Table of Contents

1. Launch file parameters

1.1. Overview

The autonomous driving of Willy is controlled by external ROS plugins. These plugins are:

Move_base → Used for sending Twist messages to Willy out of goals send to the ros navigation stack.
Hector_Mapping → Used for creating a map based on the LiDaR and transforms.
Sick_tim551 → Used for initilization of the LiDaR and creating the /scan topic for the map.
Transform → Used the send the transformation and rotations from devices on the robot to the rotation point of the robot.
Kinect → Used for the recognition of people and creating the /camera topic.
Rviz → Used for visualization of the map and sensors on the base frame.

All of these plugins have their own launch file. Every launch file has it’s own parameters which can be tuned to finetune Willy.

1.2. parameters

Move base

Move_base_param.yaml

This param file can be found in the following directory: `/home/willy/WTGD/willy/src/willy_navigation/param/move_base_params.yaml`
It’s possible that some parameters are not used yet. These can be added when needed.

- `base_global_planner` (string, default: "navfn/NavfnROS")
  The name of the plugin for the global planner to use with move_base.
- `base_local_planner` (string, default: "base_local_planner/TrajectoryPlannerROS")
  The name of the plugin for the local planner to use with move_base.
- `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}])
  A list of recovery behavior plugins to use with move_base, see 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.
- `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 at which obstacles will be cleared from the costmap 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.
- `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`
- `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.

- `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.

1.3. Base local planner

base_local_planner_params.yaml

This param file can be found in the following directory: `/home/willy/WTGD/willy/src/willy_navigation/param/base_local_planner_params.yaml`
It’s possible that some parameters are not used yet. These can be added when needed.

--Robot configuration parameters--
- `acc_lim_x` (double, default: 2.5)
The x acceleration limit of the robot in meters/sec^2
- `acc_lim_y` (double, default: 2.5)
The y acceleration limit of the robot in meters/sec^2
- `acc_lim_theta` (double, default: 3.2)
The rotational acceleration limit of the robot in radians/sec^2
- `max_vel_x` (double, default: 0.5)
The maximum forward velocity allowed for the base in meters/sec
- `min_vel_x` (double, default: 0.1)
The minimum forward velocity allowed for the base in meters/sec.
- `max_vel_theta` (double, default: 1.0)
The maximum rotational velocity allowed for the base in radians/sec
- `min_vel_theta` (double, default: -1.0)
The minimum rotational velocity allowed for the base in radians/sec
- `min_in_place_vel_theta` (double, default: 0.4)
The minimum rotational velocity allowed for the base while performing in-place rotations in radians/sec
- `escape_vel` (double, default: -0.1)
Speed used for driving during escapes in meters/sec.
- `holonomic_robot` (bool, default: true)
Determines whether velocity commands are generated for a holonomic or non-holonomic robot. For holonomic robots, strafing velocity commands may be issued to the base. For non-holonomic robots, no strafing velocity commands will be issued.

The following parameter is only used if holonomic_robot is set to true:
- `y_vels` (list, default: [-0.3, -0.1, 0.1, 0.3])
The strafing velocities that a holonomic robot will consider in meters/sec.

--Goal Tolerance Parameters--
- `yaw_goal_tolerance` (double, defaultv: 0.05)
The tolerance in radians for the controller in yaw/rotation when achieving its goal.
- `xy_goal_tolerance` (double, default: 0.10)
The tolerance in meters for the controller in the x & y distance when achieving a goal.
- `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--
- `sim_time` (double, default: 1.0)
The amount of time to forward-simulate trajectories in seconds
- `sim_granularity` (double, default: 0.025)
The step size, in meters, to take between points on a given trajectory
- `angular_sim_granularity` (double, default: `sim_granularity`)
The step size, in radians, to take between angular samples on a given trajectory.
- `vx_samples` (integer, default: 3)
The number of samples to use when exploring the x velocity space
- `vtheta_samples` (integer, default: 20)
The number of samples to use when exploring the theta velocity space
- `controller_frequency` (double, default: 20.0)
The frequency at which this controller will be called in Hz.

--Trajectory Scoring Parameters--
The cost function used to score each trajectory is in the following form:
cost =
pdist_scale * (distance to path from the endpoint of the trajectory in map cells or meters depending on the meter_scoring parameter)
+ gdist_scale * (distance to local goal from the endpoint of the trajectory in map cells or meters depending on the meter_scoring parameter)
+ occdist_scale * (maximum obstacle cost along the trajectory in obstacle cost (0-254))

- `meter_scoring` (bool, default: false)
  Whether the gdist_scale and pdist_scale parameters should assume that goal_distance and path_distance are expressed in units of meters or cells. Cells are assumed by default.
- `pdist_scale` (double, default: 0.6)
  The weighting for how much the controller should stay close to the path it was given, maximal possible value is 5.0.
- `gdist_scale` (double, default: 0.8)
  The weighting for how much the controller should attempt to reach its local goal, also controls speed, maximal possible value is 5.0.
- `occdist_scale` (double, default: 0.01)
  The weighting for how much the controller should attempt to avoid obstacles.
- `heading_lookahead` (double, default: 0.325)
  How far to look ahead in meters when scoring different in-place-rotation trajectories.
- `heading_scoring` (bool, default: false)
  Whether to score based on the robot's heading to the path or its distance from the path.
- `heading_scoring_timestep` (double, default: 0.8)
  How far to look ahead in time in seconds along the simulated trajectory when using heading scoring.
- `dwa` (bool, default: true)
  Whether to use the Dynamic Window Approach (DWA)_ or whether to use Trajectory Rollout.

- `publish_cost_grid_pc` (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 `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.
- `global_frame_id` (string, default: odom)
  The frame to set for the cost_cloud. Should be set to the same frame as the local costmap's global frame.

--Oscillation Prevention Parameters--
- `oscillation_reset_dist` (double, default: 0.05)
How far the robot must travel in meters before oscillation flags are reset

--Global Plan Parameters--
- `prune_plan*` (bool, default: true)
Defines whether or not to eat up the plan as the robot moves along the path.

1.4. Common costmap

costmap_common_params.yaml

This param file can be found in the following directory: `/home/willy/WTGD/willy/src/willy_navigation/param/costmap_common_params.yaml`
It's possible that some parameters are not used yet. These can be added when needed.

- `max_obstacle_height` (double, default: 1.0)
  The max obstacle height the robot can handle in meters.
- `obstacle_range` (double, default: 2.5)
  the maximum range sensor reading that will result in an obstacle being put into the costmap.
- `raytrace_range` (double, default: 3.0)
  determines the range to which we will raytrace freespace given a sensor reading.
- `robot_radius` (double, default: 0.8)
- `footprint` (double array, default: [[0.3, 0.3], [0.3, -0.3], [-1.0, -0.3], [-1.0, 0.3]])
  Here we set either the footprint of the robot or the radius of the robot if it is circular.
- `cost_scaling_factor` (double, default: 2.58)
  A scaling factor to apply to cost values during inflation.
- `inflation_radius` (double, default: 1.75)
  The radius in meters to which the map inflates obstacle cost values.
- `observation_sources` (string, default: scan kinect) The sensor sensors used for the navigation. Listed below.

scan: {data_type: LaserScan, topic: scan, marking: true, clearing: true, min_obstacle_height: -1, max_obstacle_height: 1.0}
kinect: {data_type: PointCloud2, topic: camera/depth_registered/cloud, marking: true, clearing: false, min_obstacle_height: -1, max_obstacle_height: 1}

1.5. Global costmap

global_costmap_params.yaml

This param file can be found in the following directory: `/home/willy/WTGD/willy/src/willy_navigation/param/global_costmap_params.yaml`
It’s possible that some parameters are not used yet. These can be added when needed.

- `global_frame` (string, default: /map)
  The topic where the map is published.
- `robot_base_frame` (string, default: /base_link)
  The topic where the robot base frame is published and transformed.
- `update_frequency` (double, default: 1.0)
  Determines the frequency, in Hz, at which the costmap will run its update loop
- `static_map` (bool, default: false)
  Determines whether or not the costmap should initialize itself based on a map served by the map_server.

1.6. Local costmap

local_costmap_params.yaml

This param file can be found in the following directory: `/home/willy/WTGD/willy/src/willy_navigation/param/local_costmap_params.yaml`
It’s possible that some parameters are not used yet. These can be added when needed.

- `global_frame` (string, default: /odom)
  The topic where the map is published.
- `robot_base_frame` (string, default: /base_link)
  The topic where the robot base frame is published and transformed.
- `update_frequency` (double, default: 5.0)
  Determines the frequency, in Hz, at which the costmap will run its update loop
- `publish_frequency` (double, default: 2.0)
  Determines the rate, in Hz, at which the costmap will publish visualization information.
- `static_map` (bool, default: false)
  Determines whether or not the costmap should initialize itself based on a map served by the map_server.
- `rolling_window` (bool, default: true)
  Setting the "rolling_window" parameter to true means that the costmap will remain centered around the robot as the robot moves through the world.
- `width` (double, default: 6.0)
  The width in meters of the costmap.
- `height` (double, default: 6.0)
  The height in meters of the costmap.
- `resolution` (double, default: 0.05)

The resolution in meters / cell of the costmap.