Lorenzi Flavio

June 2020 Mat. 1662963

Least squares for SLAM; Bartolomeo Della Corte

Report and deep analysis on this seminar topic

Introduction and fundamentals

Nowadays thanks to the advancement of scientific research there are a lot of cutting-edge mobile robots, which have now become part of our life, not only in the experimental field (e.g. Roomba), reaching a certain movement autonomy. In this area, the aim of engineers is to constantly try to obtain a robust system, reaching optimality in three main fields: calibration, SLAM and navigation. So we will see a Least Square solver method which leads to the resolution of all these problems.

This important Robotics area is obviously based on probability theory and the basic aim is to solve a classic estimation problem: given a stationary system with non observable state variable x and measurement z, the state is distribuited according to a prior distribution P(x) while the observation model P(z|x) is known. So the goal is to find the most likely distribution of states given a certain mesurement P(x|z).

Considering a mobile platform equipped with N sensors the goal of **Calibration** is to estimate simultaneously: the kinematic parameters of the platform, extrinsics parameters of the sensors and time offset of each sensor. So it is very important to avoid some errors (for example a wheel bigger than the other) that can bring the robot out of the way and break its *robustness*. All this defines a data synchronization problem that can be solved in several ways. To calibrate a mobile robot (in a very low level) for example we could mesure on the ground distances of a square and make it follows a squared path in order to read odometry and compare it with ground thruth. In this work instead is shown how this can be done in a higher level, by tuning an *unsupervised* type of Calibration. It is realized by analyzing the outcome of calibration, using the Hessain Matrix of the Least Square solver, in particular by following an exploitation-exploration approach, with the goal to find the most promising H matrix [1].

Once we have the extrinsic parameters of platform calibrated, we can start to map the environment with **SLAM**, which stands for Simultaneous Localization And Mapping [2]; it is the computational problem of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it. Currently, there are many open-source systems that are able to deliver fast and accurate estimation in typical real-world scenarios. Still, all these systems often provide an ad-hoc implementation that entailed to predefined sensor configurations.

Indeed a different (“*not common*”) approach such as that of **Least Squares** is very useful here.

This method is (in general) a form of mathematical regression analysis used to determine the line of best fit for a set of data (machine learning theory) providing a visual demonstration of the relationship between the data points. This is done by minimizing the sum of the offsets of points from the plotted curve. Each point of data represents the relationship between a known independent variable and an unknown dependent variable (so it is used to predict the behavior of dependent variables) [5] .

Least Squares for SLAM: MPR and MPC

Nowdays there are a lot of approaches (point, plane, line based…) for mapping the environment, once we have well-tuned the calibration of the robot, but as we said in this work we want to break schemes and see something new. We will see how Least Square method is applied and its results.

Iterative Least-Squares (ILS) solvers are core building blocks of many robotic applications [3], systems. This technique has been traditionally used for calibration, registration and global optimization. In particular, modern SLAM systems typically employ multiple ILS solvers at different levels: in computing the incremental ego motion of the sensor, in refining the localization of a robot upon loop closure and, most notably, to obtain a globally **consistent map**. Similarly, in several computer vision systems, ILS is used to compute/refine camera parameters, estimating the structure of a scene, the position of the camera or both.

A very interesting algorithm that uses ILS is the **MPR** that aims to register data from robot sensors as RGB cameras, and so on… with a unified and flexible framework. This is achieved by exploiting the chain rule for Jacobian, by mixing numerical and analytical Jacobians. As a result, MPR is able to operate on depth images capturing different cues and obtained with arbitrary projection functions [1]. To operate on-line, most of all the registration works rely on ad-hoc dense and non-stationary ILS solvers that leverage on the specific problem's structure to reduce the computation. The used strategy here for registering point clouds is the **Iterative Closest Point** (ICP): thanks to that, it was possible to estimate the robot pose that better explain the point measurements. A non-stationary aspect arises from the heuristic used to estimate the data association, based on the current pose estimate. So according to the type of sensor used by the robot in question, different types of point cloud mappings can be made [Fig. 1]; this particular type of measurement (*point based*) is really excellent when compared with the state of the art results.

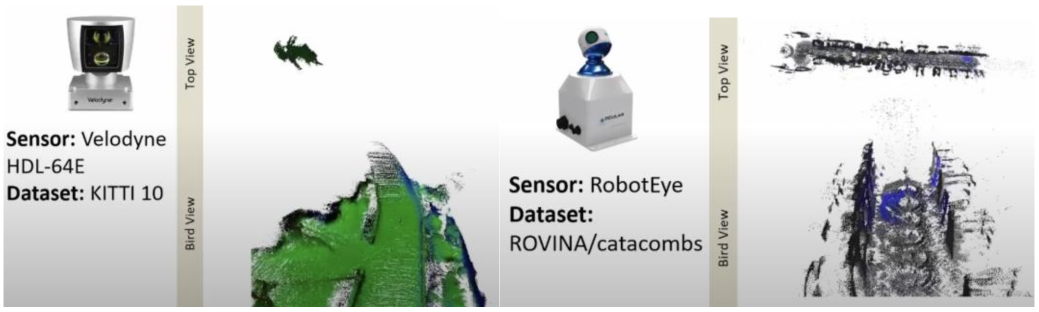


Figure 1

So the MPR can be seen as a flexible approach that is able to work with different sensors: it does not need of data association nor specific sensor adaptation; so in the same code takes all this into account with multiple cues [Fig 2]. Going into a little detail it seeks to minimize the pixel-wise difference between the current image J and the predicted one $\hat{J}$(M,X).

So it is realized by exploiting a photometric error function as following:

$$\mathbf{X}^* = \underset{\mathbf{X}}{\operatorname{argmin}} \sum_{u,v,c} \| \underbrace{\hat{\mathfrak{I}}^{c}_{u,v}(\mathcal{M},\mathbf{X}) - \mathfrak{I}^{c}_{u,v}}_{\mathbf{e}^{c}_{u,v}(\mathcal{M},\mathbf{X})} \|^2_{\mathbf{\Omega}^{c}}$$

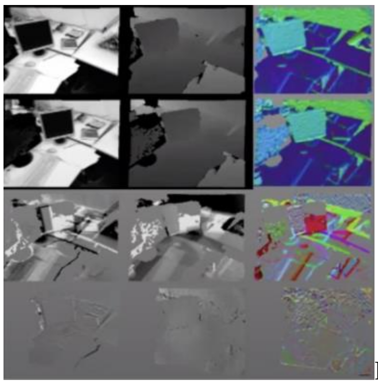
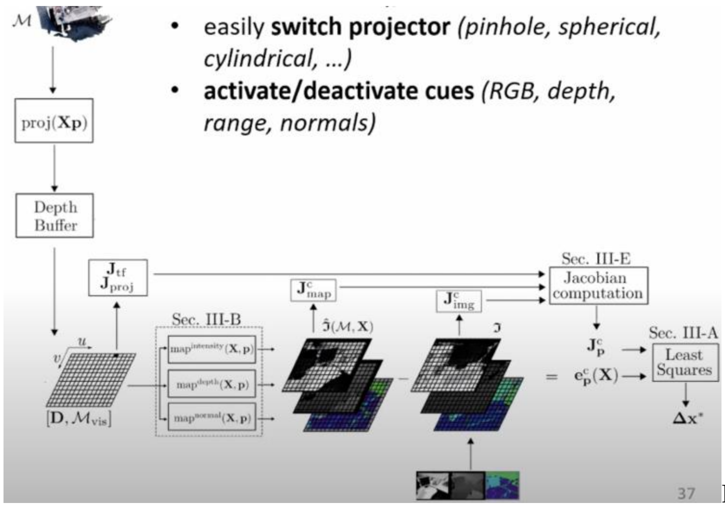


Figure 2



- easily **switch projector** *(pinhole, spherical, cylindrical, ...)*
- **activate/deactivate cues** *(RGB, depth, range, normals)*

$\mathcal{M}$
proj(Xp)
Depth Buffer
$\mathbf{J}_{\mathrm{tf}}$ $\mathbf{J}_{\mathrm{proj}}$
u
v
$[\mathbf{D}, \mathcal{M}_{\mathrm{vis}}]$
Sec. III-B
map$^{\mathrm{intensity}}(\mathbf{X},\mathbf{p})$
map$^{\mathrm{depth}}(\mathbf{X},\mathbf{p})$
map$^{\mathrm{normal}}(\mathbf{X},\mathbf{p})$
$\mathbf{J}^{c}_{\mathrm{map}}$
$\hat{\mathfrak{I}}(\mathcal{M},\mathbf{X})$
$\mathbf{J}^{c}_{\mathrm{img}}$
$\mathfrak{I}$
Sec. III-E
Jacobian computation
$\mathbf{J}^{c}_{\mathbf{p}}$
$= \mathbf{e}^{c}_{\mathbf{p}}(\mathbf{X})$
Sec. III-A
Least Squares
$\Delta\mathbf{x}^*$
37
Figure 3

Note: Figure 3 is the general scheme of MPR mechanism and operation.

Depending on the projection function we can have different models projected: so given a 3D projected model (M), this was processed to obtain $\hat{J}$ and the final error was computed with respect to the current image [Fig.3]. The least squares method is applied here taking the error from the photometric function e_p and the Jacobian part J_p, by computing the min distance and returning Δx.

As we already said an high modularity is ensured: we can immagine that if we change sensor, for example moving from RGB camera to a 3D slider camera, we obviously change the projection function, so instead of the pinhole projection we take a spherical one in according with its intrinsic properties; it turns out that we can easily switch sensors only by changing each Jacobian block.

Once we have our representation and a robot that is perfect calibrated, we can start think how to move the robot and generate *plans* and *trajectories*. Nowdays the most prominent approach to fast replan trajectory or in general handle dynamic objects is working with **MPC** (model predictive controller). Unfortunately these techniques has a very big problem: adding additional contraints (as obstacle avoidance, impose certain speed…) and/or larger horizons usually lead to lose the real-time performance. So a very good solution is proposed [1], that jointly reduces the dimensionality problem by using a smaller number of variables (via flat output) and exploits a dynamic lattice to obtain a finer grid in the initial part of the trajectory.

In particular to achieve this solution the attention is restricted to a non-linear differential flat system using flat outputs [4], that allow a substantial dimensionality reduction:

$$\mathcal{C}(\zeta) = \hat{\mathcal{C}}_0(\zeta) + \frac{1}{2}\sum_{k=1}^{N-1}\left(\|\nu_k(\zeta)\|^2_Q + \|\phi_k(\zeta)\|^2_R + \|\gamma_k(\zeta)\|^2_{A_l}\right)$$

From here on, a Least Square solver is used to minimize the cost function, using the f.o. as *state* (composed by all the poses of the trajectories and the input to get to that poses).

$$\zeta = h(x, u, \dot{u}, \cdots, u^{(r)}) \qquad\qquad \zeta \in \mathbb{R}^m$$

In the end to improve the quality of the trajectory [Fig. 4], without losing too much in terms of real time, could be applied a **Time Mesh Refinement strategy** that 1) Scale the OCP problem; 2) Perform a discretization error computation of first N states; if the error is high, use the Cubic-hermite to interpolation; 3) Call Least Square Solver with high order approximation [1][3][6].

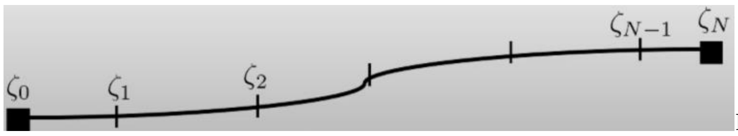


Figure 4

Conclusion

In conclusion we can say that today there are many techniques aimed at obtaining a balanced mobile robot, which is efficient and programmable in real time. Unfortunately it is not always possible to use predefined techniques for each robot, therefore in this field it is excellent to have and/or develop more solutions to achieve a certain robustness. So we have seen the importance of the modularity in this important Robotic field, and how using Least Squares method and its variants it is possible to implement techniques aimed at Calibration and SLAM, obtaining very good results in autonomous navigation.

References

- (1) Seminars in Artificial Intelligence and Robotics, Least Squares for SLAM, La Sapienza, April the 7th seminars, YouTube channel: https://www.youtube.com/watch?v=kwV4gBpd1xk
- (2) Plug-and-Play SLAM: A Unified SLAM Architecture for Modularity and Ease of Use; Mirco Colosi, Irvin Aloise, Bartolomeo Della Corte, Giorgio Grisetti
- (3) Least Square Optimization: from Theory to Practice; Mirco Colosi, Irvin Aloise, Bartolomeo Della Corte, Giorgio Grisetti
- (4) Robotics: Modelling, Planning and Control; Siciliano, Sciavicco, Villani, Oriolo, McGraw-Hill, Third Edition
- (5) The method of Least Squares, Steven J. Miller, Brown University, Math department
- (6) A finite Element Mesh Refinement, Multiphysics Ciclopedia