Robotic Localization

Carlos Lacerda

Abstract—The target of this project is to study some techniques used to estimate the location of a robot in the world - a process known as localization. Localization involves one question: Where is the robot now? Or, robo-centrically, where am I, keeping in mind that ”here” is relative to some landmark (usually the point of origin or the destination) and that you are never lost if you don’t care where you are. Although a simple question, answering it isn’t easy, as the answer is different depending on the characteristics of your robot. So, this project also target the design and model configuration of a robot in addition of creating a simulated environment using ROS, Gazebo and Rviz.

Index Terms—Robot, IEEEtran, Udacity, LaTeX, Localization.

1 Introduction

Navigation is one of the most challenging competencies required of a mobile robot. Success in navigation requires success at the four building blocks of navigation (fig. 2):

- *perception*- the robot must interpret its sensors to extract meaningful data;
- *localization*- the robot must determine its position in the environment
- cognition- the robot must decide how to act to achieve its goals
- **motion control** the robot must modulate its motor outputs to achieve the desired trajectory.

Of these four components, localization has received the greatest research attention in the past decade and, as a result, significant advances have been made on this front. Mobile robots need to answer three fundamental questions:

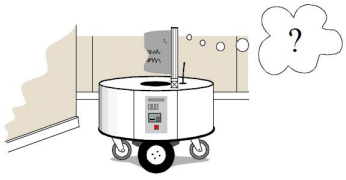

Fig. 1. Where Am I?

Where am I? Where am I going? How do I get there? To answer these questions the robot must first:

- Make measurements
- Model the environment
- Localize it self
- Plan a path to its goal

In the next section we will discuss some methods used by mobile robot to localize itself.

2 Background

Today GPS makes outdoor localization so easy that we often take this capability for granted. Unfortunately GPS is a far from perfect sensor since it relies on very weak radio signals received from distant orbiting satellites. This means that GPS cannot work where there is no line of sight radio reception, for instance indoors, underwater, underground, in urban canyons or in deep mining pits. GPS signals are also extremely weak and can be easily jammed and this is not acceptable for some applications. We can define the

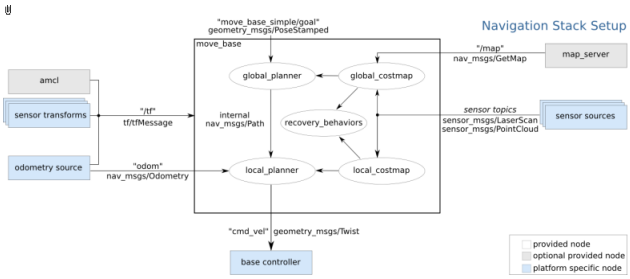

Fig. 2. ROS navigation stack

localization problem more formally where $\boldsymbol{x}$ is the true, but unknown, position of the robot and $\hat{x}$ is our best estimate of that position. We also wish to know the *uncertainty* of the estimate which we can consider in statistical terms as the standard deviation associated with the position estimate $\hat{x}$. So, considering the environment (map), the observations (sensors), the actuators (encoders), we can draw a general schematic localization as in Fig. 2. There are different meth-

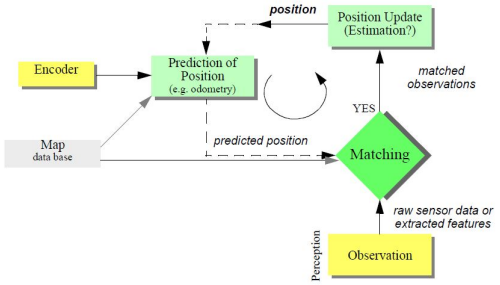

Fig. 3. General schematic for mobile robot localization

ods we can use to localize a robot where each one has its pros and cons.

2.1 Dead reckoning

Dead reckoning is the estimation of a robots pose based on its estimated speed, direction and time of travel with respect to a previous estimate.

- Relative to initial conditions
- Prone to drift and slip
- Unbounded error growth
- Easy

Result: Over time, robot belief does not match reality.

2.2 Sensor Fusion

This method combines measurements from different sensors to reduce overall error.

- Using probability theory, multiple error models combine to produce better measurements
- Any additional information, with properly modeled error, will only improve the measurement
- Unbounded error growth
- Easy

2.3 Kalman Filtering

This method uses a series of measurements observed over time, containing statistical noise and other inaccuracies, and produces estimates of unknown variables that tend to be more accurate than those based on a single measurement alone, by estimating a joint probability distribution over the variables for each timeframe.

- Assumes zero mean error
- Uses Gaussian PDF
- Unbounded error growth
- Easy

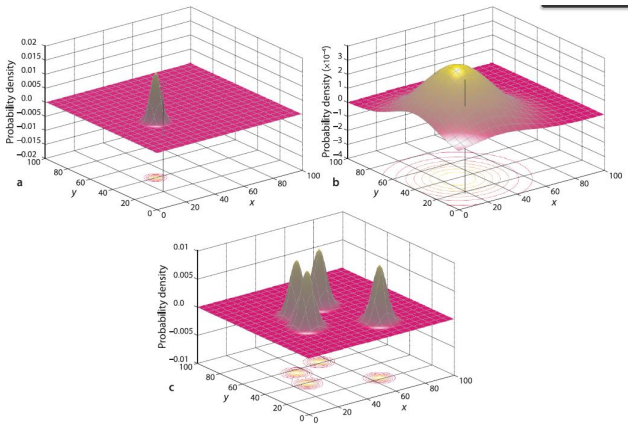

Fig. 4. Vehicle position and uncertainty in the xy-plane. Contour lines are displayed on the lower plane. **a**, the vehicle has low position uncertainty, σ = 1. **b**, the vehicle has much higher position uncertainty, σ = 20. **c**, the vehicle has multiple hypothesis for its position, σ = 1

2.4 Bayesian Filtering

The powerful Monte Carlo localization algorithm estimates the posterior distribution of a robots position and orientation based on sensory information. This process is known as a recursive Bayes filter. Using a Bayes filtering approach, roboticists can estimate the state of a dynamical system from sensor measurements.

- Can model non-linear systems
- Do not assume Gaussian PDF
- Can produce likely solutions without initial estimate of state
- Multiple belief system
- Slower

2.5 Monte Carlo methods

Are a class of computational algorithms that rely on repeated random sampling to compute their results

- Also refereed as Particle Filter Localization Algorithm
- Often used when simulating systems with a large number of coupled degrees of freedom with significant uncertainty in inputs
- Used when it is infeasible or impossible to compute an exact result with a deterministic algorithm
- Well suited to calculation by a computer
- Local localization
- Global localization
- Easy to use

Each particle as a position and orientation and represent a guess of where the robot might be located. These particles are re-sampled each time the robot moves and sense its environment. Based on these features, this is the algorithm chosen to be used on this project.

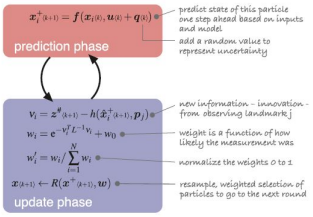

Fig. 5. he particle filter estimator showing the prediction and update phases

2.6 The Challenge of Localization: noise and aliasing

Sensor noise – Sensors are the fundamental robot input for the process of perception, and therefore the degree to which sensors can discriminate world state is critical. Sensor noise induces a limitation on the consistency of sensor readings in the same environmental state and, therefore, on the number of useful bits available from each sensor reading. Often, the source of sensor noise problems is that some environmental features are not captured by the robots representation and are thus overlooked. Sensor aliasing – Sensor aliasing causes sensors to yield little information content, further exacerbating the problem of perception and, thus,

localization. The problem is a phenomenon that humans rarely encounter. The human sensory system, particularly the visual system, tends to receive unique inputs in each unique local state. In other words, every different place looks different. The power of this unique mapping is only apparent when one considers situations where this fails to hold. Consider moving through an unfamiliar building that is completely dark. When the visual system sees only black, ones localization system quickly degrades. Another useful example is that of a human-sized maze made from tall hedges. Such mazes have been created for centuries, and humans find them extremely difficult to solve without landmarks or clues because, without visual uniqueness, human localization competence degrades rapidly.

Sensor noise and aliasing are external variables that will impact the input values read by robot. This way the EKF algorithm add uncertainty variable to prediction step and Kalman gain step.

A standard linear model has some limitations that require different methods to model motion uncertainty. Nonlinear motion updates break the Gaussian properties of the state distribution. To deal with nonlinearities, EKF utilizes a small set of sample points. In EKF the prediction model and, sometimes, the observation model, are nonlinear. So, we must linearize them by taking the Tailor Series.

2.7 Particle Filters

The key idea of MCL is to represent the belief by a set of samples (also called: particles), drawn according to the posterior distribution over robot poses. In other words, rather than approximating posteriors in parametric form, as is the case for Kalman filter and Markov localization algorithms, MCL simply represents the posteriors by a random collection of weighted particles which approximates the desired distribution. Within the context of localization,

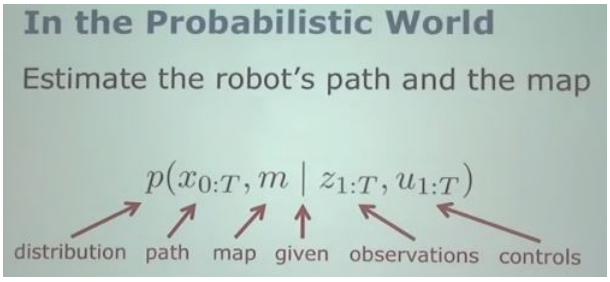

Fig. 6. Particles algorithm

the particle representation has a range of characteristics that sets it aside from previous approaches:

1) Particle filters can accommodate (almost) arbitrary sensor characteristics, motion dynamics, and noise distributions.
2) Particle filters are universal density approximators, weakening the restrictive assumptions on the shape of the posterior density when compared to previous parametric approaches.
3) Particle filters focus computational resources in areas that are most relevant, by sampling in proportion to the posterior likelihood.
4) By controlling the number of samples on-line, particle filters can adapt to the available computational resources. The same code can, thus, be executed on computers with vastly different speed without modification.
5) Finally, participle filters are surprisingly easy to implement, which makes them an attractive paradigm for mobile robot localization.

However, there are pitfalls. For example, if the sample set size is small, a well-localized robot might lose track of its position just because MCL fails to generate a sample in the right location. The regular MCL algorithm is also unfit for the kidnapped robot problem, since there might be no surviving samples nearby the robots new pose after it has been kidnapped. Somewhat counter-intuitive is the fact that the basic algorithm degrades poorly when sensors are too accurate. In the extreme, regular MCL will fail with perfect, noise-free sensors.

2.8 Comparison / Contrast

Extended Kalman Filter (EKF) and the Particle Filter (PF) are suitable alternatives for applications to detailed physically-based hydrological models. For each assimilation period, both methods use a Monte Carlo approach to approximate the state probability distribution (in terms of mean and covariance matrix) by a finite number of independent model trajectories, also called particles or realizations. The two approaches differ in the way the filtering distribution is evaluated. In the Fig. 5, $\hat{V}$ and $\hat{W}$ represents the uncertainty in prediction step and Kalman gain step, respectively.

EKF implements the classical Kalman filter, optimal only for linear dynamics and Gaussian error statistics. Particle filters, instead, use directly the recursive formula of the sequential Bayesian framework and approximate the posterior probability distributions by means of appropriate weights associated to each realization. EKF uses Importance Resampling technique, which retains only the most probable particles, in practice the trajectories closest in a statistical sense to the observations, and duplicates them when needed. In contrast to EKF, particle filters make no assumptions on the form of the prior distribution of the model state, and convergence to the true state is ensured for large enough ensemble size.

Particles filters can be considered as providing an approximate solution to the true system model, whereas a Kalman filter provides an exact solution to an approximate system model.

3 Model Configuration

In this project the model was restricted to two dimensions: x and y axis. This project used two rover robots with similar configurations: the robot label **A** with configuration previously provided and robot label **B** that was configured from scratch. Both robots use differential drive system.The differential drive is a two-wheeled drive system with independent actuators for each wheel.

The rover robot **B** was constructed from scratch to allow test different configuration values.

Input : $\hat{\boldsymbol{x}}\langle k\rangle \in \mathbb{R}^n$, $\hat{\boldsymbol{P}}\langle k\rangle \in \mathbb{R}^{n\times n}$, $\boldsymbol{u}\langle k\rangle \in \mathbb{R}^m$, $\boldsymbol{z}\langle k+1\rangle \in \mathbb{R}^p$, $\hat{\boldsymbol{V}} \in \mathbb{R}^{n\times n}$, $\hat{\boldsymbol{W}} \in \mathbb{R}^{p\times p}$

Output: $\hat{\boldsymbol{x}}\langle k+1\rangle \in \mathbb{R}^n$, $\hat{\boldsymbol{P}}\langle k+1\rangle \in \mathbb{R}^{n\times n}$

$-$ *linearize about* $\boldsymbol{x} = \hat{\boldsymbol{x}}\langle k\rangle$

compute Jacobians: $\boldsymbol{F}_x \in \mathbb{R}^{n\times n}$, $\boldsymbol{F}_v \in \mathbb{R}^{n\times n}$, $\boldsymbol{H}_x \in \mathbb{R}^{p\times n}$, $\boldsymbol{H}_w \in \mathbb{R}^{p\times p}$

$-$ *the prediction step*

$$\hat{\boldsymbol{x}}^+\langle k+1\rangle = \boldsymbol{f}\left(\hat{\boldsymbol{x}}\langle k\rangle, \boldsymbol{u}\langle k\rangle\right) \quad \textit{// predict state at next time step}$$

$$\hat{\boldsymbol{P}}^+\langle k+1\rangle = \boldsymbol{F}_x\hat{\boldsymbol{P}}\langle k\rangle\boldsymbol{F}_x^T + \boldsymbol{F}_v\hat{\boldsymbol{V}}\boldsymbol{F}_v^T \quad \textit{// predict covariance at next time step}$$

$-$ *the update step*

$$\nu = \boldsymbol{z}\langle k+1\rangle - \boldsymbol{h}\left(\hat{\boldsymbol{x}}^+\langle k+1\rangle\right) \quad \textit{// innovation : measured} - \textit{predicted sensor value}$$

$$\boldsymbol{K} = \boldsymbol{P}^+\langle k+1\rangle\boldsymbol{H}_x^T\left[\boldsymbol{H}_x\boldsymbol{P}^+\langle k+1\rangle\boldsymbol{H}_x^T + \boldsymbol{H}_w\hat{\boldsymbol{W}}\boldsymbol{H}_w^T\right]^{-1} \quad \textit{// Kalman gain}$$

$$\hat{\boldsymbol{x}}\langle k+1\rangle = \hat{\boldsymbol{x}}^+\langle k+1\rangle + \boldsymbol{K}\nu \quad \textit{// update state estimate}$$

$$\hat{\boldsymbol{P}}\langle k+1\rangle = \hat{\boldsymbol{P}}^+\langle k+1\rangle - \boldsymbol{K}\boldsymbol{H}_x\hat{\boldsymbol{P}}^+\langle k+1\rangle \quad \textit{// update covariance estimate}$$

Fig. 7. EKF algorithm

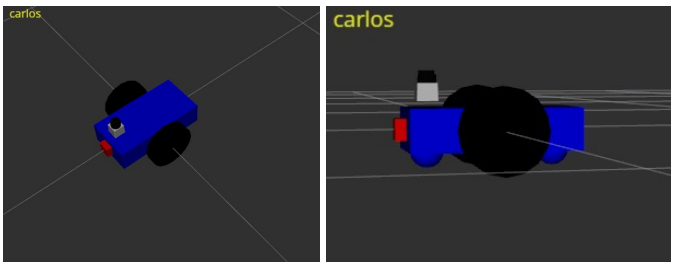

Fig. 8. Robot A: Provided configuration

The rover robot **B** that has a customized configuration when compared with rover robot **A** showed to have a better performance during the localization process.

3.1 Achievements

The rover robot **B** showed to have a better performance than rover robot **A** as showed in Table 1. The customization in robot configuration and parameters tunning showed to be more efficient for **B**. One important point to note is that robot **B** has less contact points than **A**, which means less collisions.

TABLE 1
Configuration of robot **A**

Component	Value
wheels	2
wheel radius	0.1
wheel length	0.05
caster	2
chassis size	0.4 0.2 0.1
mass	5
camera	RGB
laser scan	hokuyo

3.2 Benchmark Model

3.2.1 Model design

As showed in Table 1, robot **A** uses a RGB camera and a Hokuyo laser scan.

The camera is located in the front part of the robot and the laser scan in the up side of the robot.

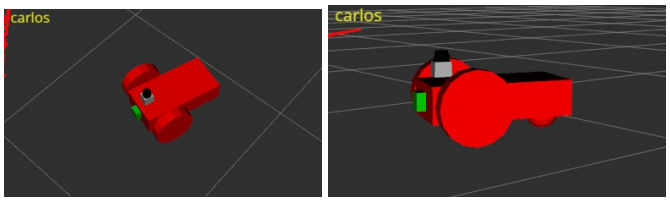

Fig. 9. Robot B: Customized configuration

3.2.2 Packages Used

The topics and nodes where populated according the Fig. 8.

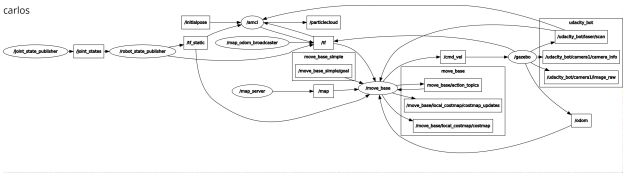

Fig. 10. Topics and nodes connections.

The following packages where used in the rover robot **A**.

- Name: differential_drive_controller, Topic: /udacity_bot/laser/scan
- Name: camera_controller, Topic: image_raw
- Name: gazebo_ros_head_hokuyo_controller, Topic: udacity_bot/laser/scan

3.2.3 AMCL Parameters tunned in robot **A**

The AMCL package is used to localize the robot in the map. The AMCL and, to provide a good performance, some values where tunned. It is the abbreviation of Adaptive Monte Carlo Localization, also known as particle filter localization. Each sample stores a position and orientation data representing the robot's pose.

Particles

The particle number must be chosen accordingly because a higher number of particles might need a higher computational resource and a lower number might not be enough to properly predict the robot location.

- min_particles = 500, max_particles = 550

Transform tolerance

Transform tolerance is the time with which post-date the transform that is published, to indicate that this transform is valid into the future.

- transform_tolerance = 0.5

3.2.4 Costmap Parameters

In ROS, *costmap* is composed of static map layer, obstacle map layer and inflation layer. Static map layer directly interprets the given static SLAM map provided to the navigation stack. Obstacle map layer includes 2D obstacles and 3D obstacles (voxel layer). Inflation layer is where obstacles are inflated to calculate cost for each 2D *costmap* cell. Besides, there is a *global costmap*, as well as a *local costmap*. *Global*

costmap is generated by inflating the obstacles on the map provided to the navigation stack. *Local costmap* is generated by inflating obstacles detected by the robot's sensors in real time.

Footprint

Footprint is the contour of the mobile base. As the robot is not circular, we have:

- footprint = [[-0.2,-0.1],[-0.2,0.1],[0.2, 0.1],[0.2,-0.1]]

Inflation

Inflation is the layer of cells with cost ranging from 0 to 255. Each cell is either occupied, free of obstacles, or unknown. *Inflation_radius* controls how far away the zero cost point is from the obstacle.

- inflation_radious = 4.0

Update and Publish frequency

The *update frequency* is frequency at which the cost map runs its main update loop. The publishing frequency of the cost map is given as *publish_frequency*.

- update_frequency = 5.0 (local costmap)
- publish_frequency = 2.0 (local costmap)
- update_frequency = 10.0 (global costmap)
- publish_frequency = 10.0 (global costmap)

The above amcl and costmap parameters are the ones most used for tunning the robot localization in this project. The complete list of parameters used in rover robot **A**, can be found in the git repository, here.

3.3 Personal Model

The rover robot **B** was designed and built from scratch. This approach allowed chose different configuration which provided some interesting results. The robot **B** model can be seen in Fig. 7.

3.3.1 Model design

The Table 2 shows the main parts of robot **B**.

TABLE 2
Configuration of robot **B**

Component	Value
wheels	2
wheel radius	0.1
wheel length	0.05
caster	1
chassis size	0.4 0.2 0.1
mass	10
camera	RGB
laser scan	hokuyo

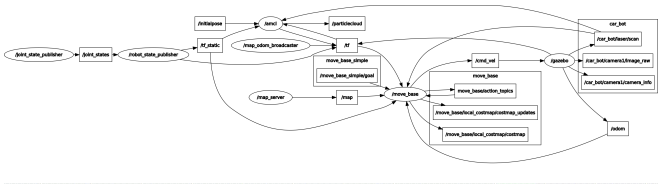

Fig. 11. Topics and nodes connections.

3.3.2 Packages Used

The topics and nodes where populated according the Fig. 9.

The following packages where used in the rover robot **B**.

- Name: differential_drive_controller, Topic: /car_bot/laser/scan
- Name: camera_controller, Topic: image_raw
- Name: gazebo_ros_head_hokuyo_controller, Topic: car_bot/laser/scan

*3.3.3 AMCL Parameters tunned in robot **B***

Particles

- min_particles = 1000, max_particles = 1500

Transform tolerance

- transform_tolerance = 0.5

*3.3.4 Costmap Parameters tunned in robot **B***

Footprint

- footprint = [[-0.2,-0.1],[-0.2,0.1],[0.2, 0.1],[0.2,-0.1]]

Inflation

- inflation_radious = 3.5

Update and Publish frequency

- update_frequency = 5.0 (local costmap)
- publish_frequency = 1.0 (local costmap)
- update_frequency = 5.0 (global costmap)
- publish_frequency = 2.0 (global costmap)

4 RESULTS

After many tries, it was verified that the time the robot **B** takes to reach the goal depends on its initial pose. If the robot is initially oriented to the same direction it must take, than the total time will be lower to reach the goal. By the other side, as much as the initial orientation is different from the correct direction then the total time to reach the goal increases. So, the lower time to reach the goal was around 82 seconds in average. In the same scenario the robot **A** took about 110 seconds. The time for the particle filters to converge takes 30 seconds in average.

During the trajectory, it was verified that robot **A** hits the front caster many times in the ground which means collision. This can explain why it takes more time to reach the goal even having acceleration and velocity values higher than robot **B**. The time for particle filters to converge takes 50 seconds in average.

The robot **B** follows a better path than robot **A**, making turns near the obstacles but without colliding as occur with robot **A**. In addition, robot **B** does a smooth path and turns around 2 times before confirm that reached the goal. The robot **A** has more difficulty to confirm the goal reached.

TABLE 3
Configuration and Results

Robot	A	B
mass	5	10
caster	2	1
turns	3	0
max_vel_x	0.5	0.7
max_vel_theta	1.5	1.5
acc_lim_theta	2.0	1.5
acc_lim_x	0.3	0.5
time	73	110

4.1 Localization Results

The table 3 shows how the difference in design model may influence the performance results.

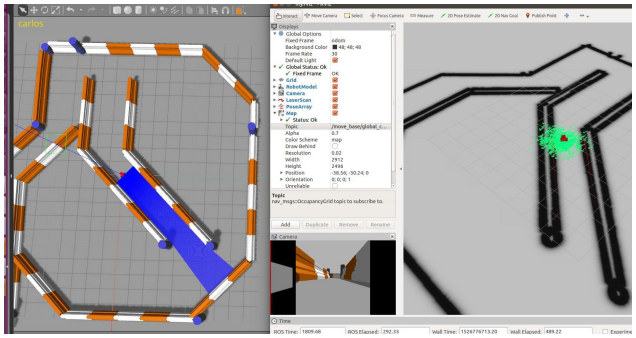

Fig. 12. Starting localization

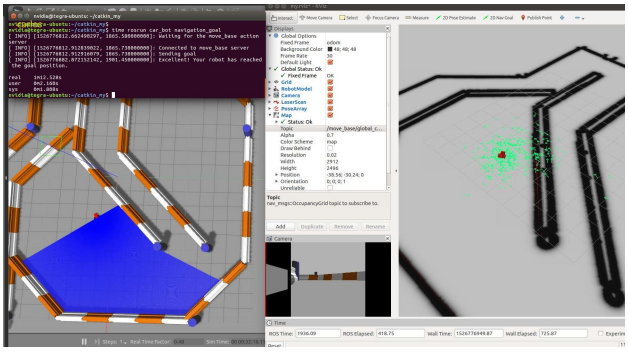

Fig. 13. Ending localization

4.1.1 Benchmark

The robot **A** takes about 110 seconds in average to complete the goal.

4.1.2 Student

The robot **B** takes about 73 seconds in average to complete the goal.

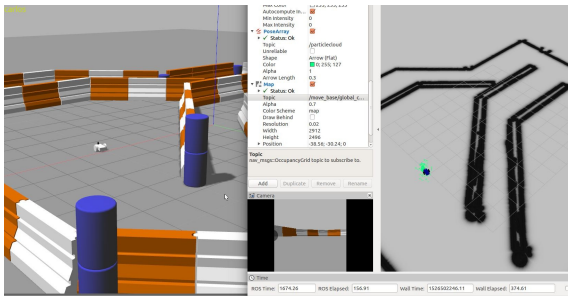

Fig. 14. Ending localization

4.2 Technical Comparison

The table 3 show that robot **B** has 2x mas of robot **A**, but robot **B** has max vel x and acc lim x, a bit higher than **A**. Yes, it was not possible set the same vel and acc to robot **A** because it turns difficult to drive, hitting the wall all the times. So, the stability makes robot **B** to hve a better performance.

During the movement it is possible to see the robot **A** tend to hit many times in the ground which does not happens with robot **B**.

5 Discussion

This project showed an interesting behavior where the design of robot **A** has the wheels located in the middle of the chassis while robot **B** has the wheels shifted a bit ahead in the chassis. While robot **A** is moving, both casters hit the ground and in robot **B** the caster has a permanent contact with ground. This lead to conclude that one caster with permanent contact with ground provide better behavior to robot when compared with two casters that keeps hitting with ground.

The localization is better with robot **B**.

One of the most flexible aspect about ROS navigation is dynamic reconfiguration, since different parameter setup might be more helpful for certain situations, e.g. when robot is close to the goal. So, thanks to rqt_reconfigure tool because using the ROS dynamic_reconfigure it was possible to reconfigure the robot dynamically without the need to restart the entire environment. The used command was:

rosrun rqt_reconfigure rqt_reconfigure

5.1 Topics

- Robot **B** performed better localization.
- The better performance is due to robot stabilization provided by better configuration.
- The increase of particle number showed to provide better behavior with 'Kidnapped Robot' problem but, again, robot **B** provided better results.
- Localization can be performed on outdoor and indoor scenarios depending on the sensors used.
- Autonomous vehicles, warehouse are some systems where amcl localization can be used.

6 Conclusion / Future work

In future project will use a similar robot with 4 wheels. This will remove the caster collision that can provide better performance.

Implement this configuration in a real robot is part of future work as well.

6.1 Modifications for Improvement

- Increase base dimension in length
- Use 4 wheels

6.2 Hardware Deployment

1) Use Jetson TX2 provided to be enough hardware resource during the simulation environment and can be a nice choice in a real robot to improve the computation time.