A Monte Carlo Localization Assignment using a Neato Vacuum with ROS

Zuozhi Yang and **Todd W. Neller**
Gettysburg College
Gettysburg, Pennsylvania, 17325 USA
tneller@gettsyburg.edu

Abstract

Monte Carlo Localization (MCL) is a sampling-based algorithm for mobile robot localization. In this paper we describe an MCL assignment and its required hardware and software. The Neato vacuum robot and a Raspberry Pi serve as the core of the robot model. The Robot Operating System (ROS) is used as the robot programming environment. Students are expected to learn the localization problem, implement the MCL algorithm, and better understand the kidnapped robot problem and the limitations of MCL by observing the performance of the algorithm in real-time application.

Introduction

This abstract offers a high-level description of an open-access robotics assignment concerning Monte Carlo Localization (MCL). We begin with a description of the assignment challenges. In later sections we described the hardware and software employed, the relevant algorithms, and the interesting design tradeoffs posed by the assignment.

One will note that this work largely derives from that of Paul Ruvolo, who has been generous in his assistance. However, we see possibilities for open-ended design to address differences between Ruvolo's assignment, classic MCL, and newer approaches that address the Kidnapped Robot Problem.

The Challenges

The mobile robot localization problem is to determine the pose (direction and position) of the robot given the map of the environment, sensor data, a sensor error model, movement data, and a movement error model. It is a very basic problem of robotics since most of robot tasks require knowledge of the position of the robot. There are three types of localization problems in increasing order of difficulty (Thrun, Burgard, and Fox 2005).

Local Position Tracking The initial pose of the robot is assumed to be known in this type of problem. Since the uncertainties are confined to region near the actual pose, this is considered to be a local problem.

Global Localization In contrast to local position tracking, global localization assumes no knowledge of initial pose. However, it subsumes the local problem since it uses knowledge gained during the process to keep tracking the position. The goal of the MCL algorithm introduced in this assignment is to perform global localization.

Kidnapped Robot Problem The kidnapped robot problem arises from the movement of a successfully localized robot to a different unknown position in the environment to see if it can globally localize. Thus it is more difficult than global localization problem since the robot has a strong but wrong belief in where it is. The original MCL algorithm does not have the ability to recover from kidnapping. Such failure is also often referred to as catastrophic failure.

Audience

This assignment is suited for an upper-level undergraduate or graduate student in an Artificial Intelligence or Mobile Robotics course.

Prerequisites

- facility with programming Python 2.7
- some experience with Unix/Linux operating system
- some exposure to linear algebra and probability theory

Resources

- Textbook:
  Probabilistic Robotics by Thrun et al.
- *A Gentle Introduction to ROS* by Jason M. O'Kane: https://cse.sc.edu/%7Ejokane/agitr/

Hardware and Software

The Hardware

The robot consist of several parts (Ruvolo 2015):

- **Neato XV vacuum robot** - This model has the essential parts we need: an actuator for moving and laser range finders for sensing.
- **A Raspberry Pi 2 Model B with an Adafruit B&W 16x2 LCD and keypad kit** - This is for communication between computer and the robot. More specifically, it will host an ad-hoc wireless connection that computer can join

using wi-fi. Thus, a wireless adapter will be necessary on both the Raspberry Pi and the computer.

- **A Unix/Linux machine** - All intensive computation must be done on a Unix/Linux computer. Sensor information is sent from the robot to the computer. The computer uses the information as input to MCL. Also movement command is sent from the computer to the robot at the same time.
- **Two TP-Link N600 Wireless Dual-Band USB Adapters** (one for the computer and one for the Raspberry Pi)

The Software

ROS The Robot Operating System (ROS) is a collection of packages and software building tools specialized for robot programming. It has become the standard robotics development environment in academia as well as some corporate settings. In this assignment we use it as a communication platform for the robot and computer. We also utilize some of its libraries.

Monte Carlo Localization

Algorithm MCL($\mathcal{X}_{t-1}, u_t, z_t, m$):
$\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
for $m = 1$ to M do
  $x_t^{[m]} = \mathbf{sample\_motion\_model}(u_t, x_{t-1}^{[m]})$
  $w_t^{[m]} = \mathbf{measurement\_model}(z_t, x_t^{[m]}, m)$
  $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
endfor
for $m = 1$ to M do
  draw i with probability $\propto w_t^{[i]}$
  add $x_t^{[i]}$ to $\mathcal{X}_t$
endfor
return $\mathcal{X}_t$

Figure 1: Pseudocode for MCL (Thrun, Burgard, and Fox 2005)

The Algorithm

Monte Carlo Localization is an algorithm that begins with a set of random hypotheses about where the robot might be all over the map and in any heading. As the robot moves and senses, a Darwinian survival-of-the-fittest process tends to multiply the most likely hypotheses and tends to kill off the least likely, gradually evolving a cloud of hypotheses where the center, i.e. average, is the most likely robot position and heading. Each of these hypotheses is called a particle, as this technique derives from a general technique called Particle Filtering. The algorithm iteratively computes and updates the probabilities of its hypotheses. Each iteration of computation has two phases (Dellaert et al. 1999):

Prediction Phase In this phase we need a motion model to predict the pose of the robot by only taking motion into account. The process is Markovian since the current pose is only dependent on one pose before. Initially we uniformly sample our particles in the map, having no prior knowledge of the pose.

Update Phase In the second phase we need a measurement model in order to incorporate sensor information. Each particle will be taking an updated weight such that the current belief of the robot pose is usually the weighted average of pose positions and of pose headings.

After the update phase, the algorithm resamples a new collection of particles according to the current particle weights and starts the next iteration of computation.

Learning Objectives

- Learning how to build and debug robotics applications with ROS
- Learning the problem of localization and the kidnapped robot problem
- Learning to work with and debug hardware
- Learning an example of a particle filtering algorithm through the implementation of Monte Carlo Localization
- Learn the limitations of MCL and trade-offs in implementation

Possible Usage of the Assignment

This is a short assignment to learn MCL and to address the kidnapped robot problem. It is adaptable to serve different teaching purposes. Our primary focus is on development towards robust success with the kidnapped robot problem in the context of an introductory AI course.

With the problem and algorithm explained and the robot and the base code supplied in early semester, several core functions, such as updating and re-weighting particles, will be required for implementation. Augmented MCL will be explained in class but not required for implementation. Student then use the rest of the semester to come up with their own ideas to deal with the kidnapped robot problem. At the end of the semester, a demonstration and discussion will be held for students to exchange ideas.

Challenges in Developing the Assignment

- **Trade-off between accuracy and efficiency:** Since this application requires low latency between receiving data and sending back instruction, the computation needs to be fast. In that regard, we need to sacrifice some accuracy and only maintain relatively few particle samples to reduce computing time between each iteration. Thus the application works better in a relatively small test arena.
- **The Limitation of the Algorithm:** The MCL algorithm suffers from and can hardly recover from a incorrectly converged localization. Initially, Ruvolo's assignment requires a given pose as approximate guidance of where the robot actually is in the map. Gaussian sampling around

this pose then supplies the initial set of particle. However, this prior knowledge simplifies the problem to a local position tracking problem. Given that we wish to teach the general MCL algorithm and make progress towards the kidnapped robot problem, a uniform sampling of the initial particle cloud is required for our adapted assignment. Motivated students can investigate more sophisticated algorithms that can deal with this problem, such as augmented MCL (aMCL), which introduces random particles to MCL as a catastrophic localization failure becomes more likely.

Available Resources

Full details concerning our adaptation of Ruvolo's assignment for the kidnapped robot problem are available via our website `http://tinyurl.com/gburgmcl`.

Acknowledgment

This assignment is adapted from one of the assignments from Paul Ruvolo's Computational Robotics taught in Fall 2015 at Olin College of Engineering. We gratefully thank him for his support, advice, and encouragement.

References

Dellaert, F.; Fox, D.; Burgard, W.; and Thrun, S. 1999. Monte carlo localization for mobile robots. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, volume 2, 1322–1328. IEEE.

Ruvolo, P. 2015. Computational robotics fall 2015. `https://sites.google.com/site/comprobo15/`.

Thrun, S.; Burgard, W.; and Fox, D. 2005. *Probabilistic robotics*. MIT press.