Robot Localization

Adam J. Carter

Abstract—Adaptive Monte Carlo Localization was implemented in simulation on two similar but slightly modified versions of a two wheeled differentially-driven caster-supported mobile robot. RGB camera and Laser range-finder data constituted the sensor array. Both robots were able to successfully localize and navigate to a distant marker when placed in an unknown portion of a known map. Relevant aspects of robotic localization and navigation are discussed.

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

1 Introduction

Localization is the problem of determining a robot's pose in a mapped environment. By 'mapped' we mean the robot has a basic map of the environment stored in memory, it just doesn't know where it is located on the map. If a robot were to suddenly 'power up' in an unknown location of a mapped environment it would need to compare it's stored map to many sensor perspectives of the environment from different angles in order to arrive at a reasonable guess as to it's location in the environment. Stationary unique environmental objects that can be seen from multiple locations on the map could be used as landmarks for orientation. After taking many measurements and iteratively eliminating improbable location guesses, the robot can narrow down on it's true location. There are four main algorithms used in robotics for localization: Extended Kalman Filter, Markov Localization, Grid Localization, and Monte Carlo Localization or 'Particle Filter'. This application uses Monte Carlo Localization, or more specifically Adaptive Monte Carlo Localization (AMCL).

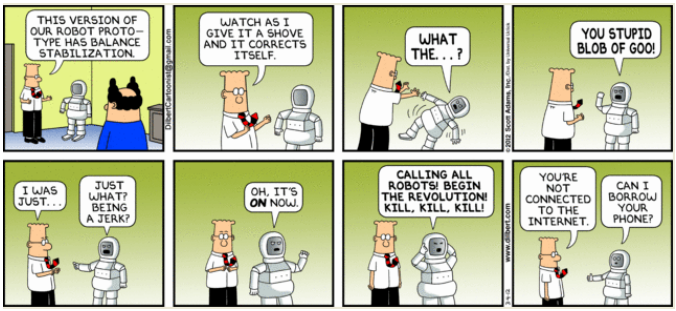

Fig. 1. Robot Revolution.

1.1 Challenges in Localization

1.1.1 Localization Problem 1/3: Position Tracking

There are three main types of localization problems. The simplest, though not trivial problem is position tracking (local localization). This consists of starting a bot with a known location in a known environment and tracking it's position closely over time as it navigates. Due to the inherent uncertainty in sensor measurements and odometry any robot will 'drift' from it's planned path over time. However, the uncertainty is contained to the region immediately surrounding the robot, not the entire map. Accounting for this uncertainty and correcting for it is reliably achieved using Kalman Filters as we will discuss.

1.1.2 Localization Problem 2/3: Global Localization

The second problem is global localization. This is analagous to the localization problem introduced in the first paragraph where a robot is introduced to a known or 'mapped' environment but in an unknown location and then must determine it's pose based on it's surroundings. Due to the larger amount of uncertainty that must be dealt with in the global map, this problem is significantly more challenging than position tracking.

1.1.3 Localization Problem 3/3: Kidnapped Robot

The 'kidnapped robot' problem provides us with the most challenging localization task to solve. It is similar to global localization but with the added difficulty that the robot can be picked up (kidnapped) and moved to a new unknown location in the mapped environment at any time. The process of global localization must then be repeated. Since most robots undergo times of uncertainty in their pose, recovering from these periods becomes a valuable skill. Learning from the kidnapped robot problem helps us build localization algorithms that are more robust. Yet another challenge is introduced when we consider dynamic environments where locations of objects change over time. For now we will keep the problem simple and just consider static environments. The importance of being able to accurately localize a robot in it's environment can not be overstated. All other tasks required of the robot become secondary as failure in localization will lead to failure of any other task.

2 Background

First used by the Apollo Space Program to guide the navigation to lunar orbit, Kalman filters have become one of the most important algorithms in controls engineering. Kalman filters are used to estimate the true value of a noisy variable in real-time. The variable being estimated could range from the temperature or pressure of a system all the way to

position tracking of a mobile robot. They are useful because they can take measurements containing uncertainty and return an accurate estimate of the real measurement value. Kalman filters provide their noise-reduced estimates after taking in very little data compared to other methods so they act quickly in real-world applications. [1]

2.1 Kalman Filters

Robots encounter uncertainty in their location due to two main sources of error:

1) The inherent noise coming from the sensor array.
2) The small but cumulative errors introduced during movement.

Movement errors are caused by environmental irregularities like wheel slip on slick or uneven terrain. **Kalman filters take into account the expected uncertainty in a sensor reading or movement and combine that with a prior estimate of the robot's state in order to formulate a hypothesis about the current state**. By combining the measurements of an array of different sensors (sensor fusion), Kalman filters can arrive at a more accurate estimate of the current state than any single sensor could on it's own. Following a movement or measurement update, the Kalman filter outputs a unimodal Gaussian distribution representing its best guess at the true value of a parameter.

2.1.1 The Kalman Cycle

We start with an initial state belief, which can be surprisingly inaccurate without causing the Kalman filter to fail. The prior state belief has a degree of uncertainty to it and is therefore represented as a Gaussian distribution around the mean or most confident value. The width of the Gaussian represents the degree of uncertainty, with a wider distribution (higher standard deviation) representing more uncertainty.

When a measurement update comes in, it is also represented with a Gaussian so as to account for the noise in the sensor. The mean of the new belief is calculated as a weighted sum of the prior and measurement means. The new belief should be biased toward the measurement update since it generally has a smaller standard deviation than the prior. We accomplish this by scaling the mean of the measurement by the uncertainty of the prior and scaling the mean of the prior by the uncertainty in the measurement. To summarize, the measurement update is a weighted sum of the prior belief and the measurement.

The variance in the new belief is calculated using the reciprocal of the sum of the squared reciprocals of the component variances. Since this uncertainty is derived from two different sources, it should carry more confidence than either of the uncertainties from which it is composed. This leaves us with an updated state estimate (posterior) that is more confident than both the prior belief and the measurement update.

In the next stage of the cycle this newly calculated state estimate is treated as the prior belief and a motion control command is executed. The result of the motion is a new Gaussian (posterior belief) whose mean and standard deviation are calculated as simple sums of the means and variances of the prior belief and the motion. At this point a full cycle of the Kalman filter has been achieved and the newly calculated posterior becomes our prior belief for the next round of updates. To summarize, the state prediction is the addition of the prior belief's mean and variance to the motion's mean and variance.

2.1.2 Multi-Dimensional Kalman Filters

This discussion has used position as the only variable being estimated. This means our Kalman filter is 1-dimensional. When x,y,z,theta pose coordinates are needed or another parameter such as velocity needs to be estimated we can achieve this through extension into multi-dimensional Kalman filters using linear algebra. Linear algebra allows us to work with multi-dimensional problems. Basically, the new variables are arranged in an array such that calculations can be carried out on them simultaneously. In the case of velocity (a hidden variable that is not directly measured), an estimate can be found by linking the relationship of velocity to distance and time into the covariance matrix. By this method multi-dimensional Kalman filters can be designed.

Previously we noted that the prior belief and the measurement update come with their own uncertainties. These can vary over time in some situations as sensor interference and other signal losses may occur. To account for this we calculate the Kalman gain which determines how much weight the state prediction should get versus the measurement update. The more uncertain of the two is given less weight. Through the magic of linear algebra and inverse matrix cancellations, it turns out the Kalman Gain is equal to the inverse of the measurement function. We update the new state's covariance using the Kalman Gain. In situations where a sensor goes offline or becomes unreliable, the Kalman gain will shift belief toward relying on the state prediction, thus recovering from inaccurate initial measurements.

2.1.3 Extended Kalman Filters

The Extended Kalman Filter allows us to deal with non-linear functions. Many movements of a robot (turns, curved paths) are non-linear. Using the multi-dimensional Kalman filter to provide updated estimates of the mean would be acceptable, but when the covariance is calculated a non-gaussian distribution is produced. Using the (first two terms of the) Taylor Series expansion we can account for non-linear inputs and outputs to the system. This approach uses locally linear approximations to non-linear functions to approximate their value.

2.2 Particle Filters

Particle Filters estimate state recursively using a set of samples (particles) drawn according to the posterior (known) distribution over robot poses. This random collection of weighted particles approximates the desired distribution and eliminates the necessity of extracting features from the sensor data.

2.3 Comparison / Contrast

Particle filters can accommodate many sensor types, motion dynamics, non-linearities and noise. They are better able to

deal with non-Gaussian distributed noise than other localization algorithms like Markov. By sampling in proportion to the posterior likelihood and controlling the number of particles used dynamically they also focus computational resources cleverly. These characteristics make them a good choice for attacking the Localization problem in this project. Kalman filters are limited by the fact that they require feature extraction from the sensory data and also are not well adapted to solving global localization or the kidnapped robot problem due to the restricted distribution of their belief representation.

3 Simulations

Differences between the Udacity Bot and the Adam Bot

1) Smaller wheels mounted to the bottom of the chassis rather than the midpoint
2) Hokuyo Laser mounting point moved to top dead center.
3) A small wing on the trunk for down-force was considered...

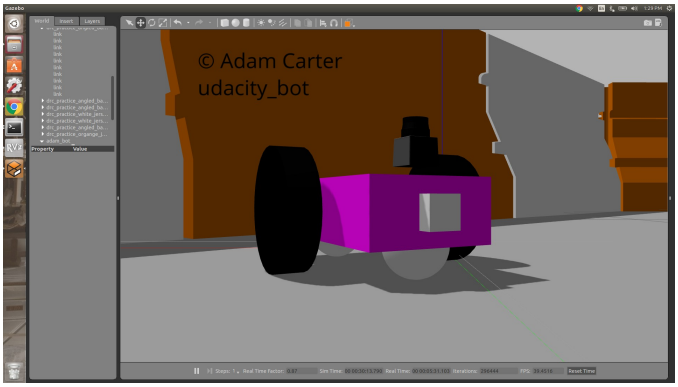

Fig. 2. Udacity Bot 1.

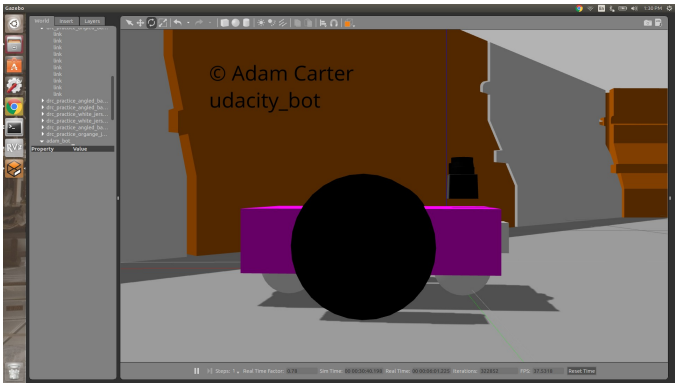

Fig. 3. Udacity Bot 2.

3.1 Achievements

Both the Udacity bot and the Adam bot were able to successfully localize when placed on an unknown location of a known map and then navigate to a distant navigation marker utilizing Adaptive Monte Carlo Localization implemented via sensor fusion of camera and laser input.

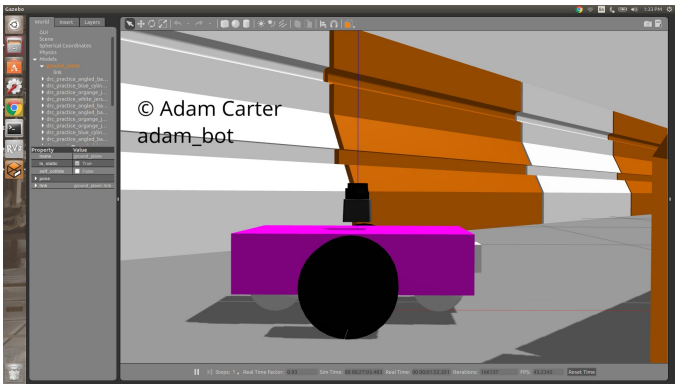

Fig. 4. Adam Bot 1.

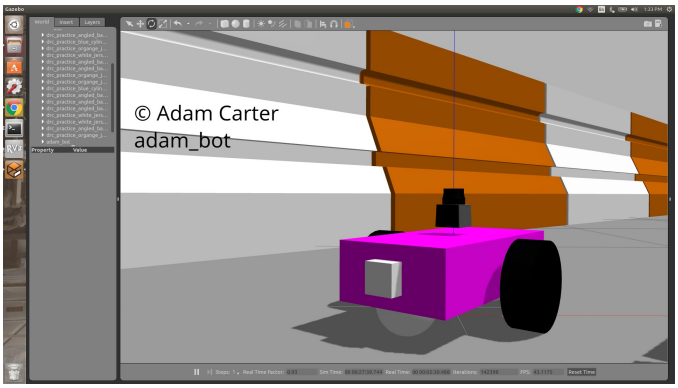

Fig. 5. Adam Bot 2.

3.2 Benchmark Model

3.2.1 Model design

The platform is a simple two wheeled diff-drive rover with a box shaped chassis measuring 40cm x 20cm x 10cm. There are casters under the nose and tail, a front mounted RGB camera and a top mounted Hokuyo laser unit.

3.2.2 Packages Used

The output of rospack list is too extensive to include here. The main packages used include the ROS navigation stack and move base package which navigates to the goal position by calculating a path from the initial position to the goal, AMCL which localizes the robot, and various other packages like Turtle Bot Teleop (used for manual navigation during tuning), Gazebo, and RViz.

The output of rostopic list is too extensive to include here but can be easily accessed through the included file system.

The output of rosservice list is too extensive to include here but can be easily accessed throught the included file system.

3.2.3 Parameters

The primary localization parameters tuned by the author are listed and described in the following table. The primary sources used to aid in tuning of these parameters were the Basic Navigation Tuning Guide available on ROS.org and a helpful ROS Navigation Tuning Guide by Kaiyu Zheng available from arXive.org.

3.2.4 Overall Filter

Min Particles: 250

This specifies the smallest number of pose belief particles that are used. When certainty in localization is high, fewer vectors can be used and computational resources can be preserved. As uncertainty rises more belief particles can be used to help determine orientation. This parameter is tunable per-system as available system resources determine it's setting.

Max Particles: 2000

This specifies the largest number of particles used in the filter. Similar to the minimum particles, it is set so as to stay withing system memory limits while allowing adequate particles for robust localization.

3.2.5 Laser

Laser min range: 2.0

This value extended the 1.5 default distance so as to avoid the laser reading anything on the robot itself as an obstacle.

Laser max range: -1

This leaves the range at the manufacturer's default setting which can be found in the Hokuyo documentation.

3.2.6 Odom

Using diff drive meant we used an algorithm (sample motion model odometry) from Thrun's Probabilistic Robotics which uses odom alpha's 1 through 4.

odom alpha 1: 0.005 Specifies the expected noise in odometry's rotation estimate from the rotational component of the robot's motion.

odom alpha 2: 0.005 Specifies the expected noise in odometry's rotation estimate from translational component of the robot's motion.

odom alpha 3: 0.010 Specifies the expected noise in odometry's translation estimate from the translational component of the robot's motion.

odom alpha 4: 0.005 Specifies the expected noise in odometry's translation estimate from the rotational component of the robot's motion.

These values were used as starting points based on weak evidence in a stack exchange thread from a Google search on the subject. They turned out to work reasonably well which suggests that the algorithm really just needs a bit of noise to work with and doesn't seem to be picky about the details.

3.2.7 Transform Tolerance

Transform Tolerance: 1.0 Time with which to post-date the transform that is published, to indicate that this transform is valid into the future.

This parameter is set by the limitations imposed by the system like many others. The anticipated latency for a transformation from the base frame to the map frame can be traced using tf monitor. The value for Transform Tolerance can then be adjusted such that this latency is taken into account.

Sim Time: 3.0 The amount of time in forward simulation of motion spent determining the best trajectory. System resource dependent. Influences the length of the proposed path planned ahead. This is a base local parameter set in the appropriate yaml file.

3.3 Personal Model

3.3.1 Model design

Similar to the Udacity Bot, the Adam Bot platform is a simple two wheeled diff-drive rover with a box shaped chassis measuring 40cm x 20cm x 10cm. There are casters under the nose and tail, a front mounted RGB camera and a top mounted Hokuyo laser unit. Differences in the Adam Bot include a lowering of the wheel mount points to the bottom of the chassis, shrinking of the wheel diameter and movement of the Hokuyo Laser from the front mid-quarter to top dead center of the platform.

3.3.2 Packages Used

The packages used for the Adam Bot were the same as those used for the Udacity Bot.

3.3.3 Parameters

The parameters used for the Adam Bot were the same as those used for the Udacity Bot.

4 Results

There was not a significant difference (in the author's opinion) in performance between the Udacity Bot and the Adam Bot. Judging by the clustering of the pose array, both bots took 10-30 seconds for the particle filters to converge. They both exhibited similar wandering behavior after initialization with an initial slow crawl in the exact opposite direction of the planned path before a stop and relatively quick turn, then more slow wandering behavior before picking up on the path and taking off. Both bots navigated successfully to the nav marker in around five minutes.

The bots showed similar behavior in their cornering attack angles and exit angles as well. After some tuning of the sim time and controller frequency a planned trajectory was obtained that stayed wide of the corner and fairly constant in it's radius. The pdist scale and gdist scale determine the bot's preference for sticking to the path vs heading toward the goal. With these set equally, the bot's actual path around the corner was consistently more 'V'-like than the planned path. The bot exhibited a rather sharp turn-in prior to apex that brought the inside wheel unnecessarily close to the inside wall of the turn. This likely comes down to the balance of path bias and goal bias being too strong toward goal, even when they are set equal. The cornering behavior did not cause collision however so it was not further tuned.

4.1 The Failed Prototype

Prior to implementation of the Adam Bot, an Adam Bot prototype was built. The main structural difference was a move of the main axle from the midpoint of the chassis back to the rear of the chassis with elimination of the rear caster. The prototype bot exhibited such poor mobility characteristics that the design was abandoned. The forward mobility was hindered by a jumping action as the front caster did not roll properly when under load. 'Dragging' the caster with the drive axle up at the front of the bot was considered but abandoned due to fears concerning anticipated poor caster performance in any type of load situation. For this reason the final build went back to a version of the Udacity Bot with a centrally located drive axle with front and rear balanced and relatively unloaded casters.

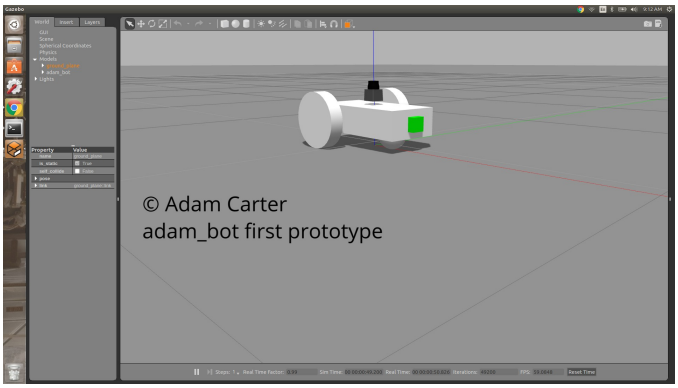

Fig. 6. Adam Bot Prototype.

4.2 Localization Results

4.2.1 Benchmark

10-30 seconds for filter convergence. 4-5 minutes to reach goal.

4.2.2 Student

10-30 seconds for filter convergence. 4-5 minutes to reach goal.

4.3 Technical Comparison

The Udacity Bot encountered two main sources of difficulty. The casters were initially slightly too large and did not allow smooth turning behavior due to collision with the chassis. When their radii were decreased slightly performance improved greatly. The other problem involved the Hokuyo Laser impinging on the front edges of the drive wheels and being blocked. This made the navigation algorithm believe there were obstacles immediately in front of the bot that were not in fact present. Modifications to the laser minimum distance parameter or laser mounting location overcame this issue. These changes were carried over to the Adam Bot.

There was not a significant difference in performance between the two bots, likely due to the overall similarity in their designs.

Significant changes to the Adam Bot relative to the Udacity Bot included shrinking the wheels so as to avoid interference with the path of the Hokuyo Laser and movement of the laser to the midpoint of the chassis in an attempt to more equally distribute weight and thereby dampen some of the vibration and bounce inherent in the platform. The wheel changes effectively solved the laser interference problem but this is a problem that can also be addressed by lengthening the minimum laser distance (laser min range parameter) in the AMCL launch file. The minor change in weight distribution did not make a significant enough difference in 'ride quality' to improve overall performance in getting to the goal.

5 Discussion

Casters are officially the worst. The author's next attempt at a mobile ground-based robot build will hopefully implement Ackerman Steering. Though this adds complexity, the gain in platform stability, ruggedness and obstacle robustness would be worth it. The AMCL algorithm worked extraordinarily well in the author's opinion. This isn't a 'typical' machine learning algorithm in the sense of utilizing a neural network but the sophistication of the probabilistic model underlying the algorithm is impressive in a similar way. By iteratively weighting and selecting pose representation vectors using probabilities, we can eliminate less likely poses until we narrow down on the most likely pose estimate for the robot in the given space. For situations where a robot can be provided with a pre-existing map, this algorithm will allow it to localize and navigate within the world represented by the map by comparing it's sensor inputs to the estimated poses.

5.1 Topics

- Which robot performed better?
  The performance of the two bots were within the noise of each other on nearly every run.
- Why it performed better? (opinion)
  The author anticipated a small improvement in performance of the Adam Bot due to the weight distribution improvement. The lack of any detectable improvement likely speaks to the fact that platform instability is not the main determinant of performance in this situation. Further tuning of the AMCL algorithm parameters to shorten the 'wandering' behavior prior to convergence seems like the next best area to focus on for improvement. Similarly the oscillating behavior exhibited prior to settling in to the target location could potentially be tuned out via manipulation of the x,y and yaw goal tolerance parameters.
- How would you approach the 'Kidnapped Robot' problem?
  The robot using AMCL will fail the kidnapped robot problem due to a lack of pose vectors available when it gets dropped in a part of the map it has not seen previously. In a closed environment like a warehouse an engineer could potentially provide a robot with residual pose estimates stored in memory from previous explorations or explorations from other bot units who are networked.
- What types of scenario could localization be performed?
  Localization using AMCL can only be performed when a map of the environment is available to the robot. The algorithm allows localization and navigation after introduction to an unknown portion of the known map, but the map is required for it to work.
- Where would you use MCL/AMCL in an industry domain?
  Any manufacturing or industrial application where the workspace of the robot is a prior known would lend itself to use of this algorithm. Many warehouse or production line environments are likely fairly static at least as far as navigable paths go. That makes for a lot of possible applications.

6 Conclusion / Future work

This project was successful in achieving the goal of implementing an autonomous localization and navigation suite

on a mobile robotic platform in simulation. The author recently received a TX-2 so with future acquisition of a mobile platform, a real-world deployment could be attempted. This will provide good practice for deployment of the AMCL algorithm in a commercial environment, for instance in the context of a mobile delivery bot on the floor of a manufacturing warehouse. The most important next step for future work will likely involve further refinement of the localization algorithm performance and further refinement of the robotic platform. Ackerman Steering may be more suitable in some situations, or possibly a better version of a caster-supported bot.

6.1 Modifications for Improvement

Examples:

- Rather than a top surface area that is 0.2 x 0.4 meters, a square platform 0.4 x 0.4 meters may be more useful for transporting things, including the sensor array and computational components of the bot. If the depth dimension could be made more shallow say by 25 to 50 percent, ground clearance would be improved. That is assuming the inner volume of the chassis is not usable real-estate. A wider platform may be more stable over rough terrain.
- Adding more cameras may be a worth-while modification. If a future mapping algorithm relied on having images of the environment from many different angles then transit time in generating data could be reduced by gathering from multiple perspectives in one sweep. Additionally a vertically sweeping laser array would allow for more precise localization in the z direction.
- If data acquisition were an objective of the localization or mapping algorithms then the more diverse the sensor array is in regard to how many perspectives it takes in, the better. Top, front, sides, back, corners could all be utilized.
- Computational resources will likely determine this. The TX-2 documentation allows for six camera inputs.

6.2 Hardware Deployment

1) What would need to be done?
   To make hardware deployment a reality, a mobile robotic platform would have to be constructed or purchased. Since there are existing examples of platforms built from RC trucks, the author was considering purchasing one for deployment with the TX-2. A more costly but potentially more useful (indoor only) platform like the turtlebot could be considered also.
2) Computation time/resource considerations?
   Considering the fact that this implementation ran without problem on the author's personal laptop, performance on the TX-2 is anticipated to be good. More sophisticated machine-learning based visual systems like a CNN-based object identification algorithm could be considered additionally.

REFERENCES

[1] L. Lamport, *LATEX: a document preparation system: user's guide and reference manual*. Addison-wesley, 1994.