# Package Summary

Released

Documented

This package provides an implementation of the Dynamic Window Approach to local robot navigation on a plane. Given a global plan to follow and a costmap, the local planner produces velocity commands to send to a mobile base. This package supports any robot who's footprint can be represented as a convex polygon or cicrle, and exposes its configuration as ROS parameters that can be set in a launch file. The parameters for this planner are also dynamically reconfigurable. This package's ROS wrapper adheres to the BaseLocalPlanner interface specified in the [nav\_core](http://wiki.ros.org/nav_core) package.

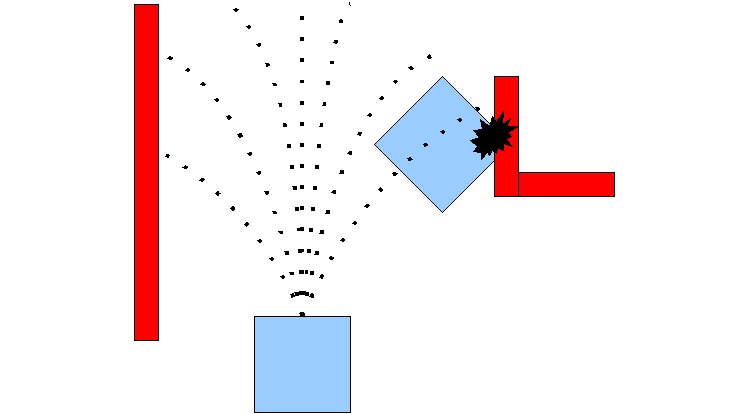
* Maintainer status: maintained
* Maintainer: David V. Lu!! <davidvlu AT gmail DOT com>, Michael Ferguson <mfergs7 AT gmail DOT com>, Aaron Hoy <ahoy AT fetchrobotics DOT com>
* Author: Eitan Marder-Eppstein, contradict@gmail.com
* License: BSD
* Source: git [https://github.com/ros-planning/navigation.git](https://github.com/ros-planning/navigation)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

Contents

1. [Overview](http://wiki.ros.org/dwa_local_planner?distro=kinetic" \l "Overview)
2. [DWAPlannerROS](http://wiki.ros.org/dwa_local_planner?distro=kinetic" \l "DWAPlannerROS)
   1. [API Stability](http://wiki.ros.org/dwa_local_planner?distro=kinetic" \l "API_Stability)
      1. [Published Topics](http://wiki.ros.org/dwa_local_planner?distro=kinetic" \l "Published_Topics)
      2. [Subscribed Topics](http://wiki.ros.org/dwa_local_planner?distro=kinetic" \l "Subscribed_Topics)
   2. [Parameters](http://wiki.ros.org/dwa_local_planner?distro=kinetic" \l "Parameters)
      1. [Robot Configuration Parameters](http://wiki.ros.org/dwa_local_planner?distro=kinetic" \l "Robot_Configuration_Parameters)
      2. [Goal Tolerance Parameters](http://wiki.ros.org/dwa_local_planner?distro=kinetic" \l "Goal_Tolerance_Parameters)
      3. [Forward Simulation Parameters](http://wiki.ros.org/dwa_local_planner?distro=kinetic" \l "Forward_Simulation_Parameters)
      4. [Trajectory Scoring Parameters](http://wiki.ros.org/dwa_local_planner?distro=kinetic" \l "Trajectory_Scoring_Parameters)
      5. [Oscillation Prevention Parameters](http://wiki.ros.org/dwa_local_planner?distro=kinetic" \l "Oscillation_Prevention_Parameters)
      6. [Global Plan Parameters](http://wiki.ros.org/dwa_local_planner?distro=kinetic" \l "Global_Plan_Parameters)
   3. [C++ API](http://wiki.ros.org/dwa_local_planner?distro=kinetic" \l "C.2B-.2B-_API)
3. [DWAPlanner](http://wiki.ros.org/dwa_local_planner?distro=kinetic" \l "DWAPlanner)
   1. [API Stability](http://wiki.ros.org/dwa_local_planner?distro=kinetic" \l "API_Stability-1)
   2. [C++ API](http://wiki.ros.org/dwa_local_planner?distro=kinetic" \l "C.2B-.2B-_API-1)

## Overview

The dwa\_local\_planner package provides a controller that drives a mobile base in the plane. This controller serves to connect the path planner to the robot. Using a map, the planner creates a kinematic trajectory for the robot to get from a start to a goal location. Along the way, the planner creates, at least locally around the robot, a value function, represented as a grid map. This value function encodes the costs of traversing through the grid cells. The controller's job is to use this value function to determine dx,dy,dtheta velocities to send to the robot.



The basic idea of the Dynamic Window Approach (DWA) algorithm is as follows:

1. Discretely sample in the robot's control space (dx,dy,dtheta)
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 (score) 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. Discard illegal trajectories (those that collide with obstacles).
4. Pick the highest-scoring trajectory and send the associated velocity to the mobile base.
5. Rinse and repeat.

Useful references:

* [D. Fox, W. Burgard, and S. Thrun. "The dynamic window approach to collision avoidance".](http://www.cs.washington.edu/ai/Mobile_Robotics/postscripts/colli-ieee.ps.gz) The Dynamic Window Approach to local control.
* [Alonzo Kelly. "An Intelligent Predictive Controller for Autonomous Vehicles"](http://www.ri.cmu.edu/pub_files/pub1/kelly_alonzo_1994_7/kelly_alonzo_1994_7.pdf). A previous system that takes a similar approach to control.
* [Brian P. Gerkey and Kurt Konolige. "Planning and Control in Unstructured Terrain "](http://pub1.willowgarage.com/apubdb_html/files_upload/8.pdf). Discussion of the Trajectory Rollout algorithm in use on the LAGR robot.

## DWAPlannerROS

The dwa\_local\_planner::DWAPlannerROS object is a [wrapper](http://wiki.ros.org/navigation/ROS_Wrappers) for a dwa\_local\_planner::DWAPlanner object that exposes its functionality as a [C++ ROS Wrapper](http://wiki.ros.org/navigation/ROS_Wrappers). It operates within a ROS namespace (assumed to be name from here on) specified on initialization. It adheres to the nav\_core::BaseLocalPlanner interface found in the [nav\_core](http://wiki.ros.org/nav_core) package.

Example creation of a dwa\_local\_planner::DWAPlannerROS object:

[Toggle line numbers](http://wiki.ros.org/dwa_local_planner?distro=kinetic)

1 #include <tf/transform\_listener.h>

2 #include <costmap\_2d/costmap\_2d\_ros.h>

3 #include <dwa\_local\_planner/dwa\_planner\_ros.h>

4

5 ...

6

7 tf::TransformListener tf(ros::Duration(10));

8 costmap\_2d::Costmap2DROS costmap("my\_costmap", tf);

9

10 dwa\_local\_planner::DWAPlannerROS dp;

11 dp.initialize("my\_dwa\_planner", &tf, &costmap);

### API Stability

* The C++ API is stable
* The ROS API is stable

#### Published Topics

~<name>/global\_plan ([nav\_msgs/Path](http://docs.ros.org/api/nav_msgs/html/msg/Path.html))

* The portion of the global plan that the local planner is currently attempting to follow. Used primarily for visualization purposes.

~<name>/local\_plan ([nav\_msgs/Path](http://docs.ros.org/api/nav_msgs/html/msg/Path.html))

* The local plan or trajectory that scored the highest on the last cycle. Used primarily for visualization purposes.

#### Subscribed Topics

odom ([nav\_msgs/Odometry](http://docs.ros.org/api/nav_msgs/html/msg/Odometry.html))

* Odometry information that gives the local planner the current speed of the robot. The velocity information in this message is assumed to be in the same coordinate frame as the robot\_base\_frame of the costmap contained within the TrajectoryPlannerROS object. See the [costmap\_2d](http://wiki.ros.org/costmap_2d) package for information about the robot\_base\_frame parameter.

### Parameters

There are a large number of ROS [Parameters](http://wiki.ros.org/Parameters) that can be set to customize the behavior of the dwa\_local\_planner::DWAPlannerROS wrapper. These parameters are grouped into several categories: robot configuration, goal tolerance, forward simulation, trajectory scoring, oscillation prevention, and global plan. Most of these parameters can also be changed using [dynamic\_reconfigure](http://wiki.ros.org/dynamic_reconfigure) to facilitate tuning the local planner in a running system.

#### Robot Configuration Parameters

~<name>/acc\_lim\_x (double, default: 2.5)

* The x acceleration limit of the robot in meters/sec^2

~<name>/acc\_lim\_y (double, default: 2.5)

* The y acceleration limit of the robot in meters/sec^2

~<name>/acc\_lim\_th (double, default: 3.2)

* The rotational acceleration limit of the robot in radians/sec^2

~<name>/max\_trans\_vel (double, default: 0.55)

* The absolute value of the maximum translational velocity for the robot in m/s

~<name>/min\_trans\_vel (double, default: 0.1)

* The absolute value of the minimum translational velocity for the robot in m/s

~<name>/max\_vel\_x (double, default: 0.55)

* The maximum x velocity for the robot in m/s.

~<name>/min\_vel\_x (double, default: 0.0)

* The minimum x velocity for the robot in m/s, negative for backwards motion.

~<name>/max\_vel\_y (double, default: 0.1)

* The maximum y velocity for the robot in m/s

~<name>/min\_vel\_y (double, default: -0.1)

* The minimum y velocity for the robot in m/s

~<name>/max\_rot\_vel (double, default: 1.0)

* The absolute value of the maximum rotational velocity for the robot in rad/s

~<name>/min\_rot\_vel (double, default: 0.4)

* The absolute value of the minimum rotational velocity for the robot in rad/s

#### Goal Tolerance Parameters

~<name>/yaw\_goal\_tolerance (double, default: 0.05)

* The tolerance in radians for the controller in yaw/rotation when achieving its goal

~<name>/xy\_goal\_tolerance (double, default: 0.10)

* The tolerance in meters for the controller in the x & y distance when achieving a goal

~<name>/latch\_xy\_goal\_tolerance (bool, default: false)

* If goal tolerance is latched, if the robot ever reaches the goal xy location it will simply rotate in place, even if it ends up outside the goal tolerance while it is doing so.

#### Forward Simulation Parameters

~<name>/sim\_time (double, default: 1.7)

* The amount of time to forward-simulate trajectories in seconds

~<name>/sim\_granularity (double, default: 0.025)

* The step size, in meters, to take between points on a given trajectory

~<name>/vx\_samples (integer, default: 3)

* The number of samples to use when exploring the x velocity space

~<name>/vy\_samples (integer, default: 10)

* The number of samples to use when exploring the y velocity space

~<name>/vth\_samples (integer, default: 20)

* The number of samples to use when exploring the theta velocity space

~<name>/controller\_frequency (double, default: 20.0)

* The frequency at which this controller will be called in Hz. Uses searchParam to read the parameter from parent namespaces if not set in the namespace of the controller. For use with [move\_base](http://wiki.ros.org/move_base), this means that you only need to set its "controller\_frequency" parameter and can safely leave this one unset.

#### Trajectory Scoring Parameters

The cost function used to score each trajectory is in the following form:

cost =

path\_distance\_bias \* (distance to path from the endpoint of the trajectory in meters)

+ goal\_distance\_bias \* (distance to local goal from the endpoint of the trajectory in meters)

+ occdist\_scale \* (maximum obstacle cost along the trajectory in obstacle cost (0-254))

~<name>/path\_distance\_bias (double, default: 32.0)

* The weighting for how much the controller should stay close to the path it was given

~<name>/goal\_distance\_bias (double, default: 24.0)

* The weighting for how much the controller should attempt to reach its local goal, also controls speed

~<name>/occdist\_scale (double, default: 0.01)

* The weighting for how much the controller should attempt to avoid obstacles

~<name>/forward\_point\_distance (double, default: 0.325)

* The distance from the center point of the robot to place an additional scoring point, in meters

~<name>/stop\_time\_buffer (double, default: 0.2)

* The amount of time that the robot must stop before a collision in order for a trajectory to be considered valid in seconds

~<name>/scaling\_speed (double, default: 0.25)

* The absolute value of the velocity at which to start scaling the robot's footprint, in m/s

~<name>/max\_scaling\_factor (double, default: 0.2)

* The maximum factor to scale the robot's footprint by

~<name>/publish\_cost\_grid (bool, default: false)

* Whether or not to publish the cost grid that the planner will use when planning. When true, a sensor\_msgs/PointCloud2 will be available on the ~<name>/cost\_cloud topic. Each point cloud represents the cost grid and has a field for each individual scoring function component as well as the overall cost for each cell, taking the scoring parameters into account.

#### Oscillation Prevention Parameters

~<name>/oscillation\_reset\_dist (double, default: 0.05)

* How far the robot must travel in meters before oscillation flags are reset

#### Global Plan Parameters

~<name>/prune\_plan (bool, default: true)

* Defines whether or not to eat up the plan as the robot moves along the path. If set to true, points will fall off the end of the plan once the robot moves 1 meter past them.

### C++ API

For C++ level API documentation on the base\_local\_planner::TrajectoryPlannerROS class, please see the following page: [DWAPlannerROS C++ API](http://www.ros.org/doc/api/dwa_local_planner/html/classdwa__local__planner_1_1DWAPlannerROS.html)

## DWAPlanner

The dwa\_local\_planner::DWAPlanner provides implementations of the DWA and Trajectory Rollout algorithms described earlier. In order to use the dwa\_local\_planner::DWAPlanner with ROS, please use the [DWAPlannerROS wrapper](http://wiki.ros.org/dwa_local_planner" \l "DWAPlannerROS). It is not recommended to use the dwa\_local\_planner::DWAPlanner on its own.

### API Stability

* The C++ API is stable. However, it is recommended that you use the [DWAPlannerROS wrapper](http://wiki.ros.org/dwa_local_planner" \l "DWAPlannerROS) instead of using the dwa\_local\_planner::TrajectoryPlanner on its own.

### C++ API

For C++ level API documentation on the dwa\_local\_planner::DWAPlanner class, please see the following page: [DWAPlanner C++ API](http://www.ros.org/doc/api/dwa_local_planner/html/classdwa__local__planner_1_1DWAPlanner.html)