1 Progress Updates

1.1 Deliverables and Milestones

I previously defined several high-level deliverables and weekly milestones with expected concrete results and deadlines. Below, I discuss each deliverable, its current status, and how it progressed (or is progressing) relative to the original milestone timeline. Since I am the only team member working on the project that is also enrolled in the class, I am responsible for all of the deliverables mentioned below (except when explicitly stated that work occurs in tandem with another RTT team member's tasks).

As a reminder for some terminology:

- *User Interface (UI) board*: this component is responsible for reading data from all of the sensors and combining it into data packets
- *On-Board Computer (OBC)*: this component receives the data packets, forwards the information to the other modules, and communicates status updates to the GCS
- *Ground Control System (GCS)*: this component is a setup located on the ground with the researchers that allows them to view the drone's status in real time

I discuss below the updates to each deliverable in decreasing order of the original importance.

1.1.1 Update the Existing Simulator

Original goal: 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.

Original timeline: Completion by the end of week 4

Updates: This work was completed on schedule. Figure 1 displays some terminal output from running

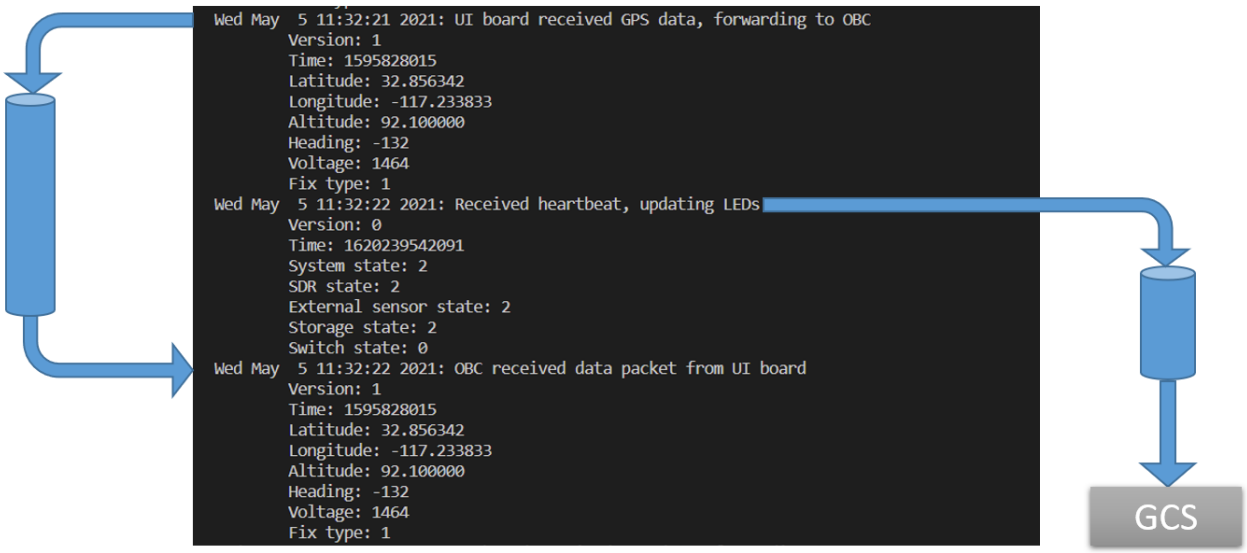

Figure 1: Example output from the UI board simulator. The left pipe represents data sent from the UI board to the OBC while the right pipe indicates a heartbeat sent from the OBC To the GCS. Notice that the data sent by the UI board and received by the OBC are identical.

the simulator. The existing testcase simply verifies that the simulator runs without crashing; hence, the ability

to generate this output indcates success. As a further step, I added the ability to log the packets sent and received to the screen for additional verification. The left pipe in the diagram indicates the sensor data being sent from the UI board to the OBC. Note that the data sent and received are identical, indicating that the communication channel within the simulation is functioning correctly. The right pipe represents a heartbeat (which contains the drone's status) being sent from the OBC to the GCS.

We have delayed opening a pull request for the work. Given the progress made by the other team members as well, the current goal is to merge and test all of the components before opening any pull requests. Instead,

Figure 2: The git log for the work done on the simulator during this quarter. This is in lieu of a pull request that is currently being delayed.

Figure 2 displays the Git log for all of the work that has been completed.

1.1.2 Implement Communication From the UI Board to the OBC

Original goal: 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 simulation, the communication occurred via FIFO pipes; on the physical drone, communication will occur via a Serial channel instead. 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.

Original timeline: Completion by the end of week 7

Updates: When I began work on this deliverable, we ran into unexpected "risk" in that we found this functionality to have already been implemented. The code to package the data was implemented and tested as part of the work on the simulator in the previous deliverable. The original goal here was to write the code that initializes the physical sensors and use the actual Serial communication, rather than the FIFO pipes. Since all of this work was completed, and testing of the actual sensors is scheduled for a separate deliverable (see Section 2.1), the next deliverable became my focus instead.

1.1.3 Implement Communication From the OBC to the UI Board

Original goal: 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. 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.

Original timeline: This was originally a stretch goal with an intended timeline of Weeks 8 - 10

Updates: As discussed previously, this was originally a stretch goal, but once the UI board's side of the communication was found to have already been implemented, I switched to this instead. There are two main components to this half of the communication:

1. Receive and parse the data received from the UI board
2. Repackage the data and send the heartbeats

Another team member and I split the work, so I focused on the first item while the other team member focused on the second item. There is some overlap between this work and that done for the simulator; namely, some basic packet processing was implemented as part of the simulation. This task mainly involves porting some of that code from C into Python. The reason for this conversion is, since this is not the first iteration of the drone, there is a significant amount of existing infrastructure around the communication pipeline between the OBC and the GCS in Python. For compatibility and ease of development, we want this code to mirror and utilize the existing code.

```
Sending packet Packet(
    payload=DataPayload(
        time=1621200853, latitude=2059382905, longitude=620254350,
        altitude=36370, heading=-21162, voltage=2422, fix_type=218, version=1,
        packet_class=5, packet_id=3),
    SYNC_CHAR_1=228,
    SYNC_CHAR_2=235)
Sending packet as bytes b'\xe4\xeb\x05\x03\x00\x18\x01\x00\x00\x00\x00`\xa1\x8f\xd5z\xbf\xb0y$\xf8T\x8e\x8e\x12\xadV\tv\xda\xed\x85'

Received packet as bytes: b'\xe4\xeb\x05\x03\x00\x18\x01\x00\x00\x00\x00`\xa1\x8f\xd5z\xbf\xb0y$\xf8T\x8e\x8e\x12\xadV\tv\xda\xed\x85'
Received packet Packet(
    payload=DataPayload(
        time=1621200853, latitude=2059382905, longitude=620254350,
        altitude=36370, heading=-21162, voltage=2422, fix_type=218, version=1,
        packet_class=5, packet_id=3), SYNC_CHAR_1=228, SYNC_CHAR_2=235)

Packets match!
```

Figure 3: Example of the OBC's ability to parse packets received over a Serial communication channel. Notice that the data entering the Serial channel and received by the OBC are identical.

Figure 3 demonstrates the functionality of the code I developed, namely the ability to receive raw bytes from a Serial channel and create a `Packet` Python object. The `Packet` object displayed in the example contains a `DataPayload` which in turn holds the data read from the sensor. The `Packet` object was developed with extensibility in mind as the payload can easily be updated to parse and store other expected payloads defined in the communication specifications.

While my half of the communication is complete, work is ongoing with the other team member to integrate all of the changes into the existing infrastructure. Some difficulty arose in that the other team member is developing purely on the Windows operating system. Opening simulated serial ports (so that we can emulate the true environment as closely as possible) was quite difficult on their machine, and so the overall merging and testing of our two halves was delayed. However, the other team member has since been able to replicate this functionality on their machine and general debugging regarding integrating the changes into the existing infrastructure is now proceeding.

1.2 MVP

My work with the RTT project originally involved implementing smaller subsections of the functionality involving the UI board, and towards this we defined two minimum viable products (MVPs):

1. Implement a UI board simulator that passes the existing end-to-end test cases
2. Implement the communication protocol from the UI board to the OBC

As discussed in Section 1.1.1, the work involving the simulator was completed on time and the goals for the MVP were met. With respect to the communication from the UI board to the OBC, we discovered that this functionality had already been implemented, and so this MVP became somewhat trivial. Instead, I pivoted to working on communication in the other direction; that is, from the OBC to the UI board. As discussed in Section 1.1.3, the half of the protocol that I was responsible for implementing is complete and is being merged with the other teammate's half.

The team's overall MVP is to have a working prototype to perform in-flight testing with the drone by the end of the summer. Given that my personal MVPs were completed on time, as well as the progress of the other team members, work towards the team's goal is progressing well.

2 Remaining Work

2.1 Deliverables

1. ***Test the UI board's functionality on the physical device***
   This task was originally a stretch goal. Given the progress with the UI board, as well as other team member's efforts towards writing firmware for the individual sensors, this task is the next priority for me This task involves deploying the UI board's software onto the physical drone and confirming that it can successfully interoperate with the sensors and OBC.

   There are a couple of caveats to the work that can currently be done. First is that the drone is located in a lab, and so the GPS module on the drone cannot get any signal. To get around this, even when deploying the code onto the drone, the GPS data will need to be simulated for now. Second, since the OBC integration is ongoing, a simple listener will be set up on the OBC side to simply receive the data via Serial and log the information for later verification.

   This task will be considered complete if: we can successfully read from the compass; we can successfully turn the LEDs on and off; we can successfully create and send a data packet once per second; and we can successfully read and log the message on the OBC side. Since I cannot observe the status of the LEDs remotely, I am hoping to have some team members that are on campus assist me in this verification.

2. ***Test the Sensor Fusion Module***
   This is a new task based on the amount of progress that has been made, both by myself and the rest of the team. There is a sensor fusion module that accepts both timestamped data packets from the UI board as well as pings from the ping detector. It then associates each ping with the corresponding data packet.

   There is some previous code for this module, but it's unclear if it is functionally correct. The first half of this task will involve reading through and debugging the existing code. Depending on what I find, the second task will either be to rewrite the code, or, if the existing code is functional, integrate it with the previous work with the UI board and verify that it is able to run correctly on the physical drone. Given the time constraints, this task is unlikely to be completed during the quarter unless work with the UI board proceeds faster than expected.

3. ***Implement a task scheduler on the UI board***
   This item is a stretch goal and may not be necessary at all. Once all of the communication between the UI board and the OBC is complete, as well as testing on the physical drone, additional 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. Given the time limit, this work will likely not be explored during this quarter.

2.2 Schedule

Week 8: Learn how to deploy the code onto the drone and successfully have it run
Week 9: Verify that we can read from the sensors and set the LEDs. Run the code for the UI board and verify that the correct data is received on the OBC side
Week 10: Begin looking through the existing sensor fusion code