Costmap_2d

1) Overview

- The costmap_2d package provides a configurable structure that maintains information about where the robot should navigate in the form of an occupancy grid.

- The costmap uses sensor data and information from the static map to store and update information about obstacles in the world through the costmap_2d::Costmap2DROS object.

- Each bit of functionality exists in a layer.

{By default, the obstacle layer maintains information three dimensionally (see voxel_grid).}

→ Maintaining 3D obstacle data allows the layer to deal with marking and clearing more intelligently.

2) Marking and Clearing

- Each sensor is used to either mark, clear, or both.

mark → just an index into an array to change the cost of a cell

clear → consists of raytracing through a grid from the origin of the sensor outwards for each observation

- If a three dimensional structure is used to store obstacle information, obstacle information from each column is projected down into two dimensions when put into the costmap.

- While each cell in the costmap can have one of 255 different cost values, the underlying structure that it uses is capable of representing only three.

Free
costmap_2d::FREE_SPACE cost

Occupied
costmap_2d::LETHAL_OBSTACLE cost

Unknown
costmap_2d::NO_INFORMATION cost

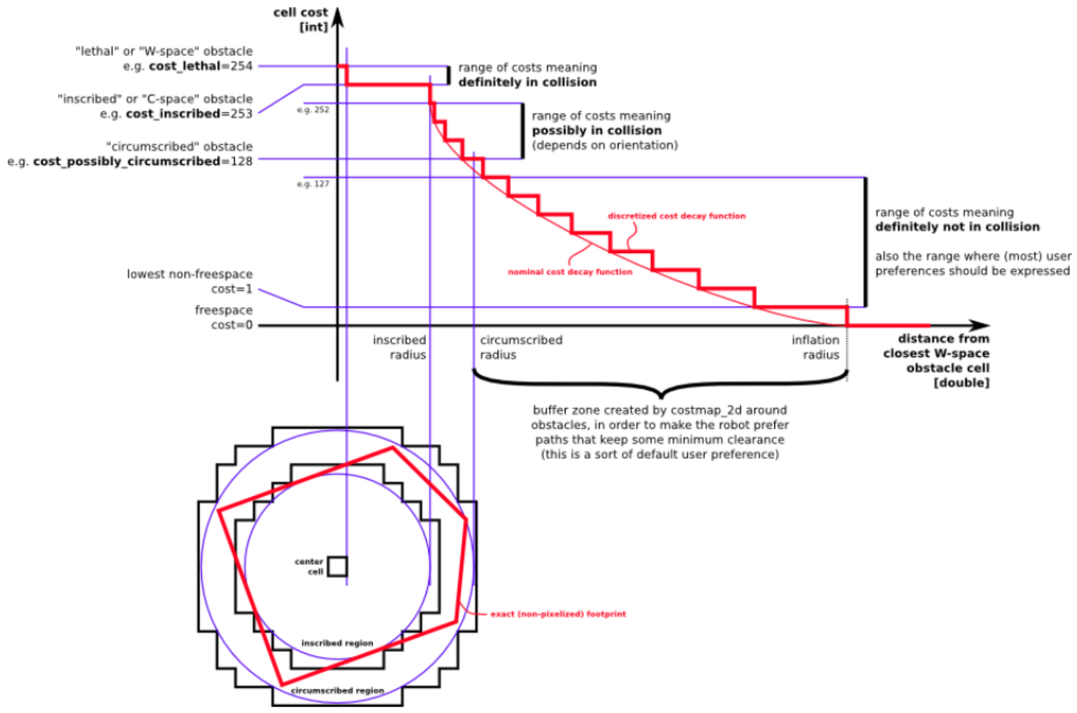
3) Map Update

- The costmap performs map update cycles at the rate specified by the update_frequency parameter.

4> tf

- It assumes that all transforms between the coordinate frames specified by the global_frame parameter, the robot_base_frame parameter, and sensor sources are connected and up-to-date.
- The transform_tolerance parameter sets the maximum amount of latency allowed between these transforms.

5> Inflation



cell cost [int]
"lethal" or "W-space" obstacle e.g. cost_lethal=254
"inscribed" or "C-space" obstacle e.g. cost_inscribed=253
"circumscribed" obstacle e.g. cost_possibly_circumscribed=128
lowest non-freespace cost=1
freespace cost=0
e.g. 252
e.g. 127
range of costs meaning definitely in collision
range of costs meaning possibly in collision (depends on orientation)
range of costs meaning definitely not in collision
also the range where (most) user preferences should be expressed
discretized cost decay function
nominal cost decay function
inscribed radius
circumscribed radius
inflation radius
distance from closest W-space obstacle cell [double]
buffer zone created by costmap_2d around obstacles, in order to make the robot prefer paths that keep some minimum clearance (this is a sort of default user preference)
center cell
exact (non-pixelized) footprint
inscribed region
circumscribed region

- Inflation is the process of propagating cost values out from occupied cells that decrease with distance.

6> Map type

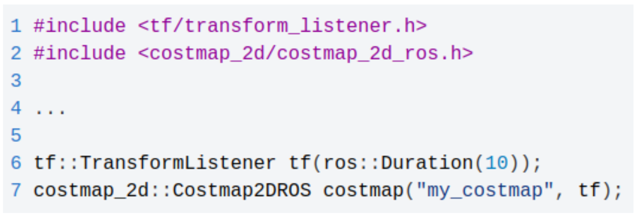
- There are two main ways to initialize a costmap_2d::Costmap2DROS object.
  - → seed it with a user-generated static map.
    - → In this case, the costmap is initialized to match the width, height, and obstacle information provided by the static map.
  - → The second way to initialize a costmap_2d::Costmap2DROS object is to give it a width and height and to set the rolling_window parameter to be true.
    - → The rolling_window parameter keeps the robot in the center of the costmap as it moves throughout the world, dropping obstacle information from the map as the robot moves too far from a given area.

7> Costmap2DROS

- The costmap_2d::Costmap2DROS object is a wrapper for a costmap_2d::Costmap2D object that exposes its functionality as a C++ ROS Wrapper.

- It operates within a ROS namespace specified on initialization.

- Example:



```cpp
#include <tf/transform_listener.h>
#include <costmap_2d/costmap_2d_ros.h>

...

tf::TransformListener tf(ros::Duration(10));
costmap_2d::Costmap2DROS costmap("my_costmap", tf);
```