Using 3D Models and Computer Vision Algorithms to Implement Monte Carlo Localization

John Allard, Alex Rich
2014 Summer Computer Science REU, Harvey Mudd College

July 6th, 2014

Contents

Abstract

The Monte Carlo Localization (MCL) algorithm has been used in the past[1] to successfully localize robots using 2D maps of an environment and a stream of range sensor information. Our research group is attempting to implement the Monte Carlo Localization algorithm using a 3D model of the environment and a feed of images from a robot in that environment. This entails the use of various computer vision algorithms to find and compare features between the image feed and our 3D model. This paper will outline the overall processes that our research group has undertaken to accomplish this task and an analysis of our resulting program.

1 Introduction

1.1 MCL Process Overview

This section is for those unfamiliar with the Monte Carlo Localization algorithm.
The overall process of having an actor[2] localize itself in an environment via the MCL algorithm is comprised of many steps. A simplified outline of these steps can be stated as follows.

Pre-Localization Attempt :

1. A map of the environment needs to be imported or constructed. This can be a 2D top-down map, a 3D map created with a laser range sensing camera, or any other type of map associates distinguishable features with spacial coordinates.

2. Some set of quantifable features about the map must be chosen. This enables a comparison between sensor readings from the actor and expected sensor readings from different places in the map.

3. This step is optional. Features from the map are computed from many different reference points and are stored for use during the localization attempt. This limits the possible particle locations compared to generating feature data at run-time, but it reduces the computational resources needed significantly and allows one to simply look up the data in a map or container. Pre-computing features also significantly reduces the complexity of the localization source code.

During Localization Attempt :

1. An actor is let loose in the environment, and a large amount of guesses as to where it could be are randomly generated according to some distribution[3]. These 'guesses' are called particles, and each particle is a data structure that contains information about its own current perspective in the environment and the sensor data it would expect to read from that perspective.

2. The program compares the current sensor readings from the actor to the expected sensor readings for each particle, and assigns a weight to each particle based on how strongly the readings correspond to one another.

3. A distribution is created according to the grouping and weighting of particles in the program. The more heavily weighted a particle is, the higher probability it has to be sampled from our distribution.

4. A new set of particles are sampled from this distibution, and a few particles are thrown into the mix from a uniform distribution. After this step the total number of particles in the program is the same as in the last step.

5. The actor is moved to a new point in the environment via some movement commands from the program. Each particle has its perspective updated according to the same commands, plus some statistical error based off of the uncertainty in the robots actual movements compared to

[1] Dieter Fox, et al. Carnegie Mellon University, University of Bonn.
[2] Any device that has sensors and can move around an environment
[3] The uniformity of this distribution depends on the users previous knowledge of the actors location.

the movement commands. The expected sensor readings for each particle are also updated to correspond to its new perspective of the environment.

6. Steps 2-4 are repeated until the particles converge on a given perspective within the map.

2 Building the 3D Map

For our research we decided on using a 3-dimensional model as our map of the environment. This model is a fully textured high-quality laser scan of the 2nd floor of the Sprague building at Harvey Mudd College. A 3D model was important for our research because we intended on using cameras as our sensors for the actor in our environment. This implies that we would have a full 3D model of the environment with which to compare images from the actor against. Building the 3D map would have been nearly impossible without the generosity of the Matterport team and specifically their COO Mike Beebe. They gave our research team a very well built and user-friendly 3D imaging camera which uses laser-range data to scan an enclosed space. The use of this camera saved our team countless headaches that would have incurred by attempting to stitch Kinect range-data together.

2.1 Taking the Scans

Our team took 42 scans of the space we work in, a roughly 3500 sqft floor consisting of 7-8 rooms and a large amount of non-static objects, such as chairs, robots, and whiteboards. The Matterport software allows about 100 scans for a single model, and the software bundled with the camera automatically merges these scans into one giant textured mesh. The software also automatically converts the data to a .obj file format for viewing on their site. We were then able to use this .obj file to determine features about our map.

2.2 Filling In the Gaps

Our model was inevitably left with many gaps from places that were obstructed from the view of our camera. To help improve the overall quality of our map we used the Meshlab software. This allowed us to remove disconnected pieces, remove unreferenced vertices, and smoothen out some jagged areas. on top of this, we rendered the background color pink to allow us to single out features on these areas and remove them from the program.

2.3 Creating a Database of Images from the Map

Our goal is to use computer vision algorithms that compare between 2D images, which meant that we had to represent our 3D map in 2D for our feature matching algorithms to work properly. To represent our 3D map as 2D information, we decided to render images from thousands of different perspectives within our map. The following steps were taken to accomplish this.

1. Establish a bounding box around our map.
2. Define a plane that sits above and parallel to the x-y plane of our map.
3. Define how tight of a grid we would like to impose over this plane.
4. Go to each grid location and take images from 8 different perspectives, rotating from 0 to 360 degrees counting by 45 degree intervals.
5. Name the images according to their location in the map and store them for later use.

What this process allows us to do is turn our 3D model into a catalogued database of 2D images from a large amount of places within our model. This allows us to use existing computer vision related algorithms (like SURF, SIFT, ORB, etc) to do the matching and weighing between places in our 3D model and the incoming image feed from the actor in the environment. Once the database of images was computed, the next step was computing another database of computer-vision derived features from these images.

3 Computing Features from the Map

Having a database of images rendered from different locations in our map is only half of the battle. The next step is to go through each image and compute a large set of general feature data about each picture. The purpose of doing the during the pre-localization phase is that it will save us a large amount of computation from occuring during run-time. Computing and cataloging this data now will allow us to load it into a map during the boot phase of the localization process and simply look it up when needed at run-time.

3.1 Types of Features

There are a variety of features that can be used when comparing two images. In its current state, the project uses extracted features via SURF and SIFT as well as grayscale and black and white images.

3.2 Storing the Features

4 MCL Implementation