Team F

Individual Lab Report 2

Progress Review 1

Yuchi Wang

Teammates

Danendra Singh

Pulkit Goyal

Rahul Ramakrishnan

Pratibha Tripathi

October 20, 2017

1 Individual Progress

My primary contribution in the last week was the implementation of a C++ program with the Bebop 2 SDK that could communicate with the drone through my laptop and send simple commands such as takeoff, land, change directions, and stream back the video feed. As well, I was in contact with a representative from Clearpath Robotics in an effort to debug the Husky AGV and get it to move.

1.1 Bebop 2 SDK

The Bebop 2 is a developer-friendly drone that supports numerous ways of custom interface and control. Parrot provides and maintains the SDK which enables Android, iOS and Linux devices to communicate with the drone in the same way that their official app does. As well, there is a ROS package that is developed upon the SDK which provides the same functionality in the form of a ROS node. Ideally, we would want to use the ROS package but it is not officially support by Parrot so we were unsure of the functionality. As a result, our team decided to explore both options – the SDK and the ROS package – to see which would be the most suitable for our application. I took on the responsibility of developing a "proof of concept" C++ program to demonstrate that drone could be controlled by a custom program.

Luckily, a sample code of C++ code was provided which detailed the steps of connecting with the Drone. First, my program had to discover the Bebop 2 through WiFi and create device controller to represent the interface between my program and the drone. The device controller takes a callback function as an argument for when the controller receives status updates from the drone. Once connected, the drone could be controlled by yaw, pitch, roll, takeoff and land commands. The takeoff and land commands take care of all the lower-level altitude commands and automatically transition the drone from a *landing* state to a *hovering* state.

The directional commands for the drone are not as high-level as the takeoff and landing maneuvers. To move the drone forward, backwards or side-to-side, the controller sends row, pitch, and yaw commands in percentages of the maximum angle. In the program, I created a *while* loop and put a keyboard read and translated the key strokes into row and pitch commands. By doing this, I was able to test the maneuverability and stability of the Drone and I was able to achieve similar performance as when I used the official Android App.

Lastly, I was able to display the video feed on my laptop (Fig 1). To do so, I had to create a video callback function at the initialization of the device controller. The code for printing the video to *videoout* was somewhat complex so sample code from the SDK was used for that.

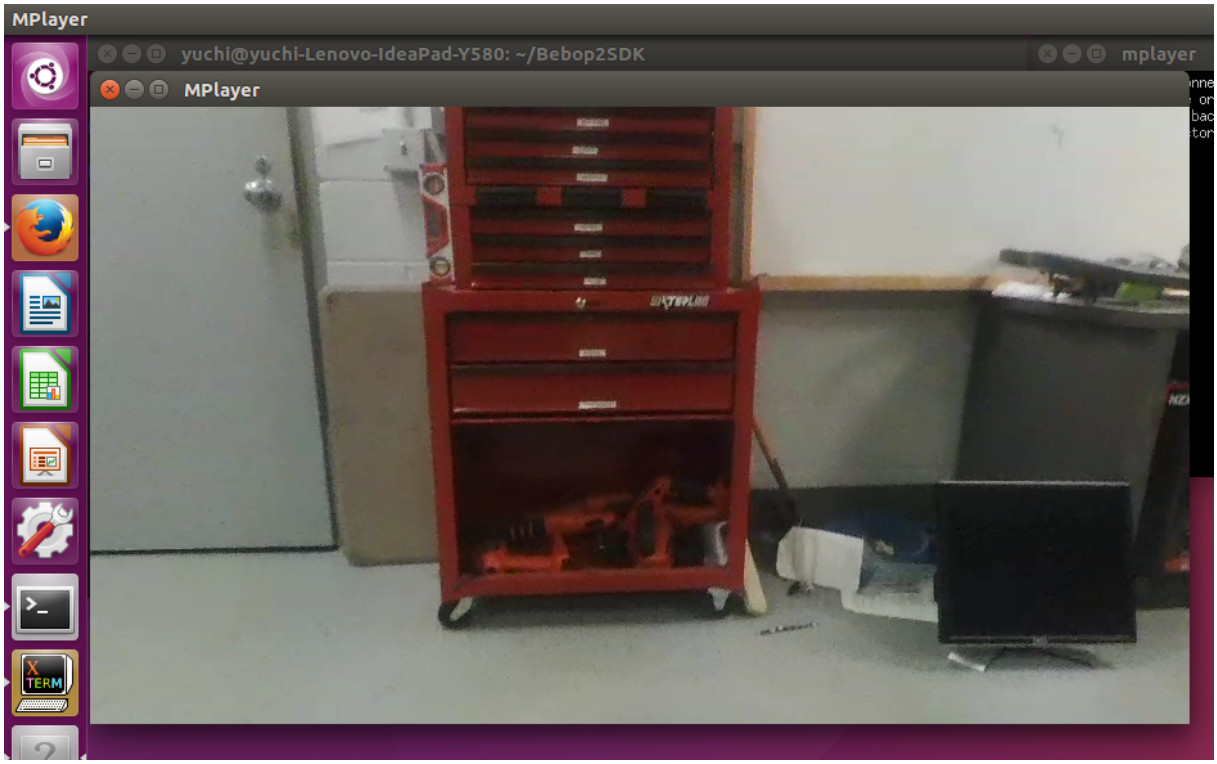

Figure 1: Video feed of the drone camera on my laptop while hovering inside FRC lab

1.2 Husky debugging

At the time of writing, we have finally managed to make the Husky move but at the time of the PR1 demonstration, we still couldn't figure out what was wrong with the Husky. The work completed in debugging the Husky is mostly a team effort and it is quite hard to separate individual effort from teamwork as most of us worked together in the lab to determine why the Husky wasn't moving. In light of that, in this section I will detail my personal approach to debugging the Husky and my attempts at fixing it.

The Husky that we received from George was in unknown shape. The included onboard PC had several custom packages builtin such as the Hokuyo but we were not told explicitly what the Husky could do nor were we given a demonstration of running it. I realized after looking at the rqt graph that the Husky control architecture was structured in layers. Specifically, the user programs operated at the highest layer and send *cmd_vel* messages to the *twist_mux* node, which selects the command from the highest priority source and forwards it to the *husky_node*, which represents the Husky itself (Fig 2).

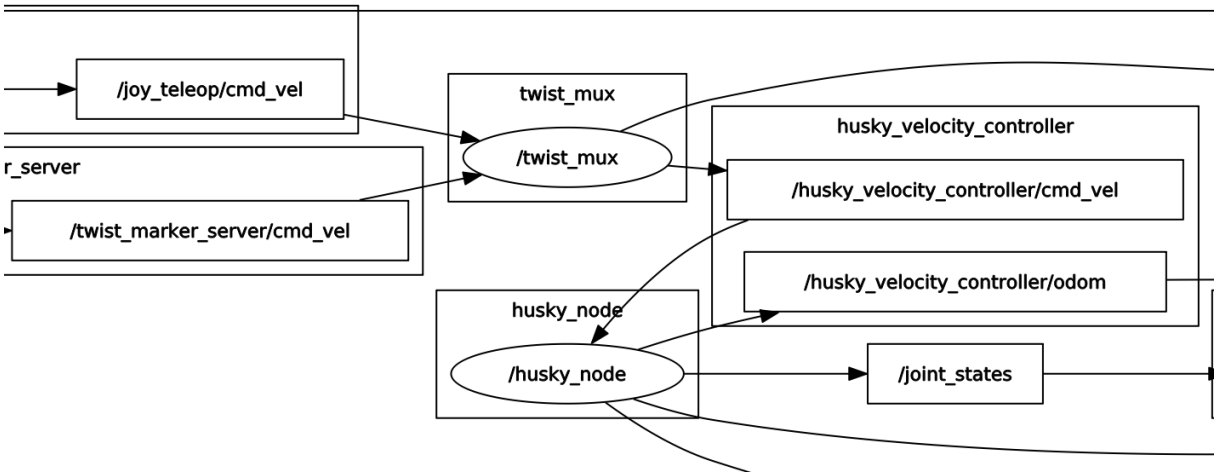

Figure 2: Control pathways for the Husky nodes

At first, I tried publishing messages to the *twist_mux* node but the Husky wasn't moving. When we determined that the *twist_mux* node was operating correctly by echoing its forwarded message, I tried to communicate with the *husky_node* directly. Unfortunately, this also did not move the Husky but we were able to receive diagnostic and status information from the Husky. Interestingly, the diagnostic information revealed that the motors were not receiving any voltage or current, but other components such as the IMU were (Fig 3).

```
mcu_and_user_port_current: 0.2
left_driver_current: 0.0
right_driver_current: 0.0
battery_voltage: 25.28
left_driver_voltage: 0.0
right_driver_voltage: 0.0
left_driver_temp: 0.0
right_driver_temp: 0.0
left_motor_temp: 0.0
right_motor_temp: 0.0
capacity_estimate: 480
charge_estimate: 0.62
```

Figure 3: Diagnostics show 0 volt and current to all motors

At this point, I believed that either the *husky_node* was not operating correctly or the onboard motor controller was damaged. I looked into reinstalling the image but we encountered network issues (since the image needed to retrieve the data from Clearpath's server). I reached out to Clearpath directly to see if they had any ideas of what was wrong but they were also unable to offer significant advice. Eventually, Pulkit was able to fix the network issue.

2 Challenges

The biggest challenge in the past few weeks was understanding what was wrong with the Husky. As no one in my team has worked with the Husky before, we spent a lot of time trying to understand the Husky architecture, searching the web for similar problems, and just trying out different ideas. Due to the number of unknown parameters, we did not have a concrete idea of how to proceed with getting the husky running. For instance, we did not know whether the problem lied in the onboard PC, the serial link, the motor drivers or the software.

When we received our second Husky from Dimi, we plugged in the onboard and the new Husky wasn't working either. However, we knew that it was working before Dimi handed it over to us. From that, we knew that the problem wasn't due to the Husky and we were able to narrow down the possible issues. After 30 minutes of debugging, we were able to identify the issue and fix it and control both Dimi and George's Husky's with a gamepad controller. It turns out that the problem was due to a broken power connection to the Husky's motors. The Husky is powered by 2 Vcc's, one for the embedded controller and the other for the motor. We could communicate with the Husky so we assumed that the motors were powered as well but that wasn't the case.

Another higher-level challenge that we experienced as a team was the rescoping of our project. Our meeting with Katia did not go as we expected, but I felt that her points and perspectives made sense. After our meeting, we have decided to refocus our project not in terms of the technology, but in terms of the application. Thus, we did not use the number of systems in our project as the basis of conversation but rather the requirements of our use case. Our new use case is as follows: we wish to implement a multimodal navigation system whereby the UAV flies in front of the AGV and surveys the environment, searching for obstacles and mapping potential routes. This provides the AGV with information that it is unable to see with its onboard sensors, and it incorporates the UAV information to make better informed path planning decisions.

3 Teamwork

The introductory work with the Bebop 2 Drone was split between myself and Danendra. Danendra looked into interfacing with the drone with the ROS and he was quite successful. He was able to control the drone in the same manner that I was with the SDK. He was able to takeoff, land, maneuver it and display the video feed on his laptop. Since we would prefer to use a ROS package directly, we will be expanding upon his work in the future. Pulkit, Pratibha and Rahul worked predominantly on the Husky. They tried a variety of different approaches to seeing where the problem with the Husky lay. For instance, they tried using the *joy_teleop* node to control the Husky and they solved the network configuration issue with the imaging.

4 Future plans

To prepare for the next PR, I will work on the Bebop 2 to implement higher-level controls. Danendra will be working on this task with me. While we are currently able to move the drone, the commands that we send are too low-level to be directly incorporated into path planning programs. Instead, we need to abstract away the low level interface and control the drone with GPS commands. Specifically, I will implement a controller that takes a GPS coordinate from the user, reads the GPS information from the drone, and provide roll-pit-yaw commands to move the drone to the desired GPS location.

My team will proceed to work on both the Husky and the Bebop 2 at the same time in order to distribute our resources. As our Husky is currently working and controllable with a joystick, Pulkit will work on developing a higher-level control stack for the Husky using encoders such that if we command it to move forward 5 meters, it will do so. Pratibha and Rahul will work on incorporating GPS data into the ROS software. There is a MediaTek GTPA010 in the MRSD inventory - they will create a ROS node that is capable of reading the GPS information from serial link.