1 Project Charter

1.1 Project Overview

Researchers often capture, tag, and release animals to study their environment and movement patterns. Because the animals can be quite small, the tag includes small sensors that intermittently emit a signal to indicate the animal's current location. The batteries attached to the animals must also be small, so the sensors' signals may be very weak to save power. A researcher who has tagged and released an animal typically must go through a laborious manual process to track the animal's movements over time. This process involves a human lugging around a large antenna while trying to detect the signal from the animal's sensor. Once a signal is detected, the human moves in that direction, looks for the signal again, and repeats the process until the animal is found. The Radio Telemetry Tracking (RTT) project intends to optimize this process by deploying drones to detect and track the signals from the animal's sensors instead. The drone has several sensors of its own on-board that allow it to optimize its flight path to more accurately determine the animal's location. Because the drone's hardware and scanning tools can be controlled via software, the drone is able to track multiple animals within a single flight, further optimizing the tracking process. To allow for ease of use and monitoring, the drone transmits its status and tracking information to a system on the ground, allowing the researchers to view the progress in real time.

1.2 Project Approach

Given that the Radio Telemetry Tracking project is an ongoing, mature effort, the feasibility of using drones to perform the tracking has been verified in previous experiments. The goal is to iterate on and improve upon the previous design. The current design includes several components, namely:

1. The drone, to which is added a compass module, GPS module, user interface (UI) board, and on-board computer (OBC)
2. The ping location estimation algorithm
3. The ground control system (GCS)

The UI board is responsible for collating the information from the compass and GPS modules before repackaging the data for the ping location estimation algorithm. The UI board also functions as a watchdog to monitor the status of the OBC. The OBC is required to occasionally send a message (known as a heartbeat) to the UI board. If the OBC fails to send too many heartbeats, the UI board can detect that there is an issue and respond by setting warning LEDs and sending the status to the GCS. The ping location algorithm takes the information regarding the drone's current location, as well the the strength of any detected sensor readings, to estimate the location of the animal. This estimate is used to optimize the drone's flight path so as to collect as many pings as possible to more accurately locate the animal. Finally, the GCS provides a visual user interface with which the researchers can monitor the drone's progress in real time. The OBC sends the drone's status and the tracking information to the GCS via WiFi so that the researchers can ensure that the system is functioning correctly.

Several other members of the team are focused on the ping location estimation algorithm, the GCS, and the firmware for the individual modules aboard the drone. My tasks for this project are focused on the UI board, so I will provide more details about that component here. First, the UI board is required to communicate with all of the other components aboard the drone; namely, the OBS, the compass, and the GPS. Given the remote work environment, a simulator was developed about ten months ago so that access to the physical hardware was not required for progress to be made on the UI board's tasks. The simulator is a set of separate C files that implement the same API that is used to read values from the physical sensors. Specifically, the simulator emulates the compass by simply outputting a random value within the expected range. The simulator opens read and write FIFO pipes for the OBC and GPS simulations, each acting like the communication channels available on the physical drone; sensor data and the package results can be read and written to these pipes as normal. By linking to these simulation files rather than the code used to read values from the actual sensors, the rest of the system can function as usual with the hardware abstracted away. The simulator has not been used for a while, so the first goal is to verify and test the existing simulator code to ensure that it is still in line with the current drone's specifications.

Once the simulator is complete, the second portion consists of allowing the UI board to collect the data from the sensors, package it into a single payload, and send it to the OBC. There are two aspects to this: the code for the UI board to package and send the data, and the code for the OBC to accept and use the data. My focus is on writing the C code that accepts the data pushed from the GPS every one second, polls the compass information, and formats it before sending it to the OBC. The code for packaging the data is partially implemented and will be verified as part of the work on the simulator. The package format for sending the data has also been defined, and the specification is available in the Google Drive used for system-level documentation.

The UI board also plays the role of monitoring the status of the OBC via the heartbeats. Since the UI board needs to multiplex these tasks, a possible extension may involve implementing a scheduler within the UI board so that it knows to prioritize retrieving and sending data over processing the heartbeats. However, since the process of retrieving and sending the data is only expected to take on the order of tens of milliseconds, testing will determine if this additional logic is necessary. Finally, the OBC must run Python code to accept and process the sensor data received from the UI board. This is currently not implemented, and while my focus is on the C code for the UI board, depending on the feasibility it may be useful to implement and test the Python code in parallel.

1.3 Minimum Viable Product

The team's overall minimum viable product (MVP) is to have a working prototype to perform in-flight testing with the drone by the end of the summer. These test flights would likely occur in partnership with the San Diego Zoo. My work towards this goal involves implementing smaller subsections of the functionality involving the UI board. Broadly speaking, the MVP for the simulation environment involves a working simulator that passes the existing end-to-end test case. The current test case does not include most of the communication between the UI board and the OBC. It instead focuses on the read APIs for the sensors, the packaging of the data into a single payload, and the formation of the heartbeat messages. The end-to-end test case will likely not be updated as part of this work; instead, implementing the communication between the UI board and the OBC will prove more useful as a full end-to-end test of the APIs. Given the time constraints and the lack of an existing testing infrastructure, unit tests for the C code will not be a requirement for the MVP.

The MVP for the communication between the UI board and the OBC involves having a working setup within the simulation environment. The main goal is to have the UI board successfully send the packaged sensor data to the OBC within the simulator according to the communication design document's specifications. Stretch goals include implementing the OBC's side of the communication, as well as having the OBC send its heartbeats. Since the OBC's side of the communication is a stretch goal, confirmation of completion of this part of the MVP will involve passing a test suite that checks the well-definedness of the messages, as well as having the OBC log the data it receives for manual verification. The simulation will abstract away some of the difficulties of the hardware, especially dealing with the timing aspects of reading data from the physical sensors; however, this means that these issues will have to be dealt with once testing on the physical device occurs. While ideally the tests on the physical device will also occur within the quarter, it's not certain if this is possible as the firmware for the individual sensors is still ongoing.

1.4 Constraints, Risk, and Feasibility

There are several potential stumbling blocks that we need to be aware of. First and foremost is the possibility that the simulation environment has become severely out of date and needs to be significantly rewritten. Based on the number of lines of code that exist in the current simulator, as well as the GitHub history as to how long it took to develop the existing simulator, rewriting it would likely take several weeks. Not using a simulated environment and instead developing and testing the new software for the UI board directly on the physical hardware presents its own challenge. Since only a single setup exists, I need to timeshare usage of the drone with the other team members; in particular, the team member writing the firmware for the individual sensors on board the drone needs to utilize the same hardware that I do. Regardless of whether or not the development occurs using the simulator, the final testing must be done on the actual hardware. Beyond the issue of timesharing, if the work on the firmware for the individual sensors is not complete, the final testing for the UI board will also be delayed.

Thankfully, few changes to the overall architecture have occurred since the original simulator was implemented. Based on my initial work, the required changes should be minimal and are focused on smaller bug fixes and relaxing assumptions in the original code around the order in which sensors are initialized. Given this, most of the work with the UI board should be feasible using only the simulator. The main concern revolves around the firmware of the individual modules. Until this work is complete, the UI board's role of acting as the middle man to collect all of the information from the individual modules cannot be fully tested on the physical hardware.

In terms of the quarter, updating the simulator and implementing the portion of the UI board that collects and repackages the data from the individual sensors is expected to be feasible. In particular, the UI board's code will be implemented and tested within the simulated environment. Ideally, if the firmware is completed for all of the individual sensors, the UI board code will be tested both in the simulated environment as well as on the drone itself. Depending on how progress proceeds with the rest of the team, full completion and testing of the UI board may be delayed in favor of implementing the UI board's and OBC's functionality in parallel for ease of testing.

2 Group Management

Given that the RTT project is quite mature, a group structure already exists. Mia Lucio is the the current project lead and Nathan Hui is the current technical lead (note that I am the only team member enrolled in the class). There are weekly meetings on Thursday evenings to provide updates about the work everybody has done and to ask questions. There are also weekly general E4E meetings in which project-wide updates are shared with the entire community. For more immediate feedback and collaboration, we all communicate via the E4E Slack group, and there is a specific channel for RTT discussions. Most decisions are discussed in the weekly team meetings, and there is flexibility in experimenting with different ideas. Much of the design documentation and low-level specifications were written by Nathan, and he provides a lot of feedback about the progress and next steps to take.

Although I am working with the E4E group, for the purposes of this class, I am responsible for all of the deliverables and milestones mentioned in this document. Since the project's goals extend far beyond the end of this quarter, schedule slips within the overall context of the project may be difficult to measure. I intend to follow as closely as possible the timeline I specify in Section 4 and monitor my own work for personal schedule slips. Should any occur, long-term goals will be reduced or cut entirely in order to focus on the main tasks (namely, the simulation environment and the UI board's side of the communication protocol). The timeline allows for some slippage by leaving Weeks 8 - 10 for stretch goals that can be delayed to continue work on the main tasks if necessary.

3 Project Development

There are several students currently working on the RTT project across a few domains, but the main focuses are: updating the GCS user interface; implementing the firmware for the compass; and optimizing the ping estimate precision calculations. I will be assisting with the firmware development, specifically around the UI board, as well as the testing of both existing and new code.

A hardware setup involving a fully-connected drone and sensors is located in a lab on campus and is remotely accessible. However, because only a single setup exists, I will need to timeshare it with all others working on the device as well. To avoid this becoming a blocking issue, my first task is to verify a previously-implemented simulation environment and enhance it as necessary. This will allow me to perform the majority of the development and testing of the software I write within the simulator. For tasks that require access to the physical device, or for the final round of testing, I will need to timeshare access to the drone with the other group members. A plan for this has not been detailed yet as it is not clear if this will be an issue or not.

There is no unit testing infrastructure set up for the existing code, and there are two integration tests for some of the simulated code. However, these tests are not implemented as part of any unit testing infrastructure, but rather are two standalone `main` functions that simply verify that calling the initialization, read, and de-initialization code for each of the sensors does not cause any errors. The actual results and side-effects from these function calls are not verified. As previously mentioned, testing my updates to the simulator will involve making sure that the existing integration tests run without failure. To test the UI board's communication code, I will build upon the existing integration test to not only read from the sensors, but also verify the values once the packaged data is sent. I intend to add unit tests for any new code.

Documentation for all of the code is hosted in GitHub. Function-level documentation is written in each file where the function is implemented, and an existing style guide is available that details how to comment and document the code. File-level documentation is located in a README in the GitHub repository. System-level documentation is available in a Google Drive. Any documentation that I add to the project will follow the existing layout, and it is likely that most of the new documentation will be limited to the code- and file-level.

4 Project Milestones and Schedule

At the highest level, the team's goal is to have a complete, working system by the end of summer that would allow for in-flight testing with the drone. Within the quarter, there are several high-level deliverables that

we would like to achieve. Since I am the only student working on the project that is also in the course, all of the below milestones will be worked on by me (time permitting). For all changes involving code, an implied requirement is that Nathan will review, approve, and merge the code. In decreasing order of importance, the deliverables are:

1. ***Update the existing simulator***
   This is an absolute requirement for the work during the quarter. This task involves: manually examining all of the existing code for the simulator; finding and fixing any bugs; and updating the simulator to work with latest API specifications. The process is mainly manual since only a single test case for the simulation code exists. This task will be considered complete when the simulator can execute the existing test case with no errors.

2. ***Implement communication from the UI board to the OBC***
   Once the simulator is working, this is the next main priority for the quarter. This task involves implementing the UI board's side of the communication between the UI board and the OBC. This functionality requires the UI board to retrieve data from all of the sensors, package it into a pre-determined format, and send it to the UI board. The specifications for the communication format have already been defined and are available in the Google Drive. In the physical device, the communication will occur via a Serial port whose specifications have also already been detailed; in the simulation, the communication will occur via FIFO pipes. This task will be considered complete once the entire functionality in the communication specifications is implemented and passing the unit tests. Since the OBC's side of the communication may not be implemented as this point, manual verification that the packets are received correctly will occur by having the OBC log the raw received messages.

3. ***Implement communication from the OBC to the UI board***
   This item is a stretch goal but may be completed in parallel with the previous item. This task involves implementing the OBC's side of the communication between the UI board and the OBC. This functionality requires the OBC to accept sensor data from the UI board before sending it to the GCS (where the ping estimation algorithm runs and the data can be visualized in real time). This task also encompasses the OBC's periodically sending a heartbeat message to the UI board. The specifications for the communication format are included in the definition available in the Google Drive. In the physical device, the communication will occur via a Serial port; in the simulation, the communication will occur via FIFO pipes. This task will be considered complete once the entire functionality in the communication specifications is implemented and is passing the unit tests. At this point, the UI board's side of the communication should also be completed, so an additional end-to-end test in which the UI board sends data, the OBC receives it, and the contents are validated must also be implemented and run.

4. ***Implement a task scheduler on the UI board***
   This item is a stretch goal and may not be necessary at all. Once the previous two items are complete, testing will occur to determine the overhead the UI board incurs when reading data from the sensors and processing heartbeat messages. Since the drone is moving relatively quickly, the UI board needs to prioritize sending these data updates to the OBC to ensure that the sensor data does not become stale. If this overhead becomes significant, we will implement a task scheduler within the UI board to enable it to prioritize the data pipeline over the heartbeats. While a simple implementation would involve occasionally skipping the processing of heartbeat messages, a much more complicated implementation would involve implementing threading. It remains to be seen what, if any of this, is necessary. The testing process will involve measuring the expected worst-case latency with which data can be read from the compass and the expected worst-case latency with which a heartbeat message can be processed. If this proves to be higher than some threshold (yet to be defined), then this item will be flushed out in more detail.

5. ***Test the UI board's functionality on the physical device***
   This item would be nice to complete by the end of the quarter, but it is heavily dependent on the work of the other team members, as well as access to the physical drone. This task involves deploying the UI board's software onto the physical device and confirming that it can successfully interoperate with the sensors and OBC. This task will be considered complete if all test cases that pass in the simulation also pass on the physical device.

More specifically, the weekly milestones for the two highest priority deliverables are defined below. Weeks 8 - 10 address some stretch goals that will allow for some flexibility if any issues arise while updating the simulator or implementing the UI board's side of the communication.

Week 4: Debug and update the existing simulator and have the code approved and merged. Completion will be demonstrated via an approved merge request, as well as a passing output from the existing

integration test. The output should confirm that a heartbeat is sent from the OBC every two seconds and that a GPS reading occurs every one second. (The work has been completed and the code is currently under review)

Week 5: Implement the UI board's functionality to read code the from simulated sensors. Completion will be demonstrated by logging the sensor values to the screen. The data should be logged every one second to emulate the rate at which real data is present on the physical hardware, and the values should be within the expected range of the actual components ([-179, 180] degrees for the compass and [0-6000] mV for the voltage). The GPS outputs data using the NMEA format, so the UI board should correctly parse and display the data packets.

Week 6: Implement the UI board's functionality to package data read from the sensors. Completion will be demonstrated by verifying that a set of unit tests that checks for the well-definedness of the packets passes. The packet structure is defined by an existing communication protocol specification.

Week 7: Implement the UI board's functionality to send data to the OBC. Completion will be demonstrated by logging the raw data that the OBC receives and verifying that it matches with the data sent by the UI board. This verification can be automated within the simulation environment and may involve some manual processes on the physical hardware.

Week 8: Complete testing within the simulator (and ideally on the physical drone) and have the code approved and merged. Completion will be demonstrated by verifying that all unit tests pass and that the merge request is approved. The unit tests should cover: the UI board's ability to package the sensor data; the UI board's ability to send the data to the OBC; and the UI board's ability to parse heartbeat messages from the OBC.

Week 9: Begin implementation of OBC's functionality to accept data from the UI board. This is a stretch goal and may occur in parallel with previous tasks, so the ideal demonstrable result is not well defined at this time. It will likely involve displaying the results of the implemented unit tests that verify the OBC's functionality to parse messages sent from the UI board.

Week 10: Continue implementation of the OBC's functionality to accept data from the UI board. This is a stretch goal and may occur in parallel with previous tasks, so the ideal demonstrable result is not well defined at this time. It will likely involve displaying the results of the implemented unit tests that verify the OBC's functionality to parse messages sent from the UI board.