CSE 483: Mobile Robotics

Lecture by: Prof. K. Madhava Krishna
Scribe:Gourav Kumar, Enna Sachdeva

Module # 09
Date: 23rd September, 2016 (Friday)

Square-Root SAM (Smoothing and Mapping)

This document discusses the theory underlying square root SAM. This is written as a subsequent module to EKF SLAM and and makes references and comparisons to it.

1 *Smoothing* vs *Filtering*

In the context of SLAM, *filtering* refers to the setting wherein, at each time step, we are interested in estimating only the most recent pose of the robot and the map of the environment. Whereas, *smoothing* refers to the setting wherein, at each time step, we estimate the entire trajectory of the robot and the map of the environment. Frequently, the *smoothing* problem is also referred to as the *full SLAM* problem.

2 Smoothing and Mapping (SAM)

Until this point, methods we have looked at are mostly based on the Extended Kalman filter(EKF), which computes a linear approximation of the nonlinear process and/or observation models, and recursively estimates the Gaussian density over the current pose of the robot and the locations of the landmarks. However, as the number of landmarks being observed by the robot increases, the state covariance matrix becomes dense, thereby increasing the computational complexity of EKF. Moreover, filtering may not provide a good-quality solution when the linearization approximation does not hold, i.e., when the function is highly nonlinear, or when a good linearization point is not available. Also, it is hard to apply EKF in an incremental fashion, i.e., when new observations arrive, EKF still takes time proportional to a polynomial of cubic order of the number of landmarks.

To address most of the above issues, SAM was proposed. SAM, short for *Smoothing and Mapping* is a framework, wherein the SLAM problem is represented using a *graphical model*, and solved in a *least squares* framework. This framework has been referred to as Square root SAM or $\sqrt{\text{SAM}}$, as it uses a *square-root filtering* technique to pose the problem in a least-squares framework.

2.1 EKF SLAM vs SAM

Before we even begin our discussion on SAM, here's a brief note on the advantages of preferring SAM over EKF SLAM.

1. In smoothing, the covariance or information matrix I stays sparse, and the optimization problem can be solved using a least squares framework.
2. The (sparse) information matrix I or the observation Jacobian A can be efficiently factorized using Cholesky or QR factorization, that speeds up the process to obtain the optimal trajectory and map.

3. It deals with the nonlinear process and measurement models in a much better way than EKF.

4. It is exact, rather than approximate as the Jacobians are calculated at the true state, unlike in EKF, where they are evaluated at estimated state.

3 Graphical Models of SLAM

A graphical model is a probabilistic model that encodes the conditional independencies between random variables. Graphs are an intuitive way of representing and visualising the relationships between variables(for example, we can answer questions like "*is A dependent on B given that we know the value of C?*", just by looking at the graph).

3.1 SAM as a Belief Net

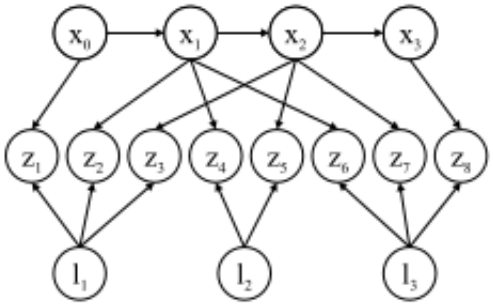

Figure 1:
Bayesian Belief Net for a SLAM problem.

A belief net is a directed acyclic graph(DAG) that encodes the conditional independent structure of a set of variables, where each variable only directly depends on its predecessors in the graph. The nodes of the graph represents variables and link represents the conditional dependence.

Consider the belief net shown in Fig. 1. Here,

- x_i: state of the robot at i^{th} time step, $i \in 0..M$
- l_j: a landmark, $j \in 0...N$
- z_k: an observation, $k \in 0...K$
- $P(x_0)$: a prior on the initial state
- $P(x_i|x_{i-1}, u_i)$: state transition probability (*motion model*). Given the previous state x_{i-1} and a control u_i, it specifies how the robot state evolves over time as a function of the control input u_i. Note that $P(x_i|x_{i-1}, u_i)$ is a probability distribution, not a deterministic function, owing to the process noise in the robot's motion.

- $P(z_k|x_{ik}, l_{jk})$: probability of obtaining a measurement z_k from state x_i to landmark l_j (*observation model*). By the *Markov assumption*, the current state x_i is sufficient to predict the (potentially noisy) measurement z_{ik}. Knowledge of any other variables, such as past measurements, controls or even past states, is irrelevant if x_i is known.

For a given known $Z = (z_1, z_2, z_3.....z_k)$,the joint probability corresponding to this network is given by

$$P(X, L|Z) = P(X|L, Z)P(L|Z) = \frac{P((X, L) \cap Z)}{P(Z)} = \frac{P(X, L, Z)}{P(Z)} \propto \quad P(X, L, Z) \tag{1}$$

$$P(X, L, Z) = P(x_0) \prod_{i=1}^{M} P(x_i|x_{i\text{-}1}, u_i) \prod_{k=1}^{K} P(z_k|x_{ik}, l_{jk}) \tag{2}$$

We assume here that the data association problem has been solved, i.e, we precisely know for each observation the global id (or a similar distinguisher) of the landmark being observed.

Assuming that all noise variables involved are Gaussian, the motion model can be written as

$$x_i = f_i(x_{i-1}, u_i) + w_i \quad \Rightarrow \quad P(x_i|x_{i-1}, u_i) \;\propto\; \exp -\frac{1}{2}||f_i(x_{i-1}, x_i) - x_i||^2_{\Lambda_i} \tag{3}$$

Here, $f_i(.)$ is the motion model, $w_i(.)$ is a random variable denoting the process noise, assumed to be normally distributed with zero mean and a covariance matrix Λ_i.

The observation model can be written as

$$z_k = h_k(x_{ik}, l_{jk}) + v_k \quad \Rightarrow \quad P(z_k|x_{ik}, l_{jk}) \;\propto\; \exp -\frac{1}{2}||h_k(x_{ik}, l_{jk}) - z_k||^2_{\Sigma_k}, \tag{4}$$

Here, $h_k(.)$ is the observation model, and $\mathbf{v_k}(.)$ is the normally distributed zero mean measurement noise with a covariance matrix Σ_k

3.2 SAM as a Least Squares Problem

Given all measurements ($Z \triangleq \{z_k\}$) and control inputs ($U \triangleq \{u_i\}$), we wish to obtain an *optimal* estimate for the set of unknowns, i.e., the entire trajectory ($X \triangleq \{x_i\}$) and the map ($L \triangleq \{l_j\}$). One way to do this is to find the Maximum a Posteriori (MAP) estimate. By maximizing the joint probability (equation(2)), the MAP estimate can be obtained (do not confuse with the usage of *map*, which refers to the location of landmarks in the environment). Let Θ be the vector consisting of unknown variables, X and L, i.e $\Theta \triangleq (X, L)$. Our objective is to find Θ^*, where

$$\Theta^* \triangleq \underset{\Theta}{\operatorname{argmax}} P(X, L|Z) = \underset{\Theta}{\operatorname{argmax}} P(X, L, Z) \tag{5}$$

Since the logarithm is a monotonically increasing function, the above problem can be expressed as follows.

$$\underset{\Theta}{\operatorname{argmax}} f(\Theta) = \underset{\Theta}{\operatorname{argmax}} \log f(\Theta) = \underset{\Theta}{\operatorname{argmin}}(-\log f(\Theta)) \tag{6}$$

Therefore, our objective function becomes

$$\Theta^* \triangleq \underset{\Theta}{\operatorname{argmin}}(-\log P(X, L, Z)) \tag{7}$$

$$\Theta^* \triangleq \underset{\Theta}{\operatorname{argmin}} \left(-\log \left(P(x_0) \prod_{i=1}^{M} P(x_i|x_{i\text{-}1}, u_i) \prod_{k=1}^{K} P(z_k|x_{ik}, l_{jk}) \right) \right) \tag{8}$$

Substituting equations 3, 4 in equation 8, we get the following nonlinear least squares problem,

$$\Theta^* \triangleq \underset{\Theta}{\operatorname{argmin}} \left(\sum_{i=1}^{M} ||f_i(x_{i-1}, u_i) - x_i||^2_{\Lambda_i} + \sum_{k=1}^{K} ||h_k(x_{ik}, l_{jk}) - z_k||^2_{\Sigma_k} \right)$$

NOTE: The mahalanobis distance associated with motion model ($f(.)$) involves process noise covariance Λ_i and that associated with measurement model ($h(.)$) involves observation covariance, Σ_k

4 Linearization of the Motion and Observation models

For most practical purposes, the process model f_i and/or the measurement equation h_k is/are nonlinear. To approximate the nonlinear models by linear ones, suitable linearization point, or an initial estimate is necessary. If a *good* linearization point is not available[1], nonlinear optimization methods such as Gauss-Newton or Levenberg-Marquart, which solve a succession of linear approximations until convergence, are to be used. In our discussion, we will assume that either a good linearization point is available, or that we are working on one particular iteration of a nonlinear optimization method.

4.1 Linearization of the Motion Model

Using Taylor series expansion, $f_i(x_{i-1}, u_i) - x_i$ can be linearized about x^0_{i-1} and x^0_i as below.

$$f_i(x_{i-1}, u_i) - x_i \approx \left(f_i(x^0_{i-1}, u_i) + F^{i-1}_i \delta x_{i-1}\right) - \left(x^0_i - G^i_i \delta x_i\right) \tag{9}$$

Here, F^{i-1}_i and G^i_i are the Jacobians of the expression to be linearized. Concretely,

$$F^{i-1}_i \triangleq \left. \frac{\partial(f_i(x_{i-1}, ui) - x_i)}{\partial x_{i-1}} \right|_{x^0_{i-1}} \tag{10}$$

$$G^i_i \triangleq \left. \frac{\partial(f_i(x_{i-1}, ui) - x_i)}{\partial x_i} \right|_{x^0_i} \tag{11}$$

Also, $a_i = x^0_i - f_i(x^0_{i-1}, u_i)$, is the odometry prediction error, which is a result of odometry perturbation($x^0_i = f_i(x^0_{i-1}, u_i) + w_i$).

[1]Here, a *good* linearization point is one about which the local linearity assumption of the function holds well, i.e., in the neighborhood of the linearization point, the linearized function very well approximates the true function.

4.2 Linearization of the Observation Model

Similarly, the observation model can also be linearized in the following fashion.

$$h_k(x_{ik}, l_{jk}) - z_k \approx \Big(h_k(x^0_{ik}, l^0_{jk}) + H^{ik}_k \delta x_{ik} + J^{jk}_k \delta l_{jk})\Big) - z_k = (H^{ik}_k \delta x_{ik} + J^{jk}_k \delta l_{jk}) - c_k \quad (12)$$

Here, H^{ik}_k and J^{ik}_k are the Jacobians involved in the Taylor series expansion about (x^0_{ik}, l^0_{jk}).

$$H^{ik}_k \triangleq \frac{\partial h_k(x_{ik}, ljk)}{\partial x_{ik}}\Big|_{(x^0_{ik}, l^0_{jk})} \quad (13)$$

$$J^{jk}_k \triangleq \frac{\partial h_k(x_{ik}, ljk)}{\partial l_{jk}}\Big|_{(x^0_{ik}, l^0_{jk})} \quad (14)$$

Also, $c_k = z_k - h_k(x^0_{ik}, l^0_{jk})$ is the measurement prediction error.

5 Getting to the Least Squares Formulation

After linearizing the motion and observation models as shown in the previous section, the MAP estimation problem reduces to the following form.

$$\delta^* = \underset{\delta}{\operatorname{argmin}} \left(\sum_{i=1}^{M} ||F^{i-1}_i \delta x_{i-1} + G^i_i \delta x_i - a_i||^2_{\Lambda_i} + \sum_{k=1}^{K} ||H^{ik}_k \delta x_{ik} + J^{ik}_k \delta l_{jk} - c_k||^2_{\Sigma_k} \right) \quad (15)$$

In the above equation, the Mahalanobis distance can be converted into a L-2 norm as follows. This technique, also used in solving weighted least squares problems, is referred to as *square root filtering*.

$$||e||^2_\Sigma \triangleq e^T \Sigma^{-1} e = (\Sigma^{-T/2} e)^T (\Sigma^{-T/2} e) = ||\Sigma^{-T/2} e||^2_2 \quad (16)$$

Converting each Mahalanobis distance to a Euclidean distance (L2-Norm) using the square root filtering method, we obtain the following system.

$$\begin{aligned} \delta^* = & \underset{\delta}{\operatorname{argmin}} \sum_{i=1}^{M} \|\Lambda_i^{-T/2} F_i^{i-1} \delta x_{i-1} + \Lambda_i^{-T/2} G_i^i \delta x_i - \Lambda_i^{-T/2} a_i\|^2_2 + \\ & \sum_{k=1}^{K} \|\Sigma_k^{-T/2} H_k^{ik} \delta x_{ik} + \Sigma_k^{-T/2} J_k^{jk} \delta l_{jk} - \Sigma_k^{-T/2} c_k\|^2_2 \end{aligned} \quad (17)$$

By slight abuse of notation - for convenience sake - let us replace $\Sigma_k^{-T/2} H_k^{ik}$ by H_k^{ik}, $\Sigma_k^{-T/2} J_k^{jk}$ by J_k^{jk}, etc. In short, lets assume that all Jacobians F, G, H, J and all constants a, c have been premultiplied by corresponding square root factors. Then, the whole problem can be written in the form

$$\delta^* = \underset{\delta}{\operatorname{argmin}} ||A\delta - b||^2_2 \quad (18)$$

The A matrix in the above equation is commonly referred to as the *observation jacobian*, as it is an accumulation of the jacobians of all observations made. An example A matrix for the factor graph shown in Fig. 2 takes the following form.

$$A = \begin{bmatrix} G_1^1 & 0 & 0 & 0 & 0 \\ F_2^1 & G_2^2 & 0 & 0 & 0 \\ 0 & F_3^2 & G_3^3 & 0 & 0 \\ H_1^1 & 0 & 0 & J_1^1 & 0 \\ H_2^1 & 0 & 0 & 0 & J_2^2 \\ 0 & H_3^2 & 0 & J_3^1 & 0 \\ 0 & 0 & H_4^3 & 0 & J_4^2 \end{bmatrix} \quad b = \begin{bmatrix} a1 \\ a2 \\ a3 \\ c1 \\ c2 \\ c3 \\ c4 \end{bmatrix} \tag{19}$$

A can accomodate landmarks and measurements of different dimensions.

5.0.1 Dimensionality of A

$d_x \implies$ number of dimensions used to represent each pose
$d_l \implies$ number of dimensions used to represent each landmark
$d_z \implies$ number of dimensions in each observation
$M \implies$ total number of states
$N \implies$ total number of landmarks
$K \implies$ total number of measurements obtained

Dimensionality of A $= (Md_x + Kd_z) \times (Md_x + Nd_l)$

5.1 Constructing the A Matrix from a Factor Graph

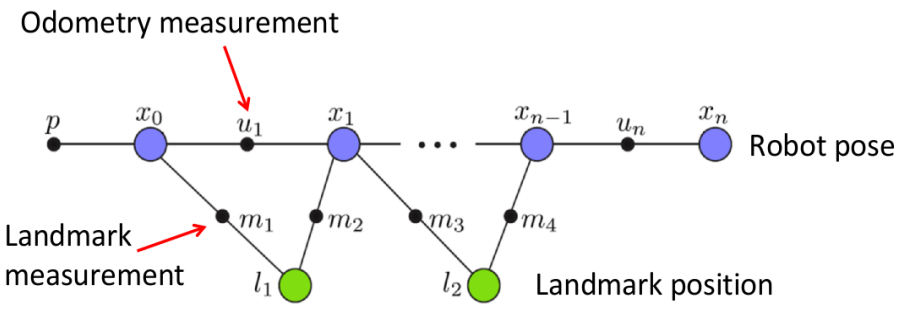

Figure 2:
Factor Graph representation for SLAM.

Factor graphs are graphical representations of the factorization of a function. A factor graph consists of two types of nodes - *variables* and *factors*. In our case, *variables* refer to the unknowns we are trying to estimate, i.e., the states ($X \triangleq \{x_i\}$) and the landmarks ($L \triangleq \{l_j\}$). *Factors* refer to the relationships among variables. These relations typically emerge in the form of odometry (which relates two pose variables) and range-bearing observations (which relates a pose and a landmark variable). In a factor graph, there are nodes for the variables (X, L), and there are nodes (also called factor nodes) for each factor (Z, U), as

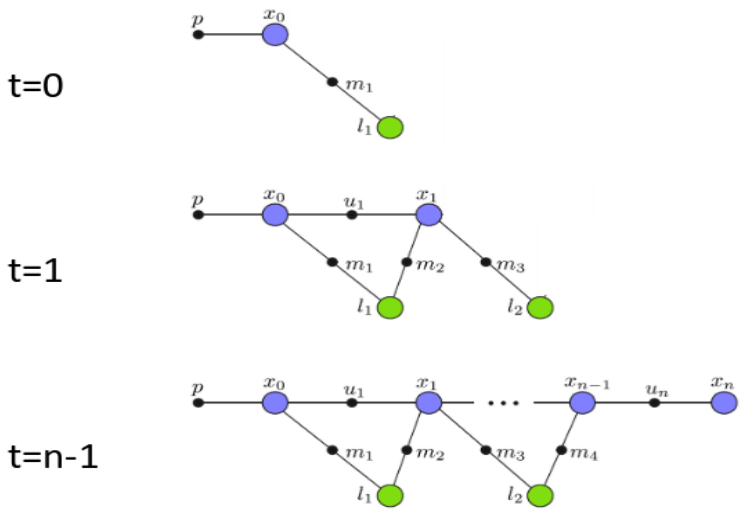

Figure 3:
Factor Graph at different time instants.

shown in Fig. 2. The evolution of a factor graph as the robot moves in its environment is shown in Fig. 3.

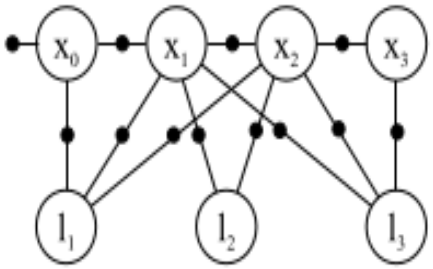

Figure 4:
Factor Graph.

Jacobian A is the measurement of the factor graphs associated with SLAM, hence can be used to construct A, directly. For instance, consider the factor graph in figure 4. to

construct its A matrix.

$$A = \begin{bmatrix} G_0^0 & 0 & 0 & 0 & 0 & 0 & 0 \\ F_1^0 & G_1^1 & 0 & 0 & 0 & 0 & 0 \\ 0 & F_2^1 & G_2^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & F_3^2 & G_3^3 & 0 & 0 & 0 \\ H_1^0 & 0 & 0 & 0 & J_1^1 & 0 & 0 \\ 0 & H_2^1 & 0 & 0 & J_2^1 & 0 & 0 \\ 0 & H_3^1 & 0 & 0 & 0 & J_3^2 & 0 \\ 0 & H_4^1 & 0 & 0 & 0 & 0 & J_4^3 \\ 0 & 0 & H_5^2 & 0 & J_5^1 & 0 & 0 \\ 0 & 0 & H_6^2 & 0 & 0 & J_6^2 & 0 \\ 0 & 0 & H_7^2 & 0 & 0 & 0 & J_7^3 \\ 0 & 0 & 0 & H_8^3 & 0 & 0 & J_8^3 \end{bmatrix}, \delta = \begin{bmatrix} \delta_{x_0} \\ \delta_{x_1} \\ \delta_{x_2} \\ \delta_{x_3} \\ \delta_{l_1} \\ \delta_{l_2} \\ \delta_{l_3} \end{bmatrix}, b = \begin{bmatrix} a0 \\ a1 \\ a2 \\ a3 \\ c1 \\ c2 \\ c3 \\ c4 \\ c5 \\ c6 \\ c7 \end{bmatrix} \tag{20}$$

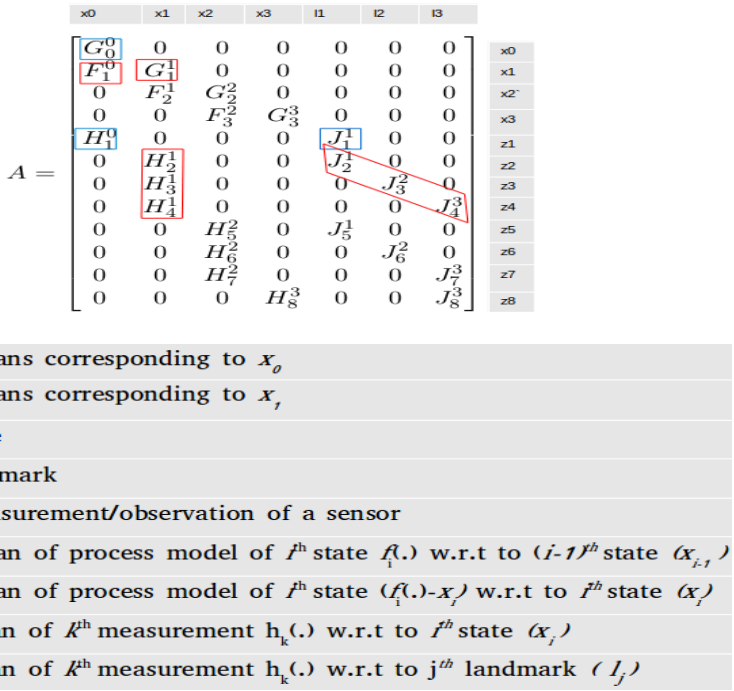

Figure 5:
A matrix

Here M=4, N=3, K=8 $\implies$ dimensions of $A = (4d_x + 8d_l) \times (3d_l + 4d_x)$. where

$d_x \implies$ dimensions of state
$d_l \implies$ dimensions of landmarks
$d_z \implies$ dimensions of measurement
$M \implies$ number of states
$N \implies$ number of landmarks
$K \implies$ number of measurements obtained till current state x_n
Every block of A corresponds to the least square criterion associated with either a landmark measurement or a odometry constraint.

5.2 Finding the solution of $A\delta = b$

For a full rank $m \times n$ matrix, the least square solution is found by solving $A^T A\delta^* = A^T b$, which is generally done by finding the Cholesky factorization of information matrix as, $I = A^T A = R^T R$, where,
R is the Cholesky traiangle $n \times n$ matrix, computed using cholesky factorization. Finding the δ^* using Cholesky factorization requires $(m + n/3)n^2$ flops.
When A matrix is ordered in canonical form(i.e, trajectory X first and then map L), the typical block structure is as below-

$$I = \begin{bmatrix} A_X^T A_X & I_X^L \\ I_X^{L^T} & A_L^T A_L \end{bmatrix} \tag{21}$$

Here, $I_X^L = A_X^T A_L$ encodes the correlation between robot state X and map L, and the diagonal blocks are band-diagonal.
An alternative approach to proceed the problem is QR factorization which has following advantages over cholesky factorization:

1. It is more accurate and numerically stable than cholesky factorization.
2. No need to compute I matrix, instead QR factorization of A if computed.

$A = QR$, where Q is an $m \times m$ orthogonal matrix and R is the upper triangular cholesky triangle. Note that A is a full column rank matrix($m > n$), $\implies Q^T Q = I$

$$Q^T A = \begin{bmatrix} R \\ 0 \end{bmatrix}, Q^T b = \begin{bmatrix} d \\ e \end{bmatrix} \tag{22}$$

Using orthogonality property of Q, the least square problem can be formulated as :
$||A\delta - b||_2^2 = ||Q^T A\delta - Q^T b||_2^2 = ||R\delta - d||_2^2 + ||e||_2^2$
Since,$||e||_2^2$ is the least-squares residual, the LS solution δ^* can be obtained by solving the square system $R\delta = d$. R is an upper triangular matrix, and the problem can be solved using back substitution, easily.

6 Square Root SAM

This section briefly describes the Square-root SAM algorithm.

6.1 Batch $\sqrt{\text{SAM}}$

In batch mode, we follow a straightforward method of solving the least-squares problem using sparse matrix methods. The graph-theoretic algorithm underlying the solution of least-squares is *variable-elimination*, hence the order in which the variables are eliminated has huge impact on running time of matrix factorization algorithm (both *Cholesky* and *QR-decomposition*). The Jacobian matrix A also represents the adjacency matrix corresponding to the factor graph, it is interesting to note that column reordering heuristics also automatically exploit the locality inherent in geographic nature of slam problem, i.e., the Jacobians corresponding to the factor nodes and hence landmarks/poses corresponding to those factor nodes lying close to each other after column reordering of the A matrix. The algorithm for the square-root SAM in batch mode is shown below.

Algorithm 1 Batch Square-root SAM

1: Construct the observation Jacobian matrix $\mathbf{A}$ and odometry/measurement prediction error vector $\mathbf{b}$ as explained in section 3.

2: Find a good column odering for $\mathbf{A}(\mathrm{A}\xrightarrow{T}A_t)$.▷ T is the transformation corresponding to optimal column ordering

3: Solve for

$$\delta_{\mathbf{t}}^{*} = \underset{\delta}{\operatorname{argmin}}||A_t\delta_t - b||_2^2$$

using either Cholesky or QR factorization method.

4: Recover δ^* from δ_t^* by:

$$\delta_{\mathbf{t}}^{*} \xrightarrow{T^{-1}} \delta^{*}$$

▷ where T^{-1} is the inverse transformation

Although this algorithm assumes linear motion/observation models or models where a good linearization point exists for optimization, the same will work for nonlinear models where nonlinear optimizers(e.g. *Levenberg Marquardt* algorithm) are used. The only difference is that the algorithm(except the ordering steps) will be called multiple time until convergence by a non-linear optimizer.

6.2 Incremental $\sqrt{\mathbf{SAM}}$

As the batch mode computes the whole Jacobian matrix for every step for each update, it turns out to be computationally expensive hence an incremental version of above algorithm is derived for both linear as well as non linear cases

6.2.1 Linear Incremental $\sqrt{\mathbf{SAM}}$

If motion/observation models are linear or if a good linearization point is available we can update the factorizations incrementally by rank 1 Cholesky update which compute R' corresponding to I':

$$I' = I + aa^T \tag{23}$$

where I is the information matrix and $I{=}QR$ is its *QR-decomposition* and a^T is the new Jacobian row corresponding to the new measurement at that step.This algorithm of Cholesky update is typically used for dense matrices, but in our case we can go for more efficient method taking into account the sparse storage scheme that we use. One such method which is easy to implement is to use a series of Givens rotations to eliminate non-zeros in the new measurement rows one by one (for reference take a look at lecture9.pdf uploaded on course portal).

6.2.2 Non-linear Incremental $\sqrt{\mathbf{SAM}}$

In case the motion model or/and the observation model is/are non-linear functions and there is no method by which we can figure out a good linearization point, we have to resort to non-linear iterative algorithms such as Gauss-Newton or the Levenberg-Marquardt algorithm, which solve a succession of linear approximations to approach the minimum. The only drawback of these algorithms over linear ones is that they find linear approximations of the

models in each iteration before the convergence point is reached(and this is done for each update step).