I2.01 Sky Dashing- GUI Development

Jorge Dozal, Ja'Brianne Cleveland, Melissa Villatoro, Ryan Huston

Ingram School of Engineering

Project Problem

- Drones are a pristine technology which is currently lacking an all-in-one Graphical User Interface (GUI) which monitors drone health and domestic flying conditions in its area.
- Developing accurate flight paths that incorporate local weather data as well as drone diagnostics from sensory data will aid in establishing sustainable drone ecosystems for commercial and residential use. This requires a multidisciplinary approach.

Project Purpose

- To further enable a "Highway In the Sky", the development of user-friendly GUI will display a geographic flight map, pre-flight diagnostics of drone, and a final FLY/NO FLY decision for each drone mission.
- The creation of an all-in-one dashboard will allow users to actively monitor if the flight mission is suitable considering current weather conditions in the area and the other variables previously mentioned.

Project Objectives

- Development of improved GUI using Python and ROS2 for drone diagnostics which gives the user pre-flight data that also determines Fly/NO-Fly Decision.
- The GUI will visually display a geographic flight path, preflight drone/weather diagnostics, and a Fly/NO-Fly decision using images and easily distinguishable choice symbols.
- Only Python will be used to :
  - Creates Flight Path
  - Store and Parch Diagnostic Data
  - Creates GUI

Background Information

- In the beginning, there existed a GUI which considers only local weather station information for preflight *wind speed, temperature, and chance of rain*.
  - The text is too small.
- User interaction with the current GUI does not allow for a complete data analysis of current conditions for the drones pre-flight.
  - Final decision should be more evident.

Figure 1: Beginning State of GUI

Graphical User Interface (GUI) Layout

Weather Data: Using Python library "requests" to parch information from online weather stations, the GUI will display the weather, temperature, and wind speed from the launch/landing stations and along the flight path. All pre-flight.

Fly/No Fly Decision: Utilizing the parameters from the pre-flight weather/drone diagnostics, if-statements are looped for each data point until the function stops. During the loops, the dashboard is updated every 500 milliseconds to display two decisions. The first being the drone data approval. This decision is based if the drone optics are satisfactory to fly or not. The second decision will be weather data approval. This decision is based if the weather optics are satisfactory to fly or not. Finally, a big FLY/NO-FLY icon will be decided based on both approvals.

Flight Path: Uses a static satellite map of flight path drone will take. Velocity is used update the direction the drone moves on the map.

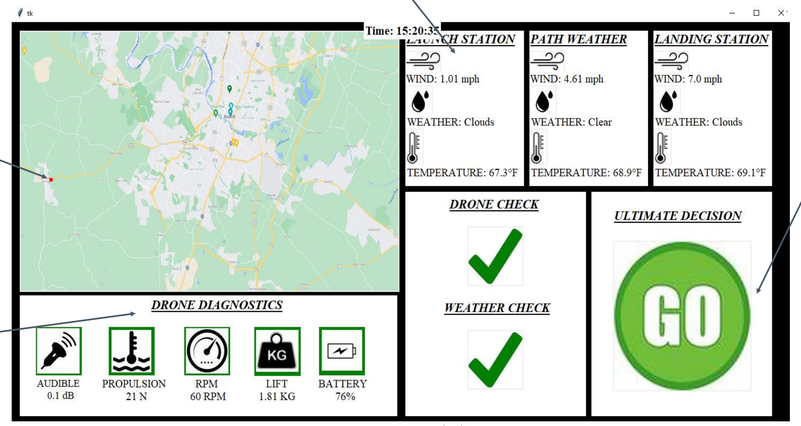

Figure 2: Pre-Flight View

Pre-Flight Drone Diagnostics: Using the provided data stored in ROS2BAGs by Electrical Engineering Team, Python library sqlite3 will display information from the *ultrasonic sensor, force mount weight sensor*, *RPM sensor*, *drone battery percentage* and *propulsion.* All pre-flight.

In-Flight Drone Diagnostics: Focused on drone optics in correlation with flight path. Python will parch the information the drone sends back and display its *flight speed, altitude, coordinates,* and *battery power.*

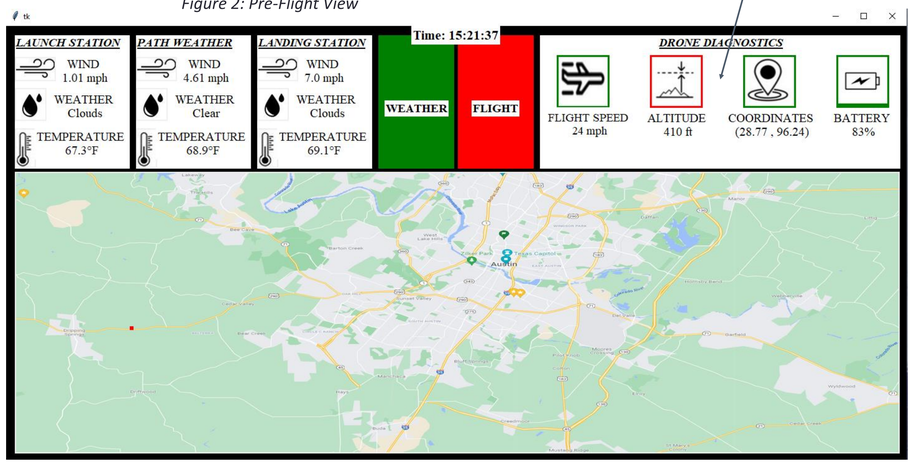

Figure 3: In-Flight View

Python and ROS integration

- Using SQLITE3
  - Opens .yaml file to find data entries in the (x,y) form
  - Sets first_float and second_float to equal respective (x,y) for simple future reference
  - Second_float is the true parameter that determines decision

```
relative_path = '../'
with open(relative_path + 'metadata.yaml', 'r') as stream:
    try:
        yaml_file = yaml.safe_load(stream)
    except yaml.YAMLError as exc:
        print(exc)

db_file_path = relative_path + yaml_file['rosbag2_bagfile_information']['relative_file_paths'][0]
conn = sqlite3.connect(db_file_path)
curs = conn.cursor()
counter = 0
curs.execute("SELECT name FROM sqlite_master WHERE type = 'table';")

df = pd.read_sql_query("SELECT * FROM messages", conn)
df['first_float'] = ""
df['second_float'] = ""
for i in range(len(df['data'])):
    a, b = struct.unpack('ff', df['data'][i])
    df['first_float'][i] = a
    df['second_float'][i] = b
conn.close()
```

Figure 4: ROS Section of Algorithim

Progress

- Throughout the duration of the GUI Development Design, the team has made tremendous progress such as:
  - Evaluation of two visual development applications and then further determining we shouldn't continue with either application.
  - Python was most optimal based upon scalability, specific project performance measures, boundaries, and constraints.
  - Creating a functional GUI using Python.
  - Pre-Flight and In-Flight views that updates using ROS2BAGs from Airogistic EE as well as parching and displaying weather diagnostics.
  - Configuring an ultimate Fly/NO-Fly decision based off drone and weather diagnostics.

Future Goals

- Future improvement to the GUI will be fulfilled by the following responsibilities:
  - Updating GUIs to direct/real-time data.
  - Dynamic Map that displays a flight path, drone, launch/landing stations, and local weather stations
  - Have second layer open by clicking icon, showing more depth diagnostics.
  - Surveying and getting proper idea of GUI before making multiple iterations

Team Members

Figure 8: Team Picture

Ryan Huston | Melissa Villatoro | Ja'Brianne Cleveland | Jorge Dozal

Acknowledgements

We would like to give a special thanks to the following individuals:

- Jeffrey Michalski, Airogistic
- Dr. Michelle Londa, Texas State
- Kathryn Budde, Texas State
- Alexander Little, Texas State
- Ethan Blagg, Nexus