indigo | kinetic | lunar | melodic | *Show EOL distros:*

Documentation Status

***navigation (/navigation?distro=melodic)**: amcl (/amcl?distro=melodic) | base_local_planner (/base_local_planner?distro=melodic) | carrot_planner (/carrot_planner?distro=melodic) | clear_costmap_recovery (/clear_costmap_recovery?distro=melodic) | costmap_2d (/costmap_2d?distro=melodic) | dwa_local_planner (/dwa_local_planner?distro=melodic) | fake_localization (/fake_localization?distro=melodic) | global_planner (/global_planner?distro=melodic) | map_server (/map_server?distro=melodic) | move_base | move_base_msgs (/move_base_msgs?distro=melodic) | move_slow_and_clear (/move_slow_and_clear?distro=melodic) | nav_core (/nav_core?distro=melodic) | navfn (/navfn?distro=melodic) | rotate_recovery (/rotate_recovery?distro=melodic) | voxel_grid (/voxel_grid?distro=melodic)*

Package Links

- **Code API (http://docs.ros.org/melodic/api/move_base/html)**
- Tutorials (/move_base/Tutorials)
- FAQ (http://answers.ros.org/questions/scope:all/sort:activity-desc/tags:move_base/page:1/)
- Changelog (http://docs.ros.org/melodic/changelogs/move_base/changelog.html)
- Change List (/navigation/ChangeList)
- Reviews (/move_base/Reviews)

Dependencies (22)
Used by (7)
Jenkins jobs (9)

Package Summary

✔ Released ✔ Continuous Integration: 91 / 91 ✔ Documented

The move_base package provides an implementation of an action (see the actionlib (http://www.ros.org/wiki/actionlib) package) that, given a goal in the world, will attempt to reach it with a mobile base. The move_base node links together a global and local planner to accomplish its global navigation task. It supports any global planner adhering to the nav_core::BaseGlobalPlanner interface specified in the nav_core (http://www.ros.org/wiki/nav_core) package and any local planner adhering to the nav_core::BaseLocalPlanner interface specified in the nav_core (http://www.ros.org/wiki/nav_core) package. The move_base node also maintains two costmaps, one for the global planner, and one for a local planner (see the costmap_2d (http://www.ros.org/wiki/costmap_2d) package) that are used to accomplish navigation tasks.

- Maintainer status: maintained
- Maintainer: David V. Lu!! <davidvlu AT gmail DOT com>, Michael Ferguson <mfergs7 AT gmail DOT com>, Aaron Hoy <ahoy AT fetchrobotics DOT com>
- Author: Eitan Marder-Eppstein, contradict@gmail.com
- License: BSD
- Source: git https://github.com/ros-planning/navigation.git (https://github.com/ros-planning/navigation) (branch: melodic-devel)

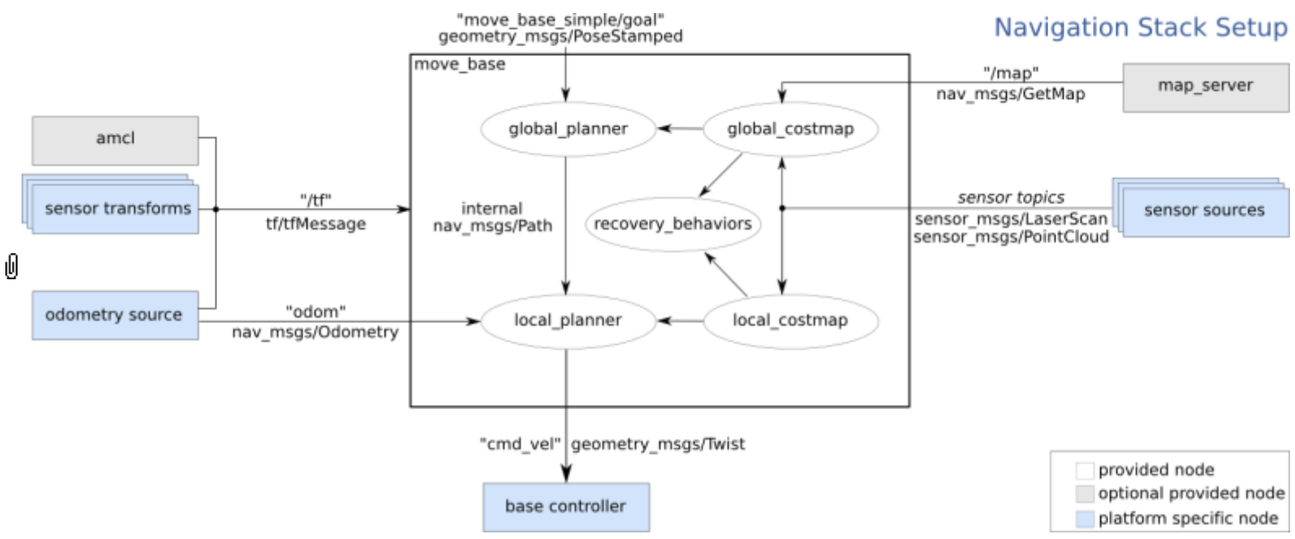
Contents

NOTE: The `move_base` package lets you move a robot to desired positions using the navigation stack. If you just want to drive the PR2 robot around in the odometry frame, you might be interested in this tutorial: pr2_controllers/Tutorials/Using the base controller with odometry and transform information (/pr2_controllers/Tutorials/Using%20the%20base%20controller%20with%20odometry%20and%20transform%20information).

1. Nodes

This package provides the `move_base` ROS Node which is a major component of the navigation stack (/navigation). A detailed description of this Node and its configuration options is found below.

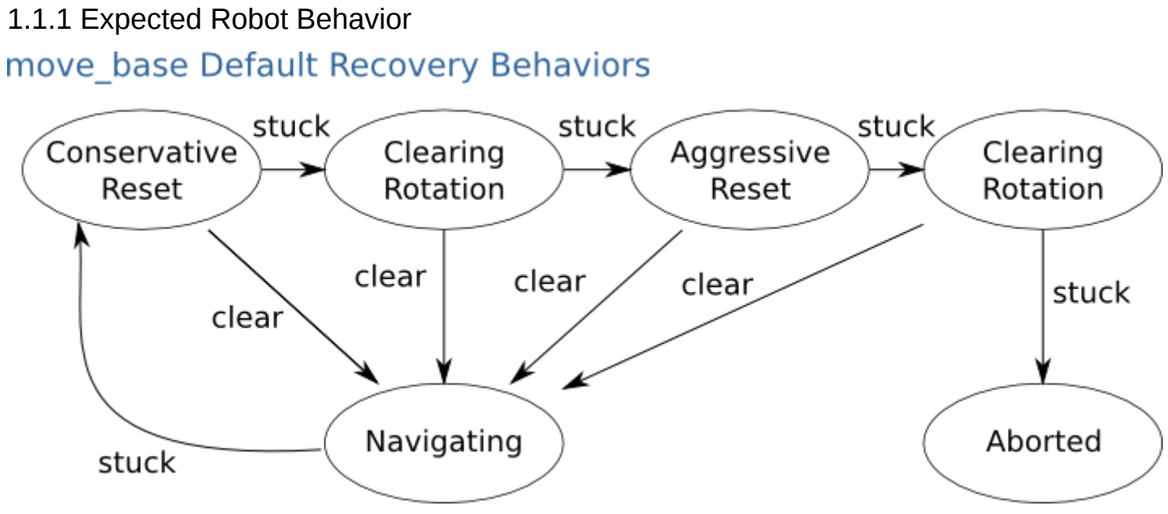
1.1 move_base



(/move_base?action=AttachFile&do=view&target=overview_tf.png)

The `move_base` node provides a ROS interface for configuring, running, and interacting with the navigation stack (/navigation) on a robot. A high-level view of the move_base node and its interaction with other components is shown above. The blue vary based on the robot platform, the gray are optional but are provided for all systems, and the white nodes are required but also provided for all systems. For more information on configuration of the `move_base` node, and the navigation stack as a whole, please see the navigation setup and configuration (/navigation/Tutorials/RobotSetup) tutorial.

1.1.1 Expected Robot Behavior



Running the `move_base` node on a robot that is properly configured (please see navigation stack documentation (/navigation) for more details) results in a robot that will attempt to achieve a goal pose with its base to within a user-specified tolerance. In the absence of dynamic obstacles, the move_base node will eventually get within this tolerance of its goal or signal failure to the user. The `move_base` node may optionally perform recovery behaviors when the robot perceives itself as stuck. By default, the move_base (/move_base) node will take the following actions to attempt to clear out space:

First, obstacles outside of a user-specified region will be cleared from the robot's map. Next, if possible, the robot will perform an in-place rotation to clear out space. If this too fails, the robot will more aggressively clear its map, removing all obstacles outside of the rectangular region in which it can rotate in place. This will be followed by another in-place rotation. If all this fails, the robot will consider its goal infeasible and notify the user that it has aborted. These recovery behaviors can be configured using the recovery_behaviors (/move_base#Parameters) parameter, and disabled using the recovery_behavior_enabled (/move_base#Parameters) parameter.

1.1.2 Action API

The `move_base` node provides an implementation of the `SimpleActionServer` (see actionlib documentation (/actionlib)), that takes in goals containing `geometry_msgs/PoseStamped` messages. You can communicate with the `move_base` node over ROS directly, but the recommended way to send goals to `move_base` if you care about tracking their status is by using the `SimpleActionClient`. Please see actionlib documentation (/actionlib) for more information.

Action Subscribed Topics

`move_base/goal` (move_base_msgs/MoveBaseActionGoal (http://docs.ros.org/api/move_base_msgs/html/msg/MoveBaseActionGoal.html))

A goal for `move_base` to pursue in the world.

`move_base/cancel` (actionlib_msgs/GoalID (http://docs.ros.org/api/actionlib_msgs/html/msg/GoalID.html))
A request to cancel a specific goal.

Action Published Topics

`move_base/feedback` (move_base_msgs/MoveBaseActionFeedback (http://docs.ros.org/api/move_base_msgs/html/msg/MoveBaseActionFeedback.html))

Feedback contains the current position of the base in the world.

`move_base/status` (actionlib_msgs/GoalStatusArray (http://docs.ros.org/api/actionlib_msgs/html/msg/GoalStatusArray.html))
Provides status information on the goals that are sent to the `move_base` action.

`move_base/result` (move_base_msgs/MoveBaseActionResult (http://docs.ros.org/api/move_base_msgs/html/msg/MoveBaseActionResult.html))
Result is empty for the `move_base` action.

1.1.3 Subscribed Topics

`move_base_simple/goal` (geometry_msgs/PoseStamped (http://docs.ros.org/api/geometry_msgs/html/msg/PoseStamped.html))
Provides a non-action interface to move_base for users that don't care about tracking the execution status of their goals.

1.1.4 Published Topics

`cmd_vel` (geometry_msgs/Twist (http://docs.ros.org/api/geometry_msgs/html/msg/Twist.html))
A stream of velocity commands meant for execution by a mobile base.

1.1.5 Services

`~make_plan` (nav_msgs/GetPlan (http://docs.ros.org/api/nav_msgs/html/srv/GetPlan.html))
Allows an external user to ask for a plan to a given pose from `move_base` without causing `move_base` to execute that plan.

`~clear_unknown_space` (std_srvs/Empty (http://docs.ros.org/api/std_srvs/html/srv/Empty.html))
Allows an external user to tell move_base to clear unknown space in the area directly around the robot. This is useful when move_base has its costmaps stopped for a long period of time and then started again in a new location in the environment. - **Available in versions from 1.1.0-groovy**

`~clear_costmaps` (std_srvs/Empty (http://docs.ros.org/api/std_srvs/html/srv/Empty.html))
Allows an external user to tell move_base to clear obstacles in the costmaps used by move_base. This could cause a robot to hit things and should be used with caution. - **New in 1.3.1**

1.1.6 Parameters

`~base_global_planner` (`string`, default: `"navfn/NavfnROS"` **For 1.1+ series**)
The name of the plugin for the global planner to use with `move_base`, see pluginlib (/pluginlib) documentation for more details on plugins. This plugin must adhere to the `nav_core::BaseGlobalPlanner` interface specified in the nav_core (/nav_core) package. (**1.0 series default:** `"NavfnROS"`)

`~base_local_planner` (`string`, default: `"base_local_planner/TrajectoryPlannerROS"` **For 1.1+ series**)
The name of the plugin for the local planner to use with `move_base` see pluginlib (/pluginlib) documentation for more details on plugins. This plugin must adhere to the `nav_core::BaseLocalPlanner` interface specified in the nav_core (/nav_core) package. (**1.0 series default:** `"TrajectoryPlannerROS"`)

`~recovery_behaviors` (`list`, default: [{name: conservative_reset, type: clear_costmap_recovery/ClearCostmapRecovery}, {name: rotate_recovery, type: rotate_recovery/RotateRecovery}, {name: aggressive_reset, type: clear_costmap_recovery/ClearCostmapRecovery}] **For 1.1+ series**)
A list of recovery behavior plugins to use with `move_base`, see pluginlib (/pluginlib) documentation for more details on plugins. These behaviors will be run when `move_base` fails to find a valid plan in the order that they are specified. After each behavior completes, `move_base` will attempt to make a plan. If planning is successful, `move_base` will continue normal operation. Otherwise, the next recovery behavior in the list will be executed. These plugins must adhere to the `nav_core::RecoveryBehavior` interface specified in the nav_core (/nav_core) package. (**1.0 series default:** [{name: conservative_reset, type: ClearCostmapRecovery}, {name: rotate_recovery, type: RotateRecovery}, {name: aggressive_reset, type: ClearCostmapRecovery}]). Note: For the default parameters, the aggressive_reset behavior will clear out to a distance of 4 * `~/local_costmap/circumscribed_radius`.

`~controller_frequency` (`double`, default: 20.0)
The rate in Hz at which to run the control loop and send velocity commands to the base.

`~planner_patience` (`double`, default: 5.0)
How long the planner will wait in seconds in an attempt to find a valid plan before space-clearing operations are performed.

`~controller_patience` (`double`, default: 15.0)
How long the controller will wait in seconds without receiving a valid control before space-clearing operations are performed.

`~conservative_reset_dist` (`double`, default: 3.0)
The distance away from the robot in meters beyond which obstacles will be cleared from the costmap (/costmap_2d) when attempting to clear space in the map. Note, this parameter is only used when the default recovery behaviors are used for `move_base`.

`~recovery_behavior_enabled` (`bool`, default: `true`)
Whether or not to enable the `move_base` recovery behaviors to attempt to clear out space.

`~clearing_rotation_allowed` (`bool`, default: `true`)
Determines whether or not the robot will attempt an in-place rotation when attempting to clear out space. Note: This parameter is only used when the default recovery behaviors are in use, meaning the user has not set the `recovery_behaviors` parameter to anything custom.

`~shutdown_costmaps` (`bool`, default: `false`)
Determines whether or not to shutdown the costmaps of the node when `move_base` is in an inactive state

`~oscillation_timeout` (`double`, default: 0.0)
How long in seconds to allow for oscillation before executing recovery behaviors. A value of 0.0 corresponds to an infinite timeout. **New in navigation 1.3.1**

`~oscillation_distance` (`double`, default: 0.5)
How far in meters the robot must move to be considered not to be oscillating. Moving this far resets the timer counting up to the `~oscillation_timeout` **New in navigation 1.3.1**

`~planner_frequency` (`double`, default: 0.0)
The rate in Hz at which to run the global planning loop. If the frequency is set to 0.0, the global planner will only run when a new goal is received or the local planner reports that its path is blocked. **New in navigation 1.6.0**

`~max_planning_retries` (`int32_t`, default: -1)
How many times to allow for planning retries before executing recovery behaviors. A value of -1.0 corresponds to an infinite retries.

1.1.7 Component APIs

The `move_base` node contains components that have their own ROS APIs. These components may vary based on the values of the `~base_global_planner`, `~base_local_planner`, and `~recovery_behaviors` respectively. Links to the APIs for the default components can be found below:

- costmap_2d (/costmap_2d) - Documentation on the costmap_2d (/costmap_2d) package used in move_base
- nav_core (/nav_core) - Documentation on the `nav_core::BaseGlobalPlanner` and `nav_core::BaseLocalPlanner` interfaces used by `move_base`.
- base_local_planner (/base_local_planner) - Documentation on the base_local_planner (/base_local_planner) used in `move_base`
- navfn (/navfn) - Documentation on the navfn (/navfn) global planner used in `move_base`
- clear_costmap_recovery (/clear_costmap_recovery) - Documentation on the clear_costmap_recovery (/clear_costmap_recovery) recovery behavior used in `move_base`
- rotate_recovery (/rotate_recovery) - Documentation on the rotate_recovery (/rotate_recovery) recovery behavior used in `move_base`

Class Diagram (partially & not strictly drawn) is available here (https://docs.google.com/drawings/d/1rKY4yxI0ibHqgQoLFZPuEayRp8Bs4FpRyaNKsMxqpX4/edit?usp=sharing).

Wiki: move_base (last edited 2018-09-27 21:33:09 by NicolasVaras (/NicolasVaras))