Where Am I?

A Localization project for the Robotics Nanodegree Program, Udacity

Muhammad Khalaf

Abstract—"Where Am I?" is a localization project that is intended to utilize some of ROS packages like amcl and the navigation stack. In addition to that, two robot models are built in a gazebo world to show the results of tuning some parameters in amcl and the navigation stack packages. Parameters are tuned in order to reach the best and most accurate localization output for the two robots in a previously mapped environment.

Index Terms—Robot, Udacity, ROS, AMCL, Localization, Navigation.

1 Introduction

ROBOT localization is the problem of determining the pose of a robot relative to a given map of the environment. It is often called position estimation or position tracking. Nearly all robotics tasks require knowledge of the location of the robots and the objects that are being manipulated [1]. There are three main types of localization. The first one is position tracking or local localization, where the initial robot pose is known. The second one is global localization, here the initial pose of the robot is unknown. And the third one is the Kidnapped robot problem, in which in which a well-localized robot is secretly teleported somewhere else without being told, and it is the hardest of the three localization problems. There are many solutions for the localization problem, such as: Markov localization, extended Kalman filter or (EKF) and Monte Carlo localization or (MCL). EKF and MCL well be discussed in more detail in the upcoming sections. In this project, a ROS package called AMCL is used for localizing a mobile robot. AMCL implements the adaptive Monte Carlo localization which uses a particle filter to track the pose of a robot against a known map [2]. The navigation stack is another ROS package that takes information from different sensors and a goal pose and outputs the velocity commands to the mobile robot [3]. After creating the robot model, that utilizes these two ROS packages, and tuning some parameters, the robot is able to navigate the environment safely and accurately.

2 Background

The three localization challenges local, global and kidnapped problem can be solved with many approaches. Kalman filters and particle filters are the two most common methods used to solve some of these challenges. In this section, the details are discussed along with the advantages and disadvantages of each approach and when to use them.

2.1 Kalman Filters

Kalman filter is basically used to filter the noise and uncertainty from noisy measurements. This approach assumes a metrical, feature-based environmental representation, in which objects can be effectively represented as points in an appropriate parameter space. The position of the robot and the locations of features form a network of uncertain spatial relationships [4]. Kalman filter runs into two iterative steps. The first step is a measurement update in which the sensors' readings are recorded and used to update the robot's state. The second step is state prediction in which the future state is predicted given the information of the current state. At the start, an initial guess is assumed. Inaccurate initial estimates have no great effect, but it is important to estimate noise parameters accurately, as it is used to determine which of the two steps should we believe more. Noise or uncertainty are represented by gaussian distributions. Computing the gaussians and their intersections requires some basic linear algebra operations. Thus the motion and measurement input to Kalman filter is assumed to be linear, however practically robot motion are usully non linear. A linearization process should be taken into account when dealing with real data. This is done using a linear approximation from Taylor series [5], and this is how the extended Kalman filter works. Despite of its relative complexity, EKF is a very powerful and fast tool that solves the problem of uncertain and noisy data very efficiently.

2.2 Particle Filters

Monte Carlo localization or MCL is the most popular method. It uses a particle filter algorithm to localize the mobile robot. A particle is a representation of the robot. Each particle has a position and an orientation. MCL can be used in both local and global localization problems. The algorithm represents the belief by a set of M particles. The accuracy of the approximation is easily determined by the size of the particle set M . Increasing the total number of particles increases the accuracy of the approximation. The number of particles M is a parameter that enables the user to trade off the accuracy of the computation and the computational resources necessary to run MCL. A measurement is recorded and an importance weight is given to each particle. Then a re-sampling process is done which leads to a new particle set with uniform importance weights, but with an increased number of particles near the actual location of the robot. The initial guess is just a set of pose particles drawn at random and uniformly over the entire pose space.

MCL can approximate almost any distribution of practical importance [1]. Adaptive Monte Carlo localization or AMCL is another variation of MCL. AMCL adjusts the number of particles dynamically as the robot navigates which leads to a significant computational advantage over MCL.

2.3 Comparison / Contrast

The next table is a comparison between EKF vs. MCL

TABLE 1
Comparison between EKF and MCL

	EKF	MCL
Measurements	landmarks	raw data
Noise distribution	Gaussian	any
Memory and time efficiency	more efficient	less efficient
Memory and resolution control	not controlled	controlled
Global localization	can't be used	can be used

In this project, Monte Carlo localization with the adaptive particle filter algorithm is used to locate the robot given a mapped environment.

3 Simulations

In this section, the two robot designs, the packages used and their parameters and the final results are discussed in much greater details.

3.1 Achievements

After tuning some important parameters in ROS packages, the localization output was very good and converges fast, however, the robot speed was kept quiet low for the calculations to be finished.

The figure below is the robot localization result from the beginning till the end of motion, pose particles are represented in green arrows.

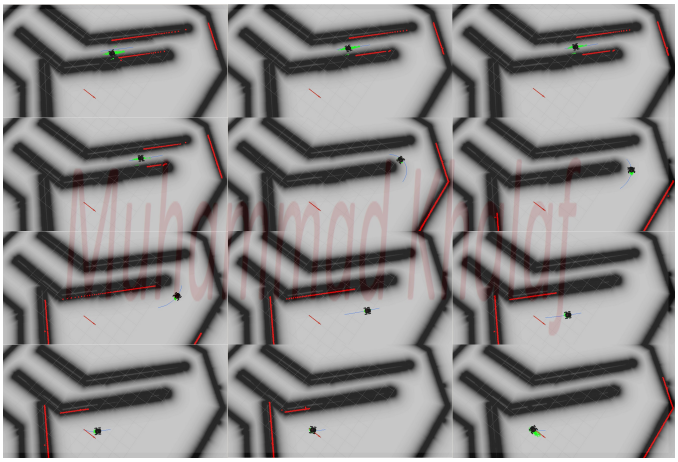

Fig. 1. Robot Localization from start to end points

3.2 Benchmark Model Design

The first robot consists of a box shaped chassis with two back and front caster wheels for the robot to be more stable. The robot moves with two wheels using a gazebo plugin for differential drive controller [6]. The robot is supplied with two sensors: a camera and a Hokuyo laser range finder. The camera is fixed to the front of the robot, and the laser range finder is on the top of the chassis. Both sensors utilize gazebo plugins to control their functionality.

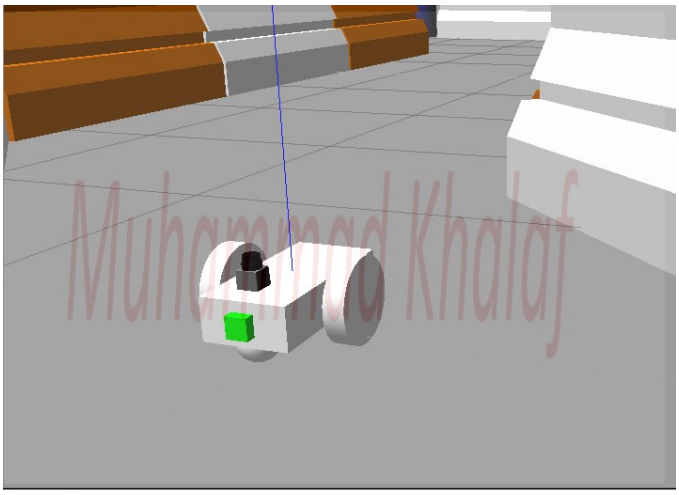

Fig. 2. Benchmark Robot

TABLE 2
Benchmark Model Size of Components

Component	Shape	Size(m)
Chassis	Box	0.4*0.2*0.1(length*width*height)
Caster Wheels	Sphere	0.05(radius)
Wheels	Cylinder	0.05*0.1(length*radius)

3.3 Personal Model Design

The second robot is a 4 wheeled mobile robot with a skid steering drive controller [7]. The size of the robot is different from the first one in order to achieve much smoother movements with these 4 wheels and The caster wheels are removed. The visual design of the chassis and the wheels are imported from mesh files for the pioneer 2 DX robot as illustrated in Gazebo tutorials [8]. Robot Sensors are the same as the benchmark robot, the only change is the minimum and maximum angle of the laser range finder to be sure that the robot wheels are out of the hokoyu field of view.

TABLE 3
Personal Model Size of Components

Component	Shape	Size(m)
Chassis	Box	0.4*0.4*0.1(length*width*height)
Wheels	Cylinder	0.05*0.1(length*radius)

3.4 Packages Used

Both the gazebo differential drive controller and skid steer drive controller uses the cmd_vel as a command topic and

Fig. 3. Personal Robot

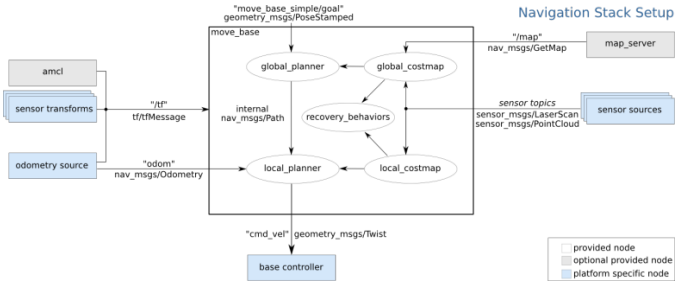

Fig. 4. move base overview

odom as an odometry topic. The camera plugin outputs what the camera sees to image_raw topic. And the hokuyo plugin outputs sensor's readings to /laser/scan topic. The same packages are used for both robot model with the same parameters. This is the list of the most important ROS packages used in the project:-

- amcl
- move_base
- base_local_planner
- global_planner
- costmap_2d

3.4.1 amcl

AMCL is a probabilistic localization system for a robot moving in 2D. It implements the adaptive Monte Carlo localization approach which uses a particle filter to track the pose of a robot against a known map [2]. amcl takes in a laser-based map, laser scans, and transform messages, and outputs pose estimates. Topics that amcl subscribes to: scan(Laser Scans), tf(Transforms), initialpose(Mean and covariance with which to (re-)initialize the particle filter) and map(retrieve the map used for laser-based localization). And it publishes: amcl_pose(Robot's estimated pose in the map, with covariance), particlecloud(The set of pose estimates being maintained by the filter) and tf(transform from odom to map).

3.4.2 move_base

Given a goal in the world, move_base will attempt to reach it with a mobile base. The move_base node links together a global and local planner to accomplish its global navigation task. The move_base node also maintains two costmaps, one for the global planner, and one for a local planner that are used to accomplish navigation tasks [9]. Subscribes to: move_base_simple/goal(Provides a non-action interface to move_base for users that don't care about tracking the execution status of their goals) Publishes: cmd_vel(velocity commands meant for execution by a mobile base).

3.4.3 base_local_planner

This package provides implementations of the Trajectory Rollout and Dynamic Window approaches to local robot navigation on a plane. Given a plan to follow and a costmap, the controller produces velocity commands to send to a mobile base. This package supports both holonomic and non-holonomic robots [10]. Subscribes to: odom(Odometry information that gives the local planner the current speed of the robot) Publishes: name/global_plan(The portion of the global plan that the local planner is currently attempting to follow), name/local_plan(The local plan or trajectory that scored the highest on the last cycle) and name/cost_cloud(The cost grid used for planning).

3.4.4 global_planner

This package provides an implementation of a fast, interpolated global planner for navigation [11]. Publishes: name/plan(The last plan computed, published every time the planner computes a new path).

3.4.5 costmap_2d

This package provides an implementation of a 2D costmap that takes in sensor data from the world, builds a 2D or 3D occupancy grid of the data (depending on whether a voxel based implementation is used), and inflates costs in a 2D costmap based on the occupancy grid and a user specified inflation radius [12]. Subscribes to: name/footprint(Specification for the footprint of the robot). Publishes: name/costmap(The values in the costmap), name/costmap_updates(The value of the updated area of the costmap) and name/voxel_grid(Optionally advertised when the underlying occupancy grid uses voxels and the user requests the voxel grid be published).

3.5 Parameters

The same parameters are applied to both robots and they almost have the same results.

3.5.1 amcl parameters

The number of particles is changed to be from 50 to 200 particles, these values have a great effect in computing time and also have a very good localization accuracy. Other filter parameters are left with their default values.
The laser sigma hit is changed to be 0.01 a greater value makes uncertainty in laser outputs very high, thus the particles would always diverge. On the other side a value of 0.95 is great for the laser z hit, and a small value of 0.05 for laser z rand result in a good estimation.

Small odom alpha values, that represent the noise in odometry, are also preferable.

Fig. 5. a.laser sigma hit=0.01, b.laser sigma hit=0.9

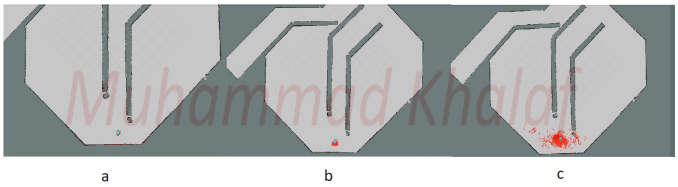

Fig. 6. a.odom alpha=0.01, b.odom alpha=0.1, c.odom alpha=0.8

3.5.2 base local planner parameters

max_vel_x was set to 0.7, with this speed the two robots were able to reach the goal in about 30 seconds, and the calculations were relatively fast enough to handle this speed, however, with higher speeds the location of the robot was not estimated correctly.
xy_goal_tolerance, which is The tolerance in meters when achieving a goal, is set to a value of 0.1, a lower value makes the robot rotate endlessly when reaching the goal. The same is applied to the yaw_goal_tolerance parameter.
sim_time, the amount of time to forward-simulate trajectories in seconds, is 1.5 and simulation is smooth. A higher value makes robot motion very very slow.
occdist_scale, the weighting for how much the controller should attempt to avoid obstacles, is set to 0.01 and a higher value makes the robot stuck in his position preferring not to move any more as it tries very hard not to get near any obstacle.

3.5.3 costmap common parameters

inflation_radius is set to 1.75. With this value both robots are able to turn around corners very well without hitting the walls.
robot_radius is 0.25 for both robots. This value covers the whole robot structures.

3.5.4 global costmap parameters

The global costmap was made static with a resolution of 0.02. The update_frequency is set to 5, and the publish_frequency is set to 2. With such values the robot was navigating very well with no, flickering in the map and no warning messages about delays and lags.

3.5.5 local costmap parameters

This map is non-static. The width and height parameters are very important. If The map size is big then the robot may stops in his position or just turns around itself. A size of 4*4 is very convenient, the robot is able to see nearby obstacles and avoid them.

4 Results

With the same parameters both robots have a good localization results. Both robots reach the goal position in a smooth manner in about 30 seconds, with the particles are packed together in the robot position and directed to the correct robot orientation.
The benchmark model seems to have a better much tighter particles though. But one can't tell the difference between the two robots.
When the robot reaches its goal, it rotates around itself to achieve the desired orientation. While doing so, the estimated particles are rotating too, but the final result is not as good as when the robot is moving in the desired direction before reaching the goal position. So whenever the robot rotates around itself, the results are getting worse.
The next Two figures are the first and second robots, the benchmark model and the personal model, after reaching the goal position.

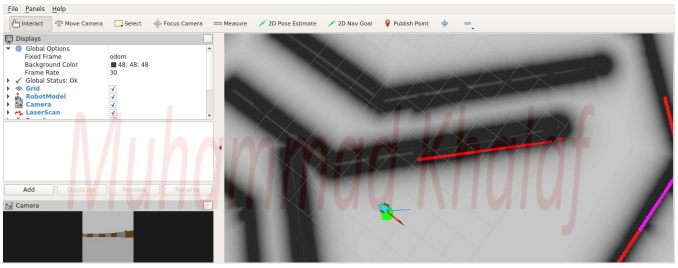

Fig. 7. The first robot at goal

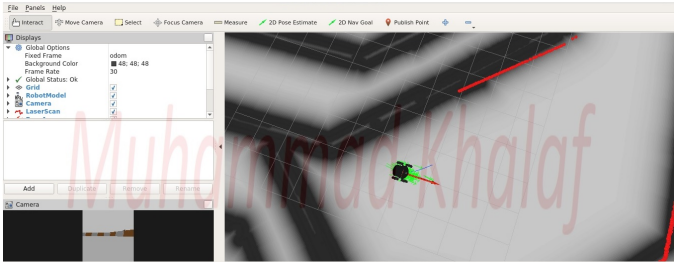

Fig. 8. The second robot at goal

5 Discussion

One thing that can be noticed is that the same parameters works well for both robots. Actually, both robots have almost the same size and same sensors setup, the main difference is in the drive controller. So it is does not require so much modifications in those parameters, except for the ones that deal with robot size, such as robot_radius and occdist_scale.
The benchmark robot performs better due to its smaller size and smoother motion than the 4 wheeled robot. However, the results for both robots are almost the same.
The 'Kidnapped Robot' problem could not be achieved using this approach, as the amcl package assume that the

robot is in a predefined location, usually at (x,y,yaw) = (0,0,0), and this is where it initializes the filter with Gaussian distribution.
Robot localization is a necessity in lots of industry domains. Ceramic factories use robots in oven rooms that have very high temperatures, these robots transports the ceramic plates through different rooms. Amazon warehouses also use robots to carry goods over large areas and they do it with not only one but with hundreds of robots working at the same place.

6 Conclusion / Future work

Both robots successfully reached to their goal position, and the estimated location was very good through the entire path. The speed was a little bit slow, but this could be faster with a better hardware.
Friction between different parts and weight of the robot should be considered when applying these methods in the real world.
Another thing that could be modified is how the trajectory planner works. So instead of reaching the goal position and then rotating the robot, it would be better if the path to the goal was smoother, like in the next figure. It would result in a better results wwhen the robot reaches its destination.

Fig. 9. trajectory without rotating in place at the goal position

References

[1] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. 2000.
[2] ROS.org, "amcl." http://wiki.ros.org/amcl.
[3] ROS.org, "navigation." http://wiki.ros.org/navigation.
[4] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*, pp. 1157–1166. Springer, 2016.
[5] Wikipedia, "Taylor series." https://en.wikipedia.org/wiki/Taylor_series.
[6] Coursera and G. Tech, "Differential drive robots." https://www.coursera.org/lecture/mobile-robot/differential-drive-robots-GnbnD.
[7] C. MIT, "Robo-rats locomotion: Skid-steer drive." https://groups.csail.mit.edu/drl/courses/cs54-2001s/skidsteer.html.
[8] Gazebo, "Attach meshes." http://gazebosim.org/tutorials/?tut=attach_meshes.
[9] ROS.org, "move_base." http://wiki.ros.org/move_base.
[10] ROS.org, "base_local_planner." http://wiki.ros.org/base_local_planner.
[11] ROS.org, "global_planner." http://wiki.ros.org/global_planner.
[12] ROS.org, "costmap_2d." http://wiki.ros.org/costmap_2d.