Planar Monocular SLAM

Cecilia Aponte

Sapienza Universita di Roma

Master of Artificial Intelligence and Robotics

Probabilisitic Robotics

January 28, 2020

1 Introduction

In order to move a robot, it has to be able to localize itself in the environment. To do this, it has to also understand the map of the environment in order to move about it. The use of GPS is one of those methods, however when a robot is located indoors or in a remote location this is not possible. Therefore, other type of data about the robot are gathered and synthesized to localize the robot and map the environment.

Even when these measurement data is gathered, there are disturbances that affect the system and therefore predictions that differ from the observed one. In order to provide a better solution, probabilistic inference is used that is picked from a set of potential solutions that are more likely than others. These are given by a model that describes the previous states and distribution of possible observations given the current states.

With the use of the Gaussian Filtering Tool, minimum error of the appearance Euclidean distance for Data Association, and Sparse Least Squares Maximum Likelihood Estimation, the Simultaneous Localization and Mapping (SLAM) is solved for a vacuum cleaner robot simulation.

2 Problem

In order to solve this SLAM problem using all the above tools, measurement data are necessary. For the estimate of the robot motion, odometry measurements at each time step are retrieved. To recover from noise errors given by this

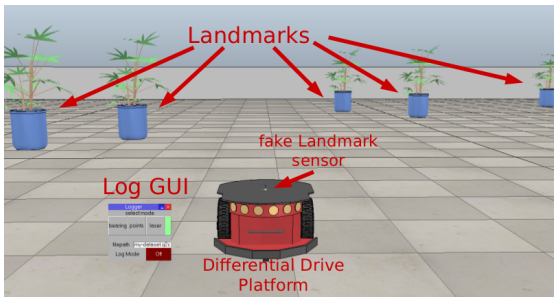

Figure 1: Robot and Environment setup scene where the robot has to localize and map (SLAM).

odometry estimate, external references (observations) are needed. The observation is captured by a monocular camera that retrieves the location of landmarks in the image and the appearance of the landmark itself in order to identify it. A sample environment of the vacuum cleaner robot is seen in Figure 1.

The camera information is given to analyze the gathered data. Camera matrix describes the describes the mapping of a pinhole camera from 3D points in the world to 2D points in an image. It contains the focal length and center location of the image in camera pixels. Camera transform is used to change robot pose from camera coordinate to robot coordinate for both rotation and position.

3 Model

The model used to solve the SLAM problem is Graph-based using Least Sqaures. The graph is constructed with both the robot poses and landmarks seen by it. It's connected with each consecutive pose (a node in the graph) as a constraint (an edge between two node poses in the graph). As well as creating a constraint when observing previously seen areas.

The graph is built with the information acquired by the sensors and then the configuration of the nodes are solved to minimize the error given by the constraints. Therefore to build this system two main sections are used for the graph: construction and optimization; and supporting sections that are necessary: landmark association and triangulation.

3.1 Graph Construction

The input of the construction of the graph is the raw data and the output is the graph containing the nodes and edges between pose and landmark constraints.

The state of the robot given by the odometry creates the node of the robot pose in the graph as seen in Figure 2. When two consecutive poses are acquired, the trajectory of the robot about the second robot pose seen from the first robot pose is computed and becomes the edge constraint in the graph. These measurements are affected by noise, therefore, the information matrix Ω_{ij} encodes this uncertainty.

Into the graph, the landmark information is also added to generate the map of the area the robot moves about. The landmark location is represented as a node and the observations of the camera images for each identified landmark is entered in the graph as edge constraints. This is done for every robot pose, creating constraints of re-observed areas as 'virtual measurements'.

Optimization then minimizes the error of landmark locations and robot poses.

3.2 Graph Optimization

The goal to solve the SLAM problem is to do a least squares error minimization. The optimized state $\hat{x}$ would be:

$$\hat{x} = \underset{x}{\operatorname{argmin}} \sum_{ij} e_{ij}^{T}(x_i, x_j)\Omega_{ij}e_{ij}(x_i, x_j) \tag{1}$$

The information matrix Ω_k is as explained earlier; higher values for Ω_k defines a higher importance edge during optimization.

3.2.1 Calculating the Errors

The error function $e_{ij}(x_i, x_j)$ is defined by the real measurements z_k (odometry or image location) and the predicted location of the robot or landmark $f_k(x) = \hat{z_k}$:

$$e_k = z_{ij} - \hat{z_{ij}} \tag{2}$$

The state is optimized for both the landmark and robot poses, therefore the error is calculated separately and optimized together. The error for the robot poses is:

$$e_{ij}(x_i, x_j) = t2v(z_{ij}^{-1}(x_i^{-1}x_j)) \tag{3}$$

The error for the landmark position is defined by the predicted location of the landmark in u,v coordinate in the image and the measured u,v:

$$e_k = \hat{z}_{uv} - z_{uv} \tag{4}$$

where $\hat{z}_{uv}$ is calculated using the triangulated location of each landmark relative to the camera pose. Transforming this location to camera coordinate $(x, y, z)_{camera}$ and then image coordinate in 2D $(u, v)_{image}$

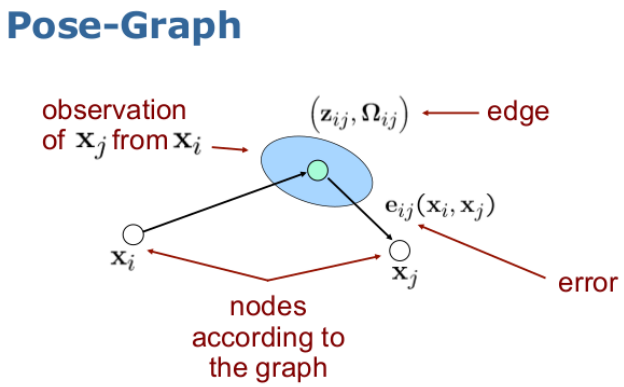

Figure 2: Diagram showing the graph creation using two consecutive robot poses and its error.

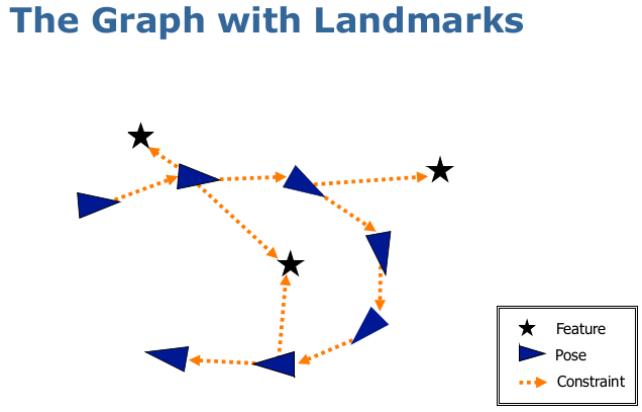

Figure 3: Diagram showing the graph creation using robot poses and landmarks.

$$\hat{z}_{uv} = \frac{\hat{z}_{xy}}{\hat{z}_z} \tag{5}$$

$$\hat{z}_{xyz} = K \cdot \hat{z}_{cam} \tag{6}$$

$$\hat{z}_{cam} = R_{cam}^{-1}(t_{land} - t_{cam}) \tag{7}$$

$$R_{cam} = [H_{rob} \cdot T]_{rotation} \tag{8}$$

where, T is the camera transform and K is the camera matrix.

The point projected of the landmark u,v is tested to see if its located within the range of the camera to capture it (height, width, near, far, etc.). If it is located within, the point error is sent to the system to be part of the optimization. Otherwise, it's discarded.

3.2.2 Calculating the Jacobians

In order to calculate the squared error, Jacobians J_{ij} are included to describe the change of the error function given the current state. THe Jacobian will be zero everywhere, except in the columns of x_i and x_j as seen in Figure 4.

$$\mathbf{J}_{ij} = \begin{pmatrix} \mathbf{0}\cdots\mathbf{0} & \underbrace{\frac{\partial \mathbf{e}(\mathbf{x}_i)}{\partial \mathbf{x}_i}}_{\mathbf{A}_{ij}} & \mathbf{0}\cdots\mathbf{0} & \underbrace{\frac{\partial \mathbf{e}(\mathbf{x}_j)}{\partial \mathbf{x}_j}}_{\mathbf{B}_{ij}} & \mathbf{0}\cdots\mathbf{0} \end{pmatrix}$$

Figure 4: Diagram showing the Jacobian matrix where nonzero elements would be located.

3.2.3 Optimization

With the error and the Jacobians for each element in the state (robot pose and landmark position), the linear matrices can the computed. The coefficient matrix H and vector b^T are described as follows:

$$H = \sum_{ij} H_{ij} = \sum_{ij} J_{ij}^T \Omega J_{ij} \tag{9}$$

$$b^T = \sum_{ij} b_{ij}^T = \sum_{ij} e_{ij}^T \Omega_{ij} J_{ij} \tag{10}$$

For each element, H is non-zero only on the main diagonal at x_i, x_j, and blocks ij and ji. While, b^T is non-zero only at x_i and x_j. The sparse system

makes it costly to solve, so a more efficient system would iteratively accumulating the contributions before solving.

The global coefficients become all the added coefficients for robot poses and landmark positions. The global coefficients are used to solve the linear system:

$$H\Delta x = -b \tag{11}$$

solving it for the increment Δx^* until the system converges:

$$\Delta x^* = -H^{-1}b \tag{12}$$

and updating the state as:

$$x \leftarrow x + \Delta x^* \tag{13}$$

3.3 Landmark Association

The landmarks that are seen by the robot as it moves need to be added to the state. Initially to do so, each landmark needs to be associated to an index so that it can be used throughout the system.

Each landmark seen contains 10 float numbers that describe its appearance. This is done previously using Hamming Distance embedding Binary Search Tree (HBST) for Descriptor Matching and Image Retrieval in Visual Place Recognition. With these 10 numbers as descriptors for each landmark seen, the Euclidean distance error of each is computed and minimized to get the closest descriptor approximation to define each landmark index.

3.4 Landmark Triangulation

These landmarks that already contain their index now need to be triangulated to acquire the location of each and add them into the state. Since only the u,v location in the image is given as information for each landmark location, a Delayed Inverse-Depth Initialization Method is used as seen in Figure 5.

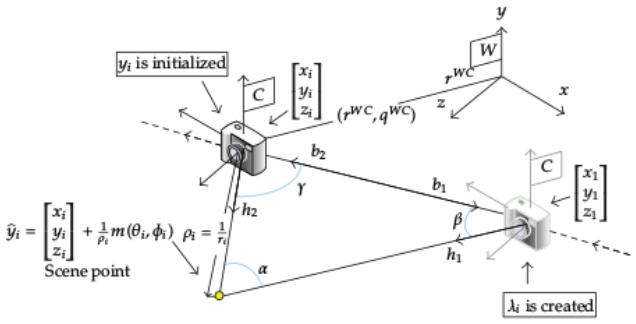

Figure 5: Diagram showing the setup for Inverse-Depth Landmark Initialization.

When only a single camera is used, the depth information cannot be obtained with just one measurement. Therefore, when a landmark is measured there is a need to wait until one or more measurements are obtained at the subsequent robot poses. This will give the system to infer the depth of the landmark as it observes the landmark move through the environment about the camera position. Doing so by estimating the angles from the landmark to the camera position, and computing the difference between those angles (parallax).

Landmarks that are closest produce a higher parallax than those farther. Therefore to increase the parallax, the robot has to keep moving in the hope that it will get closer to the landmark and have a parallax of at least 5 degrees in order to be added to the state. Parallax α is calculated using angles β and γ as follows:

$$\alpha = \pi - (\beta + \gamma) \tag{14}$$

$$\beta = cos^{-1} \frac{h_1 \cdot b_1}{\|h_1\| \|b_1\|} \qquad \gamma = cos^{-1} \frac{h_2 \cdot b_2}{\|h_2\| \|b_2\|} \tag{15}$$

where $h_{1,2}$ corresponds to directional ray vector computed using camera position and observed point observed at the first or second time; b_1 is the vector representing the camera base-line between camera optical center position where the point is observed the first time and second time; and b_2 is the opposite direction as b_1.

Once the parallax has reached a minimum 5 degree value, the next step is to calculate the inverse depth between the feature and the robot by:

$$\rho = \frac{sin(\alpha)}{b \cdot sin(\beta)} \tag{16}$$

Followed by computation of the angles of azimuth θ and elevation ϕ with respect to the world reference:

$$\theta = atan2\left(\frac{h_{2,z}}{h_{2,y}}\right) \qquad \phi = atan2\left(\frac{\sqrt{h_{2,y}^2 + h_{2,z}^2}}{h_{2,x}}\right) \tag{17}$$

To be used to calculate the directional vector of where the landmark is located relative to the second time the robot is seen:

$$m(\theta, \phi) = \begin{bmatrix} cos(\phi) \\ cos(\theta) \cdot sin(\phi) \\ sin(\theta) \cdot sin(\phi) \end{bmatrix} \tag{18}$$

Finally to calculate the location of the landmark:

$$t_{land} = t_{cam,2} + \frac{m(\theta, \phi)}{\rho} \tag{19}$$

This landmark location is added into the state space and used to solve the SLAM problem.

3.5 Parameters

The parameters needed to run the system are as follows:

1. Information Matrix (Ω): A diagonal matrix (inverse of the covariance matrix Σ^{-1}) that encodes the uncertainty due to noise of the pose and projection measurements. A smaller value corresponds to weaker edges and less it matters during optimization. Ω_{proj} for the image landmark projections is set to $I_2 \cdot 1e^{-4}$, while Ω_{pose} for the poses is set to $I_{12} \cdot 1e^{-1}$. Where I is the identity matrix and the subscript its size.

2. Kernel (ϵ): To establish a threshold of outliers that could exist during optimization for both poses and landmarks. This will decrease the contribution of measurements that have higher error and could result in performance loss. The kernel threshold for landmark projections is 50, and for poses is 3.

3. Damping (λ): To regulate the convergence. The higher is the damping factor, the smaller are the increments. $\lambda = 1e^{-2}$.

After many tests done with multiple different combinations of the parameters, the final parameters chosen are as seen in Table 1.

Variables	Trained Quantities
Ω_{proj}	$I_2 \cdot 1e^{-3}$
Ω_{pose}	$I_{12} \cdot 1e^{-1}$
Damping	$1e^{-4}$
$Kernel_{proj}$	50
$Kernel_{pose}$	3

Table 1: Parameters chosen for both algorithms tested.

4 Results

Two algorithms were implemented to test and compare the results. To understand the graph results, refer to the key in Table 2. The results are as follows:

Key for Result Graphs	
Item	**Meaning**
Yellow dots	Measured odometry values of the robot postion
Blue dots	System's calculated values of the robot position
Green dots	Correct values of the robot position (might not be visible since the blue dots might be on top, meaning the guess is as expected or with minimal error)
Red dots with numbers	Correct landmark position and its number is the index assigned to it, in order to corroborate each properly
Purple dots with numbers	System's calculated landmark position and its number is the index assigned to it, in order to corroborate each properly

Table 2: Key to be used to understand the graphs showing the results of the system performance.

One-step SLAM, where the system acquires the data once, then calculates the landmark triangulation, and lastly optimized. Therefore, the system solves the SLAM problem in one step, after all data from all timesteps are gathered.

The final complete graph of this system is shown in Figure 6.

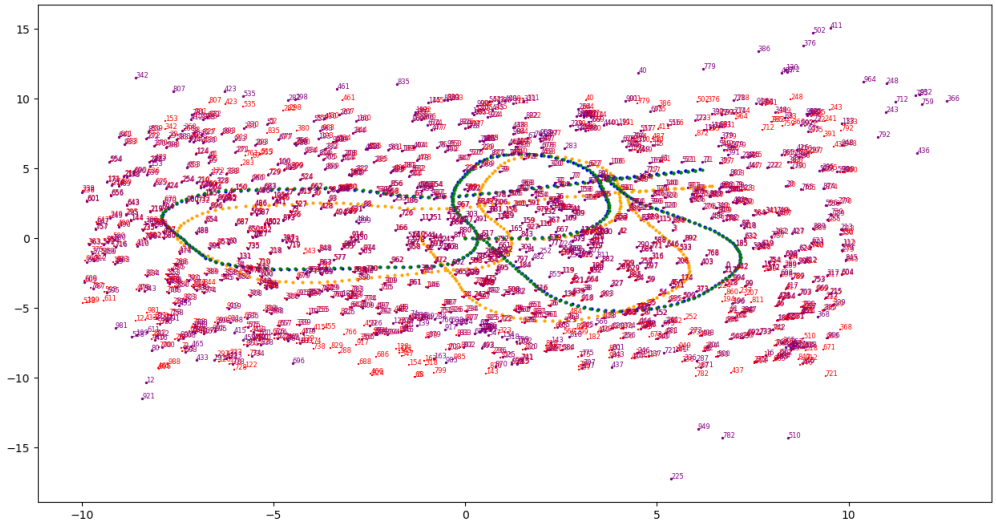

Figure 6: Diagram showing the complete graph with robot and landmark locations for the One-Step SLAM problem.

Analyzing the results only for the poses, as seen in Figure 7, shows that the system correctly localized the robot position. The image on the left shows the error that the odometry measurements (yellow dots) have compared to the correct robot position (green). This error is visibly negligible seen in the right image where the optimize robot position (blue dots) are synchronized to the correct robot position (green dots, unseen due to syncronization).

The pose (x, y, theta angle about z-axis) for the last timestep during optimization is compared in Table 3.

Robot position	Optimized	Correct	% Error
x	0.31970648	0.322606	0.899%
y	0.27858502	0.280024	0.514%
θ_z	1.9672791	1.95864	0.441%

Table 3: Robot pose for the last time timestep after optimization, the correct value, and its percent error.

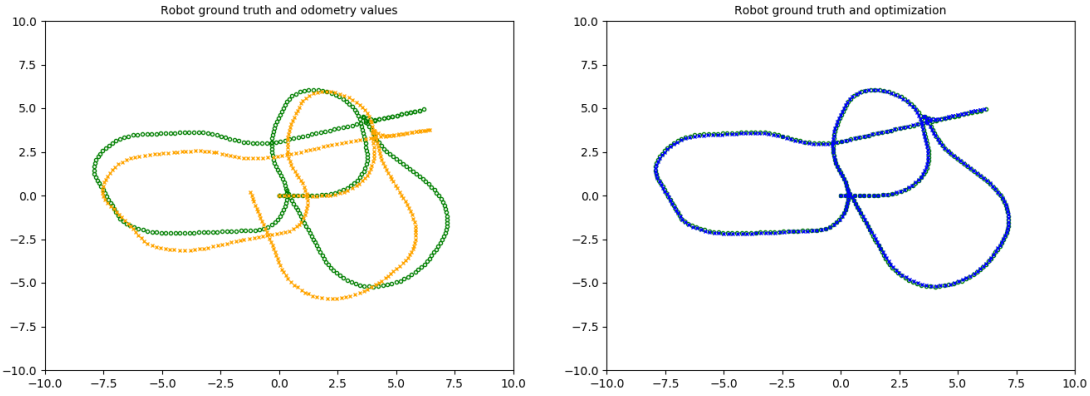

Figure 7: Diagram showing graph with robot locations for the One-Step SLAM problem. Without and with optimization, left to right images respectively.

Looking only at the landmarks as seen in Figure 8, the right image shows this result as dots excluding outliers. The comparison is evident considering the left image that shows the landmark locations only with triangulation before the optimization. Figure 9 shows the same landmaarks with the optimized robot position. It recognizes that the landmarks are indeed placed closer to its real value after the optimization by the system on the right image, unlike the left image.

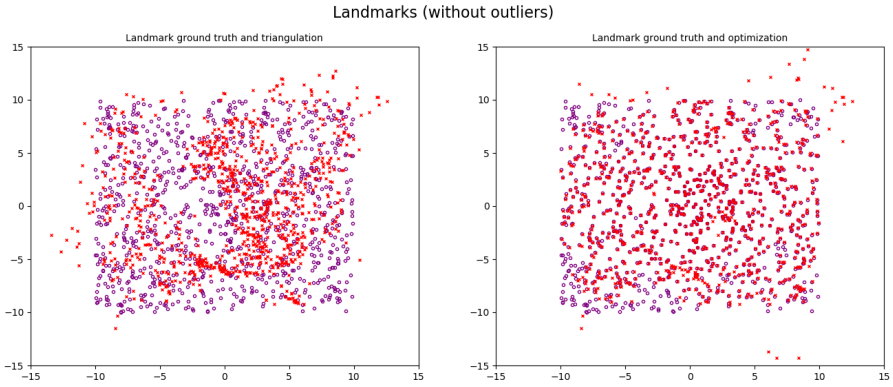

Figure 8: Diagram showing graph with landmark locations for the One-Step SLAM problem. Without and with optimization, left to right images respectively.

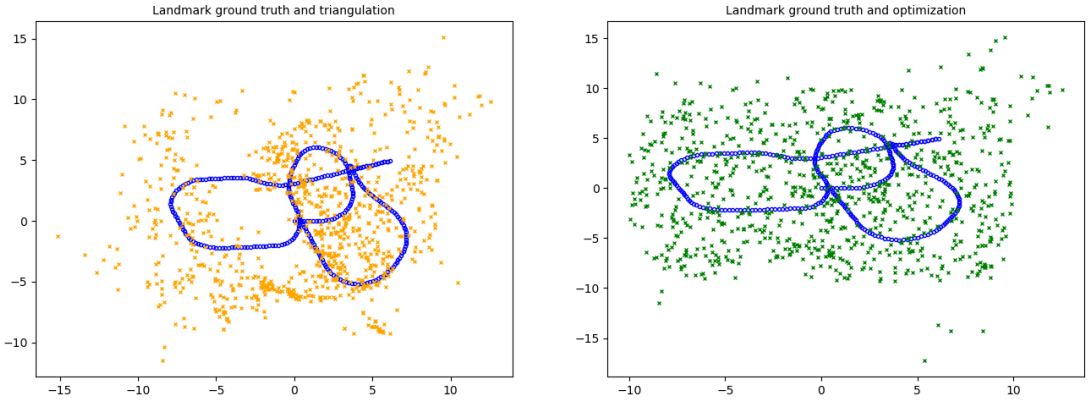

Figure 9: Diagram showing graph with the optimized robot and landmark locations for the One-Step SLAM problem. Without and with optimization for the landmark locations, left to right images respectively.

5 Discussion and Conclusion

The results presented for One-Step SLAM show convergence to the correct robot positions with minimal error. It also shows that the majority of the landmarks were mapped correctly in the environment after the optimization.

Since this implementation calculates the mapping and localization of the robot in one step, the computation cost is basic. However in a more robust im-

plementation, continuous SLAM would have to be implemented since this would be most beneficial for real-scenario where the robot has to map and localize during each time-step movement. I implemented this algorithm and although it was having positive results, the cost of computation was too high and therefore not added to my results. Therefore to sustain this implementation, a faster implementation would have to be used. For example, using C++ rather than python to create the algorithm, finding a more robust way to solve the linear system of the least-square, or even removing some extra past robot poses or landmarks that do not add much information for the creation of the map or localization but do add to the size and sparcity of the matrix to be solved.

6 Bibliography

1. Munguía, Rodrigo Grau, Antoni. (2012). Monocular SLAM for Visual Odometry: A Full Approach to the Delayed Inverse-Depth Feature Initialization Method. Mathematical Problems in Engineering. 2012. 10.1155/2012/676385.

2. Schlegel, Dominik Grisetti, Giorgio. (2018). HBST: A Hamming Distance embedding Binary Search Tree for Visual Place Recognition. IEEE Robotics and Automation Letters. PP. 10.1109/LRA.2018.2856542.

3. Stachniss, Cyrill Grisetti, Giorgio Burgard, Wolfram. Various Lecture Slides on Least Squares and Graph-based SLAM. Universität Freiburg.