minimal packet overhead and delivery properties, which minimizes latency. An alternative way to go was to setup an actual online database where we would upload the data from one end and acquire the data from the other end. This would have given us a three-way communication, which also comes with loading latency, storing latency, and reliability overhead. We quickly realized this approach would not be suitable. So we went with the UDP transmitter, which only requires communication between two devices, with packets forwarded by the router and an access point.

A second challenge involved actually implementing a UDP transmitter. Because our server uses JavaScript, we were initially skeptical about having to use two, different programming languages for communicating data on both ends. We thought we would need to start an additional JavaScript server, on the Raspberry Pi; however, it turns out, JavaScript UDP library, *dgram*, is compatible with Python UDP library, *socket*. Because of the compatibility, we decided to use Python on the Raspberry Pi for both, sending and receiving, UDP messages.

For reliable communication, which we may require for our stretch goal, we also attempted to implement a TCP transmitter. Up until this point, we were not able to get Python TCP library, *socket*, to communicate with JavaScript TCP library, *net*. There may very likely be a compatibility issue with JavaScript and Python TCP libraries. As mentioned earlier, however, TCP is not as significant and we can still use UDP for transmitting all the important messages.

2.5 Camera Feed

For the user interface, we added a stream for the front and the package camera feed across a wireless network. We use mjpg-streamer library on the Raspberry Pi for sending both, the front camera and the package camera streams to the server. At the moment, we have only implemented the package camera for the display on the server; however, because mjpg-streamer will also be used for the front camera, the code will turn out nearly identical for the front camera.

For our front facing camera, we are planning to use an Eys3D camera. However, the weight of the camera exceeded our weight limit for the helicopter. But we were able to incorporate the camera by removing other features of the MAV. Additionally, we ran into compatibility issues during the initial step up of the camera.

To fix the compatibility issues, the first step was to find solution to the Eys3D camera problem was hooking it up to a raspberry pi. It became clear very quickly that things were not working correctly. After extensively troubleshooting the camera, it was determined that the functionality was not going to get better. At this point the CS team contacted the ECEs and informed them of our situation. Over these past two terms the teamwork between the ECEs and the CS team has improved dramatically. The ECE team was quickly caught up on the situation and they found a replacement camera that suited the competition's needs. This new camera can still provide stereographic data, however, unlike the Eys3D camera where the depth map was constructed by the camera's hardware, this new camera's video feeds will need to be processed on the pilot's laptop to create a depth map. This may lead to problems down the road like too much latency from the image processing for the heat map to be useful. After the new camera was ordered, the MEs were given the new camera's weight and dimensions so they could rebalance the MAV. Balancing the MAV is extremely important to the competition because

hover is not possible without a weight balanced airframe. Overall, it was very impressive for a large interdisciplinary team react so quickly and fluidly to a problem at such a late stage in production.

3 What is Left to Do

While all of our planned functionalities are completed for our projects, we may extend the helicopter's capabilities to assist the pilot while flying through the competition. Even though they are most required, we may implement a TCP communication network, Collision Avoidance, Collision Avoidance Kill Switch, and Helicopter control.

3.1 MAV Stretch Goals

3.1.1 MAV Controls Transmitter

Currently, we have only the sensor data transmitter, for sending data from the MAV to the server. For collision avoidance, a stretch goal, we will need to also receive and transmit the MAV yaw, pitch, roll, and additional required controls. The ECE team has established a gateway in their code for us to either modify the MAV controls or pipe them back to the server. To pipe the controls, we have to add an additional UDP or TCP transmitter on both, the server and the Raspberry Pi.

3.1.2 Collision Avoidance

If time remains after building the core features of our application, we will pursue the collision avoidance attributes to prevent accidental crashes during the competition. The electrical engineering team has worked on a motor interface enabling the Computer Science team to access the helicopter's controls by providing details of the radio frequencies the remote will use. So, we can leverage the data we receive from the device on the aircraft to override controls if the MAV comes too close to an obstacle.

3.1.3 Collision Avoidance Kill Switch

If we implement collision avoidance we also need to implement a toggle feature for turning the collision avoidance off and on. We would want the collision avoidance to start on then we should be able to toggle it off if we encounter any issues while flying. Specifically, the helicopter needs to fly through a tunnel and this is the greatest point of concern for collision avoidance. If the collision avoidance code determines that the walls of the tunnel are too close to the aircraft, we will want to be able to shut down the collision avoidance feature to allow the helicopter to fly through the tunnel. Once passed the tunnel, we want to be able to turn the collision avoidance on again.

3.1.4 Helicopter Control

With the interface provided to us by the electrical engineering team, we can implement crud controls for the pilot to control the helicopter with the same laptop that will provide the camera feed GUI. The arrow keys, or "WASD" keys are potential candidates for directing the helicopter. Our controls of the helicopter will exclude the functionality to back the helicopter up. There has been no plans from the mechanical engineers or any other subteam to implement controls for the helicopter to move in reverse, thus our controls would also exclude this function.

This feature is of the lowest priority to implement and will likely not be included in the beta because of a lack of excess of time. Adding this movement functionality would please the ego of the CS team but is not necessary for the success in the MAV competition. The functionality is redundant because the team is already required to have a remote controller with a kill switch that already has this functionality.