Project: Where Am I?

Vladislav Artamonov

Abstract—A mobile robot model for Gazebo is developed to create a dedicated ROS package, and the AMCL and Navigation ROS packages are integrated for localizing the robot in the provided map. The parameters are tuned for the robots in the package to improve the localization results in the map. Two different robot models, UdacityBot and VladBot, are considered for performance evaluation. After achieving the desired results for UdacityBot, VladBot is implemented with significant changes to the robot's base and the sensor locations. The parameter settings with VladBot are checked, and the localization results are improved upon as required for VladBot.

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

1 Introduction

In Robotics, Localization is the challenge of determining the robot's pose in a mapped environment. This is done by implementing a probabilistic algorithm to filter noisy sensor measurements and track the robot's position and orientation. The robot is moving around, taking measurements, trying to figure out where it can be positioned. Since this is a probabilistic model, the robot might have a few guesses as to where it is located. [1]

There are three different types of localization problems. Firstly, the easiest localization problem is called Position Tracking. It is also known as Local Localization. In this problem, the robot knows its initial pose and the localization challenge entails estimating the robot's pose as it moves out on the environment. This problem is not as trivial since there is always some uncertainty in robot motion. However, the uncertainty is limited to regions surrounding the robot. Secondarily, a more complicated localization challenge is Global Localization. In this case, the robot's initial pose is unknown and the robot must determine its pose relative to the ground truth map. The amount of uncertainty in Global Localization is much greater than in Position Tracking, making it a much more difficult problem. Finally, the most challenging localization problem is the Kidnapped Robot problem. This problem is just like Global Localization except that the robot may be kidnapped at any time and moved to a new location on the map.

In the real world, it has to be considered whether environment is static meaning unchanging, or dynamic where objects may shift over time. Dynamic environments are more challenging to localize in.

In this project, ROS packages is utilized to accurately localize a mobile robot inside a provided map (Fig. 1) in the Gazebo and RViz simulation environments. There are several aspects of robotics to learn about with a focus on ROS. Those includes building a mobile robot for simulated tasks, creating a ROS package that launches a custom robot model in a Gazebo world and utilizes packages like AMCL and the Navigation Stack, and exploring, adding, and tuning specific parameters corresponding to each package to achieve the best possible localization results.

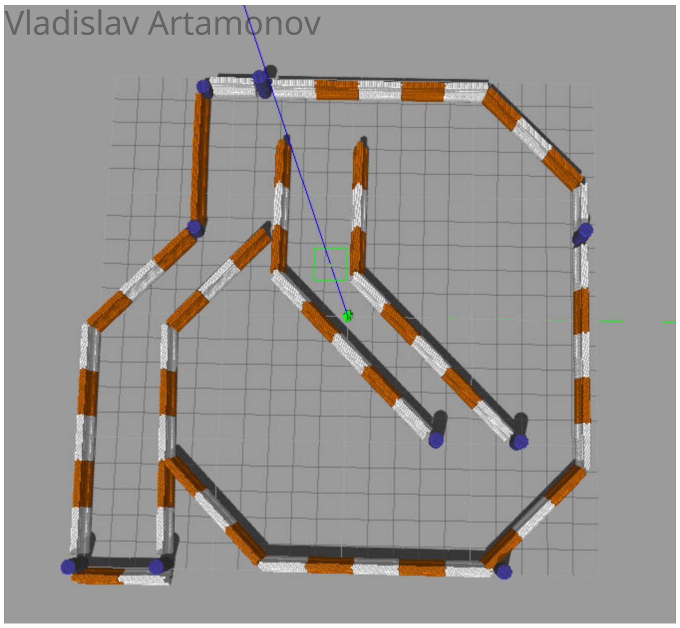


Fig. 1. Localization map. Robot at the starting position

2 Background

Localization is a base of developing autonomous mobile robots. In order to decide where to move, the robot first needs to estimate its position. In the real world, sensors are noisy and actuators might not be very precise. Also, there are external influences such as wind, slippage and many others. This combination increases the challenges of implementing a precise localization.

The localization problem can be separated into three different types: local localization, global localization, and kidnapped robot. The local localization is the simplest, where the robot's pose is known and it is required to update its position as it moves despite the sensors noise. The global localization is when the initial pose of the robot is unknown. The last problem is known as the kidnapped robot, where at any given time the robot can be moved to another location in the map.

In this paper, Kalman Filters (KF) and the Monte Carlo Localization (MCL) algorithms are compared, and the

Adaptative Monte Carlo Localization algorithm is evaluated in a simulated environment.

2.1 Kalman Filters

The Kalman Filter (KF) is one of the most practical algorithms to estimate the state of a system based on uncertain information such as noisy sensors. It was first introduced by Rudolf E. Kálmán and used in trajectory estimation for the Apollo program. It can achieve accurate results without a lot of data and does not require many computational resources.

Kalman filters have a probabilistic approach to improve the accuracy of the estimations and can be used to solve local localization problems. First, an initial prediction of the state is made. Next, starts a cycle of state prediction and measurement update. This cycle is continuously repeated using sensor data provided in real time.

Traditional Kalman Filters assume that the system is linear. However, it is not the case in many applications. One solution is to approximate a non-linear equation using some terms from the Taylor Series. This is known as Extend Kalman Filters (EKF) and they are widely used in the field of Robotics.

2.2 Particle Filters

Particle filters are another common approach to solve to Localization problem in Robotics. The particles are virtual elements that represent possible positions and orientations of the robot. The algorithms can estimate the actual pose of the robot by randomly distributing the particles across the map and filtering the ones that are more likely to be correct by comparing with sensor information.

Opposed to the EKF, the MCL algorithm can be successfully used in Global Localization problems. Moreover, the measurement noise is not restricted to Gaussian distribution which makes it a good option in several applications.

The computational power required proportional to the number of particles used. The Adaptative Monte Carlo Localization algorithm adjusts the number of particles according to the overall accuracy of the estimation, making it possible to lower the computational resources required and the particle distribution converges.

2.3 Comparison / Contrast

MCL has many advantages over EKF. First, MCL is easy to program compared to EKF. Second, MCL represents non-Gaussian distributions and can approximate any other practical important distribution. This means MCL is unrestricted by a linear Gaussian states based assumption as is the case for EKF. This allows MCL to model a much greater variety of environments especially since the real world can't always be modeled with Gaussian distributions. Third, in MCL, you can control the computational memory and resolution of your solution by changing the number of particles distributed uniformly and randomly throughout the map. [2]

In this project, Adaptive Monte Carlo Localization (AMCL) is used. AMCL dynamically adjusts the number of particles over a period of time, as the robot navigates around in a map. This adaptive process offers a significant computational advantage over MCL.

3 Simulations

ROS packages are utilized to accurately localize a mobile robot inside a provided map in the Gazebo and RViz simulation environments. There are several aspects of robotics with a focus on ROS. They include building a mobile robot for simulated tasks, creating a ROS package that launches a custom robot model in a Gazebo world and utilizes packages like AMCL and the Navigation Stack, and tuning specific parameters corresponding to each package to achieve the best possible localization results. The navigation stack assumes that the robot is config- ured in a particular manner in order to run.

3.1 Achievements

Using a provided C++ node, the robot navigates to a specific goal position while localizing itself along the way. In this project, the navigation is achieved for localization with the benchmark model (UdacityBot) and the own robot(VladBot).

3.2 Benchmark Model

3.2.1 Model design

The mobile robot model is built by creating its own Unified Robot Description Format (URDF) file.

For this model, a cuboidal base (box size=".4 .2 .1") with two caster wheels was created. The caster wheels will help stabilize this model and it can help with weight distribution, preventing the robot from tilting along the z-axis at times. There is a single link, with the name defined as "chassis", encompassing the base as well as the caster wheels. For this robot, two wheels are added to the robot. Each wheel is represented as a link and is connected to the base link (the chassis) with a joint.

For this robot, two sensors, a camera and a laser rangefinder, were added. TABLE.1 shows some of the specifications for the sensors.

TABLE 1
Camera Sensor and Laser Sensor: UdacityBot

Camera Sensor	
link name	Camera
link origin	[0, 0, 0, 0, 0, 0]
joint name	camera joint
joint origin	[0.2, 0, 0, 0, 0, 0]
geometry	box with size "0.05"
joint parent link	chassis
joint child link	camera
Laser Sensor	
link name	hokuyo
link origin	[0, 0, 0, 0, 0, 0]
joint name	hokuyo joint
joint origin	[.15, 0, .1, 0, 0, 0]
geometry	box with size "0.1" for collision
geometry	a mesh file for visual
joint parent link	chasis
joint child link	hokuyo

3.2.2 Packages Used

A ROS package that launches a custom robot model in a Gazebo world is created and the packages like AMCL and the Navigation Stack are utilized.

3.2.3 Parameters

To achieve the best possible localization results, specific parameters corresponding to each package is explored, added, and tuned.

TABLE.2 shows Localization parameters in the AMCL node. The amcl package will localize the robot. For the project, min particles is set to 20 and max particles is set to 200 as an input. This range is tuned based on the system specifications.

TABLE 2
AMCL parameters: UdacityBot

Overall Filter	
min particles	20
max particles	200
transform tolerance	0.5
initial pose x	0.0
initial pose y	0.0
initial pose a	0.0
Laser	
laser model type	likelihood field
Odometry	
odom model type	diff-corrected
odom alpha1	0.005
odom alpha2	0.005
odom alpha3	0.005
odom alpha4	0.005

TABLE.3, TABLE.4 and TABLE.5 show move base parameters in the configuration files. The move base package helps navigate the robot to the goal position by creating or calculating a path from the initial position to the goal. In TABLE.3, Local or Global costmap parameters specify the behavior associated with the local (or global) costmap. In TABLE.4, the laser sensor is defined as the source for observations for the list of parameters common to both types of costmaps. In TABLE.5, the set of parameters in the configuration file customize the particular behavior. The base local planner package provides a controller that drives a mobile base in the plane. For the project, max vel x is set to 0.5 and min vel x is set to 0.1 as an input.

TABLE 3
Local and Global costmap: UdacityBot

Local costmap	
global frame	odom
robot base frame	robot footprint
update frequency	10
publish frequency	10
width	20.0
height	20.0
resolution	0.05
static map	false
rolling window	true
Global costmap	
global frame	map
robot base frame	robot footprint
update frequency	10
publish frequency	10
width	40.0
height	40.0
resolution	0.02
static map	true
rolling window	false

TABLE 4
Common costmap Parameters: UdacityBot

Common costmap	
map type	costmap
obstacle range	5.0
raytrace range	8.0
transform tolerance	0.2
robot radius	0.5
inflation radius	0.55
observation sources	laser scan sensor

3.3 Personal Model: VladBot

3.3.1 Model design

Fig.2 shows a mobile robot (VladBot) that is built for simulated tasks. Vladbot is based on UdacityBot. To place a camera sensor and a laser sensor 0.2 higher than UdacityBot, ”face” (box size=”.1 .1 .2”) is added on the ”chassis”. TABLE.6 shows some of the specifications for the sensors.

3.3.2 Packages Used

A ROS package that launches a custom robot model in a Gazebo world is created and the packages like AMCL and the Navigation Stack are utilized.

3.3.3 Parameters

To achieve the best possible localization results, specific parameters corresponding to each package is explored, added, and tuned.

TABLE.7 shows Localization parameters in the AMCL node. For the project, min particles is set to 40 and max particles is set to 400 as an input. This range is tuned based on the system specifications.

TABLE.8, TABLE.9 and TABLE.10 show move base parameters in the configuration file. In TABLE.8, Local or Global costmap parameters specify the behavior associated with the local (or global) costmap. In TABLE.9, the laser sensor is defined as the source for observations for the list of parameters common to both types of costmaps. In TABLE.10, the set of parameters in the configuration file customize the particular behavior. For the project, max vel x is set to 0.7 and min vel x is set to 0.2 as an input.

4 Results

Both robots, UdacityBot and vladBot navigate to a specific goal position. The navigation is achieved for localization.

4.1 Localization Results

4.1.1 Benchmark

Fig.**??** shows UdacityBot at the goal position and with the PoseArray being displayed in RViz. It takes about 6 min for UdacityBot to reach the goal. The localization results look reasonable. UdacityBot does not collide with obstacles and takes a smooth path to the goal.

TABLE 5
Base Local Parameters: UdacityBot

TrajectoryPlannerROS	
holonomic robot	false
max vel x	0.5
min vel x	0.1
max vel theta	1.0
min vel theta	-1.0
min in place vel theta	0.2
acc lim x	2.5
acc lim y	2.5
acc lim theta	3.2
yaw goal tolerance	0.05
xy goal tolerance	0.05
sim time	1.0
meter scoring	false
pdistscale	0.6
controller frequency	10

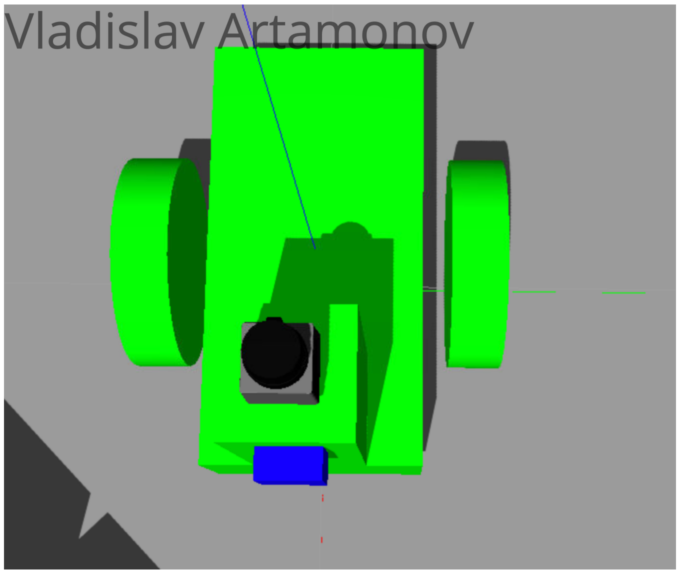


Fig. 2. VladBot

4.1.2 *Student*

Fig.?? shows VladBot at the goal position and with the PoseArray being displayed in RViz. It takes about 5 min for VladBot to reach the goal. The localization results look reasonable. VladBot does not collide with obstacles and takes a smooth path to the goal.

TABLE 6
Camera Sensor and Laser Sensor: VladBot

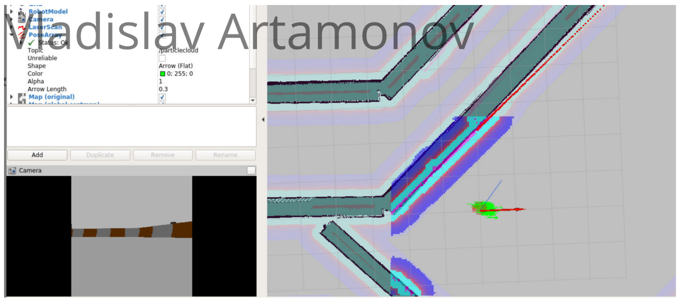
Camera Sensor	
link name	Camera
link origin	[0, 0, 0, 0, 0, 0]
joint name	camera joint
joint origin	[0.2, 0, 0.2, 0, 0, 0]
geometry	box with size ”0.05”
joint parent link	chassis
joint child link	camera
Laser Sensor	
link name	hokuyo
link origin	[0, 0, 0, 0, 0, 0]
joint name	hokuyo joint
joint origin	[.15, 0, .3, 0, 0, 0]
geometry	box with size ”0.1” for collision
geometry	a mesh file for visual
joint parent link	chasis
joint child link	hokuyo

TABLE 7
AMCL parameters: VladBot

Overall Filter	
min particles	40
max particles	400
transform tolerance	0.8
initial pose x	0.0
initial pose y	0.0
initial pose a	0.0
Laser	
laser model type	likelihood field
Odometry	
odom model type	diff-corrected
odom alpha1	0.005
odom alpha2	0.005
odom alpha3	0.010
odom alpha4	0.005

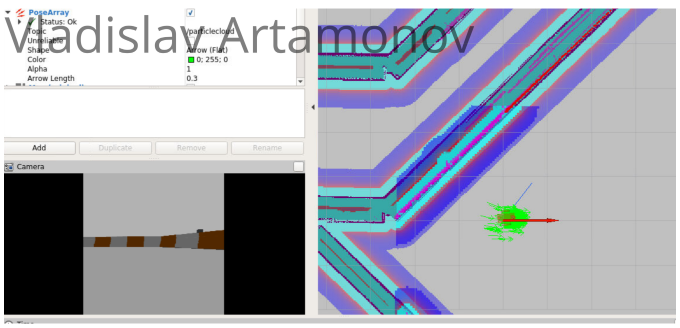
4.2 Technical Comparison

VladBot performs better than UdacityBot. It takes about 5 min for VladBot to reach the goal, but on the other hand it takes about 6 min for UdacityBot to reach the goal. The main difference between VladBot and UdacityBot is the layout of the robot. VladBot has ”face” (box size=”.1 .1 .2”) on the ”chassis” to place a camera sensor and a laser sensor 0.2 higher than UdacityBot.



Status: Ok
Topic /particlecloud
Unreliable
Shape Arrow (Flat)
Color 🟩 0; 255; 0
Alpha 1
Arrow Length 0.3

Fig. 3. UdacityBot at the goal position



Status: Ok
Topic /particlecloud
Unreliable
Shape Arrow (Flat)
Color 🟩 0; 255; 0
Alpha 1
Arrow Length 0.3

Fig. 4. VladBot at the goal position

TABLE 8
Local and Global costmap: VladBot

Local costmap	
global frame	odom
robot base frame	robot footprint
update frequency	10.0
publish frequency	10.0
width	20.0
height	20.0
resolution	0.05
static map	false
rolling window	true
Global costmap	
global frame	map
robot base frame	robot footprint
update frequency	10.0
publish frequency	10.0
width	40.0
height	40.0
resolution	0.02
static map	true
rolling window	false

TABLE 9
Common costmap Parameters: VladBot

Common costmap	
map type	costmap
obstacle range	5.0
raytrace range	8.0
transform tolerance	0.2
robot radius	0.5
inflation radius	0.55
observation sources	laser scan sensor

5 Discussion

The localization performance of VladBot is better than UdacityBot as mentioned above because the sensor layout might be able to provide more information for localization. A camera sensor and a laser sensor of VladBot are placed 0.2 higher than those of UdacityBot.

In the process, the robot may be kidnapped and moved to a new location on the map. The 'Kidnapped Robot' problem would be approached by the parameter tuning such as the reduction of the number of particles. MCL/AMCL in an industry domain would used to localize robustly in factory buildings and halls.

6 Conclusion / Future work

A mobile robot model for Gazebo is developed to create the own ROS package, and the AMCL and Navigation ROS packages are integrated for localizing the robot in the provided map. The parameters are tuned for the ROS packages to improve the localization results in the map.

Two different robot models, UdacityBot and VladBot, are considered for performance evaluation. After achieving the desired results for UdacityBot, VladBot is implemented with significant changes to the robot's base and the sensor locations. The parameter settings with VladBot is checked, and the localization results are improved upon as required for VladBot.

TABLE 10
Base Local Parameters: VladBot

TrajectoryPlannerROS	
holonomic robot	false
max vel x	0.7
min vel x	0.2
max vel theta	1.0
min vel theta	-1.0
min in place vel theta	0.2
acc lim x	2.5
acc lim y	2.5
acc lim theta	3.2
yaw goal tolerance	0.05
xy goal tolerance	0.05
sim time	1.0
meter scoring	false
pdistscale	0.6
controller frequency	10

For future Work, some enhancements could be made to the model to increase accuracy and/or decrease processing time. For the trade-offs in accuracy and processing time, all that is needed is to benchmark some of the processing times and choose the parameters that best meets the constraints of processing time or accuracy required.

Furthermore, the addition of more sensors and the modification of the robot base size also might be required for the improvement.

References

[1] Udacity, "Robotics software engineer nanodegree program term 2: Localization in robotics."
[2] Udacity, "Monte carlo localization."