Where am I?

Thomas Legleu

Abstract—Robot navigation is a robot's ability to determine its own position within a frame of reference and generate a plan to attain a specific goal. Robot localization is the process of determining where a mobile robot is located according to a known environment. Localization is one of the most fundamental competencies required by a mobile robot and is essential to navigation. Here we will look at localization and implement the monte carlo localization algorithm through various ROS packages and nodes.

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

1 Introduction

NAVIGATION is the key competency required of an autonomous mobile robot. It is based on four key notions that are integral to any robot that is considered mobile:

1) perception: interpret with sensors and extract data.
2) localization: determine its pose in the environment.
3) cognition: a decision on how to act to achieve its goals.
4) motion control: actuation of motors in a manner to achieve a goal [1]

1.1 Localization where am I

Localization answers the question: "Where am I?" The knowledge of a current location and orientation is essential for deciding what to do as a mobile robot works through a series of task. Without precise localization the robot would not be able to carry out applications or task in a meaningful way. The problem revolves around estimating the robot pose (position and orientation) relative to the coordinate frame in which the map is defined. The information available for computing the robot location is gathered using onboard sensors, while the robot uses these sensors to observe its environment and its own motion.

2 Background

Four various algorithms have been developed as a solution to the localization problem. Extended Kalman Filter, Markov localization, grid localization, and monte carlo localization . In this paper we will look at the implementation of the monte carlo localization algorithm.

2.1 Kalman Filters

The Kalman Filter Algorithm uses a system's dynamic model, known control inputs, and sequential measurements to form an estimate of the system's state. This will be a better estimate than only using one measurement, it is recursive, and can run in real time. The KF uses only the present input measurements, the previously calculated state, and its uncertainty matrix. It requires no additional information. They work on a two-step process:

1) prediction step: The Kalman filter produces estimates of the current state variables, along with their uncertainties.

2) measurement step: Estimates are updated using a weighted average with more weight to higher certainty.

They are used for localization because their ability to precisley locate a robot model in a unknown environment. Kalman Filters do have a draw back because they can only compute for linear conditions. Extended Kalman Filters (EKFs) linearizes about an estimate of the current mean and covariance and is this used as the nonlinear version of the Kalman filter. The EKF is used as industry standard for most robotic applications due to the non-linear motion of most robots. [2]

2.2 Particle Filters

Particle filters methods are a set of Monte Carlo algorithms used to solve filtering problems. Most often found in Bayesian statistical inference and signal processing. The monte carlo localization algorithm uses a particle filter to keep track of the robots pose (position and orientation).They are used for localization because their accuracy and computation advantages when locating a robot model in an unknown environment.

To achieve a robot's position and orientation based on sensory information MCL estimates a posterior distribution of these attributes. Whenever the robot senses something, recursive Bayesian estimation is used as a filter to resample particles. The idea is the particles will converge towards the position of the robot. From sensor measurements, this filter estimates the state of a dynamical system . To gather accurate estimate of the probability density over the state space compiled from the measurements is the goal of Bayes filtering. [3]

Pseudocode of the Monte Carlo Localization is seen below in Alogrithm 1 Table

Algorithm 1 Algoritm Monte Carlo Localization

```
1:  procedure ALGORITHM MCL($X_{t-1}, u_t, z_t$)
2:      $\bar{X}_t = X_t = \phi$
3:      for $m$ = 1 to $M$ do
4:          $x_t^{[m]}$ = motion-update($u_t, x_{t-1}^{[m]}$)
5:          $w_t^{[m]}$ = sensor-update($z_t, x_t^{[m]}, m_{t-1}^{[m]}$)
6:          $\bar{X}_t = \bar{X}_t + < x_t^{[m]}, m_t^{[m]}, w_t^{[m]} >$
7:      for $m$ = 1 to $M$ do
8:          draw $x_t^{[m]}$ with probability $\propto w_t^{[m]}$
9:          $\bar{X}_t = X_t = \phi$
10:     Return $X_t$
```

2.3 Comparison / Contrast

Table 1 shows some a comparison of the Monte Carlo Localization and Extended Kalman Filter. The main things to note from this table is in contrast to Kalman filtering based techniques a Particle Filter is able to represent multi-modal distributions which allows it to globally localize a robot, it uses reduced memory, integrate measurements at higher frequency, it is very accurate, and easy to implement.

TABLE 1
Monte Carlo Localization vs Extended Kalman Filter Localization

variables	MCL	EKF
Measeurements	Raw Measurements	Landmarks
Measurement Noise	Any	Gaussian
Posterior	Particles	Gaussian
Efficiency (memeory)	good	very good
Efficiency(time)	good	very good
Ease of Implementation	very good	good
Resolution	good	very good
Robustness	very good	not good
Memory Resolution Control	yes	no
Global Localization	yes	no
State Space	Multimodel Discrete	Unimodal Continuous

After comparing the table and looking at the higher level results of each based on comparision for the reasons mentioned above a Particle Filter is chosen to localize a robot in this project. [4]

3 Simulations

The main goal of this project is to create a ROS workspace in which we localize a robot as it approaches a goal. To do this a robot, custom map, environment, two launch files, YAML files, and a c++ node are generated for simulation. Let's take a closer look at the elements that make up the environment to better understand the context for implementing the monte carlo localization algorithm.

3.1 Gazebo World

Our world is a collection of models such as your robot and the environment. This is a file that describes our robot environment. Custom worlds can be modeled and deployed in the world to generate any environment. They are typically a separate world file accessed by the main launch file. Gazebo plugins help utilize all available gazebo functionality are allowed to be used in order to implement specific use-cases for specific models. An example of this would be particular characteristics to the camera like its height, width, fov, and other attributes important to simulation.

3.2 Robot Model Design (unified robot description format)

The Robot is built from a unified robot description format. A Robot model explains the various aspects of the robot for 3d representation and physics attributes. For this study the robot is made up of wheels, caster, chassis, camera, and a rangefinder sensor. Key elements such as links and joints are also set in our urdf file. Other entities like color, shape, mass, orientation, and origins are also set in this file. In this study two robots are designed and tested for localization.

3.3 Imported Map

In order to create more dynamic world we can import predefined or generate a new map. The robot in this study is placed in an environment using a map created by Clearpath Robotics. These maps are crucial to collision avoidance and path planning of the robot as they serve as the basis of the static elements of the robot environment.

3.4 Rviz Integration

Rviz is used to visualize the information of the parameters we have generated for the robot. Without Rviz robot state, camera information, and sensor data would not be visualized making it difficult to study the behavior of the robot. In order to integrate rviz a custom rviz environment based on our world, robot, and maps is generated.The robot model is used as our reference frame, optometry for fixed frame, add a RobotModel, camera, LaserScanner, map, and particles are added to the environment. [5]

3.5 Main.Launch

The main launch file spawns a gazebo world and our robot model. It also imports the map.

3.6 AMCL.Launch

Adaptive Monte Carlo Localization (AMCL) is used in the implementation of our localization instead of a typical monte carlo localization algorithm. AMCL sets the number of particles over a period of time by dynamically updating them . It is used because of the computational advantages over the mcl algorithm. ROS provides an AMCL package that will be used to perform localization.

The amcl launch file loads the provided map using a new node for the map server package, add a node that will launch the amcl package for localization, and a move base node to move the robot towards a specific goal. The amcl launch files have

3.7 Navigation Goal.cpp

Our navigation goal.cpp is a c++ node that allows out mobile bot to move towards a specific goal with all the parameters set in place. It will attempt to move the robot to the specific target. In order to run this node you must launch the udacity world and amcl launch files.

4 ACHIEVEMENTS

The goal target was achieved in this project and using adaptive monte carlo localization was implemented as the technique as the robot achieved its goal. The Udacity Bot Model has model. Udacity Bot Legleu has

4.1 Packages: Subscribers/Topics

The packages greatly effect what topics are received and published by the robot. Also the services it used and provided are also . Looking back at our launch, node, and, yaml files we can adreess. Please see the Table 2 for detailed parameter's of the udacity bot.

4.1.1 Map Server Package

The map server provides a new node. It displays a map generated as a ROS service. A key element of this package is command-line utility. This allows dynamically generated maps to be saved. With out this service we would not be able to update information of our map. [6]

4.1.2 AMCL Package

For a robot moving in 2D the amcl package provides a probabilistic localization using a particle filter to track the pose of a robot against a known map. In order to implement the ROS package we generate a ROS node in our launch file and set specific parameters to perform our localization. [7]

4.1.3 Move Base Package

The move base package will attempt to reach a goal with a mobile base given that particular goal. It links together a global and local planner. The global and local planner are up of a costmap. A costmap builds a 2d or 3d map of the environment based sensor information. The local costmap, in relation to the global costmap, keeps getting updated allowing the package to define the continuous path for the robot to move. The package has built in corrective behaviors or maneuvers that attempt to move around obstacles or find a clear path. [8]

4.2 YAML Files (Ain't Markup Language)

In order to access the packages a custom launch file that uses various ROS nodes and packages accompanied by yaml files is used to set and tune specific parameters.

4.2.1 Costmap common parameters:

This yaml file use 2D navigation stack from ROS that takes in information from odometry, sensor updates, and a goal pose and outputs the parameters for the local costmap. The obstacle range parameter is the maximum range sensor reading, raytrace range parameter sets the range to raytrace freespace, transform tolerance sets the maximum amount of latency between transforms, inflation radius sets the maximum distance from obstacles for the costmap, and observation sources parameter defines a list of sensors that are going to be passing information to the costmap. Finally, the frame name, data type, topic name, marking and clearing are set for the particular laser scanner. [9]

4.2.2 Local and Global Costmap Parameters:

The local and global costmap parameters use 2D navigation stack from ROS that takes in information from odometry, sensor streams, and a goal pose and outputs the parameters for the costmap. The two specific packages used in this are the costmap2d and map server package. The local costmap parameter used for tuning the amcl are the global frame for the costmap, the frame for the base link of the robot, the frequency in Hz for the map to be updated ,the frequency in Hz for the map to be publish display information, the width and height of the map in meters, and the resolution of the map [10]

4.2.3 Base Local Planner Parameters:

The base local planner provides the parameters to move the base with a certain velocity, acceleration, and rotation for the x,y and theta of our robot.The weighting for how much the controller should attempt to avoid obstacles. The weighting for how much the controller should stay close to the path it was given. The frequency of the controller to be updated

4.3 Benchmark Model

The Udacity Robot was built from ground up with simple primitive shapes which include the robot's chassis, wheels, caster, camera, and one imported rangefinder sensor.

4.3.1 Model design

The Udacity Bot is rectangular in shape. The robot's twoo wheels are set on the middle of the long sides middle while the two caster's are under the bot on the long sides of the rectangle. This design s helps the robots stability. The robot is equipped with a rgb camera and a rangefinder sensor (Hukuyo). They were both placed on the very front of the bot in order for unobstructed vision and range data. Please see image Fig.1 For a few views of the robot.

4.3.2 Parameters

See Table 1 fro the parameters set for the Udacity Bot on an home laptop. The table breaks down all the parameters set from the amcl launch file which is accompanied by several yaml files that set specific parameters in order for the launch file to work.

4.4 Personal Model

4.4.1 Model design

The Udacity Robot was built from ground up with simple primitive shapes which include the robot's chassis, wheels, caster, camera, and one imported rangefinder sensor. The Udacity Bot is rectangular in shape. The robot's 2 wheels set in the sides middle while the two caster's are under the bot on the long sides of the rectangle. This design s helps the robots stability. The robot is equipped with a rgb camera

TABLE 2
Udacity Bot Parameters

Costmap Common Parameters	Inputs
map type	costmap
obstacle range:	2.5
raytrace range:	3.0
transform tolerance	0.7
inflation radius	0.6
observation sources	laser scan sensor
sensor frame	hokuyo
data type	LaserScan
topic	/udacity bot/laser/scan
marking	true
clearing	true
Global Costmap Parameters	Inputs
global frame	map
robot base frame	robot footprint
update frequency	7.0
publish frequency	7.0
width + height	40.0
resolution	0.02
static map	true
rolling window	false
Local Costmap Parameters	Inputs
global frame	odom
robot base frame	robot footprint
update frequency	7.0
publish frequency	7.0
width + height	4.0
resolution	0.02
static map	false
rolling window	true
Base Local Parameters	Inputs
holonomic robot	false
robot base frame	robot footprint
max vel x	0.1
min vel x	0.08
max vel theta	2.0
acc lim theta	5.0
acc lim x	5.0
acc lim y	5.0

and a rangefinder sensor (Hyuoko). They were both placed on the very front of the bot in order for unobstructed vision and range data. Please see image Fig.1 For a few views of the robot.

4.4.2 Parameters

See Table 1 for the parameters set for the Udacity Bot-tl on an home laptop. The table breaks down all the parmaeters set from the amcl launch file which is accompanied by seceral yaml files that set specific paramters in order for the launch file to work.

5 Results

The main.launch and amcl.launch file worked properly and the robot was able to localize it self through the particle filter and move itself to the goal. The built in navigation tools from the move-base package and avoidance of the map generated by costmap 2d package were causing the most computational lag. The particle filter caused little or no lag in the process. Thus, the parameter settings for the costamp lag and moving the bot were of most important to tweak in order to achieve the goal on a home laptop.

TABLE 3
Udacity Bot Parameters TL

Costmap Common Parameters	Inputs
map type	costmap
obstacle range:	2.5
raytrace range:	3.0
transform tolerance	0.7
inflation radius	0.6
observation sources	laser scan sensor
sensor frame	hokuyo
data type	LaserScan
topic	/udacity bot/laser/scan
marking	true
clearing	true
Global Costmap Parameters	Inputs
global frame	map
robot base frame	robot footprint
update frequency	7.0
publish frequency	7.0
width + height	40.0
resolution	0.02
static map	true
rolling window	false
Local Costmap Parameters	Inputs
global frame	odom
robot base frame	robot footprint
update frequency	7.0
publish frequency	7.0
width + height	4.0
resolution	0.02
static map	false
rolling window	true
Base Local Parameters	Inputs
holonomic robot	false
robot base frame	robot footprint
yaw goal tolerance	0.10
xy goal tolerance	0.10
occdist scale	0.2
pdist scale	0.2
gdist scale	1.0
max vel x	0.1
min vel x	0.08
max vel theta	2.0
acc lim theta	5.0
acc lim x	5.0
acc lim y	5.0

5.1 Localization Results

The Localization results from the adaptive monte carlo localization package looked reasonable. The algorithm worked nicely and latency was not an issue when it came to the computational power available. The Particle Filters diverged very quickly. Within amount of frames

The robot did however take a long time to reach the goal. The tranform tolerance, speed of the robot, and inflation radius had to be adjusted in order for the transform tolerance to work well with the costmap. The bot's steering information was primitive as well. It was overv scanning causing it to not move very fluidly and constantyl jogging itself around.

The biggest issue came as the Udacity bot which is rectangular in shape got stuck getting around the corner quite a few times. This made for odd movements and before not finding the proper parameters the robot would get stuck and not be ale to reach the goal.

6 Discussion/Conclusion

One may believe the biggest hindrance on this submission is the computational power. The AMCL package did not cause any or really no lag at all. Performance issues were to be blamed on the update and publish frequency of the costmaps. The transform tolerance was another parameter that really needed to be tuned down as it causing a warning error. Problems were resolved by slowing the mobile robot significantly and really cutting back on the transform tolerance rate as well as the update/publish frequency.

The personal bot edited performed significantly better than the Udacity-Bot. It performed better because the shape allowed it to deal with getting around the corner a bit better than the the rectangular shape.

6.1 Topics

Localization is crucial to any robotic problem where one is interested in knowing the orientation and location of the the robot. It is also crucial in order to use certain localization strategies one must be aware if a map is available as certain algorithms will not work with out a known map like MCL algorithm. It is good to note that without Localization we would knot know where the robot is located. This leads me to believe that anytime one needs a to know where a robot is in space is a good reason to go through the effort to localize your robot.

Any autonomous situation is where one may use localization in the autonomous industry. Search and Rescue, Construction, autonomous cars, uavs. If the robot is running autonomously better to have a localization algorithm helping keeping track of it's position and orientation.

References

[1] R. Siegwart, *Mobile Robot Localization*. http://www.cs.cmu.edu/ rasc/Download/AMRobots5.pdf, 2002.

[2] M. S. Grewal, *Kalman Filtering Theory and Practice*. Prentice Hall, 1993.

[3] D. Fox, *Monte Carlo Localization: Efficient Position Estimation for Mobile Robots*. John Wiley Sons Ltd, 1999.

[4] "Udacity robotics software engineer mcl vs ekf." https://classroom.udacity.com/nanodegrees/nd209/parts. Accessed: 2019-08-29. Udacity.

[5] "Udacity robotics software engineer: Where am i module." https://classroom.udacity.com/nanodegrees/nd209/parts. Accessed: 2019-08-29. Udacity.

[6] "Map server package." http://wiki.ros.org/map-server. Accessed: 2019-08-29. Open Source Robotics Foundation.

[7] "Adaptive monte carlo lozalization package." http://http://wiki.ros.org/amcl. Accessed: 2019-08-29. Open Source Robotics Foundation.

[8] "Move base package." http://wiki.ros.org/move-base. Accessed: 2019-08-29. Open Source Robotics Foundation.

[9] "Navigation stack package." http://wiki.ros.org/navigation. Accessed: 2019-08-29. Open Source Robotics Foundation.

[10] "Costmap 2d package." http://wiki.ros.org/costmap-2d. Accessed: 2019-08-29. Open Source Robotics Foundation.