Data association algorithm for SLAM using bearing only sensor.

Gualandi - Nenci

February 11, 2013

Contents

Chapter 1

Introduction to the problem

SLAM stays for Simultaneous Localization And Mapping. It is a particular field of robotics/AI that aims to guess the actual state of the world, given some data acquired by the robot. Data could be robot poses, camera images associated to these poses, sonar acquisitions and many more. Needless to say, these data are usually affected by noise, meaning that solving SLAM problems usually involves a certain level of uncertainties.

1.1 Graph model

A quite suitable way to represent our knowledge about the world is using a graph. In such a graph (see figure 1.1):

- Nodes are either robot poses or landmarks.
- Edges are constraints between nodes.

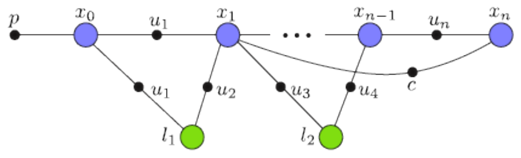

Figure 1.1: World representation graph. Blue nodes are poses, green ones are landmarks.

This way, you can imagine the graph as a lot of blocks (the nodes), with springs (the edges) that "pull and push" these blocks. If there were no errors, these springs wouldn't conflict each other. The presence of errors leads to the needing of finding a compromise state. The solution that minimizes the overall "conflict force" is the one that we choose as guess about the world. The search for such a state is called optimization.

1.2 Our case: bearing only sensor

In order to infer something about the world, the first step is to gather data, and to associate them in a proper way.

The algorithm for data association that we developed assumes that the robot is equipped with a bearing sensor. A bearing sensor is a sensor that detects an interesting point and returns, as information, the "direction" (actually an angle) where it is situated with respect to the robot. This sensor is of course an abstraction, but can be assimilated to real sensors in certain conditions.[1]

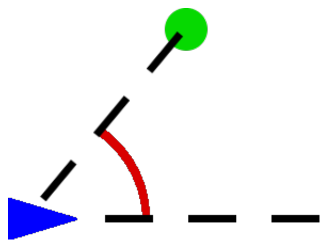

Figure 1.2: The sensor detects the red angle.

Since the only information we get about the landmarks is an angle with respect to a robot pose, introducing landmarks in the graph is not as trivial as it would be with other sensor types. This part will be examined in chapter 2.

Even after the association has been done, the graph needs a dedicated kind of edge that links a robot pose to a landmark via the direction in which the landmark is seen from the robot pose. This actually constrains the detected angle and the angle computed backprojecting the estimated position of the landmark.

[1]For instance an omnidirectional camera used for detecting coloured landmarks

Chapter 2

The algorithm

Chapter 3

Side projects

During the development of the main project, we also developed some "collateral" program. We put in this section all these secundary projects.

3.1 Simulator in Matlab

3.2 Simulator in C++

This is another simulator for the same kind of experiment. Again, the robot moves and, when the landmark is in range, the robot detects the relative bearing.

It has been developed in C++, and uses the SFML libraries.

This simulator is simpler than the previous one, to the cost of a coarser approach to errors. Noise is simply modeled as uniformly distributed between two values, while in the Matlab simulator a gaussian distribution was used (see section 3.1 for details).

The user can move the robot, resize the sensor range and add landmarks.

This program outputs two files: the first one contains the "real" odometry and measures, while the other one contains the "noised version" of the same data.

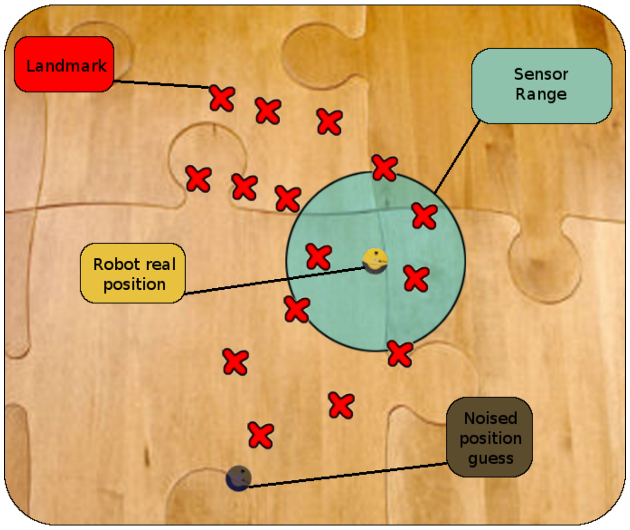

Figure 3.1: Cropped from a screenshot of the C++ simulator. Elements have been labeled.

3.3 Maps Merger

This is not properly a side project, since it is, has it is, a completely uncorrelated project. It has been placed altogheter with this project because, looking at the possible future developments, there will be the needing of merging two maps.

For our purposes, a map is a set of landmarks, where each landmark is defined by a couple of values (its **x** and **y** coordinates). The assumption made is that the two maps overlap in some part, and we want to find out which is the transformation needed to overlap them. Once this is found, a global map can be extracted from the two.

The program uses the RANSAC algorithm to find the "best" transformation. A model is identified from two couples of landmarks, each couple from one of the maps. In fact with the first correspondance we detect the translation, and with the second correspondance we detect the rotation. For each of these models we count the overlapping landmarks amongst the whole sets and give a "score" to the model. The chosen model will be the one with the highest score.

Another useful assumption is that the two maps have approximatively the same scale, so when we analyze a model, if the distance between the two landmarks in the first map is very different from the distance between the landmarks in the second map, we can skip that model without the need of additional examinations. This usually reduces the number of possible models by some orders of magnitude.

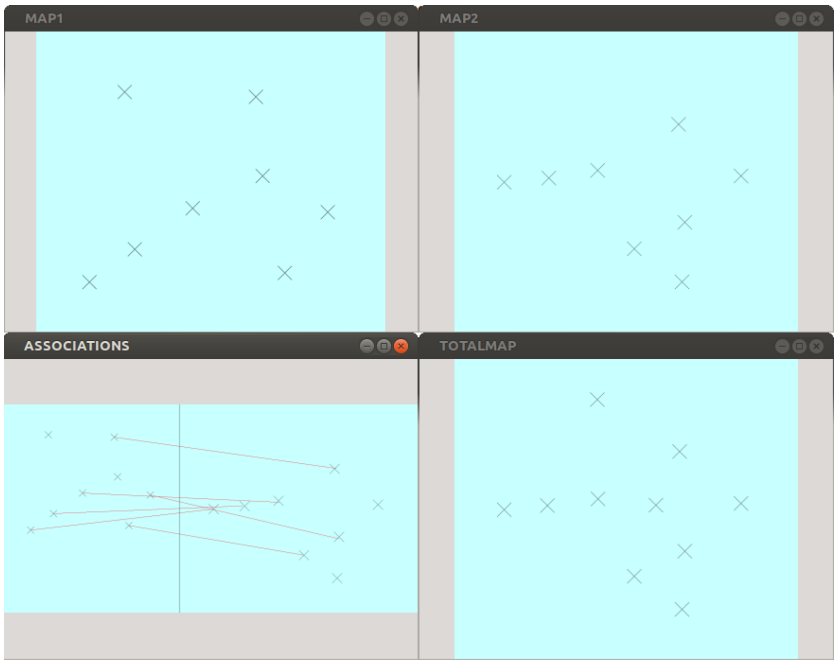

Figure 3.2: Screenshot of the MapsMerger program. In addition to rotation and translation, in the second map the landmarks have also been "moved" a little

Chapter 4

Conclusions