Multi-robot Systems with ROS

27 February, 2017

Janyl Jumadinova
The navigation stack handles moving a robot from one position to another position safely (without crashing or getting lost).

- It takes in information from odometry and the sensors, and a goal pose and outputs safe velocity commands.
ROS Navigation Stack
# ROS Navigation Stack

<table>
<thead>
<tr>
<th>Feature</th>
<th>Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>offers map data as a ROS Service</td>
<td>map_server</td>
</tr>
<tr>
<td>provides laser-based SLAM (Simultaneous Localization and Mapping)</td>
<td>gmapping</td>
</tr>
<tr>
<td>a probabilistic localization system</td>
<td>amcl</td>
</tr>
<tr>
<td>implementation of a fast global planner for navigation</td>
<td>global_planner</td>
</tr>
<tr>
<td>implementations of the Trajectory Rollout and Dynamic Window approaches to local robot navigation</td>
<td>local_planner</td>
</tr>
<tr>
<td>links together the global and local planner to accomplish the navigation task</td>
<td>move_base</td>
</tr>
</tbody>
</table>
ROS Navigation Main Steps

- Goal
- AMCL
- Path Planner
- move_base
- /cmd_vel
  + /odom
- Base Controller
Two types of navigation – global and local

The **global planner** is used to create paths for a goal in the map or a far-off distance

The **local planner** is used to create paths in the nearby distances and avoid obstacles
Global Planner

http://wiki.ros.org/navfn

- **NavFn** provides a fast interpolated navigation function that creates plans for a mobile base.
- The global plan is computed before the robot starts moving toward the next destination.
- The planner uses Dijkstra’s algorithm to find a minimum cost plan from a start point to an end point in a grid.
- The global planner generates a series of waypoints for the local planner to follow.
Local Planner

http://wiki.ros.org/base_local_planner

- Chooses appropriate velocity commands for the robot to traverse the current segment of the global path.
- Combines sensory and odometry data with both global and local cost maps.
- Can recompute the robot’s path on the fly to keep the robot from striking objects yet still allowing it to reach its destination.
- Implements the Trajectory Rollout and Dynamic Window algorithm
Trajectory Rollout Algorithm

1. Discretely sample in the robot’s control space (dx,dy,d).
Trajectory Rollout Algorithm

1. Discretely sample in the robot’s control space \((dx,dy,d)\).
2. For each sampled velocity, perform forward simulation from the robot’s current state to predict what would happen if the sampled velocity were applied for some (short) period of time.
3. Evaluate each trajectory resulting from the forward simulation, using a metric that incorporates characteristics such as: proximity to obstacles, proximity to the goal, proximity to the global path, and speed.
4. Discard illegal trajectories (those that collide with obstacles).
5. Pick the highest-scoring trajectory and send the associated velocity to the mobile base.
6. Rinse and repeat.
Trajectory Rollout Algorithm

1. Discretely sample in the robot’s control space (dx,dy,d).

2. For each sampled velocity, perform forward simulation from the robot’s current state to predict what would happen if the sampled velocity were applied for some (short) period of time.

3. Evaluate each trajectory resulting from the forward simulation, using a metric that incorporates characteristics such as: proximity to obstacles, proximity to the goal, proximity to the global path, and speed.

4. Discard illegal trajectories (those that collide with obstacles).

5. Pick the highest-scoring trajectory and send the associated velocity to the mobile base.

6. Rinse and repeat.
Trajectory Rollout Algorithm

1. Discretely sample in the robot’s control space \((dx,dy,d)\).
2. For each sampled velocity, perform forward simulation from the robot’s current state to predict what would happen if the sampled velocity were applied for some (short) period of time.
3. Evaluate each trajectory resulting from the forward simulation, using a metric that incorporates characteristics such as: proximity to obstacles, proximity to the goal, proximity to the global path, and speed.
4. Discard illegal trajectories (those that collide with obstacles).
Trajectory Rollout Algorithm

1. Discretely sample in the robot’s control space (dx,dy,d).
2. For each sampled velocity, perform forward simulation from the robot’s current state to predict what would happen if the sampled velocity were applied for some (short) period of time.
3. Evaluate each trajectory resulting from the forward simulation, using a metric that incorporates characteristics such as: proximity to obstacles, proximity to the goal, proximity to the global path, and speed.
4. Discard illegal trajectories (those that collide with obstacles).
5. Pick the highest-scoring trajectory and send the associated velocity to the mobile base.
Trajectory Rollout Algorithm

1. Discretely sample in the robot’s control space (dx,dy,d).
2. For each sampled velocity, perform forward simulation from the robot’s current state to predict what would happen if the sampled velocity were applied for some (short) period of time.
3. Evaluate each trajectory resulting from the forward simulation, using a metric that incorporates characteristics such as: proximity to obstacles, proximity to the goal, proximity to the global path, and speed.
4. Discard illegal trajectories (those that collide with obstacles).
5. Pick the highest-scoring trajectory and send the associated velocity to the mobile base.
6. Rinse and repeat.
Local Planner Parameters

- The file `base_local_planner.yaml` contains a large number of ROS Parameters that can be set to customize the behavior of the base local planner.
- Grouped into several categories:
  - robot configuration
  - goal tolerance
  - forward simulation
  - trajectory scoring
  - oscillation prevention
  - global plan
costmap

http://wiki.ros.org/costmap_2d

- A data structure that represents places that are safe for the robot to be in a grid of cells.
- It is based on the occupancy grid map of the environment and user specified inflation radius.
Costmap

http://wiki.ros.org/costmap_2d$

▶ A data structure that represents places that are safe for the robot to be in a grid of cells.
▶ It is based on the occupancy grid map of the environment and user specified inflation radius.
▶ There are two types of costmaps in ROS:
  – Global costmap is used for global navigation
  – Local costmap is used for local navigation
▶ Each cell in the costmap has an integer value in the range [0 (FREE_SPACE), 255 (UNKNOWN)]
▶ Managed by the costmap_2d package
Navigation Stack with Multiple Robots

- Get the navigation package from the shared repo.
Get the navigation package from the shared repo.

It has been modified from the navigation tutorials from git

https://github.com/ros-planning/navigation_tutorials
Navigation Stack with Multiple Robots

- Get the navigation package from the shared repo.
- It has been modified from the navigation tutorials from git
  https://github.com/ros-planning/navigation_tutorials
- Run catkin_make from your catkin_ws directory.
Fake Localization

- This node is most frequently used during simulation as a method to provide perfect localization in a computationally inexpensive manner.
Fake Localization

- This node is most frequently used during simulation as a method to provide perfect localization in a computationally inexpensive manner.

- It creates a link between the global /map frame and the robots /odom frames so they actually become equal.
  - no localization error = pose of robot in its /odom frame equals its pose in the global /map frame
Fake Localization

- This node is most frequently used during simulation as a method to provide perfect localization in a computationally inexpensive manner.

- It creates a link between the global /map frame and the robots /odom frames so they actually become equal.
  - no localization error = pose of robot in its /odom frame equals its pose in the global /map frame

- Parameters:
  - odom_frame_id: the name of the odometric frame of the robot (default = “odom”)
  - base_frame_id: the base frame of the robot (default = “base_link”)

navigation_multi.launch

Nodes that are common to all robots:

```xml
<launch>
  <master auto="start"/>
  <param name="/use_sim_time" value="true"/>

  <node pkg="map_server" type="map_server" name="map_server" args="$(find navigation_multi)/stage_config/maps/willow-full.pgm 0.1" respawn="false">
    <param name="frame_id" value="/map"/>
  </node>

  <node pkg="stage_ros" type="stageros" name="stageros" args="$(find navigation_multi)/stage_config/worlds/willow-pr2-multi.world" respawn="false">
    <param name="base_watchdog_timeout" value="0.2"/>
  </node>
</launch>
```
Run the launch file

roslaunch navigation_multi navigation_multi.launch

Stage Simulator

- 2D
- Sensor-based
- Player interface
- Kinematic (in comparison, Gazebo - dynamic)
- $O(1) \approx O(n)$ (in comparison, Gazebo $O(n) \approx O(n^3)$)
- Large teams (100's) (in comparison, Gazebo 10's)
To see the robot’s footprint in rviz change the robot footprint topic to:

/robot_N/move_base_node/local_costmap/footprint_layer/footprint_stamped

In our case, the robots have a pentagon-shape
– Defined in
move_base_config/costmap_common_params.yaml
Sending Goals

- The 2D nav goal button allows you to send a goal to the navigation by setting a desired pose for the robot to achieve.
- By default the goal is published on the topic `/move_base_simple/goal`
- However, when having multiple robots, the topic is `/robot_N/move_base_simple/goal`
- To change or to verify the topic name, first enable the Tool Properties panel via the Panels menu.
- Verify that the 2D Nav Goal topic is set to `/robot_0/move_base_simple/goal`
- Click on the 2D Nav Goal button (or press G) and select the goal for the first robot.
Now, we will use a node that will make a given robot to move to a specific location on the map.

- Locate the program called *send_goal.cpp*
- Send a goal to robot no.1:
  ```
  rosrund navigation_multi send_goal 1
  ```
- `roslaunch send_goals.launch`