AA 274A: Principles of Robot Autonomy I
Section 5

Name: Jacob Azoulay SUID: 06493371
Name: Mario Peraza SUID:06282995
Name: Pol Francesch Huc SUID: 006650276
Name: Sunny Singh SUID: 06663365

Due to time constraints in section, my team and I were only able to complete up to question 4. I re-worked this section on my own using genbu.

Problem 1

Subscribes to:

- **/map**: Subscribes to the node in order to determine which gridspace cell the robot is currently in.
- **/map_metadata**: Subrcribes to this node in order to obtain information about the map shape (i.e. grid length, width, and origin).
- **/cmd_nav**: Goal end point containing an x, y, and theta.

Publishes to:

- **/planned_path**: Publishes the path returned from A* if the time to the goal is less with this new path than with the old one.
- **/cmd_smoothed_path**: Publishes the path from A* with a cubic spline fit to it if the time to the goal is less with this new path than with the old one.
- **/cmd_smoothed_path_rejected**: Publishes the path from A* with a cubic spline fit to it if the time to the goal is greater with this new path than with the old one. Likely published for diagnostic purposes.
- **/cmd_vel**: Control inputs required to follow the current path.

Problem 2

Each mode of the state machine does ...

- **IDLE**: Holds the robot's current position until a goal and a map are provided and then switches into either the TRACK or ALIGN mode depending on pose.
- **ALIGN**: Runs the heading controller until the robot is sufficiently aligned with the starting direction and then switches to the TRACK state.

- **TRACK**: Runs the trajectory controller. If near the goal, then we switch to park mode. Otherwise, we replan since we are far from the start. If out of alignment, we switch back to the ALIGN mode. If none of those apply, then if the planned path takes too long, then we replan.
- **PARK**: Runs the pose controller. If the robot is at the goal, you IDLE and reset the goal to none.

Problem 3

The command to create a new package is ”catkin_create_pkg section5 std_msgs rospy message_generation”. Where the last three commands are the dependencies and ”section5” is the name.

Problem 4

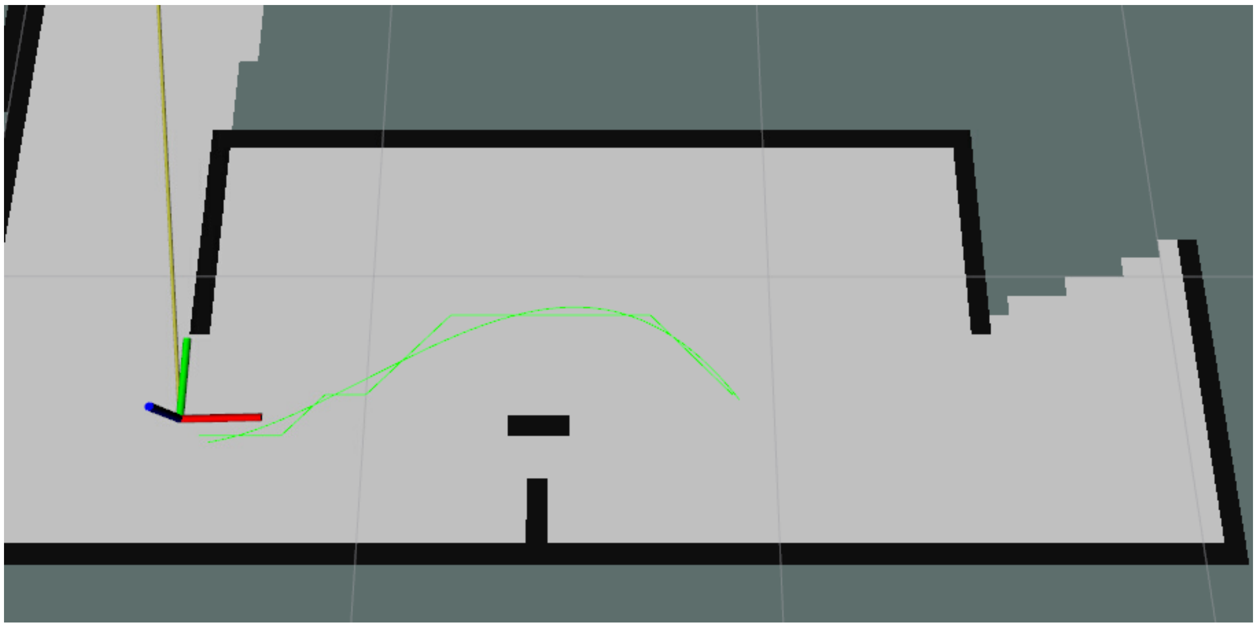

Figure 1: Rviz screenshot of the robot going back and forth in the starting area

Problem 5

The goal visualizer works by getting the location from the /cmd_nav node and publishing a marker object to the marker_node topic. The goal is updated automatically as the subscriber watches for any updates to the /cmd_nav topic and will call the publisher with the new goal location as soon it is detected.

```python
#!/usr/bin/env python3

import rospy
from visualization_msgs.msg import Marker
from geometry_msgs.msg import Pose2D

def subscriber():
    rospy.init_node('marker_node', anonymous=True)
    pub = rospy.Publisher('marker_topic', Marker, queue_size=10)
    sub = rospy.Subscriber('/cmd_nav', Pose2D, publisher, (pub))
    rospy.spin()


def publisher(data, pub):
    marker = Marker()

    marker.header.frame_id = "map"
    marker.header.stamp = rospy.Time()

    # IMPORTANT: If you're creating multiple markers,
    #            each need to have a separate marker ID.
    marker.id = 0

    marker.type = 2 # sphere

    marker.pose.position.x = data.x
    marker.pose.position.y = data.y
    marker.pose.position.z = 0

    marker.pose.orientation.x = 0.0
    marker.pose.orientation.y = 0.0
    marker.pose.orientation.z = 0.0
    marker.pose.orientation.w = 1.0

    marker.scale.x = 0.1
    marker.scale.y = 0.1
    marker.scale.z = 0.1

    marker.color.a = 1.0 # Don't forget to set the alpha!
    marker.color.r = 0.0
    marker.color.g = 1.0
    marker.color.b = 0.0

    pub.publish(marker)
    print('Published marker!')


if __name__ == '__main__':
    try:
        subscriber()
    except rospy.ROSInterruptException:
        pass
```

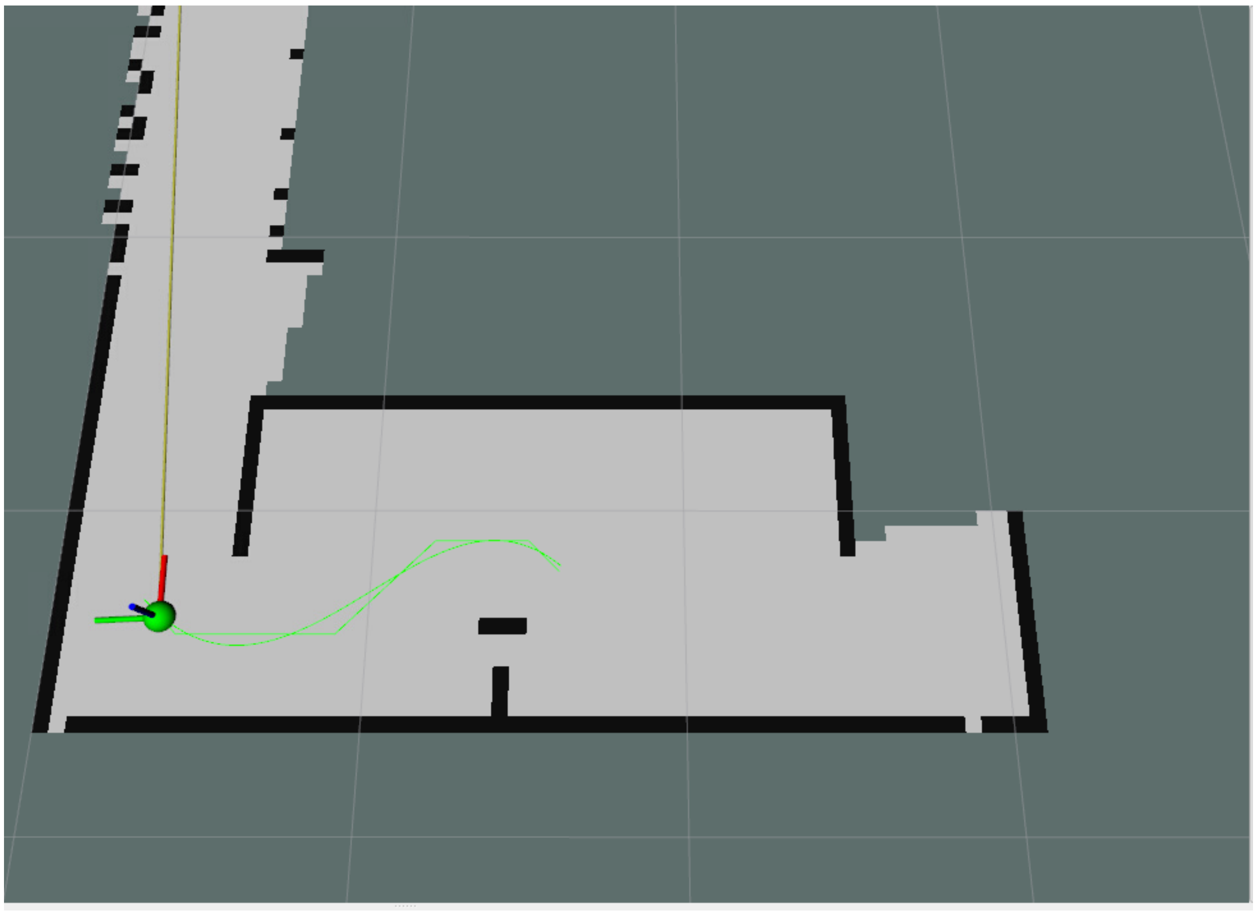

Figure 2: Rviz screenshot of the robot lining up with one of the hallways with the goal location marked.

Problem 6

This launch file has two components which launch a node each. The first node is the marker, and it effectively just runs the marker_pub.py script. The second node will launch rviz using a config file at the location specified by the argument.

As an example I used the root.launch launch file which included how to start up python scripts and rviz.

```
<launch>
    <node name="marker" pkg="section5" type="marker_pub.py" />

    <node name="rviz" pkg="rviz" type="rviz"
            args="-d $(find section5)/rviz/my_nav.rviz" />
</launch>
```