Where am I

Ahmed Khalid Ali

Abstract—This paper presents an attempt to implement robot localization in a previously mapped environment. The experiments were carried out using 2 different robots in ROS. It uses the power of AMCL package and the navigation stack to achieve good results through parameter tuning of the different used packages.

Index Terms—Robot, IEEEtran, Udacity, LaTeX, Localization, AMCL, Navigation Stack.

1 Introduction

Localization is the problem of defining position in a mapped environment and being able to navigate the environment. It is a key element in making autonomous robots. The robot won't be autonomous if the owner had to move it around manually. A lot of attempts have been done on this problem and actually it is deployed in many areas in the robotics industry. Out of the many solutions that exist for this problem, there are 2 main algorithms that are used. These are Extended Kalman Filter (EKF) and Monte Carlo Localization (MCL) both of them are discussed in the next section. A variation of MCL called Adaptive Monte Carlo Localization (AMCL) was used in this project. Simulation was done in ROS using Rviz [1] and Gazebo [2] and the AMCL package [3] and the navigation stack [4] for localization.

2 Background

This is a quick run through of the most popular algorithms used for robot localization.

2.1 Kalman Filters

Kalman filter is basically used to filter the noise and uncertainty from noisy measurements. It is used to estimate a system's state when it can not be measured directly, but an indirect measurement is available.

There are three versions of the Kalman filter:

1) Linear Kalman filter,
2) Extended Kalman filter (EKF)
3) Unscented Kalman filter (UKF).

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 robots 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 in- put to Kalman filter is assumed to be linear, however practically robot motion are usually non linear. This is where the extended kalman filter (EKF) should be used. It can deal with the non-linearity through a linearization process. This is done using a linear approximation from Taylor series [5].

2.2 Particle Filters

Another method of predicting the system states is to make random discrete guesses of where the robot is. As the robot moves around and takes measurements of the surrounding environment, it uses this reading to update each particles probability of the robots estimated position. Places of higher likelihood will over time collect more particles and therefore be more accurate of the robots posterior belief.

Monte Carlo localization (MCL) is the most popular particle filter method. In particular, adaptive Monte Carlo localization (AMCL) is used in this paper. It changes the number of particles dynamically depending on the uncertainty of the robot's state. In this paper, each particle will contain x, y, and heading states.

2.3 Comparison / Contrast

This table concludes the differences between MCL and Kalman filter (taken from the nanodegree resources):

	MCL	EKF
Measurements	Raw Measurements	Landmarks
Measurement Noise	Any	Gaussian
Posterior	Particles	Gaussian
Efficiency(memory)	✓	✓✓
Efficiency(time)	✓	✓✓
Ease of Implementation	✓✓	✓
Resolution	✓	✓✓
Robustness	✓✓	x
Memory & Resolution Control	Yes	No
Global Localization	Yes	No
State Space	Multimodel Discrete	Unimodal Continuous

Fig. 1. MCL and Kalman Filter Comparison

3 Simulations

This section discusses the details of the robots used in simulation and what packages used along with their parameters.

3.1 Achievements

We achieved reasonable localization performance with both robots. After tuning navigation stack parameters in ROS, the robot can do turns safely and go through narrow passages because of the high local map resolution. Pose predictions are shown in red arrows in the following images.

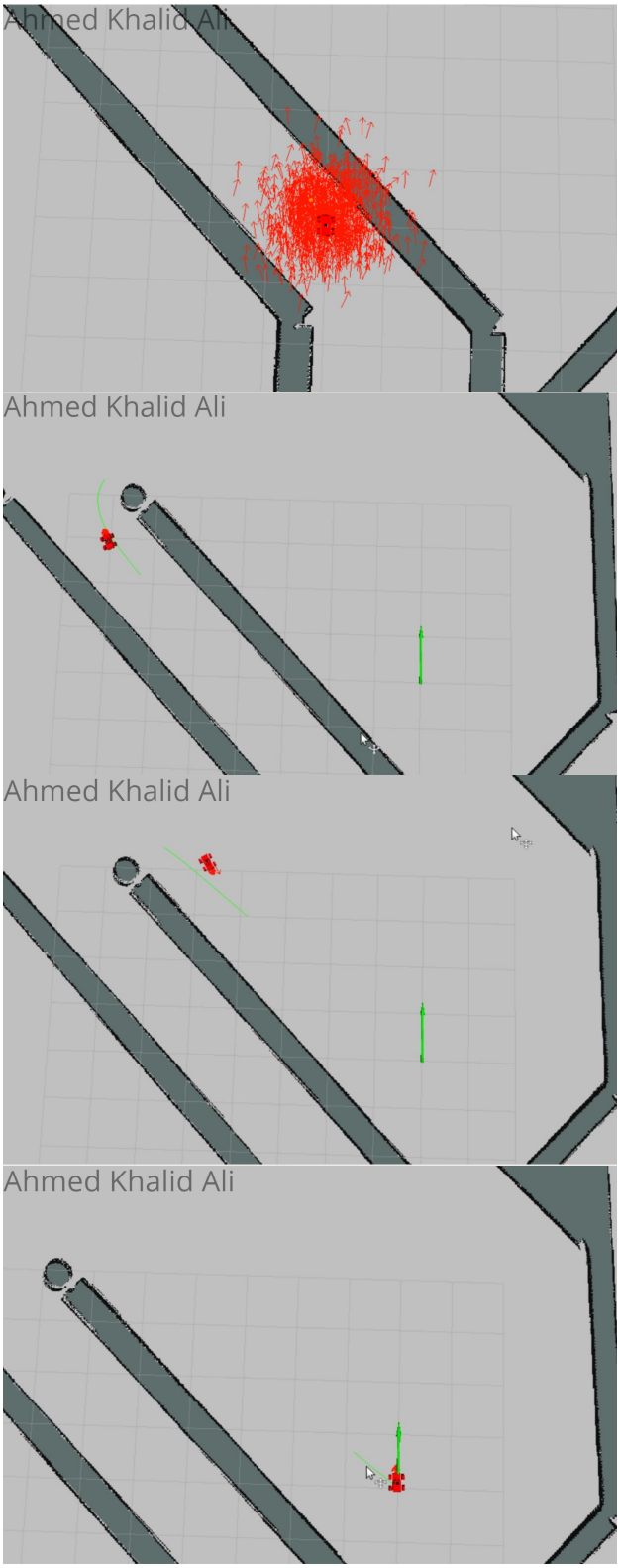

Fig. 2. Robot Localization Performance

3.2 Benchmark Model

The basic benchmark model consisted of a simple box and 2 wheels to use simple differential driving. With a laser rangefinder on top of the robot and a camera on the front of the robot both operating through gazebo plugins.

TABLE 1
Benchmark Model Details

Component	shape	Dimensions(m)
Chassis	Box	0.4 * 0.2 * 0.1
Caster Wheels	Sphere	radius=0.05
Wheels	cylinder	length=0.05, radius=0.1

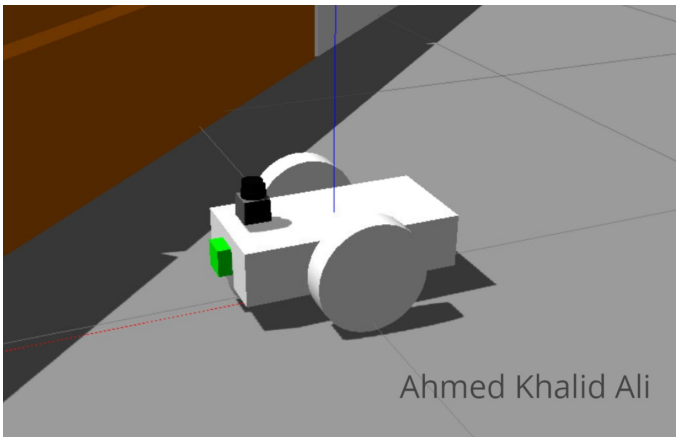

Fig. 3. Benchmark Model

3.3 Personal Model

The personal model consisted of box base with a front and back bumpers. It has a special section on top of the base to hold the laser rangefinder and the camera. It utilized skid-steering drive with 4 wheels.

TABLE 2
Personal Model Details

Component	Shape	Dimensions(m)
Chassis	Box	0.3 * 0.15 * 0.05
Front and Back Bumpers	Cylinder	radius=0.025, length=0.15
Wheels	Cylinder	radius=0.05, length=0.05
Sensor Holder	Box	0.1 * 0.1 * 0.05

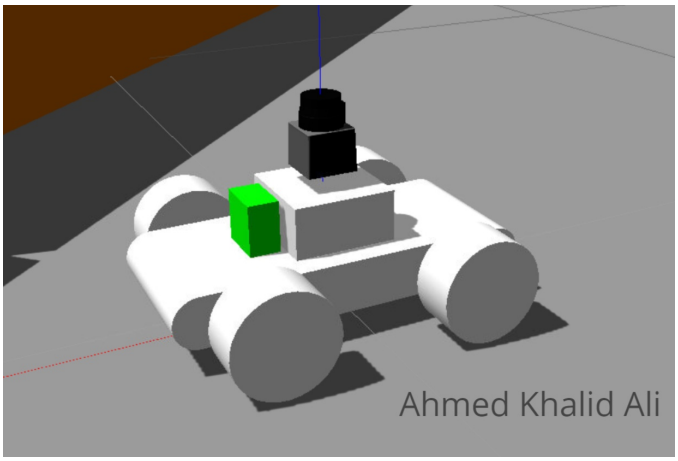

Fig. 4. Personal Model

3.4 Packages Used

The same packages are used for both robots. The main packages are:

3.4.1 amcl

A package that implements AMCL algorithm. It uses the laser scan data and map to produce a particle cloud that is used to estimate the location of the robot and a final estimate of its location.

Subscribed Topics [3]: scan (Laser scans), tf (Transforms), (initialpose) Mean and covariance with which to (re-)initialize the particle filter, map (When the use_map_topic parameter is set, AMCL subscribes to this topic to retrieve the map used for laser-based localization).

Published Topics amcl_pose (Robot's estimated pose in the map, with covariance), particle cloud (The set of pose estimates being maintained by the filter), tf (Publishes the transform from odom to map).

3.4.2 move base

Given a goal in the world, it published velocity commands to the the robot drive controller to reach the desired goal.

Subscribed Topics [6]: 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).

Published Topics: cmd_vel (A stream of 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.

Subscribed Topics [7]: odom (Odometry information that gives the local planner the current speed of the robot).

Published Topics: /global plan(The portion of the global plan that the local planner is currently attempting to follow), /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. Published Topics [8]: 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. Subscribed Topics [9]: /footprint (Specification for the footprint of the robot. This replaces the previous parameter specification of the footprint). Published Topics: /costmap (The values in the costmap), /costmap_updates (The value of the updated area of the costmap)

3.5 Parameters

Localization parameters in the AMCL node should be described, as well as move_base parameters in the configuration file. You should be able to clearly demonstrate your understanding of the impact of these parameters. Parameters from amcl and move_base packages were tuned to achieve optimal performance.
The details of the AMCL parameters:

- update_min_d: the minimum distance (in meters) required before performing an update.
- update_min_a: the minimum rotation (in radians) required before performing an update.
- max_particles: the maximum number of particles to have in any update (the lower, the less computing power needed while preserving the same accuracy).
- laser_z_hit: weight for the actual obstacles in the laser scan data.
- laser_z_rand: weight for the random unexplained laser scan data (Both z_hit and z_rand should add to 1, They define how much do you value or trust the obstacles in the laser data)
- laser_sigma_hit: std deviation to be used when dealing with obstacles in the laser scan data.
- odom_alpha*: specifies the expected noise in different elements of odometry data
- initial_pose_*: initial pose used to start the iterative process of the algorithm

The details of each parameter of costmap_2d package:

- obstacle_range: the maximum distance at which to insert obstacles into the costmap using sensor data (in meters).
- raytrace_range: the maximum distance at which to clear the costmap from obstacles using sensor data (in meters).
- transform_tolerance: Specifies the delay in transform (tf) data that is tolerable (in seconds).
- inflation_radius: The radius in meters to which the map inflates obstacle cost values(like extra padding around the obstacles to make sure that the robot does not hit it).

- footprint: the footprint of the base of robot's base to be used in inflation and obstacle detection calculations (in meters, having the center of the robot as a reference point)
- update_frequency: The frequency (in Hz) for the map to be updated.
- publish_frequency: The frequency (in Hz) for the map to be publish display information.
- width & height: dimensions of the map (in meters).
- resolution: resolution of the map (in meters/cell)

Both robots share most of the parameter values shown in the following tables. The differences are mentioned when they occur.

TABLE 3
AMCL Parameters

Parameter	value
update_min_d	0.03
update_min_a	0.03
max_particles	1000
laser_z_hit	.98
laser_z_rand	.02
laser_sigma_hit	0.05
odom_alpha*	0.01
initial_pose_x	0.0
initial_pose_y	0.0

TABLE 4
costmap_2d Parameters

Common Parameters	
Parameter	Value
obstacle_range	3.0
raytrace_range	3.0
transform_tolerance	0.2
inflation_radius	Benchmark 1.5 / Personal 1.4
footprint (Benchmark)	[[0.2, 0.175], [-0.2, 0.175], [-0.2, -0.175], [0.2, -0.175]]
footprint (Personal)	[[0.17, 0.15], [-0.17, 0.15], [-0.17, -0.15], [0.17, -0.15]]
Global Costmap Parameters	
update_frequency	Benchmark 10.0 / Personal 5.0
publish_frequency	3.0
width	10.0
height	10.0
resolution	Benchmark 0.09 / Personal 0.06
Local Costmap Parameters	
update_frequency	10.0
publish_frequency	3.0
width	3.0
height	3.0
resolution	Benchmark 0.2 / Personal 0.3

4 Results

Present an unbiased view of your robot's performance and justify your stance with facts. Do the localization results look reasonable? What is the duration for the particle filters to converge? How long does it take for the robot to reach the goal? Does it follow a smooth path to the goal? Does it have unexpected behavior in the process?
For demonstrating your results, it is incredibly useful to have some watermarked charts, tables, and/or graphs for the reader to review. This makes ingesting the information quicker and easier.

The personal robot still requires some modifications to the motors to be able to move faster and take tighter turns. It also cannot turn in-place consistently due to slipping caused by the skid steering system. It needs to move back and forth to exactly align itself exactly when it tries to rotate in place. The benchmark robot has problem moving around when obstacles are nearby. It makes a lot of small moves until eventually it can move away from the obstacle. When it is away from the obstacles though it is pretty consistent.

4.1 Benchmark

It took 35 seconds for the particles to fully converge and a total of 95 seconds to reach the goal.

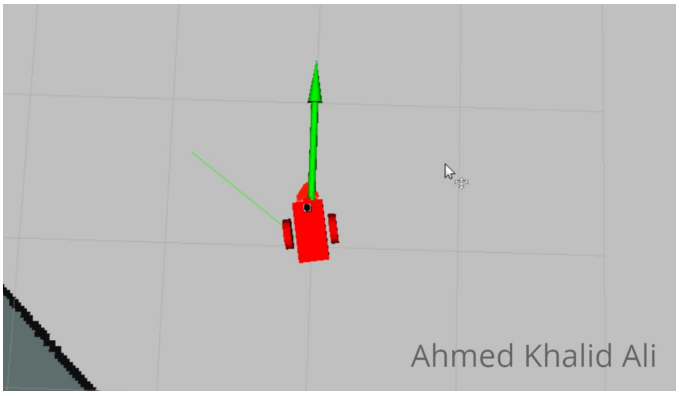

Fig. 5. Benchmark Model at Target

4.2 Student

It took around 8 seconds only for the particles to converge because it move away quickly from the wall. It took a total of 80 seconds to reach the goal.

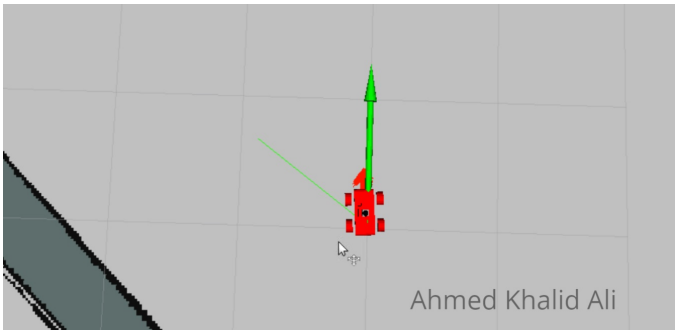

Fig. 6. Personal Model at Target

5 Discussion

Both of the robots have acceptable performance. The benchmark robot is very hesitant near walls and it is pretty slow but it can rotate in place really well and it is easier to control because of the differential driving system. The personal robot is much faster but it struggled with taking tight turns if the torque of the motors was low. it can deal with walls and obstacles much better than the benchmark robot, But the slipping caused by the skid steering driving system might result in somewhat noisier odometry data when deploying in the real life.

Overall the personal robot is more reliable as it is faster and can deal with walls and obstacles better but it needs some improvement on its driving system to reach optimal performance.

The kidnapped robot problem cannot be solved with the current implementation. It may be solved by putting some kind of ″landmarks″ in the mapped environment that the robot can communicate with to know approxmiately where it is and start working from there.

Localization has many applications in industry. For example an autonomous robot that carries stuff around can use it to determine where it is and where it should head. Autonomous robots in homes for example can carry out a lot of chores like vacuuming the floor.

6 Conclusion / Future work

The project achieved what was originally intended. Both robots could navigate the mapped environment and reach the destination successfully. For deployment in the real-world it needs a decent controller like the Jetson TX2 because the localization needs a considerable amount of computing power. This project can be fairly easily deployed in commercial products. It would need parameter tuning for the new robots and develop a way to move the robot manually first to make a map of the environment and be able to operate on it.

Future improvements include:

- Better material selection for the robot to minimize slipping.
- Better trajectory planner to minimize the need to rotate in place
- Making it faster if needed

References

[1] “rviz - ROS Wiki.”
[2] “Gazebo.”
[3] “amcl - ROS Wiki.”
[4] “navigation - ROS Wiki.”
[5] “Taylor series,” Apr. 2019. Page Version ID: 894410090.
[6] “move_base - ROS Wiki.”
[7] “base_local_planner - ROS Wiki.”
[8] “global_planner - ROS Wiki.”
[9] “costmap_2d - ROS Wiki.”