MATLAB TOOLBOX FOR SMOOTHING AND MAPPING

Viorela Ila

February 7, 2012

1 Introduction

This toolbox implements deferent optimization strategies to solve the simultaneous localization and mapping (SLAM) problem. This is needed to enable autonomous robots to localize, map and navigate previously unknown environments. This work considers smoothing rather than filtering. The advantage is that the estimation errors are reduced by using better linearization points. Both, *pose SLAM* (smoothing the robot's trajectory only) and *full SLAM* (estimating the trajectory together with landmarks in the environment) are considered. The user can also chose between *batch* and *incremental* optimization.

The purpose of the toolbox is to test the mathematics behind the smoothing and mapping

The toolbox is available here

https://wiki.laas.fr/robots/ViorelaIla?action=AttachFiledo=viewtarget=SAMToolbox.tar.gz

It includes three implementations for incremental online smoothing and mapping; a new method of incremental Cholesky updates (called *L-SLAM*) together with QR factorization of the matrix A (which in the SLAM literature is known as SAM, smoothing and mapping [**?**]) and the Cholesky decomposition of the matrix Λ (based on incremental updates of Λ from [**?**]).

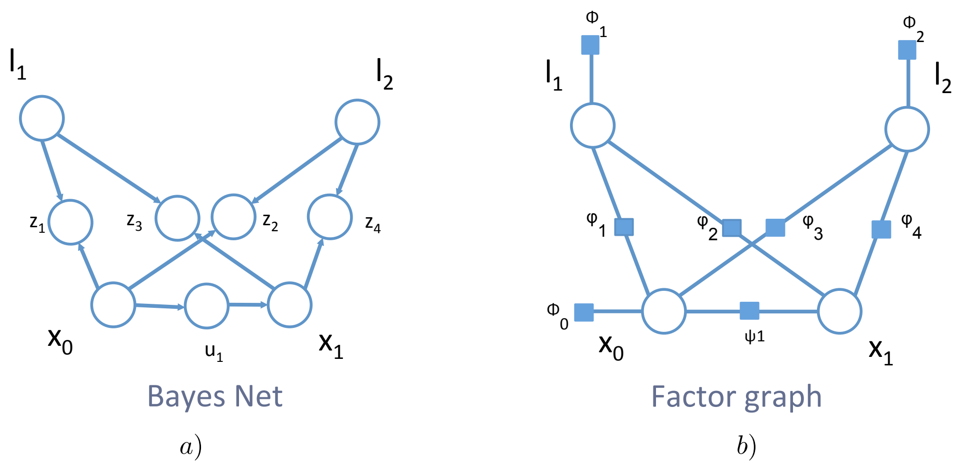

Figure 1: a) Bayes net representation of SLAM. b) Factor graph representation of SLAM.

2 Mathematical background

The SLAM problem estimates the robot trajectory $\mathbf{x} = \{x_i | i \in 0..n\}$ and the position of landmarks in the environment $\mathbf{l} = \{l_j | j \in 1..nl\}$ given a set of observations $\mathbf{z} = \{z_k | k \in 1..m\}$ and control inputs $\mathbf{u} = \{u_i | i \in 1..n\}$. Figure 1 a) shows the graphical model representation of such a distribution as a Bayes net. In general, for the Bayes net representation we can write the joint density as:

$$P(\mathbf{x}, \mathbf{l}, \mathbf{z}, \mathbf{u}) \propto P(x_0) \prod_j^{nl} P(\mathbf{l}_j) \prod_i^{n} P(x_{i+1} \mid x_i, u_i) \prod_k^{m} P(z_k \mid x_{i_k}, l_{j_k}) \tag{1}$$

where $\prod_j^{nl} P(\mathbf{l}_j)$ can be omitted as it is uninformative and the equation above becomes:

$$P(\mathbf{x}, \mathbf{l}, \mathbf{z}, \mathbf{u}) \propto P(x_0) \prod_i^{n} P(x_{i+1} \mid x_i, u_i) \prod_k^{m} P(z_k \mid x_{i_k}, l_{j_k}) \tag{2}$$

2.1 Factor graph formulation

When the factorization is made explicit we obtain the equivalent factor graph representation of the problem:

$$P(\mathbf{x}, \mathbf{l}, \mathbf{z}, \mathbf{u}) = \phi(x_0) \prod_i^{n} \psi(x_{i+1}, x_i) \prod_k^{m} \varphi(x_{i_k}, l_{j_k}) \tag{3}$$

In a typical SLAM application, there are three types of factors: the unary factors $\phi(x_0) \propto P(x_0)$, odometric factors $\psi(x_{i+1}, x_i) \propto P(x_{i+1} \mid x_i, u_i)$ and measurement factors $\varphi(x_{i_k}, l_{j_k}) \propto P(z_k \mid x_{i_k}, l_{j_k})$ and the corresponding graphical representation is shown in Figure 1 b).

2.2 Maximum likelihood estimation

The optimization problem associated with the full SLAM problem can be concisely stated in terms of a sparse least-squares problem. We want to obtain the maximum likelihood estimate (MLE) for all robot poses $\mathbf{x}$ and all landmarks $\mathbf{l}$ collected in vector $\theta = (\mathbf{x}, \mathbf{l})$:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\ P(\boldsymbol{\theta} \mid \mathbf{z}, \mathbf{u}) = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\ \{-\log(P(\theta \mid \mathbf{z}, \mathbf{u}))\}\,. \tag{4}$$

In general, for the SLAM problem, we assume Gaussian distributions and we write:

$$\begin{aligned} x_i = f_i(\mathbf{x}_{i-1}, \mathbf{u}_i) - \mathbf{w}_i \quad &\Leftrightarrow \quad P(\mathbf{x}_{i+1} \mid \mathbf{x}_i, \mathbf{u}_i) \propto \exp\left(-\tfrac{1}{2} \parallel f_i(\mathbf{x}_{i-1}, \mathbf{u}_i) - \mathbf{x}_i \parallel^2_{\Gamma_i}\right) \\ z_k = h_k(\mathbf{x}_{i_k}, \mathbf{l}_{j_k}) - \mathbf{v}_k \quad &\Leftrightarrow \quad P(\mathbf{z}_k \mid \mathbf{x}_{i_k}, \mathbf{l}_{j_k}) \propto \exp\left(-\tfrac{1}{2} \parallel h_k(\mathbf{x}_{i_k}, \mathbf{l}_{j_k}) - \mathbf{z}_k \parallel^2_{\Sigma_k}\right) \end{aligned} \tag{5}$$

where f and h are motion and measurement models, v and w are the normally distributed zero-mean motion and measurement noise with covariance matrices Γ_i and Σ_k, respectively.

2.3 Least-squares problem

Finding the MLE from equation 4 is done by solving the following non-linear least-squares problem

$$\boldsymbol{\theta}^* = \operatorname{argmin} \left\{ \sum_{i=1}^{n} \parallel f_i(\mathbf{x}_{i-1}, \mathbf{u}_i) - \mathbf{x}_i \parallel^2_{\Gamma_i} + \sum_{k=1}^{m} \|h_k(\mathbf{x}_{i_k}, \mathbf{l}_{j_k}) - \mathbf{z}_k\|^2_{\Sigma_k} \right\}. \tag{6}$$

In practice solving this typically nonconvex optimization problem is difficult and prone to local minima. Therefore, non-linear optimization methods such as Gauss-Newton or Levenberg-Marquardt are used to approximate the minimum incrementally, by solving a series of linearized problems. At each iteration, we collect the Jacobian matrices into the matrix A and the current residuals into the right-hand side vector $\mathbf{b}$ as in [**?**] ,we obtain the following standard least-squares problem in δ:

$$f(\boldsymbol{\delta}) = \frac{1}{2} \|A\boldsymbol{\delta} - \mathbf{b}\|^2 \tag{7}$$

In general, matrix A can be very large, but is quite sparse. In [**?**, **?**] it has been shown that matrix A naturally corresponds to the factor graph representation in equation 3. The normalized equation is given by:

$$f(\boldsymbol{\delta}) = \frac{1}{2} \|\Lambda\boldsymbol{\delta} - \boldsymbol{\eta}\|^2 \tag{8}$$

where $\Lambda = A^T A$ is the information matrix and $\boldsymbol{\eta} = A^T \mathbf{b}$ is the information vector. The information matrix Λ corresponds to the Markov random field associated with the SLAM problem. We call “information form” of the SLAM problem when the state is represented by the information matrix Λ and the information vector $\boldsymbol{\eta}$. The information matrix has the advantage of being sparse and remaining of the size of the state $(n + nl) \times (n + nl)$ even if the number of measurements increases.

2.4 Solving the SLAM problem

To solve the linearized version of equation 8 we can employ either iterative or direct methods. Each of these families of optimization techniques presents its own advantages and limitations.

Iterative methods

Iterative methods, on one hand, only require access to the gradient and have a small memory footprint, but can suffer from poor convergence. The current state of the art in iterative methods applied to SLAM problem stems from the work by [**?**] whose main contribution is a parameterization of the global poses in terms of increments along the trajectory. Grisetti [**?**, **?**] adopted this incremental parameterization as well, but instead defines it on a spanning tree. Both lines of work use stochastic gradient descent (SGD) to minimize the resulting objective functions in terms of incremental poses.

Direct methods

On the other hand, direct methods exhibit quadratic convergence and can be quite efficient for sparse problems like SLAM. The linear system can be solved directly either by Cholesky or QR matrix factorizations of the matrix Λ, respectively A and then performing backsubstitution.
Cholesky factorization yields $A^\top A = R^\top R$, where $R^\top$ is the Cholesky factor, and a forward and back substitutions on $R^\top \mathbf{y} = A^\top \mathbf{b}$ and $R\,\theta = \mathbf{y}$ first recovers $\mathbf{y}$, then the actual solution, the update θ.
Alternatively we can skip the normal equation in 8 and apply *QR factorization* directly to the matrix A in 7, yielding $A = Q\,R$. The solution θ can be directly obtained by backsubstitution in $R\theta = \mathbf{d}$ where $\mathbf{d} = R^{-\top}\,A^\top\,\mathbf{b}$. Note that Q is not explicitly formed; instead $\mathbf{b}$ is modified during factorization to obtain $\mathbf{d}$.

Recently has been shown in [**?**] that sparse Cholesky decomposition on $A^\top A$ performs faster than QR decomposition on A . Performing both operations; $A^\top A$ and Cholesky decomposition, on sparse matrices remains more efficient than performing QR decomposition on A.
In both strategies, finding the solution involves backsubstitution; if the factor R is sparse, the backsubstitution performs rapidly, but it becomes more inefficient when the fill-in increases. The

structure of the factor, in particular the number of non-zero elements in each row and column can be predicted by forming the "elimination tree". Forming the elimination tree depends on the chosen elimination ordering. To avoid the fill-in, efficient elimination orderings need to be found. While finding the variable ordering that leads to the minimal fill-in is NP-hard [?] for general problems, one typically uses heuristics such as the column approximate minimum degree (COLAMD) algorithm by [?], which provide close to optimal orderings for many problems.

2.5 New method to incrementally update the Cholesky factor

The insights gained from the Bayes tree data structure allowed us to develop new simplified incremental algorithms, more suitable to very large scale applications such those in the project ROSACE. For that we investigated how to incrementally update directly the Cholesky factor $L = R^\top$ in order to combine the advantages of incremental updates and sparse Cholesky decomposition.

Integrating a new measurement to the information form is additive. Updating Λ and $\boldsymbol{\eta}$ we obtain:

$$\tilde{\Lambda} = \begin{pmatrix} \Lambda_{11} & \Lambda_{21}^\top \\ \Lambda_{21} & \Lambda_{22} + \Omega \end{pmatrix} \; ; \quad \tilde{\boldsymbol{\eta}} = \begin{bmatrix} \boldsymbol{\eta}_1 \\ \boldsymbol{\eta}_2 + \boldsymbol{\omega} \end{bmatrix} \tag{9}$$

where Ω is the increment in information given by $H^\top \, \Sigma_{z_k} \, H$ and $\boldsymbol{\omega} = H \, \Sigma_{z_k}^{-1} \mathbf{e_k}$, $\mathbf{e_k}$ being the prediction-measurement error. Only a part of the information matrix is changed in the update step and the same happens with the lower diagonal factor L. The update factor becomes:

$$\tilde{L} = \begin{pmatrix} L_{11} & \\ L_{21} & \tilde{L}_{22} \end{pmatrix} \tag{10}$$

From $\tilde{\Lambda} = \tilde{L} \, \tilde{L}^\top$ and equations 9 and 10 we can write:

$$\Lambda_{22} + \Omega = L_{21} \, L_{21}^\top + \tilde{L}_{22} \, \tilde{L}_{22}^\top \tag{11}$$

and the changed part of the $\tilde{L}$ factor can be computed by applying Cholesky decomposition only to the reduced size matrix:

$$\tilde{L}_{22} = chol(\Lambda_{22} + \Omega - L_{21} \, L_{21}^\top) \tag{12}$$

The part of the RHS: $\tilde{\mathbf{d}} = \begin{bmatrix} \mathbf{d_1} \\ \tilde{\mathbf{d_2}} \end{bmatrix}$ affected by the new measurement can also be easily updated

$$\tilde{\mathbf{d_2}} = \tilde{L}_{22} \backslash (\boldsymbol{\eta}_2 + \boldsymbol{\omega} - L_{21}\, \mathbf{d_1}) \tag{13}$$

In an online applications, where the robots are mostly performing exploration task either to map the environment or to navigate towards a specific goal, the new measurements affect variables that are close in the state representation, resulting in small sizes on the information increments Ω and $\boldsymbol{\omega}$ which translates in efficiency of the update step. Furthermore, in the case of ground robots odometric measurements have higher rate but they only link consecutive robot poses. Therefore adding an odometric measurement can be considered constant time. The constant is 1 for the particular case of Pose SLAM and k for the case of landmark SLAM, where k is the number of landmarks visible at a time instant.
After updating L and $\mathbf{d}$ backsubstitution is performed to find the solution.

2.6 Optimization on Manifold

The toolbox supports 2D and 3D SLAM. For the 2D the state is represented using (x, y, θ) and for the 3D the state is representing using $(x, y, z, \phi, \theta, \psi)$. Those are parameterizations of rotation-translation transformation between robot poses in the case of pose SLAM and/or robot position and landmarks in the case of landmark SLAM. Unfortunately those are global parameterizations which exhibits singularities or other anomalies at various points in the parameter space. These anomalies can cause serious problems for gradient based minimization procedures, like Gauss-Newton or Levenberg-Marquard.

Optimization problems on a manifold such as SO(3) involves calculating incremental steps in the tangent space to the manifold.

3 Implementation Details

The toolbox implements and test the SLAM nonlinear optimization problem. To test the

3.1 Nonlinear optimization

Gauss-Newon method:

```
Algorithm: Gauss-Newton
i=1;
done=(i>=Solver.maxIT);
while ~done
    [dm,time_solve]=solveSystem(System);
    done=((norm(dm)<Solver.tol)||(i>=Solver.maxIT));
    if ~done
        Config=newConfig(Config,dm);
        ck=cputime;
        System=linearSystem(Config,Graph,System);
        i=i+1;
    end
```

Levenberg-Marquard method:

```
Algorithm: Levenberg-Marquard


factor=10;
lambda=Solver.lambda;
lambda_max=1e7;
System_new=System;


current_error=10000000;


i=0;
done=0;
while ~done
    i=i+1;
    [S,v]=prepareSystem(System_new.A,System_new.b,lambda);
    %dm=spqr_solve(S,v);
    dm=S\v;
    if ((norm(dm)<Solver.tol)||(i>=Solver.maxIT))
        done=1;
    else
```

```
        Config_new=newConfig2D(Config,dm);

        System_new=linearSystem(Config_new,Graph,System_new);

        next_error=norm(System_new.b);  %norm(System_new.A*dm-System_new.b);

        if(next_error<=current_error)&& (lambda <= lambda_max)

            done =((current_error-next_error)<.005);

            fprintf('.');

            if mod(i,50)==0

                fprintf('\n');

            end

            Config=Config_new;

            current_error=next_error;

            System=System_new;

            lambda=lambda/factor;

        else

            lambda=lambda*factor;

        end

    end


end
```

```
function [S,v]=prepareSystem(A,b,lambda)

Atr=A';

H = Atr*A;

v = Atr*b;

D = diag(diag(H));

S =(H+lambda*D);

end
```

Contents