Probabilistic Pose Estimation based on Topological Map

Autonomous Systems Lab, ETH Zurich, Switzerland

January 7, 2008

1 Overview

An autonomous mobile robot must have some method by which to determine its position with respect to known locations in the environment in order to navigate and achieve some goals. This is the localization problem. In this exercise, we will try to implement a simple localization method using the techniques from topological mapping. Since a topological map does not contain precise metric information and also to simplify the exercise to not using a dead-reckoning method, we will only focus on probabilistic sensor-based localization.

In a localization problem, we have to compute the probability $p(s_k|\Theta, z_k)$ where:

- s_k is the location of the robot at time k
- Θ is the given map
- z_k is the observation at time k

Note that we ignore the control input in this formulation since we are only concerned with sensor information. Using the Bayes rule, we have:

$$p(s_k|\Theta, z_k) = \frac{p(s_k|\Theta)\, p(z_k|s_k, \Theta)}{p(z_k|\Theta)}$$

The normalizing term $p(z_k|\Theta)$ does not depend on s_k, thus can be ignored. The term $p(s_k|\Theta)$ can be set to a constant value (uniform distribution without sensor information). Therefore, the robot location $p(s_k|\Theta, z_k)$ is proportional to the probability $p(z_k|s_k, \Theta)$. Intuitively, by using the sensor information (e.g. extracted features from the surroundings) the robot can perform a matching of places in a given map in order to determine its current location. In other words, if the places can be characterized by different distinctive patterns or fingerprints, the robot can determine which place it is currently in by extracting the fingerprint of the place.

2 Feature and Fingerprint Selection

In the first part of the exercise, we will use line segments extracted from range scans and openings (empty spaces, openings, discarded noisy points between segments) as the fingerprint elements. Figure 1 shows a sample of range scans taken inside a place (a room) and line segments extracted from the scan. In this exercise, lines are extracted using a simplified Split & Merge algorithm (that implements only the splitting process).

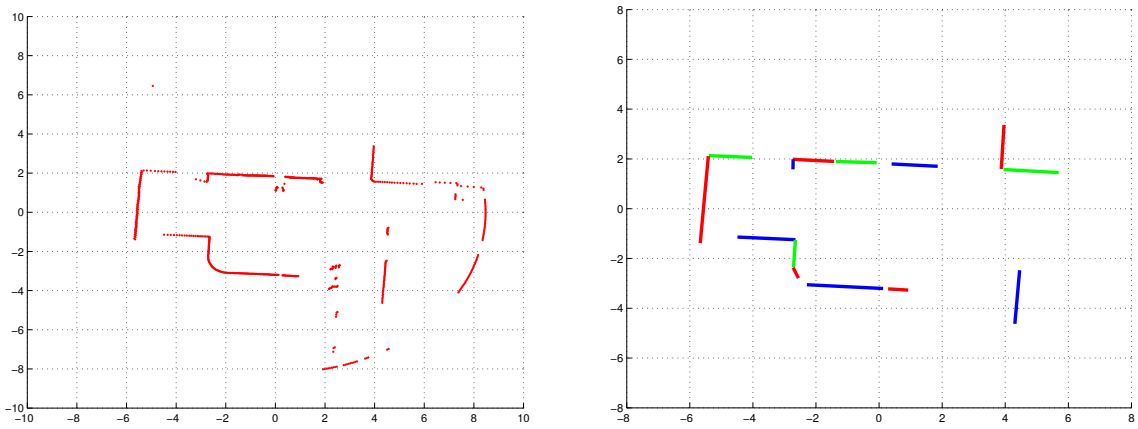

Figure 1: A scan and extracted line segments.

Now we have to characterize the place above by a pattern string or fingerprint. A simple way is to represent the features by the angles seen from the robot as shown in Figure 2. The angles are then discretized into units, e.g. one unit is say 20^o. For the place in our example, it can be represented by the pattern string "1010011101111111111" where '1' is for segments and '0' is for openings. One advantage of transforming feature angles to number of units (with rounding to nearest integer number) is that the extracted feature angles are not precise. They are affected by noise and might be slightly different between observations while the robot is standing. Using units instead of absolute values overcomes the imprecision. Another advantage is that we can use standard string comparison techniques to evaluate and score the similarity between two strings which leads us directly to the computation of the matching probability.

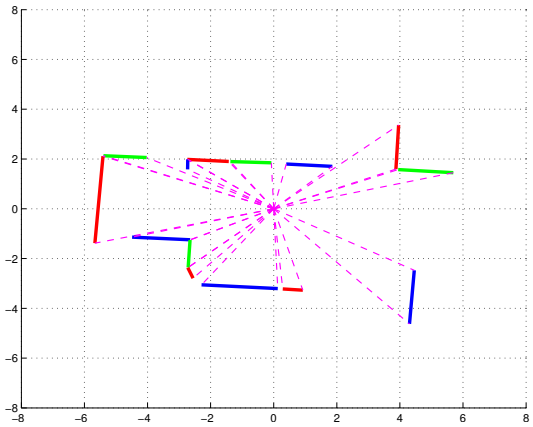

Figure 2: Angles of extracted segments and openings (spaces between segments) are used to generate the pattern string of a place.

In this exercise, we will use the Levenshtein distance for comparing pattern strings (http://en.wikipedia.org/wiki/Levenshtein_distance). The following definition and example are taken from the website. *"The Levenshtein distance between two strings is given by the minimum number of operations needed to transform one string into the other, where an operation is an insertion, deletion, or substitution of a single character."*. For example, the Levenshtein distance between "kitten" and "sitting" is 3, since these three edits change one into the other,

and there is no way to do it with fewer than three edits:

1. kitten → sitten (substitution of 'k' for 's')
2. sitten → sittin (substitution of 'e' for 'i')
3. sittin → sitting (insert 'g' at the end)

The similarity probability of 2 pattern strings can be calculated by the following equation:

$$Similarity\_Prob = \frac{String\_length - Levenshtein\_distance}{String\_length} \times 100\% \quad (1)$$

In the above example, the similarity probability of "kitten" to "sitting" is $(7-3)/7 = 57.1\%$. Notice thatwe will obtain different pattern strings from different robot orientations within the same place. Thus, the bits of a string should be rotated when comparing with another string. The desired minimal distance, which corresponds to the maximal similarity probability, can be then computed.

In the next part of the exercise, we will test the same approach presented above with color blobs as features. Color blobs can be extracted from an image by using simple color thresholding techniques. In order to separate the illumination component from the color, this exercise uses the HSV color space instead of the RGB one. Color thresholding can be done very simplistically using bounds on the colors that need to be detected and the application of some constraints. The constraints include thresholds bounding the size of the blobs detected and the spread of the blobs. Put simply, each pixel of the image is checked against the color thresholds that are predefined. The connected components in the resultant image are labeled, each label corresponding to a particular blob. After applying the constraints, the resulting blobs are deemed to have been identified.

Note that the method while simplistic, can be potentially unreliable in that it is quite sensitive to changes in lighting conditions. However, a simple method such as this one should suffice to demonstrate the concept. For the best result, it is recommended that the color to be identified be specifically chosen for the local conditions (the designed field with the local lighting levels etc.). Also, a well lighted environment is recommended; it is best to avoid shadows from the structural components, to the extent possible.

In the final part of the exercise, we will attempt to see if fusing color and line based fingerprints can result in a better localization performance. One way of doing this is to apply a simplistic Bayesian sensor fusion scheme - a weighted average of the the two estimates (from lines and color blobs) where the weights (priors) can be experimentally found (number of correct outcomes in a total number of tests, for instance). Another possibility is to fuse/concatenate the two fingerprints obtained into a single signature of the place and use this fused signature to perform topological localization.

3 Code Provided

The code provided in the exercise includes feature extraction (lines and color blobs), fingerprint generation (for lines or blobs) , fingerprint matching (the Levenshtein's distance measure). A simple testing module is also provided that can compute the "confusion matrix" for a given set of fingerprints. Code is also provided for calibrating the camera in order to estimate distance

measures from omnicam images. The reader is also recommended to check the links listed in section 6 for more information on every aspect of this exercise.

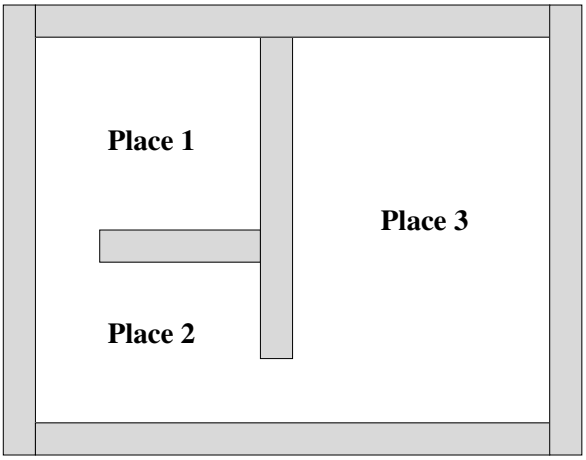

Figure 3: Testing environment. There are 3 places whose sizes (extracted segments), openings are different.

Note that the first steps that are required to be performed include

- Build a testing environment similar to the one shown in Figure 3.
- Calibrate the omnicamera by running the program ./scans/InitCamera.m.
- Use imtool command in MATLAB (or the color picker from an image manipulation program such as GIMP or XnView) to choose the colors that you want to identify

4 Objectives and Tasks

- *Test line fingerprints*: Use the GetLinePattern.m function to record a set of fingerprints(try changing the positions and angles of the robot). Try using the testPlaceLines.m function to see how distinctive the signatures are. Also, write a function CheckPattern.m to compute the probability of localization. You may want to write a few lines of code in order to test newly generated fingerprints against a previously stored database of fingerprints. This should give you a fair understanding of the exercise and how it can be implemented using line features.
- *Test color blob based fingerprints*: Like in the previous case, use GetBlobPattern.m to obtain a color blob based fingerprint. Use the other blob related functions (as in previous step) to assess for your self the reliability of color blobs (as implemented here) towards localization. Also, compute a probabilistic estimate of the topological localization.
- *Test the combination of lines and blobs*: As mentioned before, a "sensor fusion" approach can be adopted or otherwise a "multi-modal fingerprint" approach can be adopted. The latter approach could be done by either fusing or concatenating the individual fingerprints. Try each of them and test them against each other. Try placing random obstacles in the field and check the response of the system. In principle, would you expect the performance

of the multi-feature case to be better than the single feature case or would it be worse ? Why ?

5 Nice extensions that may be attempted

- *Topological Mapping using fingerprints* - Extend this code by incorporating a clustering module that can cluster various fingerprints to form topological "places" (a region of space such as a room or each enclosure of the field constructed for this work) that are connected to each other. You can also attempt to move the robot and perform some or all of the above processes in an online fashion.
- *Improved Features or distance measure* - One can see that using feature angles (angles seen from the robot) is not the best solution, since the angles are not invariant to the robot pose. As a result, at different positions within the same place, the extracted fingerprints are different (not considering obstacles). Can you suggest an improvement ? For example, considering segment lengths because they are invariant to robot pose. Replace the function ComputePatStringX.m by your method and repeat the process. How else would you propose to improve the approach presented ?

6 Useful Information

- How to build a Range Finder using an Omnidirectional Camera - http://www.asl.ethz.ch/education/master/mobile_robotics/Exercise_3.pdf
- Split & Merge method - http://asl.epfl.ch/aslInternalWeb/ASL/publications/uploadedFiles/nguyen_2005_a_comparison_of.pdf
- HSV color space (http://en.wikipedia.org/wiki/HSV_color_space)
- Levenshtein's distance (http://en.wikipedia.org/wiki/Levenshtein_distance)
- MATLAB - Matlab help is your best reference
- Mapping with Fingerprints (exercise) - http://asl.epfl.ch/member.php?SCIPER=156345
- Cognitive Maps (lecture) - https://www.asl.ethz.ch/people/vasudevs/personal