

Cornell Adversary: Magpie Project (FA 25)

Jonathan Distler

Mechanical and Aerospace Engineering
Cornell University, Ithaca, NY, USA

Abstract—This document represents my contributions to Cornell University’s Aerospace Adversary Laboratory and their ongoing Magpie project. I was responsible for configuring a ROS 2 *ws* to stream Marvelmind GPS data. I also set up a Gazebo environment with a replica of the Magpie project’s real drone, then I used a PX4 configuration (alongside external plugins) to publish (and subscribe via an additional node) telemetry data within the simulated environment. As a side project I flashed an NVIDIA Jetson Orin Nano NX with a similar ROS 2 *ws* to more expediently stream GPS data, as well as for its future use cases involving onboard computation and training.

I. PROJECT GOALS

The work throughout the Fall 2025 semester was a direct followup to the preceding two semesters of work I’ve contributed towards the Aerospace Adversary Lab. The most relevant contributions involved configuring an indoor GPS-setup known as Marvelmind that tracks a drone using pulses of ultrasound. With the Marvelmind system setup and tested, I was then tasked with creating a ROS 2 environment that would stream the output of the Marvelmind GPS data, such that it could be more accurately tested, and such that it could be implemented alongside various algorithms involved in satellite cat-and-mouse maneuvers. Following my work setting up the ROS 2 workspace (*ws*), I setup a simulation in a Gazebo environment using a replica of our lab’s drone; I also configured the PX4 software alongside the simulation along with various external plugins to output realistic telemetry data (i.e. GPS location, Euler angles, acceleration, velocity, etc.). Then, I created a ROS 2 publisher and listener node to output the telemetry data, then to save the telemetry data for future analysis. Finally, to tie all of the work together I flashed an NVIDIA Jetson Orin Nano NX with the ROS 2 *ws* as was used originally, for future goals of the project.

II. MARVELMIND GPS SUBSCRIPTIONS

As a continuation from last year’s work, for the initial calibration of the marvelmind sensors, after measuring the heights of the beacon holders, I added a total of .525” to the bottom of the beacons, and I needed to make sure to add that same amount to the bottom of the modem holder. Assuming no deformation (to change the height drastically), I brought in adhesive tapes to hold the modules in place in their “holders”. I measured out the ideal height of the beacons. The literature said 3m, given my wingspan and the obstacles in the room, I settled on a height of 2.313 m high on all of the walls. I moved the beacons so that the beacons on the wall were directly above those on the ground, and those on the ground were parallel to each other. I also 3D printed a new modem holder, it was

measured such that the height of the beacons and modems is equal for all of them. Then, I created a Matlab script to plot the 3D-position of the drone with a color gradient corresponding to time, located here: Matlab 3D Positional Plotting.

To build the first ROS 2 workspace, first I downloaded the newest ROS 2 package and fixed the dependency issues (in our case it was ROS 2 humble, which is slightly outdated, but it still works). As for setting up a ros2 workspace for the Marvelmind Publisher, the steps are listed below:

- `mkdir -p /gps_ws/src`
- `cd /gps_ws`
- `colcon build`

Then, I tried to build a package to publish the GPS data, initially I had issues calling functions from external classes (even within the same directory), so I just compiled all of the code into one large class in the same file. The following code initializes a package for drone control and is documented in the reference Enlarged Class for Function Calling. The corresponding source file is located at `/home/labdesktop1/gps_ws/.../gps_listener/`.

Then one needs to execute the following lines in terminal to get Marvelmind data:

- `cd ~/gps_ws`
- `source install/setup.bash`
- `ros2 run gps_listener listener`

III. GAZEBO TELEMETRY DATA

Next, I followed Cameron’s directions to setup the Gazebo, PX4, QGC/MAVProxy pipeline:

- 1) Install Gazebo by following the instructions at: https://gazebo.org/docs/harmonic/install_ubuntu/
- 2) Clone the PX4 Git repository using the following command:

```
git clone https://github.com/PX4/PX4-Autopilot.git/ --recursive
```

- 3) Build the simulation:

- To build the standard simulation: `make px4_sitl gz_x500`
- To build headless (no GUI): `HEADLESS=1 make px4_sitl gz_x500`
- To build with GUI (gz_x599): `make px4_sitl_gz_x599`

- 4) Run QGroundControl (necessary for PX4 to believe all preflight requirements are checked). This does **not** have to be run if pymavlink is used to engage offboard mode.

- a) If QGroundControl is not installed, follow instructions at: https://docs.qgroundcontrol.com/master/en/qgc-user-guide/getting_started/Download_and_install.html
- b) Must run on GUI. If running only in terminal, use MAVProxy (workaround for arming/flight check parameters; can read/write parameters from the drone):

- i) Preferred method: install MAVProxy

```
pip install MAVProxy
```

- ii) Start MAVProxy as the ground station

```
mavproxy.py --master=udp
  ↳ :127.0.0.1:14550
# MUST ALWAYS BE RUNNING IN
  ↳ TERMINAL, ACTS AS GROUND
  ↳ STATION
```

- iii) Python code to connect via MAVProxy:

```
from pymavlink import mavutil
# Connect to PX4 (not necessary
  ↳ with new ROS 2 implementation
  ↳ )
master = mavutil.mavlink_connection
  ↳ ('udp:127.0.0.1:14550')
master.wait_heartbeat()
print("Heartbeat received. PX4
  ↳ preflight checks should now
  ↳ pass.")
```

If running the `make px4_sitl gz_x500` command instead of the headless version, expect to wait a few minutes (approximately 1–2) for the simulation world to instantiate before proceeding to the next step.

To create a more versatile pipeline, I altered the code for MAVProxy to include the following Command and Listener nodes. Both are kept locally at:

```
C:/home/labdesktop1/ros2_ws/src/mavsdk_ros2/
  ↳ mavsdk_ros2
```

- Commander Node
- Listener Node

This is the current state of the pipeline. The PX4 repository was cloned into Distler_PX4. I also created a new ROS 2 workspace which I sourced and added the aforementioned Command and Listener Nodes (which might create issues in the future as it differs from the Marvelmind publisher):

- 1) **Terminal 1:**

```
cd Distler_PX4/PX4-Autopilot
HEADLESS=1 make px4_sitl gz_x500
```

- 2) **Terminal 2:**

```
cd Distler_PX4/PX4-Autopilot
mavproxy.py --master=udp:127.0.0.1:14550
```

- 3) **Terminal 3:**

```
cd ros2_ws
ros2 run mavsdk_ros2 mavsdk_command_node
```

- 4) **Terminal 4:**

```
cd ros2_ws
ros2 run mavsdk_ros2 mavsdk_listener_node
```

This setup results in telemetry data being written to a CSV at `/home/labdesktop1/drone_logs`. The following code produces graphical results from the CSV. In the future, I plan to produce a better input into the CSV to make a more accurate and digestible data stream.

For reference, see: [CSV Plotting File](#), replacing the file location with that of the CSV you want to read in.

IV. NVIDIA JETSON ORIN NANO NX FLASH

For flashing the NVIDIA Jetson Orin Nano NX Flash, I followed the following steps:

```
ssh magpie@192.168.55.1
# [password: magpie]

sudo apt update && sudo apt upgrade -y
sudo apt install ros-humble-desktop -y

# Set up ROS 2 environment
echo "source /opt/ros/*/setup.bash" >> ~/.
  ↳ bashrc
source ~/.bashrc

# Create ROS 2 workspace
mkdir -p ~/ros2_ws/src
cd ~/ros2_ws
colcon build
source install/setup.bash
echo "source ~/ros2_ws/install/setup.bash" >>
  ↳ ~/.bashrc

# Create Marvelmind ROS 2 package
cd ~/ros2_ws/src
ros2 pkg create --build-type ament_python
  ↳ marvelmind_mavsdk

# Move your scripts to the package directory
# ~/ros2_ws/src/marvelmind_mavsdk/
  ↳ marvelmind_mavsdk/
cd ~/ros2_ws/src/marvelmind_mavsdk/
  ↳ marvelmind_mavsdk
chmod +x ROS2_Listener_Mavsdk.py
chmod +x ROS_Publisher_Mavsdk.py

# Edit setup.py to define console scripts
nano ~/ros2_ws/src/marvelmind_mavsdk/setup.py
# Replace entry points with:
# entry_points={
#   'console_scripts': [
#     'mavsdk_listener = marvelmind_mavsdk.
      ↳ ROS2_Listener_Mavsdk:main',
#     'mavsdk_publisher = marvelmind_mavsdk.
      ↳ ROS_Publisher_Mavsdk:main',
#   ],
# }
```

```
# Build and source the workspace
cd ~/ros2_ws
colcon build
source install/setup.bash
```