Robot Localization problem in a virtual environment

Carlos Olcina

Abstract—A probabilistic approach to the localization problem of a mobile robotic platform is conducted in this work. Mobile localization is one of the basic problems in robotics. This works starts with a comparison between the Extended Kalman Filter and the Monte Carlos Localization algorthim (MCL), aslo known as particles filter, after that an analisys and a software implementation of the Monte Carlo Localization algorithm is made for two basic robots.

Index Terms—Localization, Particle filters, Udacity, Robotics, deep learning.

1 Introduction

WITHOUT localization a robot doesn't know where it is. There are a lot of robot that does not need to localize itself in a geograpich map. That is the case of the controlled numeric machines used for cutting or drillings, this machines use a limited number axis and they localize themselves using precise wheel encoders. When they lose the reference within their enviroment, a simple movement to the origin of the axis solve the localization problem. For a robot in the wild the things are much more complicated. Even the best actuators or motors have errors, the most precise laser measurements tools have errors. Additionally to this erros external and random variables such as slippery floors or "wild" animals can change the known state of a robot. That is the magnitude of the problem, an autonomous robot can not trust anything, not even itself.

A simple solution to this problem is simple, you don't trust anything. As the sensors and actuators are inerent uncertain, a probabilistic approach to the localization can be made defining both, measurements and motions as guest of where the robot is and how much the robot had move. This models working together have the ability of absorb the uncertainties an lead to a good guess of where the robot is.

In the further sections, a description of two of this probabilitic methods is made and lately they are compared. Addytionallly, an implementation of one of this algorithm is develop for a simple robot.

2 Background / Formulation

2.1 Probabilist robotics

In a software development environments, the state of a robot can as precise as the data type used in its definition. Therefore, modeling the motion of the robot with a deterministic approach gives the exact positioning/pose of the robot at any time, also know as the kinematic state x_t. This approach can be used when validating motion or controls algorithms but can lead to high inaccuracy in pose calculation for real environments. Additionally, each model should account for errors in measurements that event the most precise laser sensor have. For dealing with those uncertainties a probabilistic approach is made for both. This sections vaguely describes models and approaches to probabilistic robotics, for the interested reader extensive and detailed documentation can be found in probabilistic robotics by Sebastian Thrun [1].

2.1.1 Motion update

In probabilist robotic the state of the robot is uncertain and defined as a group of possible states, whenever there is a movement the uncertainty grows an more possible states are generated for the robot. This states need to be calculate for each movement and is defined as the state transition probability $p(x_t|u_t, x_{t-1})$. So, the state (x_t) in a given time is a probability density conditioned by the past state x_{t-1} and the motion commands passed to the robot u_t. For calculating this probability a motion model needs to be defined, and changes in each different robotic implementation. Nevertheless, in comparison with a deterministic approach, a probabilistic model is capable of recovering the state under different kinds of errors. For example, when a motion command is passed to a robot, if the wheels get stuck or slip a little bit a deterministic model would miss the correct final state, a probabilistic model on the other hand would have a set of possible states that accounts for the sum of all those errors whether they are know or not. In a real environment, both approaches need to measure or "sense" the environment to account for this errors and give a good estimation.

2.1.2 Measurement update

As the nature of measurement is uncertain by nature probabilistic robotics define the measurement as a conditional probability distribution $p(z_t|x_t, m)$.

$$p(z_t|x_t, m) \tag{1}$$

In equation 1 ,z_t is the measurement in time t and m the map. A good advantage of a probabilistic model instead a deterministic one is that allows to accommodate the uncertainties of the measurement, such as inaccuracies in measurement or reflexions in wall materials, without the need of explicitly define those variables. Normally, sensors

make more than one measurement at a particular time, so assuming the independence between measurements the probability distribution (2) can be obtained as the product of the individual ones.

$$p(z_t|x_t, m) = \prod_{k=1}^{K} p(z_t^k|x_t, m) \tag{2}$$

There are different and broadly documented probabilistic measurement models when dealing with range-finders two of the most commons are the beam range finder and the likelihood field range-finder.

In the beam range-finder model, defining z_t^{z*} as the true measurement an approximation to the physical model of the beam can be formulated mixin up four probability densities:

- **Correct range with local noise.** This can be represented as a normalized Gaussian distribution centered around the measurement,z_t, with a standard deviation σ_{hit}
- **Unexpected objects.** Represented as an exponential distribution
- **Failures.** Represented as a point distribution centered at the maximum range of the sensor, z_{max}
- **Random measurements.** Account for noise and unexplained measurements as a uniform distribution.

The likelihood field model takes another approach to model a range finder sensor. It projects the endpoints of the scans are mapped through trigonometric transformations and then sources of noise are added modeled as probability distributions, similarly as in the beam range model. Nevertheless, this model discard completely all the maximum reads of the range sensor.

2.2 Enviroment Definition

2.2.1 Map

In localization problems, the assumption of a known map is made. A map can be represented in multiples ways and it varies upon the localization technique used.

Feature-based maps use set of distinct characteristics of the environment, an example of this could be a field with a series of defined landmarks. The robot measures the distance to the landmarks and infers the pose from this measurements, in a feature-based map only the position of the objects/landmarks are known. kalman filters localization methods use feature-based maps to calculate the posterior Gaussian of the pose estimation.

Location-based maps model the environment as a labeled world. Assigning the free or occupied label to a grid of points for defining the robot world defines the occupancy grid maps. Grid localization and Monte Carlos Localization filters use location-based maps to estimate the robot pose.

2.2.2 Local vs Global

If the initial pose of the robot is known the filter only needs to account for the measurements of the robots to effectively reduce the uncertainty and track the position of the robot pose.

If the initial pose of the robot is unknown the problem to solve is named the global localization, in this case, the uncertainty is huge an covers all the map, the algorithm should be able to find the robot after the position is found and defined, the problem is reduced to the position tracking problem.

The last and more difficult of the localization problems is the kidnapped robot problem. Here the robot is teleported randomly to other position. A successful algorithm should be never totally confident of its position to recover from this problem.

In this work, only the position tracking and global localization problems are accounted.

2.2.3 Static vs Dynamic

An environment can be static or dynamic. In the previous section, the models for the measurement step of a robot account for unexpected objects and random values. This model is already capable of dealing with a dynamic environment such as moving objects or people passing by treating them as noise. A static environment never changes so the only variable to account for is the robot pose.

In this work, the map is defined as a static environment.

2.3 Localization Algorithms

Localization algorithms in probabilistic robotics are defined as variants of Bayes filters. A Bayes filter calculate the belief distribution from measurement and motion probabilities as described in 2.1.2 and 2.1.1 respectively. The belief over the state is an abbreviation for the posterior density and can be described as the probability of a state density at a time t based in all the past measurements and motions (see equation 3).

$$bel(x_t) = p(x_t|z_{1:t}, u_{1:t}) \tag{3}$$

The straightforward implementation of the Bayes filter to a localization problem is called Markov localization. This algorithm adds the map variable as influences directly the measurement model (2)

2.3.1 EKF

The extended Kalman filter needs an initial estimation of the robot time as a Gaussian distribution. As we inference, the position value from the velocity calculation this type of filter is highly susceptible to position drift and only can solve the tracking position problem

EKF can be extended to global localization problem with a Multi hypothesis tracking algorithm. Here, instead of using only one Gaussian to represent the posterior belief a mixture of Gaussian created dynamically is used. Using a defined threshold the algorithm keeps only the gaussian more likely to define the state of the robot.

2.3.2 Monte Carlo Localization

The Monte Carlo localization algorithm uses a particle filter and the probabilist motion and measurement models to infer the position of the robot. Particles filters are an alternative nonparametric implementation of the Bayes filter. The main idea behind the particles filters is to generate an approximation of the posterior belief through a finite number of samples (know as particles), these particles are

randomly generated through a sample motion model and resampled after the measurement step to keep only the most likely particles to represent the final state.

Furthermore, the MCL can solve the tracking position and global localization problems. The particle initial definition is not bound to a Gaussian distribution, so particles can be distributed throughout the entire map if the initial position is unknown leading to the solution of the global localization problem after a few steps of measurement and motion updates. Additional, applying a random particle generation technique the MCL can solve the kidnapped robot problem. This technique generates new particles randomly throughout the map, that way if the robot is suddenly translated to another position a set of those randomly generated particles became more likely than the previously believed ones.

2.3.3 EKF vs MCL

As talked before an essential difference between the EKF and the MCL is that the second one can solve the global localization and the kidnapped robot problem. Nevertheless, the EKF has some advantages in front of the MCL, it is more efficient and therefore can achieve better resolutions. On the other hand, MCL is easy to implement and don't relay in Gaussian distributions, also is the most robust of the probabilist filters as can recover for wide errors in the initial belief. The table 1 summarizes the main differences between the EKF and the MCL.

TABLE 1: Comparison between EKF and MCL

	EKF	MCL
Measurements	landmarks	raw measurements
Measurement noise	gaussian	any
Posterior	gaussian	particles
Efficiency	+	-
Implementation	-	+
Resolution	+	-
Robustness	-	+
Global localization	no	yes

2.4 Challenges

Localization is one of the most relevant problems to solve when developing an autonomous robot. When a robotic platform loses track of its position is not able to navigate the environment anymore and become a useless machine. There is where the importance of the localization algorithms resides, a fully autonomous robot is able to recover by itself from the uncertainties of the environment.

In real applications both filters have advantages and normally they are used in conjunction to make use of the best features of each. The main problem for real application of the particle filters is that it needs extensive memory and computational time to render and sample the particles. In embedded systems, the maximum number of particles should be set to a minimum number for the filter to achieve localization on the given map. At the same time, all the selection of parameters should ensure that the whole systems work smoothly reserving computing capabilities and memory for other systems and actuators that the robot may have. KLD-sampling is a technique to adapt the number of particles of an MCL based on the quality of these particles and can least the computational needs of the MCL.

Given a finite computing capacity for the robot and an environment the engineer should choose wisely between this filters, its variables and more parameters such as the resolution of the grid/feature map, to ensure the proper and smooth function of the whole system.

3 Results

The figure 1 shows the steps of a typical process of localization for the given robot, being the selected parameters the ones in the table 2.

TABLE 2: Parameters selection

parameter	value	package
min_particles	20	amcl
max_particles	500	amcl
initial_cov_xx	5	amcl
initial_cov_yy	5	amcl
initial_cov_xx	5	amcl
update_min_d	0.05	amcl
update_min_a	0.2	amcl
resample_interval	3	amcl
transform_tolerance	0.1	amcl
odom_alpha1	0.02	amcl
odom_alpha1	0.02	amcl
odom_alpha1	0.02	amcl
odom_alpha1	0.02	amcl
laser_z_hit	0.95	amcl
laser_sigma_hit	0.1	amcl
laser_z_rand	0.05	amcl
laser_likelihood_max_dist	2	amcl
kld_error	0.05	amcl
kld_z	0.99	amcl
recovery_alpha_slow	0.05	amcl
recovery_alpha_fast	0.2	amcl
sim_time	3	move base
meter_scoring	TRUE	move base
pdis_scale	2	move base
oscillation_reset_dist	0.5	move base
controller_frequency	10	move base
obstacle_range	2.5	costmap_2d
raytrace_range	3	costmap_2d
robot_radius	0.25	costmap_2d
inflation_radius	0.5	costmap_2d
update_frequency	6	globalcostmap
publish_frequency	2	globalcostmap
width	80	globalcostmap
height	80	globalcostmap
update_frequency	6	localcostmap
publish_frequency	2	localcostmap
width	10	localcostmap
height	10	localcostmap

The final position for the classroom robot can be observed in the figure 3

An additional result with a modified robot is shown in the figure REF, This robot has an additional multipurpose grip base in a circular rotating base. In this case, the extra a multiperspective camera to provide 3D scanning capabilities and an imu to improve tracking through EKF had been added to the base.

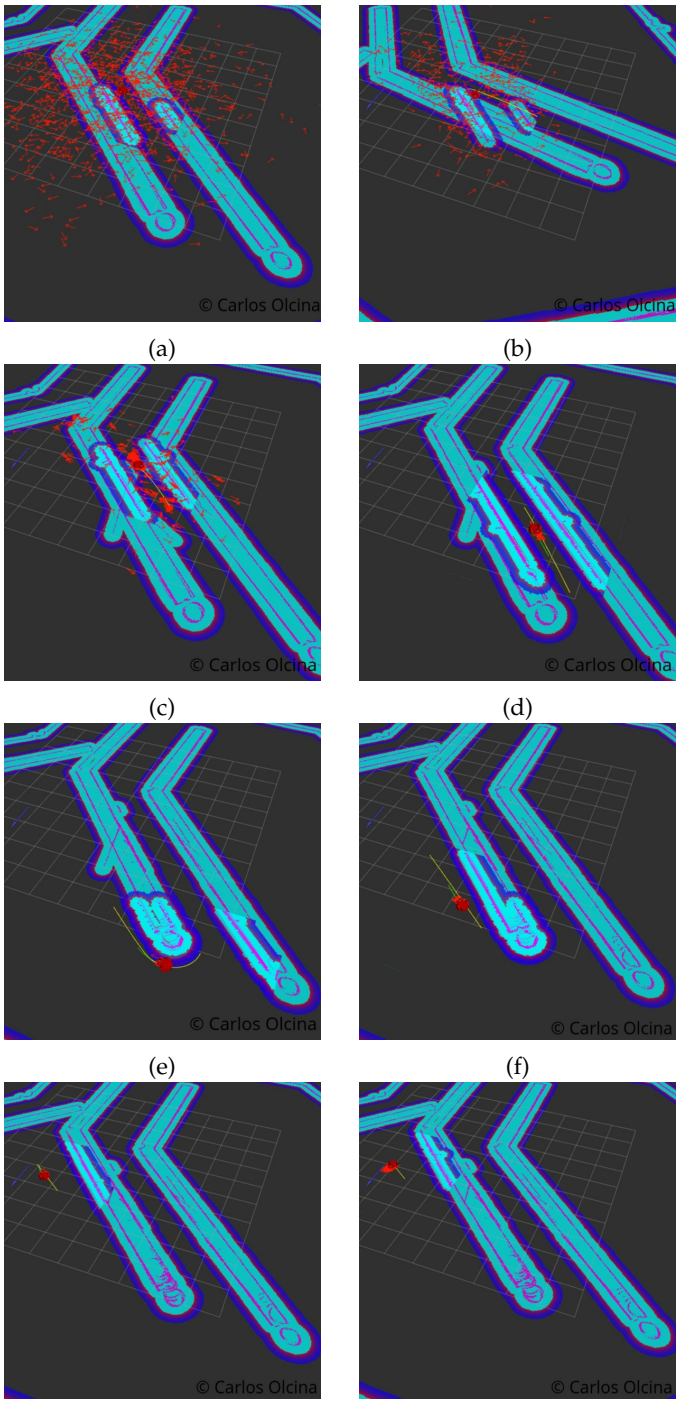


Fig. 1: Steps of a robot localization problem, from uncertainty to final goal

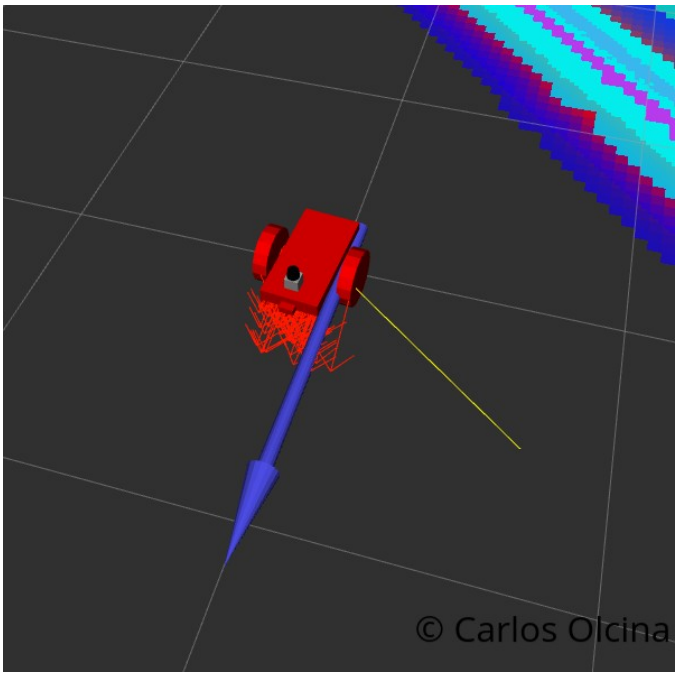


Fig. 2: final position udacity bot

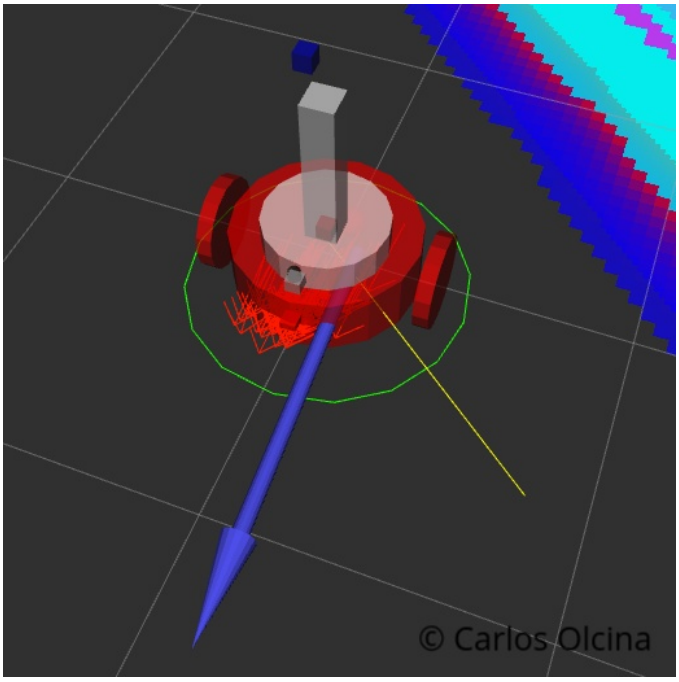


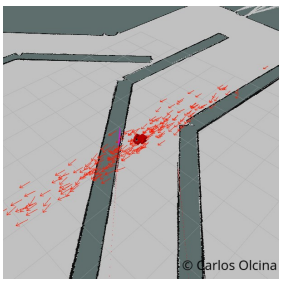
Fig. 3: final position my bot

4 Model Configuration

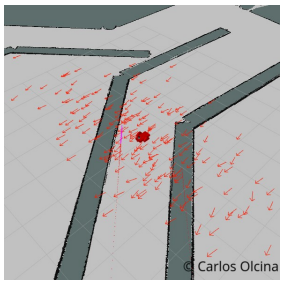
This section covers the different parameters of the necessary ROS packages to localize and navigate an autonomous robot as described in the ros navigation stack packages [2].

4.1 amcl

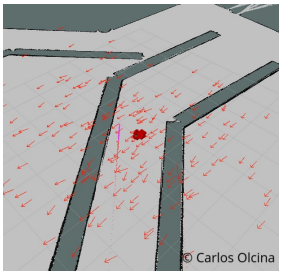
AMCL stands for Augmented Monte Carlo Localization and as the name describes, it handles the localization of the robot. This package is the one that computes the measurement and motion steps described in section 2.



(a) high initial x covariance



(b) high initial y covariance



(c) high initial x,y and yaw covariance

Fig. 4: initial covariance effect

4.1.1 particles

The first step of a MCL algorithm is the generation of the initial particles distribution. In the amcl ros package, the initial particles is set by the parameter max_particles and the initial covariances for the particles are defined in the parameters:

- initial_cov_xx (figure 4a)
- initial_cov_yy (figure 4b)
- initial_cov_aa (figure 4c)

An illustration of the effect of this initial parameters can be observed in the figure 4, the definition of this parameters should be made according to the initial information about the whereabouts of the robot.

kld_err and kld_err are the two parameters that control the generation or suppression of particles for the Kullback-Leibler-Divergence sampling (KLD-sampling) algorithm that this package implemented. The maximum particles are defined by the max_particles parameter already mentioned and the minimum by the min_particles parameter. KLD-sampling allows the MCL algorithm to adapt the number of particles over time-based in a probabilistic method. As the MCL algorithm has an accurate estimate of the robot position fewer particles are necessary to describe the pose and the problem of localization is simplified to a tracking problem, this parameters aim to ease the computational power required dropping down the number of particles as they become unnecessary.

For solving the kidnapped robot problem the Augmented-MCL needs to extra parameters recovery_alpha_slow and recovery_alpha_fast. They control the generation of new and random particles away from the estimated pose of the robot and the purpose of this is to recover the pose from failures or relocations of the robot.

4.1.2 odom model

The odometry motion model describe each movevement step as a rotation followed by a translation and a final rotation. The sampling algorithm for $p(x_t|u_t, x_{t-1})$ takes four parameters as inputs $(\alpha_1, \alpha_2, \alpha_3, \alpha_4)$, defined as odom_alpha1 ,odom_alpha2 ,odom_alpha3 and odom_alpha4 respectively.

α_1 and α_2 account for the noise in the rotations and a high value in this parameters leads to particle distributions with different angles (picture 5 left). α_3 and α_4 are in charge of the noise for the translational movement and the results for a high value in this parameters is a distribution with good oriented particles but widely spread in longitude (figure 5 right).

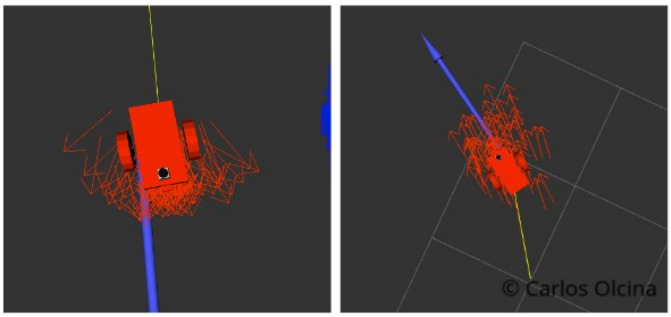


Fig. 5: odom alpha 12vs34

The figure 6 shows two different settings, both using the same value for all the parameters. A value of 0.2 was used to obtain the left figure and a value of 0.002 for the right one. Logically, a low value in the noise of the Odom parameters leads to a more accurate distribution of particles.

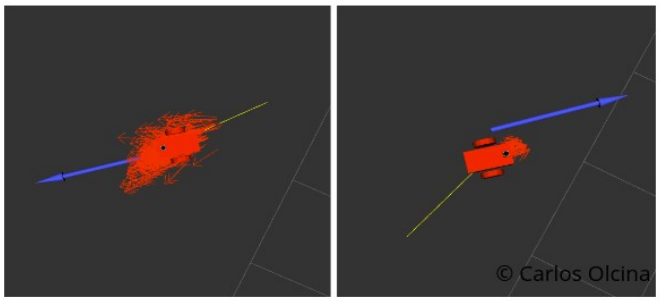


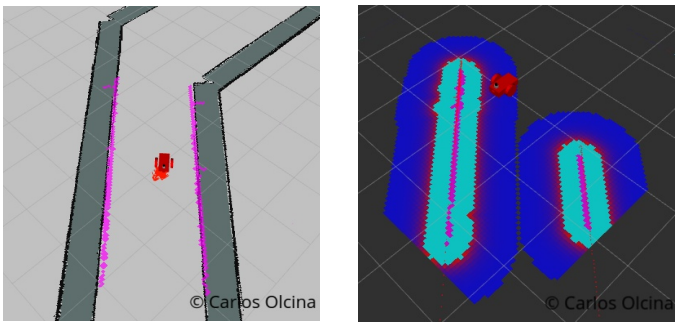
Fig. 6: odom alpha1234 variations

4.1.3 laser range model

The measurement noise is assessed using four parameters:

- laser_z_hit
- laser_sigma_hit
- laser_z_rand
- laser_likelihood_max_distance

Together they define the probabilistic model for the beam and therefore the likelihood model used to obtain the proposed obstacles for each measurement. z_{hit} and z_{rand} define the importance of the direct measurement noise from laser readings and the random measurements caused by moving objects respectively. laser_sigma_hit (σ_{hit}) defines



(a) inflation_radius = 0.0 (b) inflation_radius = 1.0

Fig. 7: Inflation radius effect

the standard deviation of a zero-centered Gaussian distribution to model the "hit" noise. In practice, a high value for σ_{hit} leads to more uncertain measurements and more steps are need to eliminate undesired particles.

4.2 cost map

The navigation step of the ROS stack needs a map to locate the robot. In this case, a cost map is used, the cost map is defined as a 2D array of cells that define where the robot can and can not be. These cells have a unique cost value within a range of 255 different values. Nevertheless, after define the parameters *robot_radius* and *inflation_radius*, cells are defined as either Lethal, Inscribed, Possibly circumscribed, Freespace or Unknown. Based on those values a cost map is generated, the *inflation_radius* acts as a safety measure when choosing the path to be followed by the robot. In the figure 7, the difference on a cost map with a high inflation radius and low one can be observed, the different colors in the right account for the different regions described above is the green area the Inscribed area and the blue area the Possibly circumscribed. Whit this information the path planning algorithm can choose safer routes for the robot avoiding these areas when possible and even canceling all motion when no safe path is found.

The parameters *obstacle_range* and *raytrace_range*, are used to add or clean new values to the global cost map when new data from the sensors is available. When the initial position is unknown the global map is not perfectly oriented, this adds objects and free spaces that are not real an should be clean with better measurements. A ray-trace range of zero will never clean the map for new measurements leading to unoccupied zones marked as occupied one. A good overview of this cost "cleaning" process can be observed in the section 3 between the figure 1e and the figure 1f, where a little appendix appeared in the previous localization step is erased with new measures when the robot pass by that precise area of the map.

5 Discussion

In my opinion, one of the main features of the MCL is the robustness of the algorithm under several conditions and scenarios. I tested this algorithm, in the given environment, with parameters to simulate a bad sensor with a good odometry, the other way around and both bad sensor and odometry. Even with so bad conditions in most cases, the robot was able to localize itself, not with the best accuracy of course, but a "not so bad" estimate of the pose was made in all the test. The robustness and ease of implementation of the MCL make this the perfect candidate for developing all kind of robots that need 2D localization. For example, in the toys industry, a smart developer could create a robot with cheap components and focus on a good tuning of the AMCL to obtain good performance. For other industries, that may need more accurate solutions, other ways of tracking such as Kalman filters should be added in conjunction with the MCL to improve the resolution of the localization when needed, that way we add the best qualities of each filter. Resolution and computing efficiency to the tracking problem for the KF and global localization and kidnapped robot problems for the MCL.

As the main purpose of this work is localization the navigation stack has not been developed in detail. That lead to situations where the robot get stuck even when it has options to continue navigating through the goal. One curious effect is found when the initial covariance for the particle distribution is too uncertain. In this cases sometimes the robot gets stuck in the first steps of localization because the cost of the global maps blocks all the possible navigation paths (see figure 8).

Another key scenario that we have not discussed before is what happened when the robot encounters a dynamic environment. In such environments, doors or temporary objects change the provided static map and can lead to a point where the robot is not capable of locale itself correctly even whit the generation of new particles as discussed before. A more advanced algorithm should take care of changing environments or maps. A real autonomous robot should be able to modify the given map to account this temporary changes and at the same time filter moving objects such as people or domestic animals. Nevertheless, people or animals passing by can be filtered if they are treated as random noise.

6 Conclusion / Future work

For improving this result and test a better dynamic model of the robot should be provided. In this work, we have tune the parameters for the localization in a virtual environment but for a real implementation of this project, we should refine the parameters with data from the real sensors. Also, mass values and dimensions should be provided to the gazebo worlds to test possible dangerous environments.

Navigations problems as the one presented in the previous section could be solved modifying the unstuck maneuvers. For example, when no path is found because of the orientation of the global cost map, a way to navigate and recover the orientation based only on the local map should be provided.

Another problem with particles filter occurs when there are several similar features on the map. For example, a map of the plant of a hotel has several rooms that look similar and a hallway that is not good for eliminating particles. In this cases, active exploration algorithms should be provided.

One of the main limitations of this model is that a good accurate map of the real environment is necessary. So, once

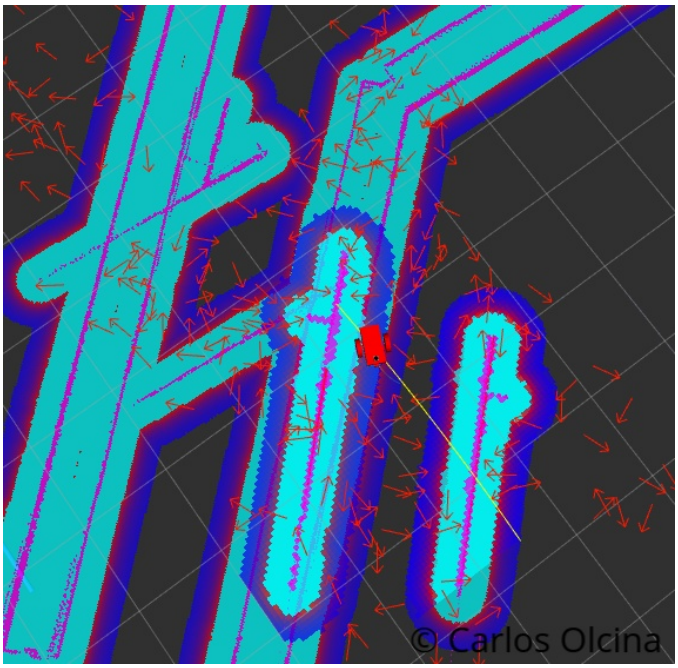


Fig. 8: A robot virtually stuck

that we have tested our model in a virtual environment, the next step should provide the autonomous robot with the capabilities of creating the same map on the fly.

REFERENCES

[1] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.
[2] B. P. Gerkey, "Ros navigation stack." https://github.com/ros-planning/navigation, 2017.