Team F

Individual Lab Report 4

Progress Review 3

Yuchi Wang

Teammates

Danendra Singh

Pulkit Goyal

Rahul Ramakrishnan

Pratibha Tripathi

November 10, 2017

1 Individual Progress

My primary contribution since the last PR was the serial-ROS interface between the SE100 GPS, the multi-system network setup with ROS, GPS localization and navigation with the Bebop 2 Drone.

1.1 RadioLink SE100 GPS

For the last PR, I was able to connect the SE100 GPS to an Arduino UNO and read the NMEA format data from the RX serial line. My goal for this PR was to fully integrate the RadioLink SE100 GPS with ROS such that GPS data could be published in either *sensor_msgs/NavSatMsg* or *gps_common/GPSFix* format. Online resources and quick searches revealed that there was already a ROS package that took in NMEA data from a serial link and publishes it on a topic. This is the **nmea_navsat_driver** package and it is composed of 3 different nodes: the *nmea_topic_driver* node takes in NMEA formatted ROS messages and publishes them in *nmea_msgs/Sentence* format, the *nmea_topic_serial_reader* reads NMEA sentences from the specified serial port and publishes them to a ROS topic, and the *nmea_serial_driver* is essentially a combination of the previous two nodes. During testing I realized that while the *nmea_topic_serial_reader* node worked well, the *nmea_topic_driver* node was unable to correctly parse the NMEA messages. By extension, that meant the *nmea_serial_driver* node also didn't work. Further debugging revealed that the messages were being published in real time by the *nmea_topic_serial_reader* node but the *nmea_serial_driver* node could not verify the message's checksum. At this point, Rahul took over the work on the SE100 and was able to implement a custom node that converts the NMEA ROS messages to *sensor_msgs/NavSatMsg* messages.

1.2 Multi-system network

The work detailed in this section is split between Pulkit and me. The multi-system network task is composed of two sub-tasks. First, we want to be able to run ROS across multiple machines over WiFi with only 1 master. Secondly, we want to run the Bebop 2 Drone as a client and not as a host.

The first task is to distribute a ROS network over WiFi. This means that while the ROS master is hosted by a single machine, different ROS nodes can be run on different computers without needing its own master. This task was accomplished very simply but with a lot of effort and time due to debugging reasons. First, we had to setup our own router (named **teamf**) because we needed our systems to have static IP addresses. This cannot be accomplished with university networks such as **CMU-SECURE**. Secondly, on the remote computers, we had to change the system variable **ROS_HOSTNAME** to be the IP address of the host computer. After completing these two steps, it seemed as though we were able to use `rostopic list`, `rostopic echo`, `rostopic pub`. However, testing revealed that while we could see the topics of the host machine on remote machines, we could not see remote topics on the host. To fully enable bidirectional communication, we had to edit the `/etc/hosts` of all the machines to include the IP address and the names of all the other machines. After doing this, we were able to teleoperate the Husky over the WiFi.

Unfortunately, we were not as successful in our endeavour to force the Bebop 2 Drone to act as a client. Currently, the Bebop 2 broadcasts its own SSID and is a WiFi host. However, using the drone in this configuration is very problematic for us. As previously explained, we would like our system to be run across a single distributed network. If the Bebop 2 is a WiFi host, that means we will have two networks and at least 2 WiFi adapters on a single computer to connect to both networks. Furthermore, the connectivity of all the systems depends on the ROS node that interacts as the network bridge, which means that if the node were to crash, we would lose control of the drone.

Online sources have reported to be successful in converting the Bebop 1 Drone (not Bebop 2) to a WiFi client but the techniques used are not officially supported by Parrot. First, we used `telnet` to gain remote access to the Bebop 2 OS. Then, we ran scripts that supposedly changed the network mode of the Bebop 2. Unfortunately, this did not work and the Bebop 2 refused to connect to our network. Hours

of changing various networks parameters proved unfruitful. In the end, we decided that this task wasn't important enough to justify spending more time.

1.3 GPS localization on Bebop 2

The work completed in this section is split between Danendra and me. For this task, we were interested in the accuracy of the Bebop 2 GPS. To do so, we set the Bebop 2 on the ground and recorded the GPS updates for 1 minute. The drone updates at approximately 1Hz so this resulted in roughly 60 data points. These points were then plotted on a map (Fig 1) to visualize the drift. We calculated the drift to be approximately 3 meters. While this amount of variance isn't amazing, it does lie in accordance with our expectations and more importantly, it is tighter than our FVE criteria of 5m.

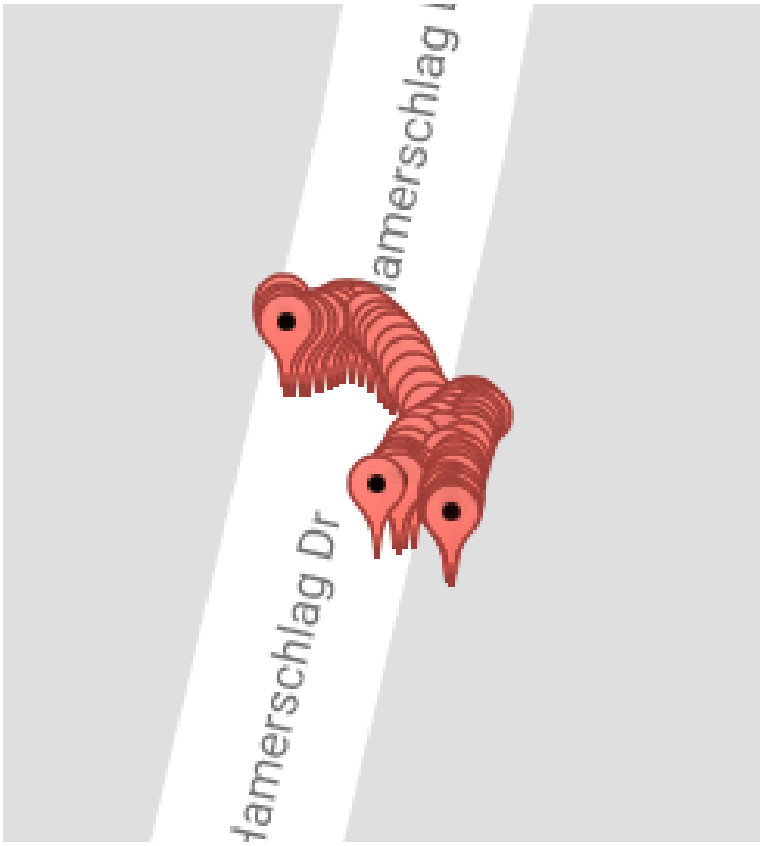

Figure 1: Drift from the Bebop 2 GPS module - recorded over a minute

Secondly, we wanted to know how accurately we can trust successive GPS updates. As one can see from Fig 1, it seems rather unlikely that the GPS locations are distributed evenly across the time intervals. Instead, it seems as though the drone started at the top left corner and drifted in an arc toward the bottom right. This means that while the variance over 1 minute may be 3 meters, the variance over 5 seconds is likely less.

To test this theory, I implemented the *Haversine* formula in our custom ROS node to calculate the difference between two GPS locations. Afterward, we placed the drone on the ground and also let it hover autonomously for 1 minute. In both cases, the difference in successive GPS locations was never more than 1 meter.

1.4 GPS Navigation on Bebop 2

This task can be thought of as an expansion of the previous task (GPS localization is a prerequisite). The end goal of this task is to implement a flight controller that can autonomously navigate the Bebop 2 Drone to a GPS location. This task has great significance to us as it relates directly to our 3rd FVE task. There are two approaches to accomplish this task. First, the Bebop 2 has a built in Mission Planner functionality that uses onboard `.mavlink` files to execute a flight plan. While it would be great to use this feature, there is a major problem: the files cannot be accessed by the ROS driver which means that we cannot plan paths dynamically. Nevertheless, we thought it would be helpful to explore this option in case we find it useful. Danendra has completed work on using that feature while I have focused on creating a custom autonomous navigation controller.

So far, we have been only using the Drone in teleoperation mode with a joystick controller. The design challenge in switching from teleoperation mode to autonomous flight mode is how to effectively transfer control and how to effectively implement an emergency stop system. Chris Baker's lectures offered some insight into this design issue. First, Chris suggests to implement a controller that needs continuous input - all commands expires within a short time and the robotic system always reverts to a neutral/stopped position. Fortunately, the Bebop 2 Autonomy Driver has this feature built in but it's not as effective as it would be with a ground system. This is because the Bebop 2 reverts to a default hovering state and a hovering state can still drift and crash, particularly in windy weather. Importantly, this means that we cannot just shutdown our ROS node (`control-c`) to perform an emergency stop. Our current implementation combines manual teleoperation with our autonomous controller and makes use of the fact that the Bebop 2 will automatically ignore any move commands if it is in a *landed* state. This means that we can use the land command as an emergency stop and the liftoff command to resume operation. We limit the functionality of the joystick to these two commands and we give our custom controller access to all the row/pitch/yaw commands. In essence, the joystick acts as a parent to the autonomous controller. A depiction of this architecture is shown in Fig 2.

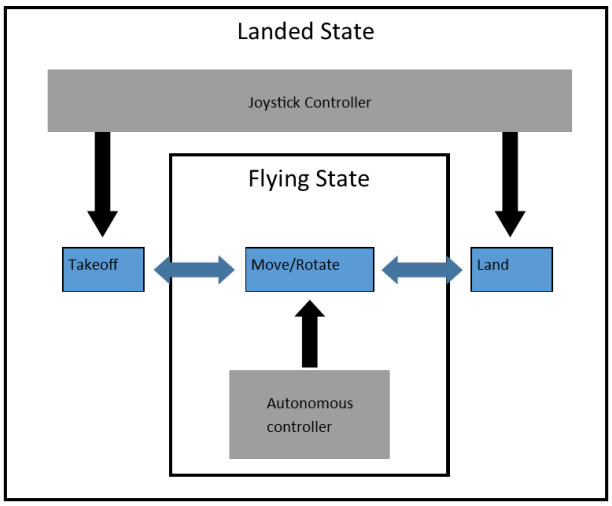

Figure 2: Design philosophy for the Bebop 2. We use the controller to enter and exit the flying state and limit the commands for the autonomous controller to row/pitch/yaw. By doing so, we ensure that we have full control of when the drone starts and stops flying.

In terms of the autonomous controller's functionality, I have implemented a simple program that moves forward (changes the pitch to 0.1) until it is within 5 meters of a target GPS location. So far, this program works well in terms of moving forward but not so well in terms of stopping (definitely a place for future work). A video of our first test flight is available at `https://drive.google.com/open?id=1rrRVxevJwfeJGrMYzBGoedAS3O6lNu8n`.

2 Challenges

There are many difficulties with each of the previously discussed tasks. For the GPS SE100, a challenge was finding out why the *nmea_topic_driver* node did not parse the NMEA data. This issue was particularly hard to debug because I installed the package with `apt-get` and not with the source code so I couldn't make changes to the node. However, I talked to Aaron and he was able to successfully use the node with a Pre-Go PPP GPS so I assumed that the problem was caused by the NMEA data being formatted slightly differently.

For the WiFi network with the Bebop 2, the first challenge was connecting to the onboard OS. The connection specifically required `telnet` and not `ssh`, which took Pulkit and me some time to figure out. In addition, the main challenge was figuring out why the drone refused to change WiFi configurations. As said before, the instructions online were for the Bebop 1 Drone and thus I could not make any assumptions on the validity of these approaches for the Bebop 2. In fact, a few of the files that were mentioned in the instructions did not even exist on the Bebop 2. Certain online developers believed that Parrot has limited the permissions of developers to make high-level changes to the system configuration - including the network setup. As a result, some of the more aggressive approaches involved flashing the firmware to remove this restriction; however, this comes with the risk of "bricking" the drone and after consideration, we concluded that this wasn't a risk we were willing to take.

For the localization with the GPS, our main challenge was visualizing the positions in real-time. Ideally, we would like to visualize the drift in GPS locations in real-time so we can have a better idea of the accuracy. To that end, I tried installing 2 different ROS packages that subscribed to the *fix* topic. First, I tried using **gpsd_viewer** but the compilation involved a plethora of missing dependencies that I couldn't resolve in the end. Then I tried to use the **mapviz** package. I were able to run this node and even subscribe to the GPS messages, but for some reason it kept saying "no messages received" even though I could `echo` it perfectly fine.

For the autonomous navigation, the current challenge is to find out why the drone didn't stop when it was within 5 meters of the target destination during the test run. Because of this error, the drone crashed into a wall and broke a propeller. While we have spares, we would like to avoid more of these accidents if possible. A simple solution to this is to reduce the speed and I have considered slowing down the speed at which the drone flies (currently pitch is set to 0.1). However, I've also noticed that strong winds sometimes push back against the drone. This is another challenge since the presence of these winds may necessitate a PID controller.

3 Teamwork

As previously described, I performed joint work in the network setup (with Pulkit), Bebop 2 GPS localization and Bebop 2 autonomous navigation (with Danendra). For other tasks, Pulkit and Pratibha worked on setting up and connecting the IMU for the husky and incorporating that information into the RVIZ visualization. Rahul was able to complete the work on the SE100 GPS and implement a ROS node that converted the NMEA data to GPS coordinates. Rahul and Pratibha worked together on constructing the sensor mount for the Husky. In addition, Rahul has also started working on Velodyne obstacle detection. Danendra and Pratibha were responsible for the Power Distribution Board design for the assignment.

As a team, we talked with John about the feedback for our project in light of the questions from the FVE presentation. He has said that a comparative study between the performance with and without the drone would help motivate the project. To that end, we have decided to add an additional component to the SVE which is the navigation of the Husky without drone support. Furthermore, we have presented our progress and this new component to our sponsor, Katia. She has suggest improvements on making the comparison more fair (since using April Tags can assume prior knowledge that isn't realistic).

4 Future plans

Our future plans is dictated by the goals for our FVE. The first two goals are related to April Tag detection which has been assigned to be (internally). Most of the work on this front has been completed. As shown in PR3, we are able to localize the April Tags with respect to each other through tf2. However, the problem still remains that the localization isn't very stable. To address this issue, we talked to Sasanka (our PhD advisor) who recommended that we use a simple low-pass filter. Rahul and I will plan to implement that by the next PR. The other FVE goal is autonomous GPS navigation with the Bebop 2. As described in the previous two sections, some work has been completed on this front but some more work is certainly required. First, I will need to find out why the drone isn't stopping within 5 meters of the target location. Second, I will need to implement omnidirectional control (currently drone only goes forward). Lastly, I will need to test it in various weather conditions to see if the speed can be set to a static number or a dynamic controller such as a PID controller is needed. Danendra will also be assisting me with the Bebop 2 work. Lastly, Pulkit and Pratibha will be working toward setting up the WiFi network for the Bebop 2.