Weston Robot

# RSE2107A - Lecture 5

ROS Navigation Part 1

Weston Robot

# Agenda

## 01
Navigation stack

## 02
Localisation

## 03
Planners

# Navigation Stack

# Map Based Navigation

- Robot Navigation
  - *Where is the robot?*
    - Localisation: helps the robot know its location
  - *Where is the robot going?*
    - Mapping: robot requires a map of its environment to know where it has been moving around thus far
  - *How does the robot get there?*
    - Motion/Path planning: goal of robot needs to be well defined for the robot to understand

# Navigation in ROS

- Three packages that ROS has in the Navigation Stack
  - *gmapping*
    - create maps using laser scan data
  - *amcl*
    - responsible for localisation using existing map
  - *move_base*
    - allows robot to navigate and move to a goal pose with respect to a given reference frame

# Localisation

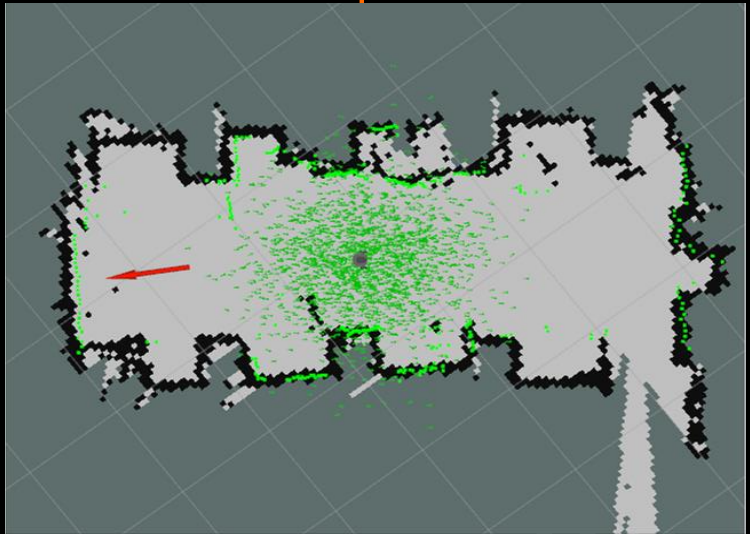
# What is localisation?

- The process of determining where a mobile robot is located with respect to its environment.
- The knowledge of the robot’s location is important for making decisions about its navigation path.
- ROS offers the **AMCL** localisation package.

# AMCL

Weston Robot

- Adaptive Monte Carlo Localisation.
- Probabilistic localisation system for robot moving in 2D space.
    - Predicts possible locations of robot based on map information and sensor data.



# AMCL node

- Requires a generated map before it can be used.
- Subscribes to
  - data of the laser via the topic /scan.
  - laser-based map via the topic /map.
  - transformation of the robot via the topic /tf.
- Publishes the estimated positions of the robot in the map via
  - /amcl_pose
  - /particle_cloud

# AMCL node

- Services provided
  - global_localization(std_srvs/Empty)
    - Takes no arguments
    - Initiate global localization by dispersing particles randomly throughout the free spaces in the map.
- Services called
  - static_map(nav_msgs/GetMap)
    - This service can be called to retrieve the map.

# Setting up the amcl node

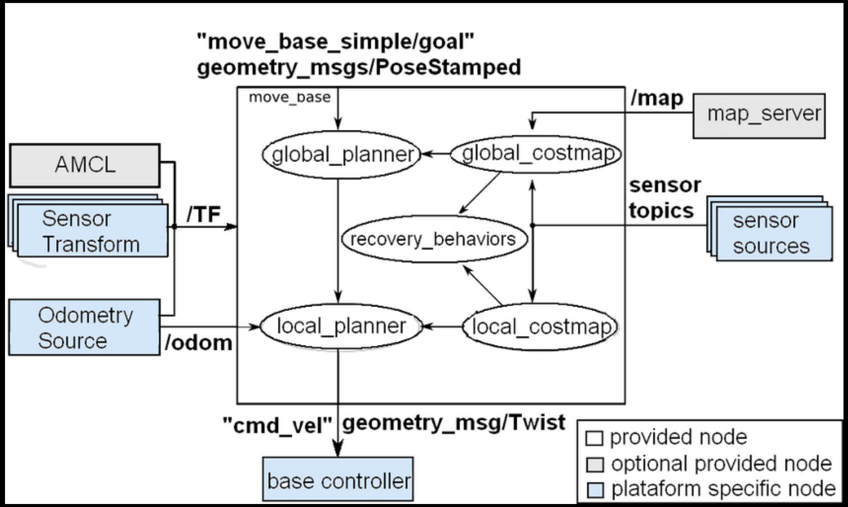
- General parameters
  - odom_model_type (default: “diff”)
    - Depends on the robot. Can be diff, omni, diff-corrected, etc.
  - odom_frame_id: default (“odom”)
    - Which frame to use for odometry.
  - base_frame_id (default: “base_link”)
    - Which frame to use for robot base.
  - global_frame_id (default: “map”)
    - Name of the coordinate frame published by localisation system.

# move_base

Weston Robot

# move_base node

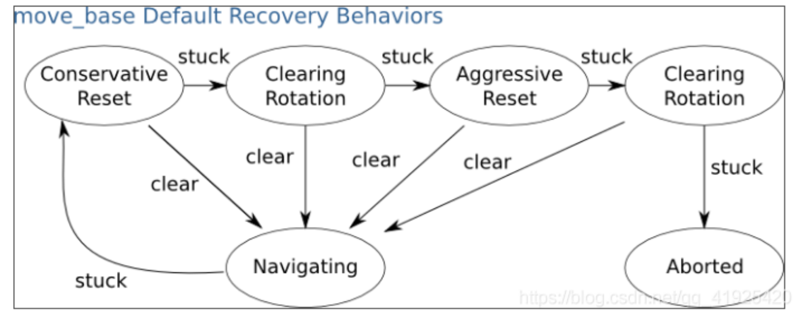
- Allows movement of robot to desired location using navigation stack
    - recovery behaviours
    - global planner
    - local planner
    - global costmap
    - local costmap
- Subscribed topic:
    - move_base_simple/goal
- Published topic:
    - cmd_vel



Weston Robot

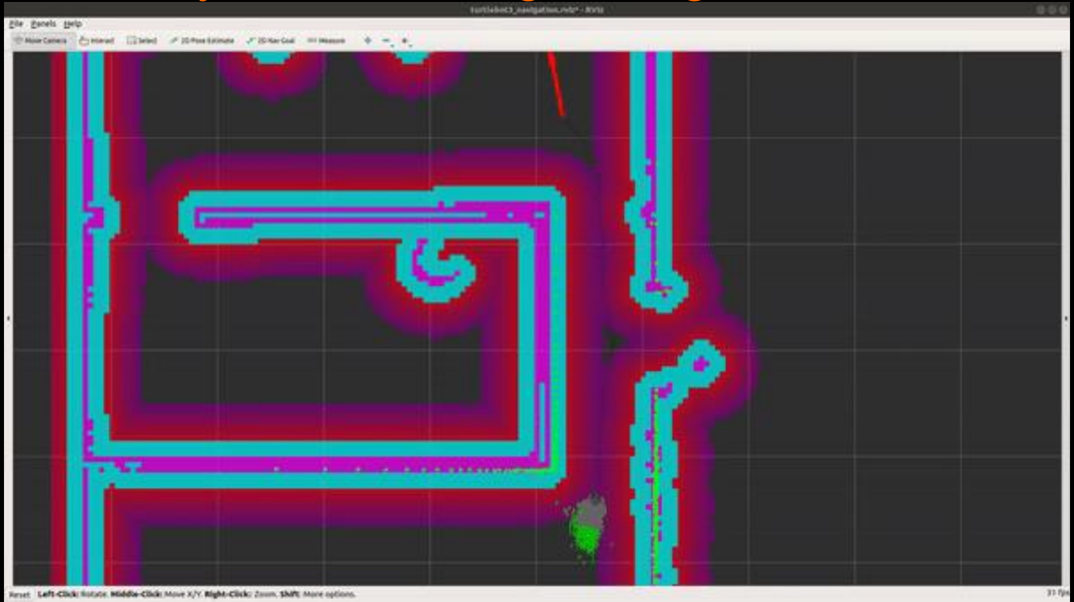
# move_base recovery behaviours

- When robot perceives itself as stuck, the move_base node will try to do the following:
  - Clearing of obstacle outside of user specified region in the map and perform in-place rotation to clear out space
  - [failed] robot will move more aggressively to clear its map, and try to rotate in-place again
  - [failed again] it will deem goal as not feasible



# What are Costmaps

- Costmap is a grid map where each cell is assigned a specific cost. The cost represents the “difficulty” in traversing through different areas of the map.



# Types of Costmaps

- Global costmap
  - Generated using data from static map.
  - Inflates the lines on the map.
  - Used by global planner to generate route.
- Local costmap
  - Generated using data from sensors (Eg Lidar, Ultrasonic)
  - Used by local planner to detect obstacles and plans path to avoid obstacle collision.

# ROS costmap package

- Subscribes to
  - ~<name>/footprint (geometry_msgs/Polygon)
    - Specifies the footprint of the robot.
- Publishes to
  - ~<name>/costmap (nav_msgs/OccupancyGrid)
    - Values in the costmap
  - ~<name>/costmap_updates (map_msgs/OccupancyGridUpdate)
    - The value of the updated area of the costmap.

Weston Robot

# ROS costmap package

- Costmaps consist of multiple layers. The most important layers are:
  - Static Map Layer
    - Represents the part of the costmap that is generally fixed, like those generated from using SLAM (Lab 4).
  - Obstacle Map layer
    - Tracks obstacles based on data received from sensor data.
  - Inflation Layer

# Inflation Layer

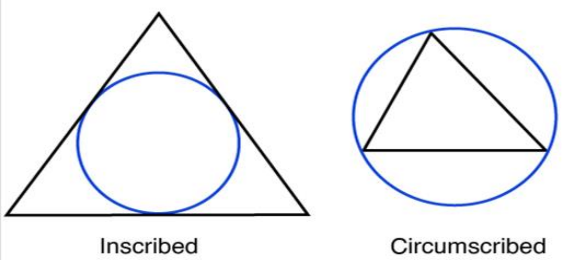
Weston Robot

- Each cell in the costmap can either be free, occupied or unknown.
- The inflation layer allocates cost values for each cell from 0 to 254. Where the cost value decreases with distance from the obstacle.
- There are 5 defined stages for costmap values:
  1. Lethal
     - Actual obstacle exists within the cell.

# Inflation Layer

Weston Robot

2. Inscribed
   - Cell is less than the inscribed radius of the robot from obstacle. Definite collision if the robot is within the cell.
3. Possibly circumscribed
   - The cell is more than the circumscribed radius of the robot. Collision may not be imminent and maybe dependent on orientation of robot.



# Inflation Layer

4. Freespace
    - No obstacle, robot should be free to move there.
5. Unknown
    - No cost information available about a given cell.

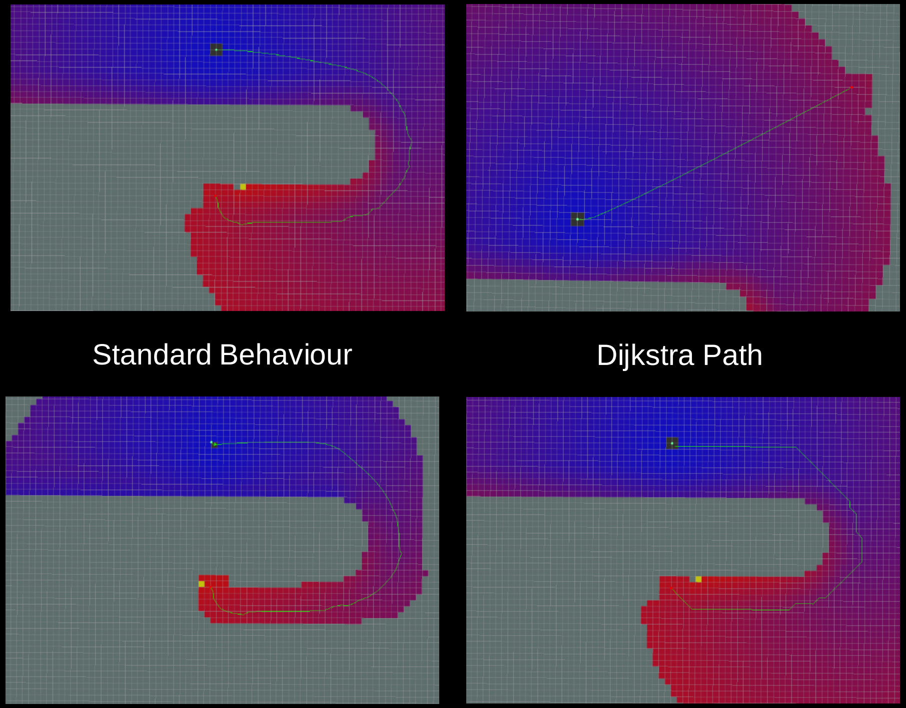
Weston Robot

# Types of planners

<table>
<tr><th>Global Planner</th><th>Local Planner</th></tr>
<tr><td>uses a prior information (from mapping) of the environment to create best possible path</td><td>transforms the global path to suitable waypoints, while taking into consideration of dynamic obstacles and vehicle constraints</td></tr>
<tr><td colspan="2"><ul><li>Global planner will plan a global path around existing and new obstacles (specified by planner_frequency parameter).</li><li>Local planner will do obstacle avoidance (where cmd_vel is produced and based of controller_frequency parameter), and try to follow global plan closely (taking into consideration a part of the global planner at a time)</li></ul></td></tr>
</table>

# Global Planners

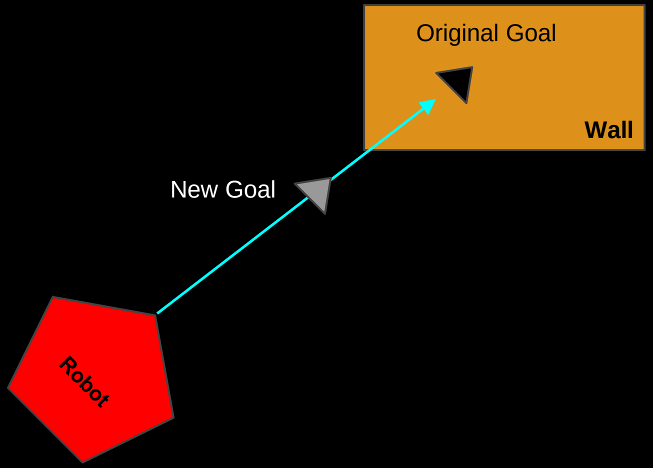
- navfn
  - grid based global planner using Dijkstra's algorithm
- global_planner
  - flexible replacement of navfn
    - supports A*
    - can use grid path



Standard Behaviour | Dijkstra Path | A* Path | Grid Path

# Global Planners

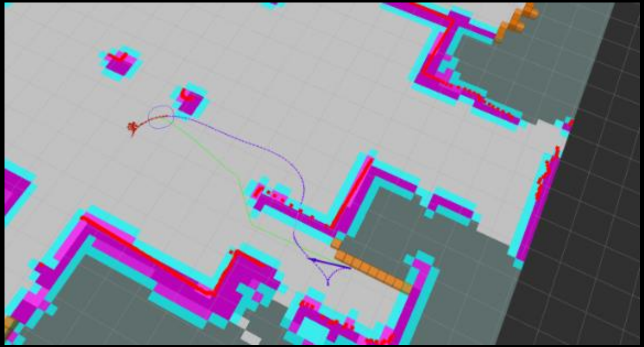
- carrot_planner
  - checks if goal is an obstacle
    - if yes: moves goal back along vector between robot and goal
    - if no: passes goal point as plan to local planner



Weston Robot

# Local Planners

- base_local_planner
  - Implementation of DWA and trajectory rollout approach
- dwa_local_planner
  - More flexible and modular compared to base_local_planner’s DWA implementation
- eband_local_planner
  - Elastic Band method
- teb_local_planner
  - Timed-Elastic-Band method
- mpc_local_planner
  - Model predictive control approaches



# limo_bringup

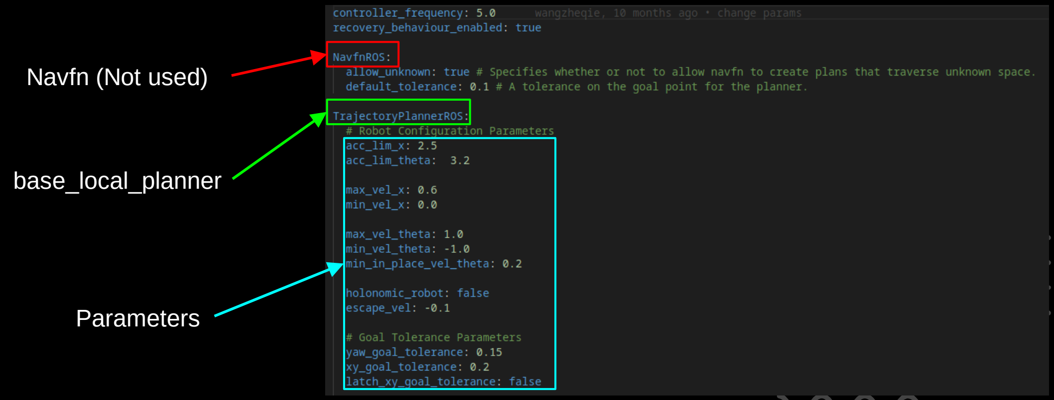
- Specifying which planners to use under …/limo_bringup/launch folder, within the limo_navigation_diff.launch file:
  - Global Planner: *global_planner*
  - Local Planner: *base_local_planner*

```xml
<!--  ************** Navigation ***************  -->
<node pkg="move_base" type="move_base" respawn="false" name="move_base" output="screen">
    <rosparam file="$(find limo_bringup)/param/diff/costmap_common_params.yaml" command="load" ns="global_costmap" />
    <rosparam file="$(find limo_bringup)/param/diff/costmap_common_params.yaml" command="load" ns="local_costmap" />
    <rosparam file="$(find limo_bringup)/param/diff/local_costmap_params.yaml" command="load" />
    <rosparam file="$(find limo_bringup)/param/diff/global_costmap_params.yaml" command="load" />
    <rosparam file="$(find limo_bringup)/param/diff/planner.yaml" command="load" />

    <param name="base_global_planner" value="global_planner/GlobalPlanner" />
    <param name="planner_frequency" value="1.0" />
    <param name="planner_patience" value="5.0" />
    <param name="base_local_planner" value="base_local_planner/TrajectoryPlannerROS" />
    <param name="controller_frequency" value="5.0" />
    <param name="controller_patience" value="15.0" />
        <param name="clearing_rotation_allowed" value="true" />
</node>
```

# limo_bringup

- Parameters used for the different Planners
  - File with the parameters is in the param folder under the …/param/diff/planner.yaml file
  - To avoid confusion: Navfn can be seen in the file, but we are using global_planner so those lines under Navfn ROS will not be part of the robot's planning
  - global_planner will use its default parameters since it is not specified within the .yaml file
- Further readings: http://wiki.ros.org/base_local_planner ; http://wiki.ros.org/global_planner#Parameters



Weston Robot

# Trajectory Tuning

- Cost Function to score each trajectory
  - pdist_scale (path distance bias): weighing for how much controller should stay within given path
  - gdist_scale (goal distance bias): weighing for how much controller should attempt to reach its local goal (controls speed as well)
  - occdist_scale: weighing on how much controller should avoid obstacles
- Further readings: http://wiki.ros.org/navigation/Tutorials/Navigation%20Tuning%20Guide

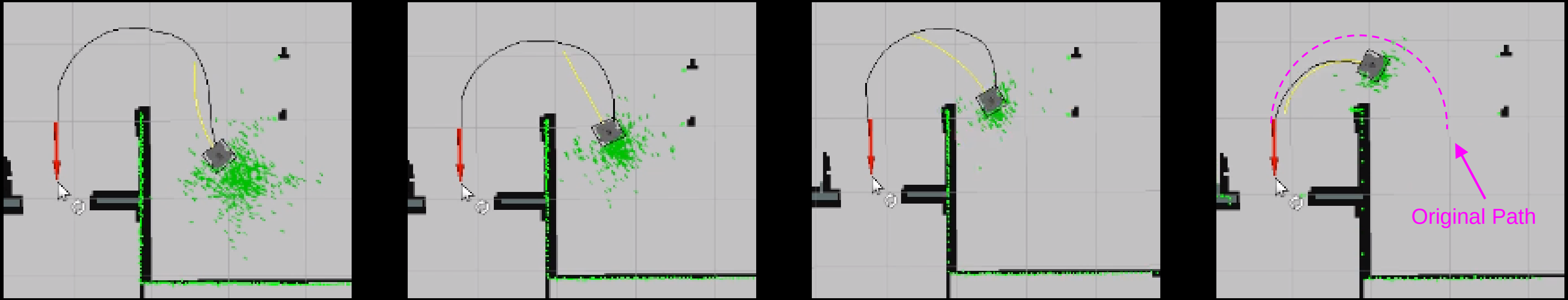
```yaml
# Trajectory scoring parameters
meter_scoring: true # Whether the gdist_scale and pdist_scale parameters should assume that goal_distance and path_distance are expressed in units of meters or cells.
occdist_scale:  0.1 #The weighting for how much the controller should attempt to avoid obstacles. default 0.01
pdist_scale: 2.5  #     The weighting for how much the controller should stay close to the path it was given . default 0.6
gdist_scale: 1.0 #     The weighting for how much the controller should attempt to reach its local goal, also controls speed  default 0.8

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

Trajectory scoring parameters in planner.yaml

# Cost Function

$$
\begin{aligned}
cost &= \texttt{path\_distance\_bias} * (\text{distance}(m) \text{ to path from the endpoint of the trajectory}) \\
&+ \texttt{goal\_distance\_bias} * (\text{distance}(m) \text{ to local goal from the endpoint of the trajectory}) \\
&+ \texttt{occdist\_scale} * (\text{maximum obstacle cost along the trajectory in obstacle cost (0-254)})
\end{aligned}
$$




Trying to stay within path

Steering from path and attempting to reach goal

Changing path and trying to stay within new path