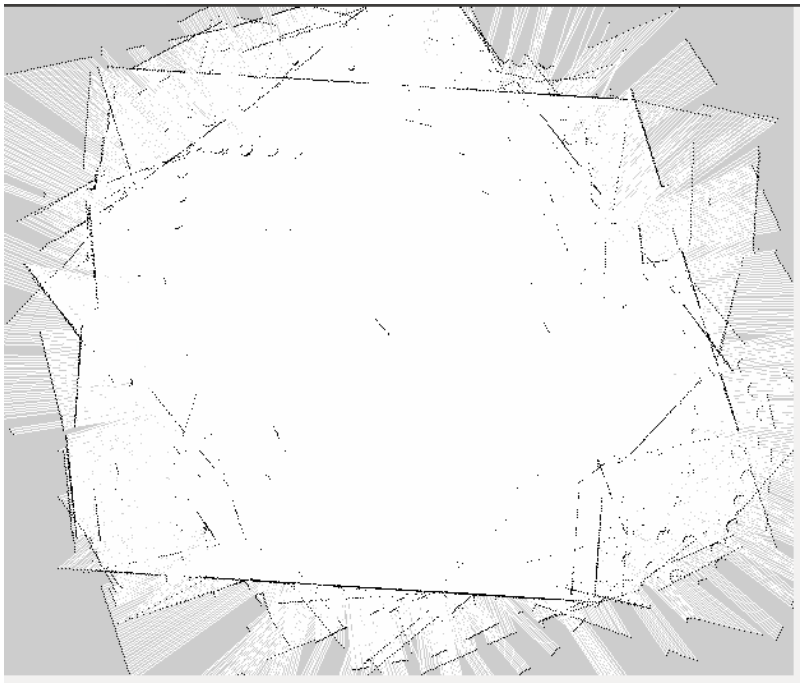
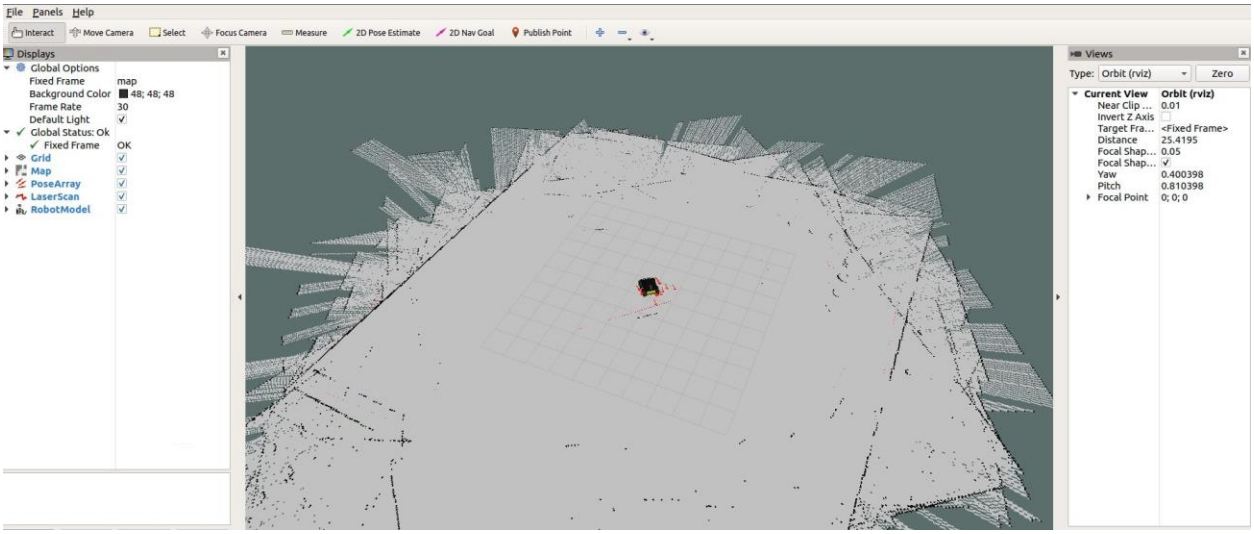
Lab#9

Student ID: 21060007

Task1->

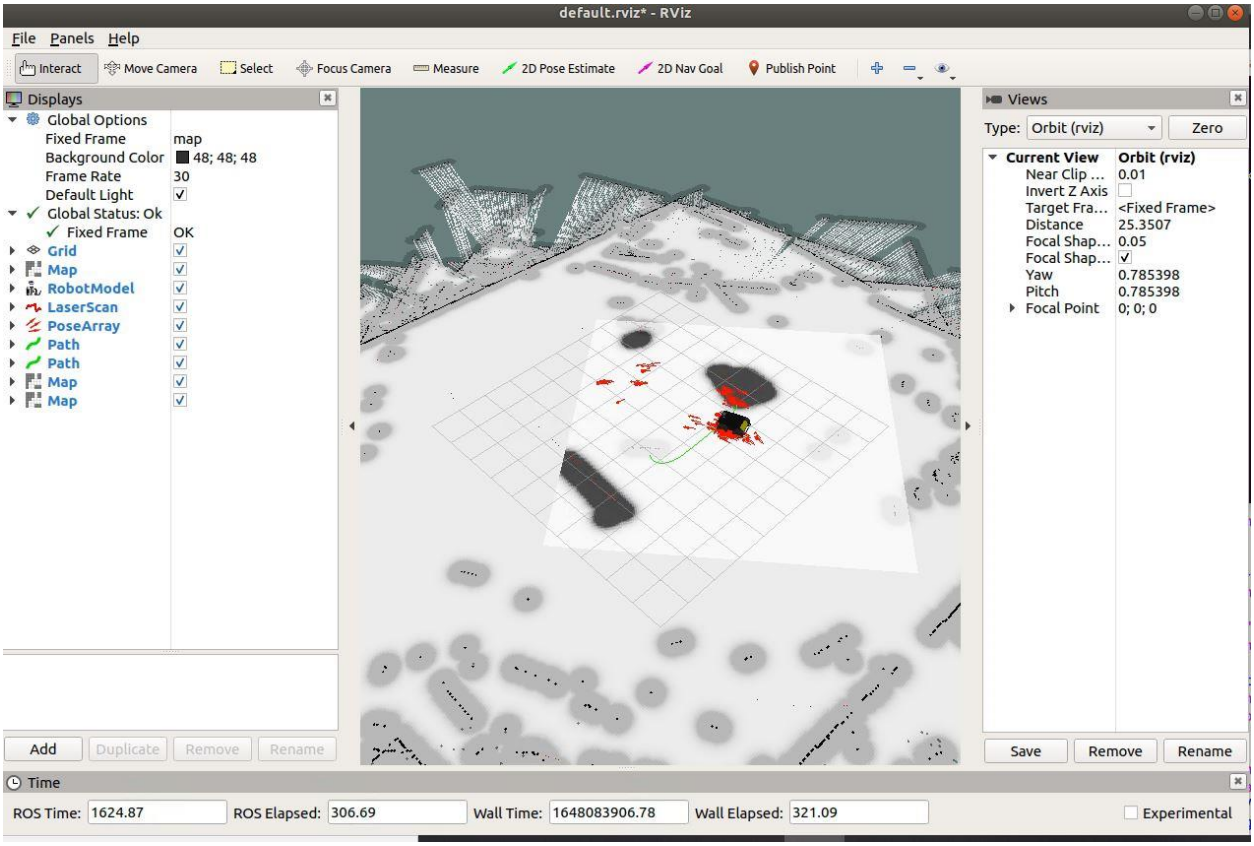
Map:





Observations: The number of particles for min and max range in amcl.launch file allows improved estimation of the robot pose as well as accurate movement of the robot with teleop, however it slows the system as well.

Task2:



Observations:

Inflation radius: The inflation radius should be set to the maximum distance from obstacles at which a cost should be incurred. For example, setting the inflation radius at 0.55 meters means that the robot will treat all paths that stay 0.55 meters or more away from obstacles as having equal obstacle cost

Obstacle_range: determines the maximum range sensor reading that will result in an obstacle being put into the costmap. Here, we have it set at 2.5 meters, which means that the robot will only update its map with information about obstacles that are within 2.5 meters of the base.

Raytrace_range: parameter determines the range to which we will raytrace freespace given a sensor reading. Setting it to 3.0 meters as we have above means that the robot will attempt to clear out space in front of it up to 3.0 meters away given a sensor reading.

Task3:

```
#!/usr/bin/env  python
import rospy
import actionlib
from move_base_msgs.msg import MoveBaseAction,     MoveBaseGoal
waypoints=[[(2.1,2.2,0.0),(0.0,    0.0,0.0,1.0)],[(6.5,4.43,0.0),(0.0,0.0,-0.984047240305,
          0.177907360295)]]
def goal_pose(pose):
```

```
        goal_pose=MoveBaseGoal()
        goal_pose.target_pose.header.frame_id          =        'map'
        goal_pose.target_pose.pose.position.x  =        pose[0][0]
        goal_pose.target_pose.pose.position.y  =        pose[0][1]
        goal_pose.target_pose.pose.position.z  =        pose[0][2]
        goal_pose.target_pose.pose.orientation.x       =        pose[1][0]
        goal_pose.target_pose.pose.orientation.y       =        pose[1][1]
        goal_pose.target_pose.pose.orientation.z       =        pose[1][2]
        goal_pose.target_pose.pose.orientation.w       =        pose[1][3]
        return goal_pose


if __name__=='__main__':
        rospy.init_node('patrol')
        client    =actionlib.SimpleActionClient('move_base',MoveBaseAction)
        client.wait_for_server()

while True:
        for pose in waypoints:
                goal=goal_pose(pose)
                client.send_goal(goal)
         client.wait_for_result()
                print("done")
```