INDIVIDUAL LAB REPORT 8

Progress Review - 9

Shubham Garg

TEAM H (PHOENIX)

Teammates:

Akshit, Parv, Steve

October 10, 2019

Individual Contribution:

1. Trade study of different available WiFi routers
2. Sensor fusion of T265 tracking camera, wheel odometry & IMU (3DM-GX5-45 GNSS/INS)
3. Establishing wireless communication link between Husky (AGV) and DJI (UAV)

Trade study of different available WiFi routers:

Most WiFi router usually works on 2.4 GHz or 5 GHz because they are in ISM radio bands which are reserved internationally for the use of radio frequency energy for industrial, scientific and medical purposes other than telecommunications. While the use of other frequencies is regulated to prevent someone from setting up a transmitter that blocks radio signals and telephones from working.
Moreover, different things happen because of different frequencies. At 5 GHz, more data can be carried, because there is more number of bits that can be sent in one second. But the problem is it's harder for the higher frequency to go far. With the higher frequency, it is harder to move through solid objects like walls, and the energy dissipates faster with high-frequency signals versus lower frequency ones. While at 2.4 GHz, not as much information can be transmitted, but because it's not as high frequency, it can go further before the signal degrades. And for us (as per our performance requirement), the UAV and AGV should be able to communicate with each other within a 25m radius. So, I found TP-Link AC1750 Smart WiFi Router which operates on the required frequency and range.

Sensor fusion of T265 tracking camera, wheel odometry & IMU (3DM-GX5-45 GNSS/INS):

Figure 1. shows the integrated Husky hardware with UR5 arm and multiple stereo cameras. And I worked on fusing the multiple sensor data using robot localization.
Robot localization package in ROS is a very useful package for fusing any number of sensors using various flavors of Kalman Filters. There are two kinds of pose estimates, one is the robot's local position (which is continuous and drifts over time), and the other is the robot's estimated position globally (which is discontinuous but more accurate in the long run). And these pose estimates affect different transforms between the three coordinate frames of the map, odom and base link.
We can't solely rely on the wheel odometry for the pose estimation as wheels skids and it is also subjected to the errors caused by uncertainty in the components of the robot and unevenness of the surface. The way to solve this issue is to introduce redundancy since it

is less likely for multiple sensors to fail all at once. Moreover, each sensor has its strengths and weakness. Like in our case T265 tracking works accurately in the indoor environment while the pose estimates drift in the outdoor environment. But the combination of IMU & GPS works best in the outdoor environment. So, we are using complementary features of these sensors for accurate pose estimation in indoor & outdoor environments.
Robot localization ROS package in particular (the node we are using to fuse our data) provides two kinds of Kalman filter nodes (Extended Kalman Filter, and Unscented Kalman Filter (EKF and UKF)). UKFs are slower as compared to EKFs but more accurate for non-linear transformations. But we found that EKFs are just as effective as UKFs for sensor fusion, so we are just using EKF filter.
Husky is a wheeled, nonholonomic robot that works in a planar environment. So, we need to ignore the subtle effects of variations in the ground plane as it might be reported from an IMU. This is done by setting the 2D mode true in the robot localization node which effectively zeros out the 3D pose variables in every measurement. Another advantage in using this ROS package is that we can independently select different parameters from $(X, Y, Z, \theta, \phi, \psi, \dot{X}, \dot{Y}, \dot{Z}, \dot{\theta}, \dot{\phi}, \dot{\psi}, \ddot{X}, \ddot{Y}, \ddot{Z})$ for fusion. Inbuilt wheel encoders are used to estimate instantaneous X & Y velocity as well as the absolute pose information. However, we don't fuse the angular velocities from the wheel odometry and rely on IMU for the same. Similarly, we fuse the absolute pose, linear velocities & angular velocities from the tracking camera. Figure 2. shows the pose estimation before and after sensor fusion.

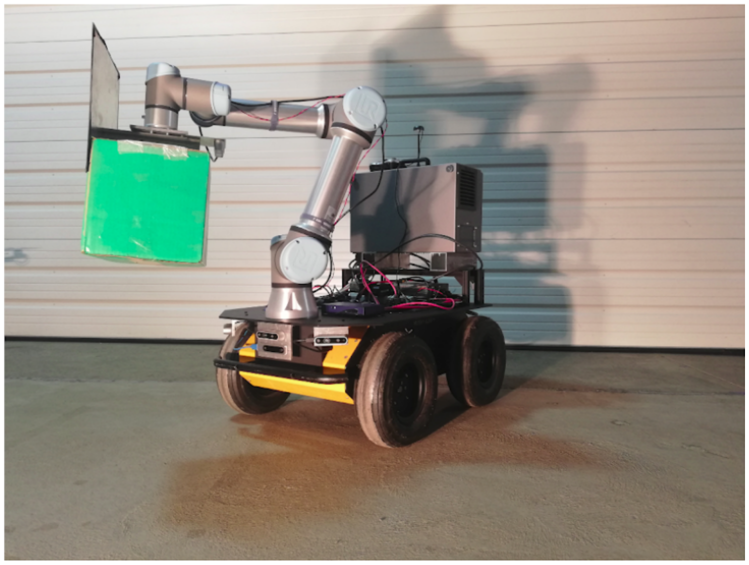
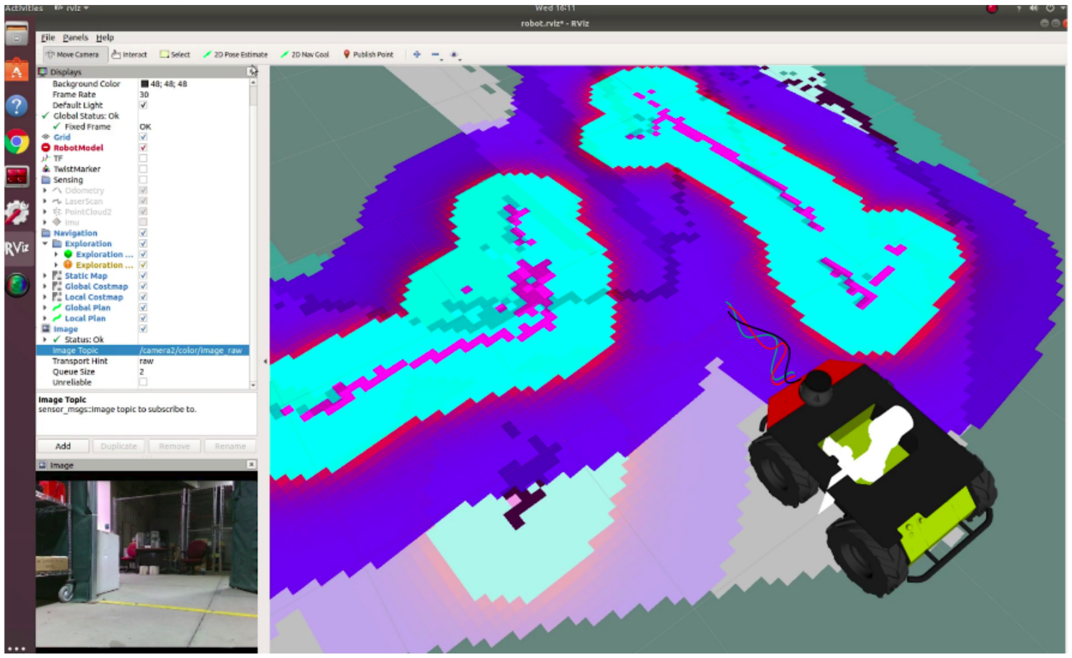


Figure 1: Integrated Husky hardware



```
Background Color      48; 48; 48
Frame Rate            30
Default Light         ✓
✓ Global Status: Ok
  ✓ Fixed Frame       OK
Grid                  ✓
RobotModel            ✓
TF
TwistMarker
Sensing
  Odometry            ✓
  LaserScan           ✓
  PointCloud2         ✓
  Imu
Navigation            ✓
  Exploration         ✓
    Exploration ...   ✓
    Exploration ...   ✓
  Static Map          ✓
  Global Costmap      ✓
  Local Costmap       ✓
  Global Plan         ✓
  Local Plan          ✓
Image                 ✓
  ✓ Status: Ok
  Image Topic         /camera2/color/image_raw
  Transport Hint      raw
  Queue Size          2
  Unreliable

Image Topic
sensor_msgs::Image topic to subscribe to.
```

Figure 2: Black line shows the trajectory with wheel odometry, red line show the trajectory with fused output & green line shows the actual trajectory (based on the given waypoints)

Establishing wireless communication link between Husky (AGV) and DJI (UAV):

For this, I had to understand the python socket programming. There are three main entities involved in socket programming – Socket Server, Socket Client, and Socket. A server is a software that waits for client requests and serves or processes them accordingly. On the other hand, a client is a requester of this service. A client program requests for some resources to the server and the server responds to that request. The socket is the endpoint of a bidirectional communications channel between server and client. Sockets may communicate within a process, between processes on the same machine, or between processes on different machines. For any communication with a remote program, we have to connect through a socket port.

So, I have designed both server and client model such that each can communicate with them. Communication flow looks like this.

1. Python socket server program executes at first and waits for any request.
2. Python socket client program will initiate the conversation at first.
3. Then server program will respond accordingly to client requests.
4. The client program will terminate if the user enters presses escape message. Server program will also terminate when client program terminates.

In our case, Husky acts as the server and DJI acts as the client. Both the nodes update a text file (which acts as the database).

Challenges:

Challenges faced in the last two weeks are discussed below:
1. It took a lot of time in debugging and testing the DJI drone using velocity controller where understanding the Airlab's core stack was most challenging.
2. Understanding how increasing the frequency for publishing the velocity commands helped.
3. As of now, our Husky is in collaboration with another DOE team. So, we need to properly schedule the testing. And we have to remove and reattach the UR5 arm on the Husky for their testing.
4. Integrating multiple sensors in the robot localization node and getting the required transformation between each sensor and the base link.
5. Understanding socket programming for WiFi communication.

Teamwork

Initially, we all debugged the issue faced in controlling the DJI drone using a velocity controller. And we found that increasing the update
Shubham worked on calibrating the T265 camera and D425i camera mounted on the UR5 arm. Further, he worked on fusing the sensor data from wheel odometry, IMU and the T265 tracking camera.
Akshit & **Parv** performed the WiFi range test between the UAV and UGV to ensure that they can communicate with each other within a 25m radius.
Akshit & **Shubham** worked on setting up the WiFi router as a base station with static IP assignment for Husky and DJI.
Shubham wrote the script to establish the wireless communication link between UAV and AGV. **Akshit** helped in making the communication between UAV and AGV completely asynchronous by using multithreading.
Shubham collected the point cloud data for Steve to test his door detection algorithm.
Steve worked on writing the door detection algorithm using stitched point cloud data.
Parv and **Akshit** extensively worked on debugging and testing the UAV velocity controller and image-based visual servoing.
Parv also performed few tests to conclude that the NUC does not have enough processing power to fuse the data at the input frame rate i.e. 2Hz.

Future plans

1. **Shubham** and **Parv** will be working on cleaning up the Husky Software as the current Husky is being shared with another team. We will also work on writing and testing the scripts for longer missions.
2. **Shubham** will be responsible for testing the sensor fusion extensively for longer runs.
3. **Akshit** will work on testing the pose controller on DJI using AirLab's core stack.
4. **Steve** and **Shubham** will be responsible for writing the generic opening detection algorithm using point cloud data which can be used for both the window and door.
5. **Steve** and **Parv** will work on designing the extinguisher and procuring the parts for the same.
6. **Akshit** will be creating the mounts for extinguishers and will be responsible for testing it extensively.
7. **Akshit** and **Steve** will be responsible for testing the autonomous UAV opening detection and entering through it with the new strategy using a position controller instead of velocity controller.