Individual Lab Report- 9

By Danendra Singh

Team F: Falcon Eye

Danendra Singh

Yuchi Wang

Pulkit Goyal

Pratibha Tripathi

Rahul Ramakrishnan

22nd March 2018

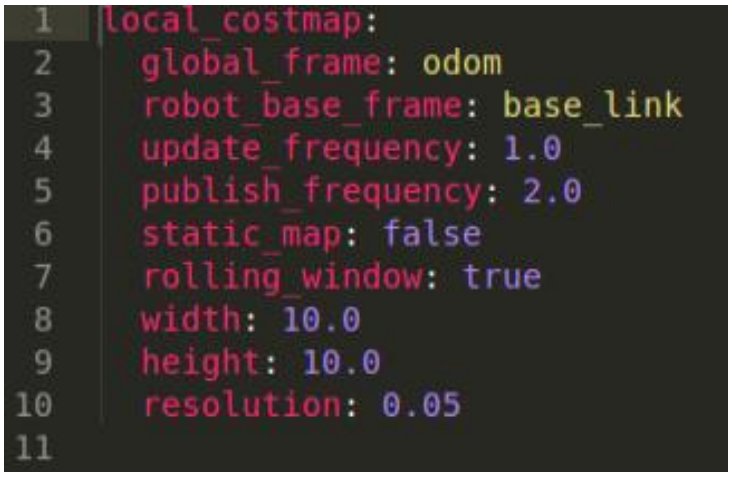
1. Individual Progress:

As my contribution to the MRSD Project for progress review 10, I have successfully developed Cost map for GPS based waypoint navigation and Obstacle avoidance. This involved developing Global and Local Cost maps based on ROS costmap_2d package and integrating the 2-D Hokuyo lidar for the package. Additionally, I worked with Pulkit, Pratibha, and Rahul to implement the husky waypoint_nav package that is built on ROS navigation stack.

1.1 Costmap Generation:

Using costmap_2d package in ROS, we were able to generate a 2D cost map that takes in Hokuyo LIDAR sensor data, builds a 2D occupancy grid of the data, and inflate the costs in the 2D cost map based on the occupancy grid and a specified inflation radius.

Hokuyo's data is used to either mark (insert obstacle information into the cost map), clear (remove obstacle information from the cost map), or both. A marking operation is just an index into an array to change the cost of a cell. A clearing operation, however, consists of raytracing through a grid from the origin of the sensor outwards for each observation reported. Figure 1 depicts the parameters we set in the local_costmap.yaml file.

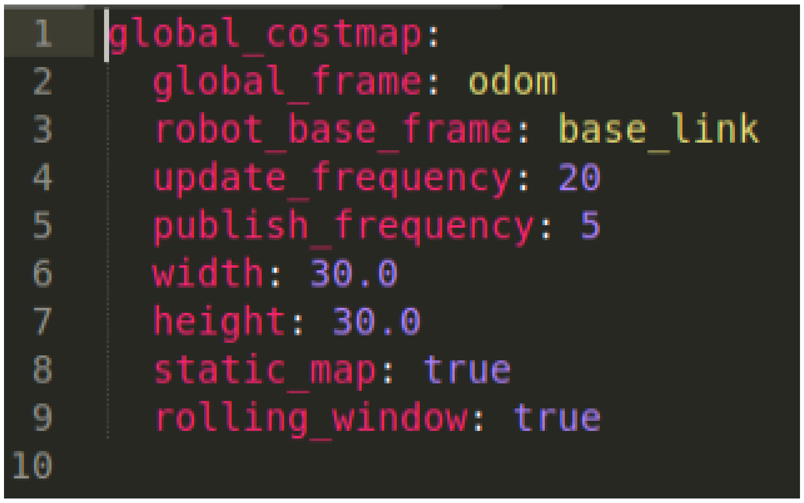


```
local_costmap:
  global_frame: odom
  robot_base_frame: base_link
  update_frequency: 1.0
  publish_frequency: 2.0
  static_map: false
  rolling_window: true
  width: 10.0
  height: 10.0
  resolution: 0.05
```

Figure 1: Local Cost Map parameters

The package can take in multiple sensor inputs and based on the sensor used it can generate 3-D cost map as well. Our next test would be to evaluate the data from the Velodyne-Puck LIDAR.

We are using a publish frequency of 5 Hz to read the data from the sensor and update the cost map. Figure 2 depicts the parameters set for global_costmap.yaml

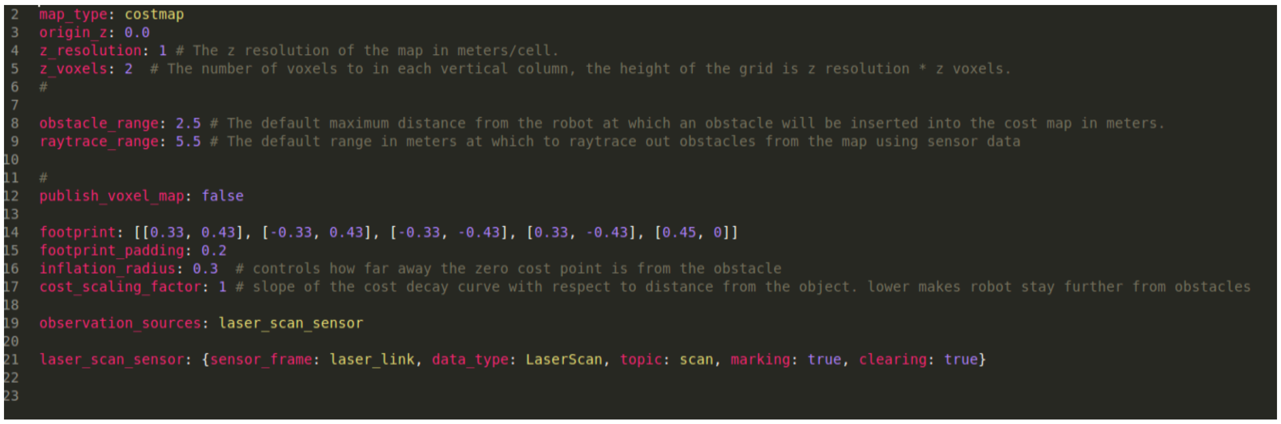


```yaml
global_costmap:
  global_frame: odom
  robot_base_frame: base_link
  update_frequency: 20
  publish_frequency: 5
  width: 30.0
  height: 30.0
  static_map: true
  rolling_window: true

```

Figure 2: Global Cost Map parameters

Since we would be operating outdoors, instead of using a static map in which the robot can map and avoid obstacles, we are using a rolling_window parameter. This sets up a map centered at the Husky whose size can be adjusted as a width and height parameter. As the robot moves too far away from the current area, previous obstacles are dropped from the map and new ones are inserted.

This suite our purpose as we care only for the obstacles in a local vicinity of the area to be avoided. Figure 3 explains the general parameters we set in our general_costmap.yaml file. These include footprint of our robot, obstacle inflation, sensor source etc.



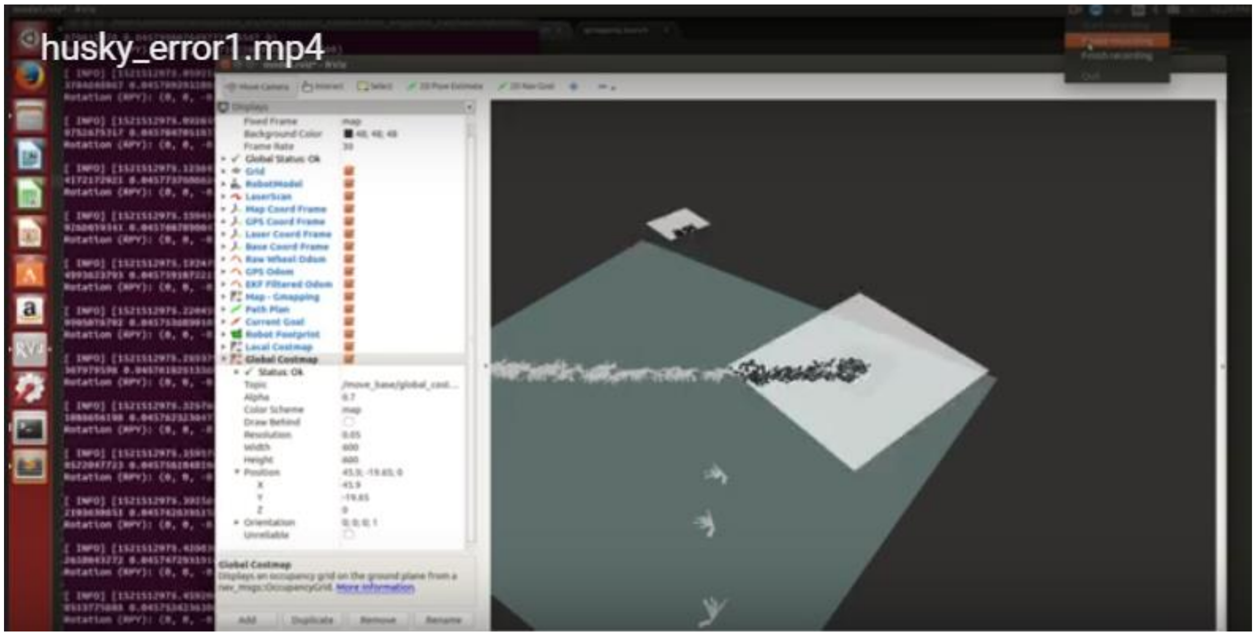
```yaml
map_type: costmap
origin_z: 0.0
z_resolution: 1 # The z resolution of the map in meters/cell.
z_voxels: 2  # The number of voxels to in each vertical column, the height of the grid is z resolution * z voxels.
#

obstacle_range: 2.5 # The default maximum distance from the robot at which an obstacle will be inserted into the cost map in meters.
raytrace_range: 5.5 # The default range in meters at which to raytrace out obstacles from the map using sensor data

#
publish_voxel_map: false

footprint: [[0.33, 0.43], [-0.33, 0.43], [-0.33, -0.43], [0.33, -0.43], [0.45, 0]]
footprint_padding: 0.2
inflation_radius: 0.3  # controls how far away the zero cost point is from the obstacle
cost_scaling_factor: 1 # slope of the cost decay curve with respect to distance from the object. lower makes robot stay further from obstacles

observation_sources: laser_scan_sensor

laser_scan_sensor: {sensor_frame: laser_link, data_type: LaserScan, topic: scan, marking: true, clearing: true}


```

Figure 3: General Sensor settings and defining robot footprint

1.2 Husky Waypoint_Nav

The Waypoint_nav package of Clearpath Husky combines ekf_localization to fuse odometry data with IMU and GPS data - navsat_transform to convert GPS data to odometry and to convert latitude and longitude points to the robot's odometry coordinate system - GMapping to create a map and detect obstacles - move_base to navigate to the goals while avoiding obstacles.

Initially, we were getting unacceptable hoping by the husky between two frames within the Rviz visualization as shown in Figure 4.



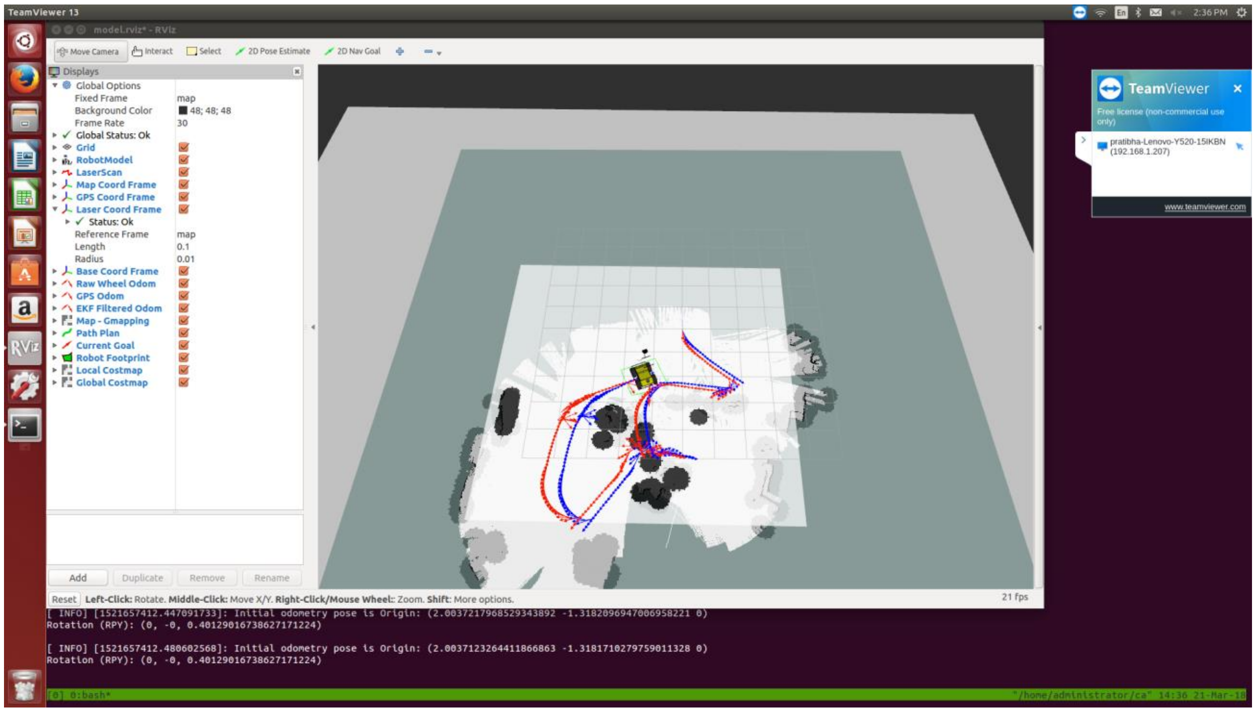
```
Fixed Frame          map
Background Color     48; 48; 48
Frame Rate           30
Global Status: Ok
Grid
RobotModel
LaserScan
Map Coord Frame
GPS Coord Frame
Laser Coord Frame
Base Coord Frame
Raw Wheel Odom
GPS Odom
EKF Filtered Odom
Map - Gmapping
Path Plan
Current Goal
Robot Footprint
Local Costmap
Global Costmap
  Status: Ok
  Topic              /move_base/global_cost...
  Alpha              0.7
  Color Scheme       map
  Draw Behind
  Resolution         0.05
  Width              600
  Height             600
  Position           45.9; -19.65; 0
    X                45.9
    Y                -19.65
    Z                0
  Orientation        0; 0; 0; 1
  Unreliable
```

Global Costmap
Displays an occupancy grid on the ground plane from a nav_msgs::OccupancyGrid. More Information.

Figure 4: Husky frame hopping between two map positions

After a long haul of trying various debugging options, we were suggested that this hopping could be because of 2 different launch files publishing EKF values. This indeed turned out to be true. The Husky-ROS-Indigo package had a package for Husky_control having the same launch file- ekf_localization as the one in our Waypoint_nav package.

ROS kind of makes a cache of a launch file and hence when we were launching our waypoint_nav EKF launch file, the other launch file in the system was also being launched and hence the observed hopping was occurring.We solved this problem by commenting out the system EKF launch file and hence the problem was mitigated. The results after eliminating this issue were desirable to our application as shown in Figure 5. Here black spots denote obstacles, blue lines depict EKF localization path and red lines depict odometry path. Figure 6 shows that Husky is stable at its origin and registering obstacles in cost map.

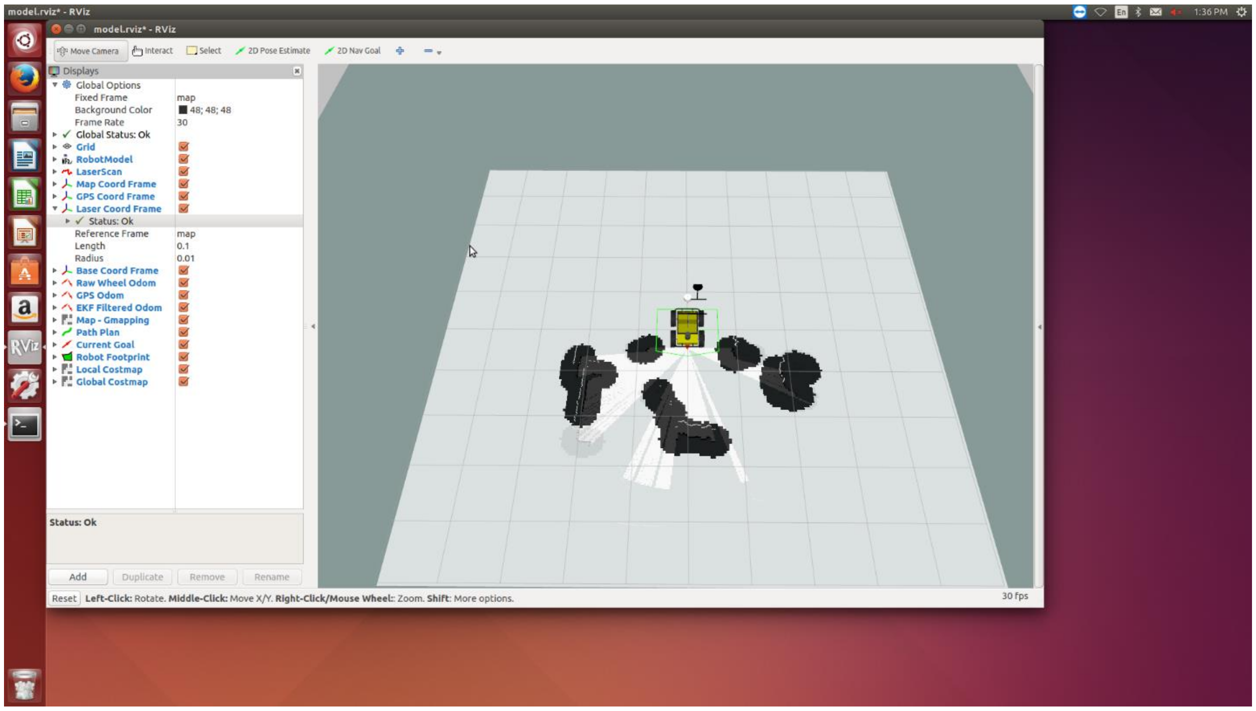


```
Global Options
  Fixed Frame        map
  Background Color   48; 48; 48
  Frame Rate         30
Global Status: Ok
Laser Coord Frame
  Status: Ok
  Reference Frame    map
  Length             0.1
  Radius             0.01
```

Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel:: Zoom. Shift: More options.

```
[ INFO] [1521657412.447091733]: Initial odometry pose is Origin: (2.0037217968529343892 -1.3182096947006958221 0)
Rotation (RPY): (0, -0, 0.40129016738627171224)

[ INFO] [1521657412.480602568]: Initial odometry pose is Origin: (2.0037123264411866863 -1.3181710279759011328 0)
Rotation (RPY): (0, -0, 0.40129016738627171224)

[0] 0:bash*
```

Figure 5: Correct Waypoint_nav implementation

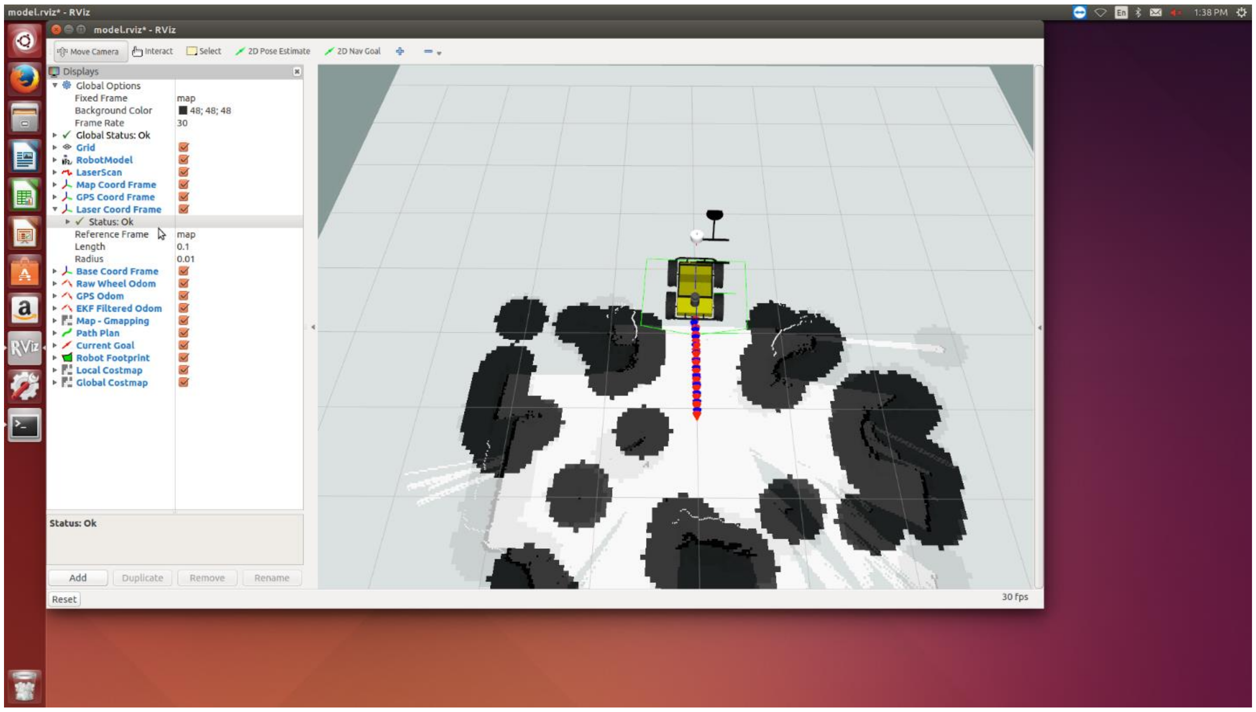


```
Global Options
  Fixed Frame        map
  Background Color   48; 48; 48
  Frame Rate         30
Global Status: Ok
Laser Coord Frame
  Status: Ok
  Reference Frame    map
  Length             0.1
  Radius             0.01
Status: Ok
```

Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel:: Zoom. Shift: More options.

Figure 6: Husky stable at its origin and registering nearby obstacles

Figure 7 depicts the husky moving in straight line and registering new obstacles. Observe exact overlap between EKF path(blue line) and odometry path(red line).



Global Options	
Fixed Frame	map
Background Color	■ 48; 48; 48
Frame Rate	30
Global Status: Ok	
Grid	✓
RobotModel	✓
LaserScan	✓
Map Coord Frame	✓
GPS Coord Frame	✓
Laser Coord Frame	✓
Status: Ok	
Reference Frame	map
Length	0.1
Radius	0.01
Base Coord Frame	✓
Raw Wheel Odom	✓
GPS Odom	✓
EKF Filtered Odom	✓
Map - Gmapping	✓
Path Plan	✓
Current Goal	✓
Robot Footprint	✓
Local Costmap	✓
Global Costmap	✓

Status: Ok

Figure 7: Straight line movement from Figure 6 pose.

2. <u>Challenges</u>

For this PR, the biggest challenge we faced was integrating the navigation stack on the operating system. Debugging the Husky's frame hopping problem took most of our time since the last PR and hence reduced our time to test our system for robustness.

One common problem since the starting of the semester has been the bad weather for testing outside. However, the past couple of weeks have been little better, allowing us to test our drone outside and record test results. We are hopeful that the weather in coming weeks would be better than before and align with our goals to perform unit tests and overall system testing.

Another big risk or challenge is that we are predicting that the course loads from other courses are going to increase in the coming weeks and hence we are trying to make up a thorough work schedule that allows us to keep a balance between our MRSD project goals and other assignments/projects.

3. Teamwork

Yuchi worked on testing the exploitation algorithm for the drone in real test scenario outside. We tested our system by not just registering the GPS locations of the April Tags and navigating the drone to the last April Tag location, but also test whether the drone rejects an incomplete tree of April Tags to the target location and rather chooses a complete traversable path to the target. The results were what we expected and met our requirements.

Pratibha, Pulkit, Rahul and I worked on developing the software stack for obstacle detection and avoidance, and navigating to the GPS locations supplied by the drone.

Thus, by defining each member's goal successfully and working together as a strong team, we could achieve all the tasks for the PR-10.

4. Future plans

As discussed in the last team meeting, we moved the UGV drone proxy navigating to the destination while avoiding obstacles to the next PR and instead were able to demonstrate a robust path planning and deliberative obstacle avoidance with the UGV.

Hence, as a part of the next PR goal, we have to test the UGV drone proxy navigation. We have been able to detect and successfully avoid obstacles using a planar 2-D lidar. I am planning to test the same system by integrating the Velodyne-Puck Lidar that we decided to use initially, and test how does our system behave. We expect to detect obstacles form a farther distance and hence better plan the UGV's movement.

Another task to perform is to integrate software stack for the Husky and Bebop so that they are able to communicate directly with each other and share intelligence. Yuchi will be implementing this. Pulkit and Rahul will work to test our Husky navigation stack with GPS and improve localization for path planning. I and Pratibha will work to implement the drone proxy UGV navigation goal with the current navigation stack.