Individual Lab Report- 2

By Danendra Singh

Team F: Falcon Eye

Danendra Singh

Yuchi Wang

Pulkit Goyal

Pratibha Tripathi

Rahul Ramakrishnan

20th October 2017

<u>Individual Progress:</u>

As my contribution to the MRSD Project-1 since the past week, I have worked on bebop2 drone (provided to us by the sponsor) and controlled it via ROS interface. I also performed flight tests to test the stability and hardware capabilities of the drone. Additionally, I helped set up the battery interface for the Husky.

<u>Bebop2 Drone:</u>

Bebop2 drone is the second generation of low cost drone produced by Parrot. It has a 2700 mAh battery that gives it a flight time of approximately 20 minutes. Bebop2 has an 180^0 fish eye lens with digital image stabilization. It also has a vertical stabilization camera in the bottom, shown in Figure 1, which works as an optical flow sensor and is used to stabilize the drone when in flight. Every 16 milliseconds, an image of the ground is taken and compared to the previous one to determine the speed of the Bebop Drone. Hence, in low light conditions, the optical flow sensor is unable to view the ground image leading to an unstable flight during night time.

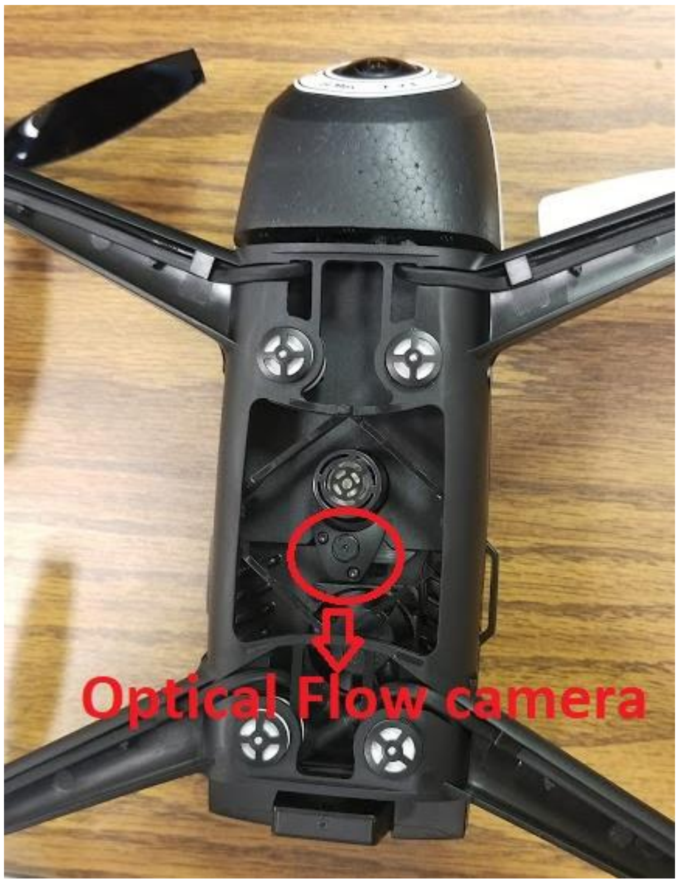


Figure 1: Optical Flow Camera

We can communicate with the Bebop2 via a dual-band 2.4 GHz and 5 GHz link that transmits power up to 21 dBm and has a range of 250 meters.

On switching on the drone, it broadcasts an open connection which I used to connect to my Linux OS.

ROS control Bebop2 Drone:

I used ‘bebop_autonomy’ ROS package to connect to bebop2. It is based on Parrot’s official ARDroneSDK3 and is supported by the online ROS community. The steps I followed to interface Bebop2 with ROS are as follows:

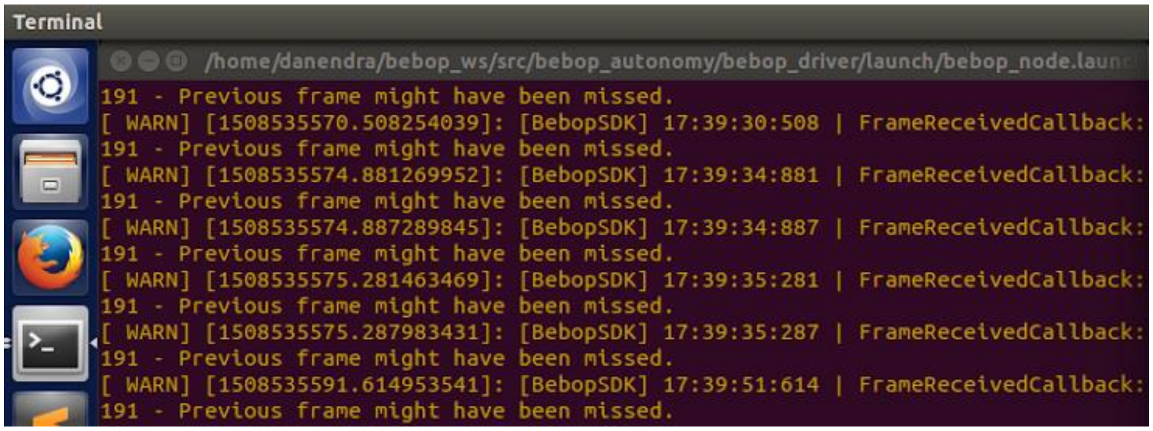
1. Get following ubuntu Packages:
    a. sudo apt-get install build-essential python-rosdep python-catkin-tools

    b. Follow these steps to Clone Parrot AR SDK
    ```
    # Create and initialize the workspace
    $ mkdir -p ~/bebop_ws/src && cd ~/bebop_ws
    $ catkin init
    $ git clone https://github.com/AutonomyLab/bebop_autonomy.git src/bebop_autonomy
    # Extra step for 64bit Ubuntu Xenial until parrot_arsdk debian package build fixed
    $ git clone https://github.com/AutonomyLab/parrot_arsdk.git src/parrot_arsdk
    # Update rosdep database and install dependencies (including parrot_arsdk)
    $ rosdep update
    $ rosdep install --from-paths src -i
    # Build the workspace
    $ catkin build
    ```

2. Connect to Bebop2 WIFI.
    a. Make sure that the ‘bebop_node.launch’ file has the correct IP address of the drone.

3. Run Bebop’s ROS driver as a Master Node:
    a. Source catkin workspace
    `source ~/bebop_ws/devel/setup.bash`

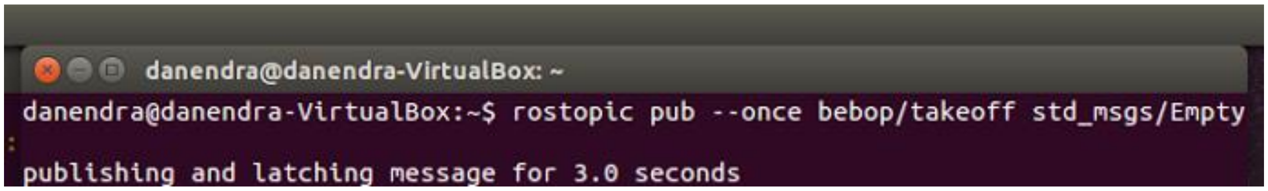
    b. roslaunch bebop_driver bebop_node.launch
        i. Ignore if you get few warnings, the input might be missing some camera feed frames. This can be seen in Figure 2.



```
Terminal
/home/danendra/bebop_ws/src/bebop_autonomy/bebop_driver/launch/bebop_node.launc
191 - Previous frame might have been missed.
[ WARN] [1508535570.508254039]: [BebopSDK] 17:39:30:508 | FrameReceivedCallback:
191 - Previous frame might have been missed.
[ WARN] [1508535574.881269952]: [BebopSDK] 17:39:34:881 | FrameReceivedCallback:
191 - Previous frame might have been missed.
[ WARN] [1508535574.887289845]: [BebopSDK] 17:39:34:887 | FrameReceivedCallback:
191 - Previous frame might have been missed.
[ WARN] [1508535575.281463469]: [BebopSDK] 17:39:35:281 | FrameReceivedCallback:
191 - Previous frame might have been missed.
[ WARN] [1508535575.287983431]: [BebopSDK] 17:39:35:287 | FrameReceivedCallback:
191 - Previous frame might have been missed.
[ WARN] [1508535591.614953541]: [BebopSDK] 17:39:51:614 | FrameReceivedCallback:
191 - Previous frame might have been missed.
```

Figure 2: Missing Frame Warnings while running master node

4. Send Commands to Bebop:
   a. Take-off:
      i. rostopic pub --once bebop/takeoff std_msgs/Empty

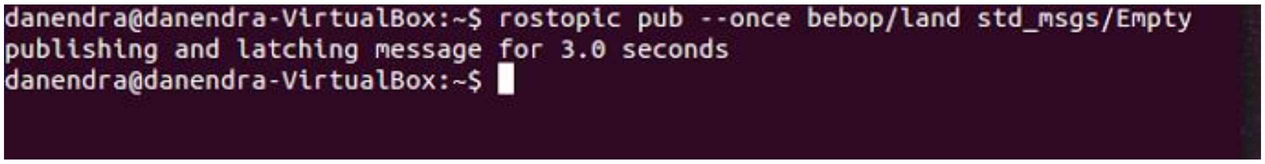


```
danendra@danendra-VirtualBox: ~
danendra@danendra-VirtualBox:~$ rostopic pub --once bebop/takeoff std_msgs/Empty

publishing and latching message for 3.0 seconds
```

Figure 3: Take-Off command to Bebop2

   b. land:
      i. rostopic pub --once bebop/land std_msgs/Empty



```
danendra@danendra-VirtualBox:~$ rostopic pub --once bebop/land std_msgs/Empty
publishing and latching message for 3.0 seconds
danendra@danendra-VirtualBox:~$
```

Figure 4: Land command to Bebop2

   c. View Camera Feed:
      i. rqt_image_view

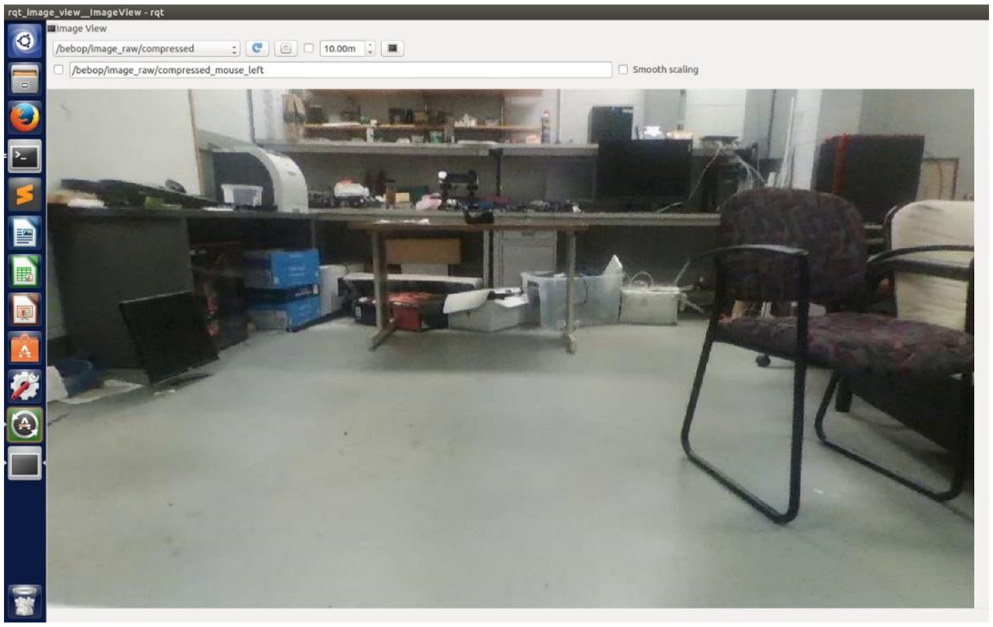

rqt_image_view__ImageView - rqt
Image View
/bebop/image_raw/compressed
10.00m
/bebop/image_raw/compressed_mouse_left
Smooth scaling

Figure 5: Video frame received from Bebop2

Challenges

The biggest challenge we faced since the last review was to get the husky moving. Initially we were able to receive the communication data from the Husky but were unable to send movement commands to it. However, after working on the Husky we got from NREC, we figured out that the problem was in the battery extension cable we were using. One of the ports of the cable was not supplying power to the Husky motors and hence it was not moving. Finally, after lot of debugging we were able to move the Husky.

Individually, I faced few challenges while working with the Bebop2 drone. Initially, since I was using Virtual Box for Linux, I was not able to connect to Bebop2 and run the Bebop master ROS node. This took up lot of my time to figure out the problem. I tried changing the network connection settings in Linux OS and bebop Launch file but got no success. Later, I found out that the default connection type in Virtual Box is 'NAT' type. However, to successfully connect in Linux to the Bebop which is originally connected via the Windows WiFi , the connection type in Virtual Box should be of the type 'Bridged'. This solved the connection problem in Linux.

Additionally, we tried testing the drone for takeoff and landing commands during the night and encountered that the drone losses its stability and moves in random direction. We initially thought that this is happening because the drone is not able to receive the GPS signals for localization and hence tried testing it in the open. However, we got the same random movements. On researching about the problem, we figured out that our thought process was wrong and got to know that Bebop uses visual odometry to stabilize itself using a vertical downward camera. Hence, during low-light conditions or night, the camera is not able to distinguish between captured frames and hence is unable to perform visual odometry.

Another problem we faced was to discuss the scope of our project with our sponsor. Our sponsor, Prof. Katia was not happy with the scope we had already defined in the first few meetings and wanted us to change the use case. Her suggestions were logical and valuable but was different from what we have originally decided. So, we had to incorporate some of these changes with our original action plan. 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 to navigate autonomously to a desired path using paths generated by the UAV.

<u>Teamwork</u>

Yuchi worked on the Husky and Bebop2 drone. He was able to control the Bebop2 drone via SDK on Linux and demonstrated takeoff and landing. He also took the initiative to setup the Husky software alongwith Rahul.

Pulkit, Pratibha and Rahul worked on the Husly and were able to diagnose and solve the communication issues of the Husky. They tried a lot of debugging on the software side and were able to finally make the Husky running. They also configured a Logitech controller to control the movements of the Husky.

I worked to control the Bebop2 via ROS. Pratibha helped me debug the errors and was also responsible for carrying out flight testing with me.

<u>Future plans</u>

As my next PR goal, I will work with Yuchi to implement high-level control for Bebop2. Since our end goal is to make the drone autonomously fly to a desired location, we plan to implement ROS commands to direct the Drone to Fly to a GPS location input by the user. We plan to take user input for GPS location, provide roll-pitch-yaw commands to make it fly to the desired GPS location.

I also plan to check the network connectivity robustness with the Bebop2. Although the manufactures claim that the Bebop has 250 m, we wanted to make sure that we have robust connection between drone and the laptop to prevent the drone straying away from us.

Pulkit will work on developing higher-level controls for the Husky using encoders or IMU so that we are able to move the Husky in a desired direction for a desired distance. Pratibha and Rahul will work on incorporating GPS data into the ROS software. We plan to use MediaTek GTPA010 from the MRSD inventory and create a ROS node that is capable of reading the GPS information from serial link.