Team F

Individual Lab Report 3

# Progress Review 2

*Yuchi Wang*

*Teammates*

Danendra Singh

Pulkit Goyal

Rahul Ramakrishnan

Pratibha Tripathi

October 27, 2017

# 1 Individual Progress

My primary contribution since the last PR was the testing of a joystick-teleoperated Bebop 2 controller, building a custom controller to track the Bebop's state variables, visualizing the TF information inside RVIZ, and reading GPS data from the Radiolink SE100 module.

## 1.1 Bebop 2 Joystick controller

Recall that last week, we demonstrated the ability to interact with the Bebop 2 through a ROS driver. At that time, we did not have an actual ROS controller node that was receiving commands from an input device or using a heuristic. Instead, we merely used the *rostopic pub* command to send the commands one at a time. This week, I extended Danendra's work by enabling full flight control of the Bebop 2 through a joystick controller node. I note that most of the joystick controller code is included with the Bebop Autonomy package so actual software development on this component is minimal. Most of my effort went into debugging compiler issues, tuning network parameters, and setting the controller configuration.

The provided joystick controller node is called *joy_teleop* and it reads inputs from the joystick controller and publishes *cmd_vel* messages to the *bebop_driver* node. An rqt diagram of this architecture is shown in figure 1. In addition, the drone publishes raw images files in the format of *image_raw*, which can be visualized with *rqt_image_view*. Putting all of this together, I was able to control the Bebop 2 Drone with the joystick while also seeing the camera feed, in essence duplicating most of the functionality of the Bebop 2 Android app.

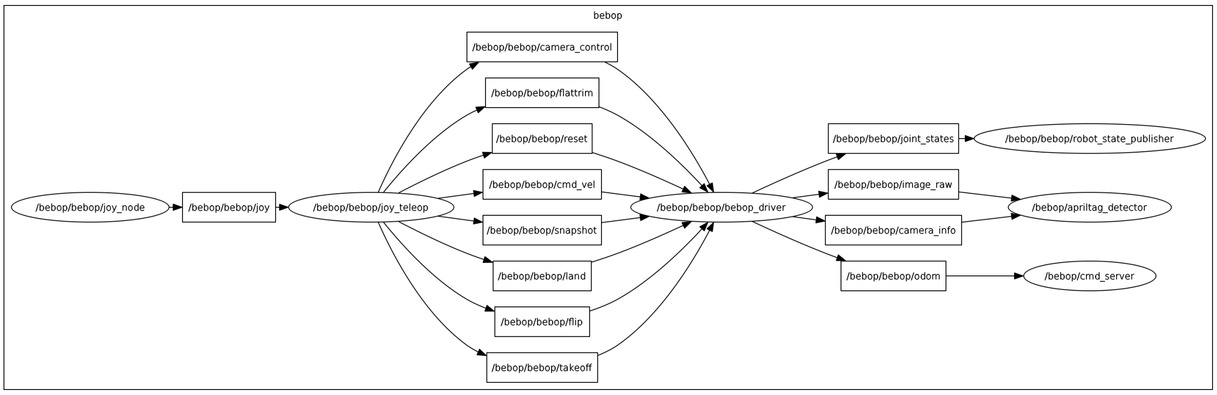


Figure 1: The rqt graph for the joystick controller system

Danendra and I tested flying the Bebop 2 drone outside with the controller to see its stability and the range of its functionality. During flight time, we noticed that the drone was very responsive and quick. In addition, the drone did retain most of its stability functionality; we did not notice excessive drift while hovering or instability during turning. Thus, we were confident that the low-level control algorithms are built into the drone itself and this allowed us to better estimate the amount of work required for developing our custom controller. A video of us controlling the drone via joystick is available at `https://drive.google.com/open?id=0B_arrx0e8EV0THJaX0M4b2d2cVE`.

## 1.2 Custom Bebop 2 Controller

I have started developing a ROS controller package for interacting with the Bebop 2 drone. The end goal is to use some form of path and motion planning heuristic to control the drone but at the moment, I've focused predominantly on implementing a state container functionality. There are many different state variables that are associated with the drone. Examples include the battery percentages, flying mode, and odometry. However, when I was exploring the SDK and the ROS driver package, I realized that the drone lacks a *get* functionality - that is, the controller cannot query the drone for specific information; instead the drone publishes these information when their state changes. Clearly, it can be quite annoying to develop a controller if the necessary information cannot be retrieved instantly so I've been working to first record and store all this information inside appropriate classes.

## 1.3 RVIZ visualization

After completing part 3 of the Programming Familiarization assignment, it is obvious that some of the data published by the Bebop 2 can be used to visualize its location in RVIZ. Furthermore, we have a need to do so because our project will certainly need to localize the Bebop with respect to the target location and the Husky AGV. There are two ways of visualizing the drone within RVIZ. First, the drone publishes *odom* data that can be read directly by RVIZ (Fig 2). As expected, when testing with this setup, we noticed that the reading is stable but over time it will drift so we will need to integrate other sensors with a Kalman Filter for more accurate results.

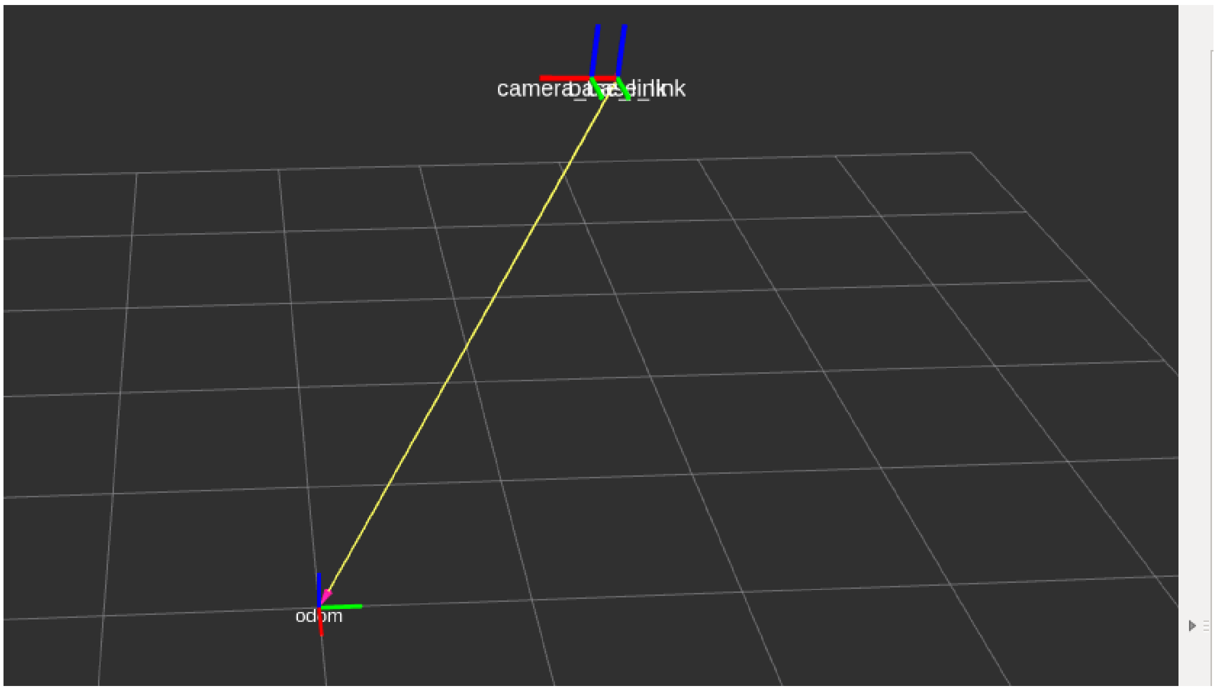


Figure 2: The Bebop 2 related frames are visualized with respect to the **odom** frame in RVIZ

Second, I was able to localize the drone through the use of April Tags placed on the ground. The use April Tags has direct significance to our project because both Katia and her PhD students have advised us to simplify the validation setup through the use of April tags to represent both the AGV, open paths, and obstacles. The *image_raw* data published by the drone was fed into an *apriltag_detector* node on my computer (this can be seen in Fig 1), which publishes *tf* messages. This information was then visualized in RVIZ. A key difference between using April Tags and odometry data was that the April Tags was relatively consistent over time (did not drift), but it fluctuated significantly from one second to the next. However, I don't think this is an inherent problem with apriltags and is instead caused by a configuration issue (I experienced the same issue with Part 3 of the assignment). My plan to address this issue is discussed in section 4.

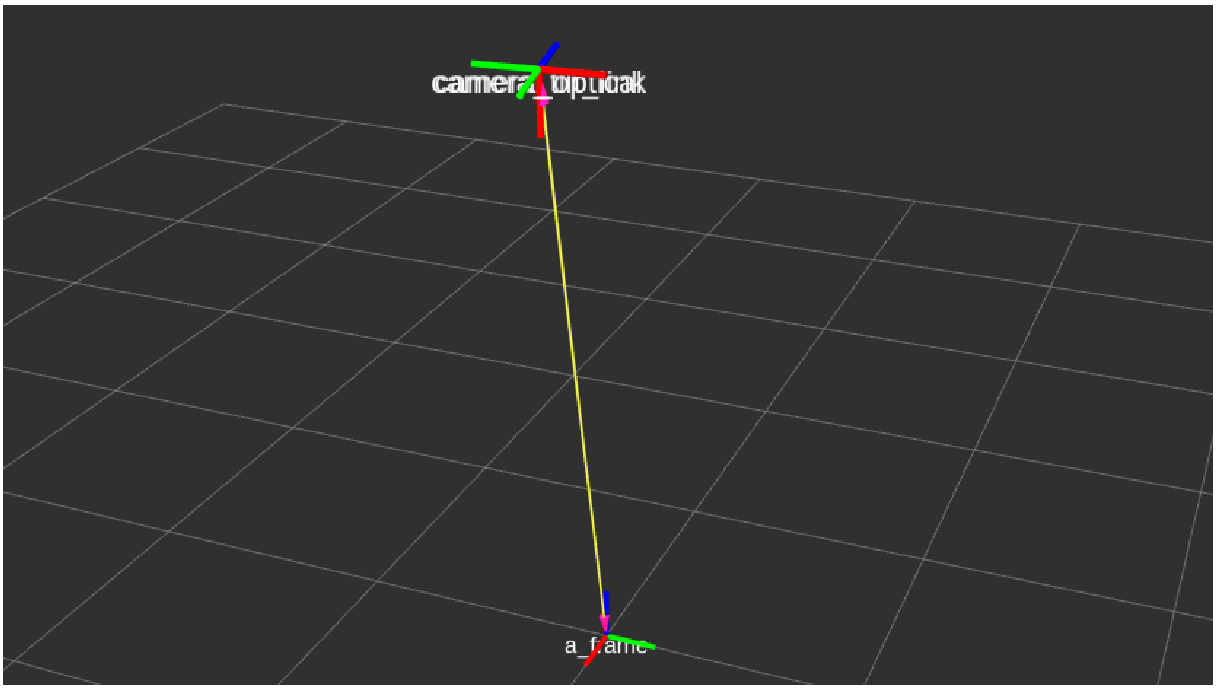


Figure 3: The camera frames are visualized with respect to the April Tag (**a_frame**) frame in RVIZ

### 1.4 RadioLink SE100

Because the Husky we acquired from George and Dimi both have no sensors, we need to select and purchase our own devices. I noticed that there was a SE100 GPS module in the MRSD inventory so I took it out to test it (I also wanted the Arduino Shield but that was not there). The SE100 is a GPS module designed for the PIX flight controller. The SE100 has both UART and I2C connectors (Fig 4) but no information was given on how to directly interact with the connections. As a result, I took a trial and error approach and assumed that the module would publish GPS info by default on TX and the RX line was used only to change specific settings - I also ignored the I2C lines. I connected the TX line to Digital 0 on Arduino and configured Digital 0 as Serial In. After tweaking the baud rate, I was able to continuously receive NMEA data (Fig 5).

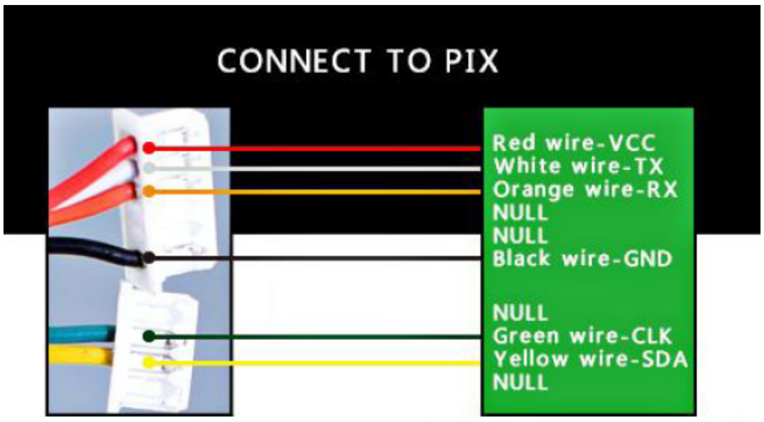
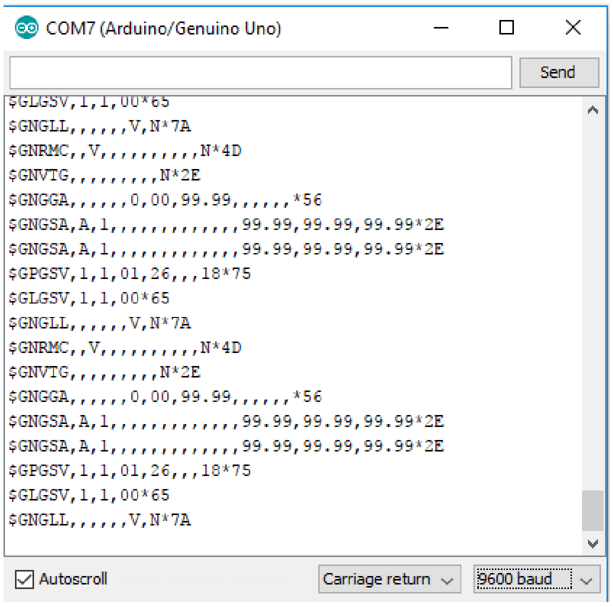


Figure 4: The connectors from the Radiolink SE100. Obtained from Radiolink SE100 User Manual.



```
$GLGSV,1,1,00*65
$GNGLL,,,,,,V,N*7A
$GNRMC,,V,,,,,,,,,,N*4D
$GNVTG,,,,,,,,,N*2E
$GNGGA,,,,,,0,00,99.99,,,,,,*56
$GNGSA,A,1,,,,,,,,,,,,,99.99,99.99,99.99*2E
$GNGSA,A,1,,,,,,,,,,,,,99.99,99.99,99.99*2E
$GPGSV,1,1,01,26,,,18*75
$GLGSV,1,1,00*65
$GNGLL,,,,,,V,N*7A
$GNRMC,,V,,,,,,,,,,N*4D
$GNVTG,,,,,,,,,N*2E
$GNGGA,,,,,,0,00,99.99,,,,,,*56
$GNGSA,A,1,,,,,,,,,,,,,99.99,99.99,99.99*2E
$GNGSA,A,1,,,,,,,,,,,,,99.99,99.99,99.99*2E
$GPGSV,1,1,01,26,,,18*75
$GLGSV,1,1,00*65
$GNGLL,,,,,,V,N*7A
```

Figure 5: Serial data in from the SE100. The GPS coordinates are invalid in this figure as the GPS has no signal.

## 2 Challenges

The main difficulty in this progress review is the interface of the GPS. Because the GPS is designed to be connected to a flight controller, there is no documentation on how to directly interact with it through a computer. As a result, I had to look at online examples of other GPS modules and try to take the same approach. I noticed that many of the examples only needed the TX, GND, and VCC lines to read from the GPS. From this, I though that perhaps the SE100 didn't require the RX or the I2C lines either. Fortunately, this was the case but I wouldn't have know otherwise.

For the Bebop 2 development, there weren't many technical challenges where I was unsure of how to do a certain task. Most of the objectives completed for this PR were not proof-of-concept objectives in which we demonstrate the viability of a certain approach. For instance, I knew that the joystick could be used to teleoperate (the code was provided) so the purpose of that objective was primarily to test the performance of the Bebop driver. As well, Part 3 of the programming assignment has shown me that localization can be accomplished with odometry data and apriltags within RVIZ. As a result, while doing most of the tasks were time consuming, they did not present a significant technical challenge.

However, because we have managed to get both the Husky and the Bebop 2 Drone running through ROS, there were certain logistical or management challenges in terms of deciding how the team should proceed. Shortly after ILR02 was submitted, we decided to scope down the objectives for PR2 as we predicted that this week would be rather busy due to the barrage of assignment deadlines. In addition to the CAD and PCB assignments that were due, there were also many aspects of the AGV/UAV that we can develop so it was not clear which parts we should focus first. For instance, we could have started with the higher-level flight control, image-stitching and map generation from UAV, or AGV Velodyne integration. To determine what was truly necessary, we had to look at our WBS, risks, functional architecture, and employ the good advice of Dimi. In the end, we decided that localizing the UAV was a prerequisite to actually implementing higher-level controls.

## 3 Teamwork

As previously described, joint work that I was involved in was the Bebop joystick testing with Danendra. For other tasks, Pulkit and Rahul were able to control the Husky with a joystick. In addition, Pulkit and Rahul are working together to complete a CAD design for a sensor mount for the Husky. This sensor mount would have to support the Velodyne Puck, the GPS module, the onboard PC and an additional 24V battery to power the sensors. As well, it should have an open and big platform for the UAV to take-off from and for an April Tag to be placed.

Pratibha and Danendra also worked on designing a power distribution board for connecting the sensors with the battery. This is the same work that needs to be accomplished for the PCB assignment. Rahul has also been looking into how to read and process the data from the Velodyne Puck (since John has managed to procure two Pucks from Velodyne). As we currently do not have the actual sensor, Rahul has downloaded .pcap files, converted them to bag format, and visuzlied that in RVIZ.

## 4 Future plans

I will continue to work on the GPS module to either connect the module directly to the onboard Linux computer through USB or to use the Arduino as a middleman. I would prefer to connect it directly to reduce the complexity of the overall system. Angad said that he had personal experience doing this so I will contact him for help if needed. Furthermore, I will convert the NMEA data into a more useful format and publish it on a ROS topic. There is a ROS package (**nmea_navsat_driver**) available so I will try to use that. I will also try to fix the April Tag issue described in section 1.4 and combine both odom, apriltag localization, and GPS data (this can be read from *fix* topic, with message type *sensor_msgs/NavSatFix*). Lastly, I will try to run both the Husky driver and the Bebop driver on one system such that the April Tags, the Husky, and the Bebop 2 can all be localized with respect to each other.

Danendra will look into a higher level controller for the UAV that takes in a location and the UAV can autonomously fly there. There is built in flight plan control on the Bebop 2 so he will try to use that. Pulkit and Rahul will continue to work on developing the CAD design for the sensor mount. Rahul has started testing the Velodyne driver but Velodyne doesn't provide the source code for the driver and we need it to integrate into our architecture. As a result, he has currently installed and integrated a Velodyne driver from a 3rd party source and he will continue to test and calibrate the settings. Pulkit and Pratibha will look into the incorporating the encoder data from the Husky to estimate its state. Furthermore, they will implement a controller that takes in a location and have the Husky autonomously drive there, using the encoder data as feedback.