Data association algorithm for SLAM using bearing only sensor.

Gualandi - Nenci

February 12, 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.

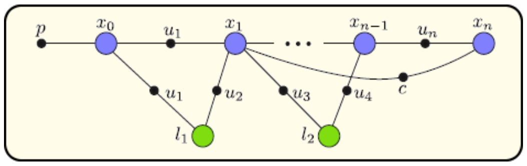

Figure 1.1: World representation graph. Blue nodes are poses, green ones are landmarks.

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]

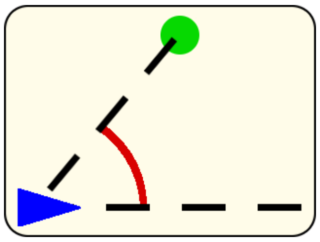

Figure 1.2: The sensor detects the red angle.

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.

In chapter 2 we discuss in details the algorithm we developed for this data association problem.

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

Chapter 2

The algorithm

We present here the algorithm we developed for data association with bearing only sensor.

The program takes as input a file that has a line for each step of the trajectory. Each line specifies: The rototranslation with respect to the previous step (x, y, theta) a list of the measurements acquired in this step.

The basic idea is that, at each step, the algorithm iterates over all the observations associated with the considered position and tries to associate them with observations from the previous step, determining if each observation should be **queued** to one of the previous tracks or it should be associated to a new landmark.

A landmark is considered **confirmed** when it has at least a certain number of associated observations.

The *track*, to which we will often refer in this chapter, is actually a landmark. When new observations are added to the track, the landmark position is re-estimated and the track is propagated to the next step. When, on the contrary, no new observations are associated to the track, it is not propagated, and the landmark itself is either pruned or kept (depending on the fact that it had been confirmed or not).

Going on with the trajectory, the algorithm creates a graph, putting in it new poses and landmarks.

Every time a certain number of steps has been performed, the graph is optimized according to the edges created up to now.

In section 2.1 we explain which problems we had to face, and how we dealed with them. After that, in section 2.2 there is a pseudo-code of the developed algorithm.

2.1 Issues

2.1.1 The main problem: vague observations

Using a bearing only sensor involves a big problem: an observation is not enough to estimate the position of a landmark. From an observation associated with a robot position we can only determine a line (actually an half line) where the landmark may be. This is a problem because next step observations can't rely on the estimated position of the landmark for determining correspondances.

The developed algorithm deals with this problem distinguishing two cases (see figure 2.1):

1. The landmark already has an estimated position.
2. A position has not been estimated for the landmark. (Usually happens when the landmark has been seen only once)

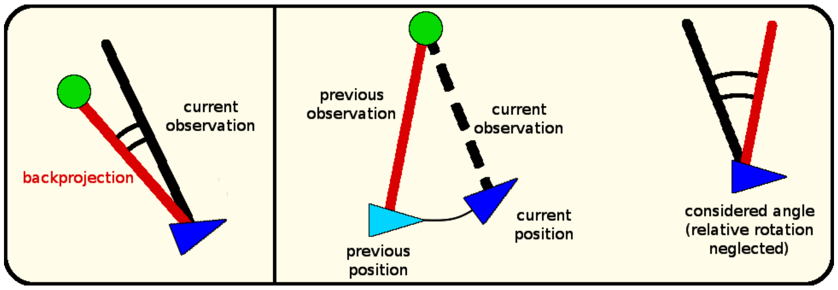

Figure 2.1: The two possible situations. The drawn angles are those to be evaluated.

In the first case, the estimated position is backprojected on the robot pose, and the resulting angle is compared to the current observations.

In the second case, an assumption is made: given that the captures are "close", the sensed bearing for the same landmark should have remained similar between two consecutive steps. The comparation is then done only with the previous angle (of course after considering positions relative rotation).

2.1.2 More than a close track

It can happen that the observation we are evaluating is close to two of the tracks. There is of course a track that is "closer" than the other, but they are both inside a certain threshold.[1] In this case, we simply ignore the new observation.

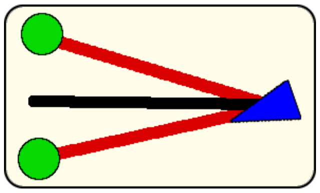
Figure 2.2: Two close observations. How to choose?

To understand the sense of this choice, it is the case to highlight the importance of avoiding wrong associations. Since we can only rely on the bearing for the landmarks position estimation, a wrong association can cause the graph to explode when optimizing. This is then wiser, where possible, to avoid every source of uncertainty. A pessimistic approach will possibly discard some good informations, but will be more reliable.

2.1.3 Two observations for the same track

In section 2.1.2 we have seen that an observation could be close to two tracks. We now face the dual problem: it can happen that two of the current observations are close to the same track.

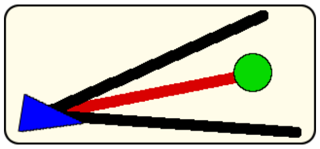
Figure 2.3: Two new observations are both close to the same track.

Just like before, we cannot risk to associate a landmark with wrong measurements. Moreover, it is obvious that a single landmark shouldn't generate two measurements. Following the idea that not knowing is better than thinking wrong, we don't add either of the two observation to the track. Anyway, the track is propagated to the next trajectory step.

[1]We actually defined a threshold for the closer and another threshold for the second closer.

2.2 Pseudo-code

In this section, a pseudo-code for the algorithm is given.

First of all, some pseudo-structures.

- RobotPosition: this represents a pose, and will contain:
  - Coordinates and orientation (x,y,theta).
  - A set of Observations.
- Observation: this represents a measurement. It will be defined by:
  - A reference RobotPosition.
  - A bearing value.
- Landmark: this represents a landmark, and is also used to propagate tracks. It will consist of:
  - Coordinates (x,y).
  - A set of references to Observations (all the observations that have been associated to this landmark)

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

This Matlab odometry simulator was realized in order to simulate the uncertainties on robot perceptions originated from physical process like wheels slippage, or sensor noise. The simulator provides a GUI that allows the user to navigate in a map filled with randomly-placed landmarks, and visualize in real-time both the real configurations sequence and the noised configurations sequence. Finally, a text file is produced, associating a sequence of robot states to the landmark perceptions from these states.

3.1.1 Representation of the configuration

Robot configuration C is rapresented by the mean of a 2D homogeneous matrix.

$$\mathbf{C}(k) = \left(\begin{array}{c:c} \mathbf{R}(k) & \mathbf{t}(k) \\ \hdashline 0 & 1 \end{array}\right)$$

with $\mathbf{t} = [x, y]^T$.

The matrix also rapresents univocally a certain state of the robot $[x, y, \theta]$. The robot model used is a planar omnidirectional robot, with the capability of changing its linear and velocity dependings on user inputs. The instantaneous linear velocity $v \in \mathbb{R}$ is intended in the direction of the x (sagittal) axis of the robot, while angular velocity $\theta \in \mathbb{R}$ is intended with respect the z axis (orthogonal to the workspace). Of course this values are intended to be deduced from the encorders on the wheels.

Given a certain configuration $\mathbf{C}$ and the two instantaneous velocities v and θ, the new configuration is computed as:

$$\mathbf{C}(k+1) = \left(\begin{array}{c|c} \mathbf{R}(k) \cdot \mathbf{H}_r & \mathbf{t}(k) + \tilde{\mathbf{t}} \\ \hline 0 & 1 \end{array}\right)$$

$$\tilde{\mathbf{t}} = [\tilde{t_x}, \tilde{t_y}]^T = \mathbf{R}(k) \cdot [v, 0]^T$$

$$\mathbf{H}_r = \begin{pmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{pmatrix}$$

3.1.2 Configurations storing

The *distance* d between two configuration matrices M_1, M_2 rapresenting respectively states $s1 = [x_1, y_1, \theta_1], \quad s2 = [x_2, y_2, \theta_2]$ is defined as

$$d = \alpha \cdot d_\theta + d_p$$

where d_θ is the absolute value of the the (circular) distance between angles θ_1,θ_2, and d_p is the norm of the vector $[x_1 - x_2, y_1 - y_2]$. The parameter α, usually > 1, can balance the importance of the rotation distance respect the linear one.
While user navigation proceeds the simulator stores in two separete list (real and noised) all the subsequent configurations $\mathbf{C}$. (IMAGE) In a certain time instant, both current noised and real configurations are appended to their lists if and only if the *distance* between the current (noised) configuration and the last saved (noised) configuration exceeds a certain threeshold.

3.1.3 Landmark locate

Every time that a configuration is appended to the list, is create a new landmark reading. A landmark reading is composed by two rows:

$$odomPose \quad <ID> \quad n_X \; n_Y \; n_\theta \; t_X \; t_Y \; t_\theta$$

$$bearing \;\; b_1 \;...\; b_n \; IDb_1 \;...\; IDb_n$$

where: $< ID >$ is the number of reading, $[n_X, n_Y, n_\theta]$ is the noised state of the robot, while $[t_X, t_Y, t_\theta]$ the real one. The n values $b_1...b_n$ gives the bearing angle of the n landmarks visible from $[t_X t_Y t_\theta]$. Each landmark is then associated to his univocal ID $< IDb_1 > ... < IDb_n >$.

3.1.4 Odometry and sensors noise

The error on the odometry is modeled as a zero-centered white noise with gaussian distribution, that influences independently the two velocities v and θ. The standard deviation of the distribution has a constant component and an additional component that is directly proportional to the norm of velocity v or θ. In this way is modelled the fact that when velocity increase we tends to have a higher noise component.

Note that the Odometry error is added while navigating, and influences the integration done on the robot. Conversely the error on sensor is added off-line, and it is indipendent

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 measurements, while the other one contains the “noised version” of the same data.

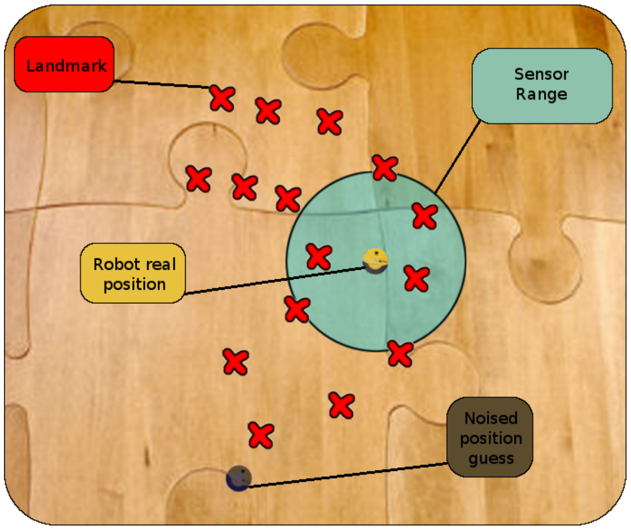

Figure 3.1: Part of a screenshot of the C++ simulator. Elements have been labeled.

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.

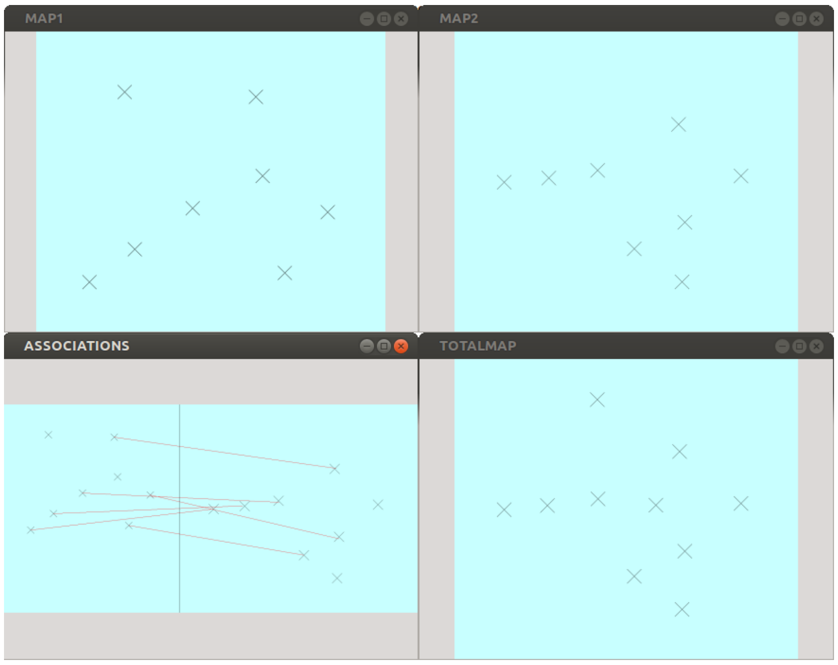

Figure 3.2: Screenshot of the MapsMerger program. In addition to rotation and translation, in the second map the landmarks have also been "moved" a little

Chapter 4

Conclusions