Solutions for Unit 4 Path Planning

Index:

Solution Exercise 4.4

Exercise 4.4

Launch File: send_goal_client.launch

In [ ]:

```
<launch>
    <node pkg="send_goals" type="send_goal_client.py" name="move_base_action_client" output="screen"
    </node>
</launch>
```

END Launch File: send_goal_client.launch

Python File: send_goal_client.py

In [ ]:

```
#! /usr/bin/env python
import rospy
import time
import actionlib
from move_base_msgs.msg import MoveBaseAction, MoveBaseGoal, MoveBaseResult, MoveBaseFeedback

# definition of the feedback callback. This will be called when feedback
# is received from the action server
# it just prints a message indicating a new message has been received
def feedback_callback(feedback):

    print('[Feedback] Going to Goal Pose...')

# initializes the action client node
rospy.init_node('move_base_action_client')

# create the connection to the action server
client = actionlib.SimpleActionClient('/move_base', MoveBaseAction)
# waits until the action server is up and running
client.wait_for_server()

# creates a goal to send to the action server
goal = MoveBaseGoal()
goal = MoveBaseGoal()
goal.target_pose.header.frame_id = 'map'
goal.target_pose.pose.position.x = 1.16
goal.target_pose.pose.position.y = -4.76
goal.target_pose.pose.position.z = 0.0
goal.target_pose.pose.orientation.x = 0.0
goal.target_pose.pose.orientation.y = 0.0
goal.target_pose.pose.orientation.z = 0.75
goal.target_pose.pose.orientation.w = 0.66

# sends the goal to the action server, specifying which feedback function
# to call when feedback received
client.send_goal(goal, feedback_cb=feedback_callback)

# Uncomment these lines to test goal preemption:
#time.sleep(3.0)
#client.cancel_goal()  # would cancel the goal 3 seconds after starting

# wait until the result is obtained
# you can do other stuff here instead of waiting
# and check for status from time to time
# status = client.get_state()
# check the client API link below for more info

client.wait_for_result()

print('[Result] State: %d'%(client.get_state()))
```

END Python File: send_goal_client.py

Solution Exercise 4.5

Exercise 4.5

Launch File: my_move_base_launch_1.launch

In [ ]:

```
<?xml version="1.0"?>
<launch>

  <!-- Run the map server -->
  <arg name="map_file" default="$(find husky_navigation)/maps/my_map.yaml"/>
  <node name="map_server" pkg="map_server" type="map_server" args="$(arg map_file)" />

  <!--- Run AMCL -->
  <include file="$(find husky_navigation)/launch/amcl.launch" />

  <!--- Run Move Base -->
  <include file="$(find my_move_base_launcher)/launch/my_move_base_launch_2.launch" />

</launch>
```

END Launch File: my_move_base_launch_1.launch

Launch File: my_move_base_launch_2.launch

In [ ]:

```
<?xml version="1.0"?>
<launch>

  <arg name="no_static_map" default="false"/>

  <arg name="base_global_planner" default="navfn/NavfnROS"/>
  <arg name="base_local_planner" default="dwa_local_planner/DWAPlannerROS"/>
  <!-- <arg name="base_local_planner" default="base_local_planner/TrajectoryPlannerROS"/> -->

  <node pkg="move_base" type="move_base" respawn="false" name="move_base" output="screen">

    <param name="base_global_planner" value="$(arg base_global_planner)"/>
    <param name="base_local_planner" value="$(arg base_local_planner)"/>
    <param name="base_local_planner" value="$(arg base_local_planner)"/>
    <rosparam file="$(find my_move_base_launcher)/params/my_move_base_params.yaml" command="load"/>

    <!-- observation sources located in costmap_common.yaml -->
    <rosparam file="$(find husky_navigation)/config/costmap_common.yaml" command="load" ns="global_c
    <rosparam file="$(find husky_navigation)/config/costmap_common.yaml" command="load" ns="local_co

    <!-- local costmap, needs size -->
    <rosparam file="$(find husky_navigation)/config/costmap_local.yaml" command="load" ns="local_cos
    <param name="local_costmap/width" value="10.0"/>
    <param name="local_costmap/height" value="10.0"/>

    <!-- static global costmap, static map provides size -->
    <rosparam file="$(find husky_navigation)/config/costmap_global_static.yaml" command="load" ns="g

    <!-- global costmap with laser, for odom_navigation_demo -->
    <rosparam file="$(find husky_navigation)/config/costmap_global_laser.yaml" command="load" ns="gl
    <param name="global_costmap/width" value="100.0" if="$(arg no_static_map)"/>
    <param name="global_costmap/height" value="100.0" if="$(arg no_static_map)"/>
  </node>

</launch>
```

END Launch File: my_move_base_launch_2.launch

Launch File: my_move_base_params.yaml

In [ ]:

```
controller_frequency: 1.0
recovery_behaviour_enabled: true

NavfnROS:
  allow_unknown: true # Specifies whether or not to allow navfn to create plans that traverse unknow
  default_tolerance: 0.1 # A tolerance on the goal point for the planner.
```

```
TrajectoryPlannerROS:
  # Robot Configuration Parameters
  acc_lim_x: 2.5
  acc_lim_theta:  3.2

  max_vel_x: 1.0
  min_vel_x: 0.0

  max_vel_theta: 1.0
  min_vel_theta: -1.0
  min_in_place_vel_theta: 0.2

  holonomic_robot: false
  escape_vel: -0.1

  # Goal Tolerance Parameters
  yaw_goal_tolerance: 0.1
  xy_goal_tolerance: 0.2
  latch_xy_goal_tolerance: false

  # Forward Simulation Parameters
  sim_time: 2.0
  sim_granularity: 0.02
  angular_sim_granularity: 0.02
  vx_samples: 6
  vtheta_samples: 20
  controller_frequency: 20.0

  # Trajectory scoring parameters
  meter_scoring: true # Whether the gdist_scale and pdist_scale parameters should assume that goal_
  occdist_scale:  0.1 #The weighting for how much the controller should attempt to avoid obstacles.
  pdist_scale: 0.75  #     The weighting for how much the controller should stay close to the path i
  gdist_scale: 1.0 #     The weighting for how much the controller should attempt to reach its local

  heading_lookahead: 0.325  #How far to look ahead in meters when scoring different in-place-rotatio
  heading_scoring: false  #Whether to score based on the robot's heading to the path or its distance
  heading_scoring_timestep: 0.8   #How far to look ahead in time in seconds along the simulated traj
  dwa: true #Whether to use the Dynamic Window Approach (DWA)_ or whether to use Trajectory Rollout
  simple_attractor: false
  publish_cost_grid_pc: true

  # Oscillation Prevention Parameters
  oscillation_reset_dist: 0.25 #How far the robot must travel in meters before oscillation flags are
  escape_reset_dist: 0.1
  escape_reset_theta: 0.1

DWAPlannerROS:
  # Robot configuration parameters
  acc_lim_x: 2.5
  acc_lim_y: 0
  acc_lim_th: 3.2

  max_vel_x: 0.5
  min_vel_x: 0.0
  max_vel_y: 0
  min_vel_y: 0

  max_trans_vel: 0.5
  min_trans_vel: 0.1
  max_rot_vel: 1.0
  min_rot_vel: 0.2

  # Goal Tolerance Parameters
  yaw_goal_tolerance: 0.1
  xy_goal_tolerance: 0.2
  latch_xy_goal_tolerance: false
```

END Launch File: my_move_base_params.yaml

Solution Exercise 4.8

Exercise 4.8

Launch File: my_move_base_launch_1.launch

In [ ]:

```
<?xml version="1.0"?>
<launch>

  <!-- Run the map server -->
  <arg name="map_file" default="$(find husky_navigation)/maps/my_map.yaml"/>
  <node name="map_server" pkg="map_server" type="map_server" args="$(arg map_file)" />

  <!--- Run AMCL -->
  <include file="$(find husky_navigation)/launch/amcl.launch" />

  <!--- Run Move Base -->
  <include file="$(find my_move_base_launcher)/launch/my_move_base_launch_2.launch" />

</launch>
```

END Launch File: my_move_base_launch_1.launch

Launch File: my_move_base_launch_2.launch

In [ ]:

```
<?xml version="1.0"?>
<launch>

  <arg name="no_static_map" default="false"/>

  <arg name="base_global_planner" default="carrot_planner/CarrotPlanner"/>
  <arg name="base_local_planner" default="dwa_local_planner/DWAPlannerROS"/>
  <!-- <arg name="base_local_planner" default="base_local_planner/TrajectoryPlannerROS"/> -->
  <node pkg="move_base" type="move_base" respawn="false" name="move_base" output="screen">

    <param name="base_global_planner" value="$(arg base_global_planner)"/>
    <param name="base_local_planner" value="$(arg base_local_planner)"/>
    <rosparam file="$(find my_move_base_launcher)/params/my_move_base_params.yaml" command="load"/>

    <!-- observation sources located in costmap_common.yaml -->
    <rosparam file="$(find husky_navigation)/config/costmap_common.yaml" command="load" ns="global_c
    <rosparam file="$(find husky_navigation)/config/costmap_common.yaml" command="load" ns="local_cc

    <!-- local costmap, needs size -->
    <rosparam file="$(find husky_navigation)/config/costmap_local.yaml" command="load" ns="local_cos
    <param name="local_costmap/width" value="10.0"/>
    <param name="local_costmap/height" value="10.0"/>

    <!-- static global costmap, static map provides size -->
    <rosparam file="$(find husky_navigation)/config/costmap_global_static.yaml" command="load" ns="g

    <!-- global costmap with laser, for odom_navigation_demo -->
    <rosparam file="$(find husky_navigation)/config/costmap_global_laser.yaml" command="load" ns="gl
    <param name="global_costmap/width" value="100.0" if="$(arg no_static_map)"/>
    <param name="global_costmap/height" value="100.0" if="$(arg no_static_map)"/>
  </node>

</launch>
```

END Launch File: my_move_base_launch_2.launch

Solution Exercise 4.11

Exercise 4.11

Launch File: my_move_base_launch_1.launch

In [ ]:

```
<?xml version="1.0"?>
<launch>

  <!-- Run the map server -->
  <arg name="map_file" default="$(find husky_navigation)/maps/my_map.yaml"/>
  <node name="map_server" pkg="map_server" type="map_server" args="$(arg map_file)" />

  <!--- Run AMCL -->
  <include file="$(find husky_navigation)/launch/amcl.launch" />

  <!--- Run Move Base -->
  <include file="$(find my_move_base_launcher)/launch/my_move_base_launch_2.launch" />

</launch>
```

END Launch File: my_move_base_launch_1.launch

Launch File: my_move_base_launch_2.launch

In [ ]:

```
<?xml version="1.0"?>
<launch>

  <arg name="no_static_map" default="false"/>

  <arg name="base_global_planner" default="navfn/NavfnROS"/>
  <arg name="base_local_planner" default="dwa_local_planner/DWAPlannerROS"/>
  <!-- <arg name="base_local_planner" default="base_local_planner/TrajectoryPlannerROS"/> -->

  <node pkg="move_base" type="move_base" respawn="false" name="move_base" output="screen">
```

```
    <param name="base_global_planner" value="$(arg base_global_planner)"/>
    <param name="base_local_planner" value="$(arg base_local_planner)"/>
    <rosparam file="$(find my_move_base_launcher)/params/my_move_base_params.yaml" command="load"/>

    <!-- observation sources located in costmap_common.yaml -->
    <rosparam file="$(find husky_navigation)/config/costmap_common.yaml" command="load" ns="global_c
    <rosparam file="$(find husky_navigation)/config/costmap_common.yaml" command="load" ns="local_cc

    <!-- local costmap, needs size -->
    <rosparam file="$(find husky_navigation)/config/costmap_local.yaml" command="load" ns="local_cos
    <param name="local_costmap/width" value="10.0"/>
    <param name="local_costmap/height" value="10.0"/>

    <!-- static global costmap, static map provides size -->
    <rosparam file="$(find my_move_base_launcher)/params/my_global_costmap_params.yaml" command="loa

    <!-- global costmap with laser, for odom_navigation_demo -->
    <rosparam file="$(find husky_navigation)/config/costmap_global_laser.yaml" command="load" ns="gl
    <param name="global_costmap/width" value="100.0" if="$(arg no_static_map)"/>
    <param name="global_costmap/height" value="100.0" if="$(arg no_static_map)"/>
  </node>

</launch>
```

END Launch File: my_move_base_launch_2.launch

Launch File: my_global_costmap_params.yaml

In [ ]:

```
global_frame: map
rolling_window: true
track_unknown_space: true

plugins:
  - {name: static,                    type: "costmap_2d::StaticLayer"}
  - {name: obstacles_laser,           type: "costmap_2d::VoxelLayer"}
  - {name: inflation,                 type: "costmap_2d::InflationLayer"}
```

END Launch File: my_global_costmap_params.yaml