Neural Networks:
Project Report.

Irvin Aloise, mat. 1392066
Mirco Colosi, mat. xxxxxxx

November 29, 2016

1 Introduction

Simultaneous Localization and Mapping is one of the main tools used in mobile robotics. Solving this computational problem, agents are able to build a map of an unknown environment and to locate themselves in this environment. In general, agents that performs SLAM cannot rely on powerful sensors like GPS, but they have to solve this problem using indoor-friendly sensors, e.g. cameras, gyroscope, laser-scanners.

In this project, it has been modified a SLAM system based on a **single RGBD camera**. In particular, it was initially designed to use only image features to solve the problem and build the environment map and it has been upgraded embedding feature points' normals in the optimization process. In this way, the system became more robust with no loss in terms of computational speed.

In the next Session it will be briefly explained the concepts behind this approach and some results will be presented.

2 System overview

The RGB-D system in analysis has been built using ROS and OpenCV and it has three main components:

- **Camera tracker**: allows to record the camera motion inside the world.
- **Frame manager**: manipulates the map triggering loop closures.
- **Solver**: runs the well known least-square optimization algorithm.

Before going deeper in the analysis, it is good to underline how an RGB-D camera works. The one used here is an *ASUS Xtion*, connected to the PC through a ROS node. It has a standard RGB camera which captures 320×240 coloured images and a structured light camera that is able to gather depth information in indoor environment; this sensor is able to run at $30Hz$.

Frame Manager was based only on features correspondences and it had to go through the following core steps:

1. **Detection**: interesting points (e.g. corners) are discovered on the RGB images based on the Harris algorithm.
2. **Extraction**: interesting points are then used by a *Brief Descriptor* in order to characterize those corners and make them actual *features*.
3. **Matching**: features are compared to previous frame's ones and a *brute-force matcher* creates *point correspondences*.

Those *image points* are in (u, v, d) coordinates in the **image frame**, where the d component is retrieved using the depth map gathered directly using the sensor.

Moreover, it is possible to retrieve also the *camera points*, which are expressed in the **world frame**, e.g. (x, y, z). This points are calculated through a transformation $T_{camera \to world}$.

Given this really simple overview, it is possible to define the least-square optimization problem: its target is to estimate the best $T_{camera \to world}$, using the re-projection error between image points and world points. Obviously the camera matrix and all its parameters are already known.

As it has been said before, the *solver* will perform all the computations needed. Originally, it had to evaluate the following entities:

1. **Error** between image points and projected world points. It is a (3×1) vector defined as follows:

$$\mathbf{e}_k = K \cdot (T \cdot \mathbf{p}_{world}) - \mathbf{p}_{image} \tag{1}$$

   where K represents the camera matrix.

2. **Jacobian** is a (3×6) matrix computed using the chain rule as follows:

$$J_k = J_p K J_t \tag{2}$$

   where

$$J_p = \begin{pmatrix} 1/z & 0 & -x/z \\ 0 & 1/z & -x/z \\ 0 & 0 & 1 \end{pmatrix} \tag{3}$$

$$J_t = \left[I | 2 \cdot skew(T \cdot \mathbf{p}_{wolrd}) \right] \tag{4}$$

After those initial evaluation, it is possible to compute the H_k matrix and the $\mathbf{b}_k$ vector for the current point correspondence as follows

$$H_k = J_k^T I J_k \tag{5}$$

$$\mathbf{b}_k = J_k^T I \mathbf{e}_k \tag{6}$$

Obviously, those must be evaluated for each point correspondence in order to retrieve the increment ΔT, in formulae:

$$H = \sum_k^K H_k \tag{7}$$

$$\mathbf{b} = \sum_k^K \mathbf{b}_k \tag{8}$$

$$\Delta T = \frac{\mathbf{b}}{H} \tag{9}$$

Starting from this point, now it will be presented the upgrade brought by the project in analysis.

As it has been anticipated before, the main idea is to embed information about feature points' **normals** into the solver. They are estimated computing **eigenvectors** of the covariance matrix $\mathcal{C}_p$ of each feature point. Namely, given $\mathcal{C}_p$, the *eigenvector corresponding to the smallest eigenvalue* represents the best estimation of the point's normal.

After this computation, both *error* and *Jacobian* had been modified. In fact, now the **error** is a (6×1) vector built as follows:

$$\mathbf{e}_k^{new} = \begin{pmatrix} \mathbf{e}_k^{old} \\ \hline \mathbf{e}_k^{norm} \end{pmatrix} \tag{10}$$

where

$$\mathbf{e}_k^{old} = K \cdot (T \cdot \mathbf{p}_{world}) - \mathbf{p}_{image} \tag{11}$$

$$\mathbf{e}_k^{norm} = (R\mathbf{n}) - \mathbf{n} \tag{12}$$

In formula 12, matrix R represents the rotational part of matrix T, since normals are direction and the translational part of T does not affect them.

The **Jacobian** is now a (6×6) matrix, composed as follows:

$$J_k^{new} = \begin{pmatrix} J_k^{old} \\ \hline J_k^{norm} \end{pmatrix} \tag{13}$$

where

$$J_k^{old} = J_p K J_t \tag{14}$$

$$J_k^{norm} = [0_{(3\times3)} | 2 \cdot skew(R\mathbf{n})] \tag{15}$$

where matrices J_p and J_t are already specified in equations 3 and 4.

Once retrieved those new quantities, the remaining part have not been modified (e.g. matrix H and $\mathbf{b}$ vector).

In figure 1 is shown the output of the updated system using a real dataset. It is possible to appreciate the fact that floor and walls are better shaped: only in low light conditions some error occurs (e.g. some "holes" in the floor).

Obviously this method can be further improved in many aspects and it is not a state-of-art SLAM system. For example it is possible to deeply embed normals in the algorithm to reduce the influence that light conditions have on performances, or to take advantage of multithreading technique to speed-up the whole machine.