AA274A Section 5

Problem 1

Subscribes to

- /map
- /map_metadata
- /cmd_nav

/map has type nav_msgs.msg.OccupancyGrid. Represents occupancy on a grid map in the form of a uint8 list in row-major order. Cells have range [0,100] representing probability of occupancy.
/map_metadata has type nav_msgs.msg.MapMetaData. Stores height and width of map in cells, size of cell in meters, cell which corresponds to origin of world coordinates, and time of map load
/cmd_nav has type type geometry_msgs.msg.Pose2D. Stores 2d general coordinates x, y, theta of goal state (pose). Used as destination when replanning motion at each time step.

Subscribes to these messages to perceive current world state.

Publishes to

- /planned_path
- /cmd_smoothed_path
- /cmd_smoothed_path_rejected
- /cmd_vel

First three have type nav_msgs.msg.PathMessage,
which has an array of geometry_msgs.msgs.PoseStamped,
which are timestamped geometry_msgs.msgs.Pose messages,
which contain a Point type position in x,y,z and Quarternion type orientation in x,y,z,w.
Timestamp is in std_msgs.msg.Header in the stamp field of type time.

/cmd_vel has type geometry_msgs.msgs.Twist, which consists of linear and angular velocities in free 3d space expressed as a generic type geometry_msgs.msgs.Vector3 which has fields x,y,z.

First 3 messages published mainly(?) for logging purposes. Last one commands the robot with the velocities corresponding to the current planned trajectory.

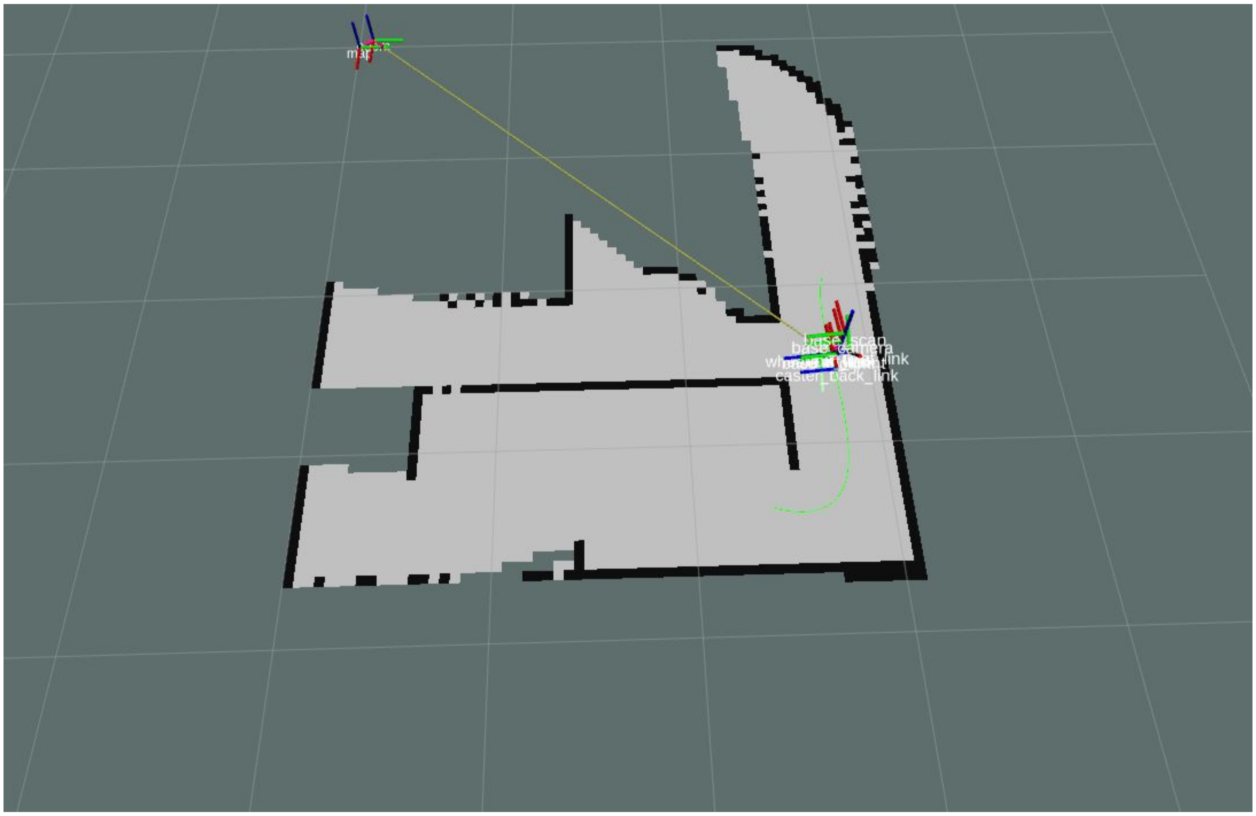
Problem 2

Observation: The robot assumes that the environment is fully observable and stationary.

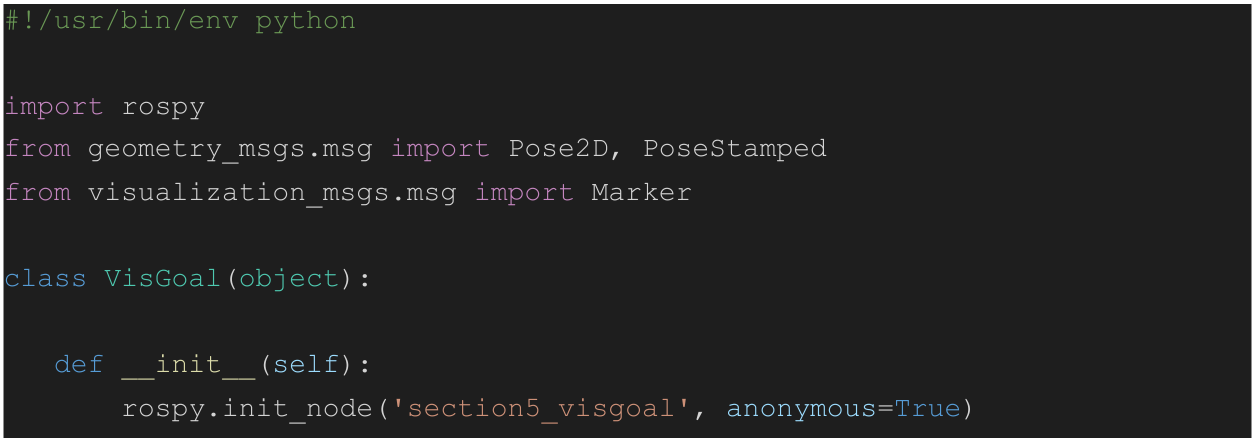
- Depending on the FSM state, the replan() call sets controls, which get published at the end of each run() loop.
- In IDLE state the robot does nothing. This is used as a sink state for FSM.
- Robot starts in ALIGN state during each replan() call, and stays in the state until self.aligned() is True. Otherwise switches to TRACK state.

- In TRACK state the robot mainly checks for conditions to replan, after having moved for a while or if it hasn't reached the goal when it expected to. When the robot is close to the goal, transitions to PARK state.
- In PARK state, the robot relies on the pose_controller (Lyapunov stability) to seek the goal state. When close enough, transitions to IDLE state.

Problem 3



Problem 4


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

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

class VisGoal(object):

   def __init__(self):
       rospy.init_node('section5_visgoal', anonymous=True)
```

```
        self.QUEUE_SIZE = 10

        # Messaging
        self.pub = rospy.Publisher('marker_topic', Marker,
queue_size=self.QUEUE_SIZE)
        rospy.Subscriber('/move_base_simple/goal', PoseStamped,
self.sub_callback)

        # State
        self.has_data = False

        self.marker = Marker()
        self.marker.header.frame_id = "map"
        self.marker.header.stamp = rospy.Time()
        # IMPORTANT: If you're creating multiple markers,
        #            each need to have a separate marker ID.
        self.marker.id = 0
        self.marker.type = 2 # sphere
        self.marker.pose.position.x = 1
        self.marker.pose.position.y = 1
        self.marker.pose.position.z = 0.2
        self.marker.pose.orientation.x = 0.0
        self.marker.pose.orientation.y = 0.0
        self.marker.pose.orientation.z = 0.0
        self.marker.pose.orientation.w = 1.0
        self.marker.scale.x = 0.1
        self.marker.scale.y = 0.1
        self.marker.scale.z = 0.1
        self.marker.color.a = 1.0
        self.marker.color.r = 1.0
        self.marker.color.g = 0.0
        self.marker.color.b = 0.0


    ## Callbacks

    def sub_callback(self, posestamped):
        # rospy.loginfo(posestamped)
        self.marker.pose.position.x = posestamped.pose.position.x
        self.marker.pose.position.y = posestamped.pose.position.y
        self.has_data = True

    def shutdown_callback(self):
        pass # do nothing

    def run(self):
        rate = rospy.Rate(10) # 10 Hz
        while not rospy.is_shutdown():

            if self.has_data:
                self.pub.publish(self.marker)

            rate.sleep()



if __name__ == '__main__':
    node = VisGoal()
    rospy.on_shutdown(node.shutdown_callback)
    node.run()
```

Problem 5

```
<launch>
    <node pkg="asl_turtlebot" type="navigator.py" name="navigator" />
    <node pkg="section5" type="visgoal.py" name="visgoal" />
    <!--<node pkg="rviz" type="rviz" name="rviz" args="-d
/home/lqkhoo/catkin_ws/src/section5/rviz/my_nav.rviz" />-->
    <node pkg="rviz" type="rviz" name="rviz" args="-d $(find
section5)/rviz/my_nav.rviz" />
</launch>
```