A Sliding Window Filter for SLAM

Gabe Sibley

University of Southern California
Center for Robotics and Embedded Systems

Technical Report No. CRES-06-004.

August 18, 2006*

Abstract

This note describes a Sliding Window Filter that is an on-line constant-time approximation to the feature-based 6-degree-of-freedom full Batch Least Squares Simultaneous Localization and Mapping (SLAM) problem. We contend that for SLAM to be useful in large environments and over extensive run-times, its computational time complexity must be constant, and its memory requirements should be at most linear. Under this constraint, the "best" algorithm will be the one that comes closest to matching the all-time maximum-likelihood estimate of the full SLAM problem, while also maintaining consistency. We start by formulating SLAM as a Batch Least Squares state estimation problem, and then show how to modify the Batch estimator into an approximate Sliding Window Batch/Recursive framework that achieves constant time complexity and linear space complexity. We argue that viewing SLAM from the Sliding Window Least Squares perspective is very useful for understanding the structure of the problem. This perspective is general, capable of subsuming a number of common estimation techniques such as Bundle Adjustment and Extended Kalman Filter SLAM. By tuning the sliding window, the algorithm can scale from exhaustive Batch solutions to fast incremental solutions; if the window encompasses all time, the solution is algebraically equivalent to full SLAM; if only one time step is maintained, the solution is algebraically equivalent to the Extended Kalman Filter SLAM solution. The Sliding Window Filter enables other interesting properties, like continuous sub-mapping, lazy data association, undelayed or delayed landmark initialization, and incremental robust estimation. We test the algorithm in simulations using stereo vision exterioceptive sensors and inertial measurement proprioceptive sensors. Initial experiments show that the SWF approaches the performance of the optimal batch estimator, even for small windows on the order of 5-10 frames.

1 Introduction

For a mobile robot, there are many situations in which accurate high-resolution local *spatial awareness* is a prerequisite for successfully performing a task. For

*Revised August 23, 2006

instance, good sensing is undoubtedly useful for tight obstacle avoidance, delicate mobile manipulation, sensing complicated terrain, subtle motion detection or moving object tracking from a moving platform.

This is a *data-fusion* problem in which a sensor (potentially undergoing uncertain, dynamic motion) must combine noisy measurements into a single underlying sensor-relative state estimate. In Robotics this problem is most commonly tackled under the banner of Simultaneous Localization and Mapping (SLAM); in Computer Vision the problem is usually referred to as Structure from Motion (SFM).

This paper develops a Sliding Window Filter for SLAM that focuses computational resources on accurately estimating the immediate spatial surroundings using a sliding time window of the most recent sensor measurements. Ideally, we would like a constant time algorithm that closely approximates the all-time maximum-likelihood estimate as well as the minimum variance Cramer Rao Lower Bound - e.g. we would like an estimator that achieves some notion of statistical optimality (quickly converges), efficiency (quickly reduces uncertainty) and consistency (avoids over-confidence). To this end we give a derivation of the SLAM problem from the Gaussian non-linear least squares optimization perspective. We find that this results in a simple, yet general, take on the SLAM problem; we think this is a useful contribution.

It is interesting to note at this point that even though the Sliding Window Filter technique focuses on the spatially immediate estimation process, it also incrementally builds a map structure (occupying $O(n)$ memory space) that is useful for non-local problems, such as loop closure, place recognition, navigation, topological planning, etc. So while it does not address loop closure *per-se*, it does build a representation amenable to solving such problems. Ultimately, we believe that loop closure is best cast as a distinct problem from local estimation. Decoupling loop closure from the core SLAM estimator allows concentrating computational resources on improving the local result, which is crucial for applications that require spatially high-resolution, dense structure estimates. With high bandwidth sensors (like cameras) focusing on the local problem is clearly important for computational reasons; this is especially true if we wish to fuse all of the sensor data (or a significant portion thereof). However, even with this local focus, once a loop closure is identified, global optimization over the map structure left behind by the Sliding Window Filter should match the global batch solution.

Sliding Window Filtering is generally applicable to a wide variety of platform configurations and environments, capable of working with or without a motion model and with or without proprioceptive sensors. This flexibility allows solving bearing only problems, such as Bundle Adjustment, as well as traditional dynamic state space problems like EKF SLAM. Further, the approach is easily applicable to problems with asynchronous sensor measurements, like for example when using GPS at $\sim$1 hz in combination with inertial sensors at $\sim$100hz.

We apply the Sliding Window Filter to SLAM with stereo vision and inertial measurements. Experiments show that the best approximate method comes close to matching the performance of the optimal estimator while attaining constant time complexity - empirically, it is often the case that the difference in their performance is is indistinguishable.

2 Problem Description

It is useful to approach SLAM from the traditional Statistical Point Estimation perspective because it helps reveal the underlying structure of the problem. This is apparent for a number of reasons. First, because it highlights the fundamental minimization principle at work in least squares, which, we would argue, is a principle that is harder to see from the recursive estimation perspective. Second, starting with the underlying probability density functions that describe our problem, it clearly shows the Gaussian probabilistic nature of SLAM - that is, SLAM is simply tracking a normal distribution through a large state space; a state space that changes dimension as we undertake the fundamental probabilistic operations of removing parameters via marginalization, and adding parameters via conditioning. A third reason to derive SLAM via statistical point estimation is because it exposes a rich body of theory about the convergence of least squares estimators - theory that is missing or difficult to come by from the recursive non-linear estimation perspective.

Without going into detail, note that from this point one can easily see the connection to many important concepts like Newton's method, Fisher Information, the Cramer Rao Lower bound, information graphs, graphical models, belief propagation and probabilistic belief networks to name just a few. All these concepts have intuitive derivations starting from traditional statistical point estimation. If these concepts are branches of a tree, then optimizing a quadratic, as in least-squares, is the fundamental trunk.

With this in mind, we carry forward in the usual way, by describing the system state vector, process model, measurement model and how we include prior information.

2.1 System Parameterization

The state vector is a temporal sequence of robot poses $\mathbf{x}_{\mathbf{p}_j}$, $1 \le j \le m$, and 3D landmark positions $\mathbf{x}_{\mathbf{m}_i}$, $1 \le i \le n$.

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_{\mathbf{p}} \\ \mathbf{x}_{\mathbf{m}} \end{bmatrix} = \begin{bmatrix} \mathbf{x}_{\mathbf{p}_1} \\ \vdots \\ \mathbf{x}_{\mathbf{p}_m} \\ \mathbf{x}_{\mathbf{m}_1} \\ \vdots \\ \mathbf{x}_{\mathbf{m}_n} \end{bmatrix}$$

Each pose $\mathbf{x}_{\mathbf{p}_j}$ is represented by a six parameter column vector comprised of a 3D point and an Euler angle $\mathbf{x}_{\mathbf{p}_j} = [x_{\mathbf{p}_j}\ y_{\mathbf{p}_j}\ z_{\mathbf{p}_j}\ r_{\mathbf{p}_j}\ p_{\mathbf{p}_j}\ q_{\mathbf{p}_j}]^T$. Landmarks are represented by their 3D position, $\mathbf{x}_{\mathbf{m}_i} = [x_{\mathbf{m}_i}\ y_{\mathbf{m}_i}\ z_{\mathbf{m}_i}]^T$. The state dimension is thus $|x| = (6m + 3n)$ and grows as the robot path increases and as new landmarks are observed.

2.2 Process Model

The process model $f_j : \mathbb{R}^6 \to \mathbb{R}^6$ for a single step describes each pose in terms of the previous pose

$$\mathbf{x}_{\mathbf{p}_j} = f_j(\mathbf{x}_{\mathbf{p}_{j-1}}, \mathbf{u}_j) + \mathbf{w}_j \tag{1}$$

where $\mathbf{u}_j$ is an input command to the robot. The noise vector $\mathbf{w}_j$ is additive and follows a normal distribution $\mathbf{w}_j \sim N(0, \mathbf{Q}_j)$, so that $\mathbf{x}_{\mathbf{p}_j} \sim N(f_j(\mathbf{x}_{\mathbf{p}_{j-1}}, \mathbf{u}_j), \mathbf{Q}_i)$. A simple and useful kinematic process model for f is the compound operation, $\oplus$, which is described in (Appendix B). The Jacobian of f, $\mathbf{F}_j = \frac{\partial f}{\partial x}\Big|_{x_{j-1}}$, which we will need in a moment, is also derived in (Appendix B). Concatenating the individual process models together we can write the system dynamic state space model as,

$$f(\mathbf{x_p}) = \begin{bmatrix} f_1(\mathbf{x}_{\mathbf{p}_0}, \mathbf{u}_1) \\ \vdots \\ f_m(\mathbf{x}_{\mathbf{p}_{m-1}}, \mathbf{u}_m) \end{bmatrix}.$$

With the above definitions and assumptions the probability density function describing the robot path is

$$P(\mathbf{x_p}) = \eta_{\mathbf{x_p}} \exp\left\{-\frac{1}{2}(\mathbf{x_p} - f(\mathbf{x_p}))^T \mathbf{Q}^{-1}(\mathbf{x_p} - f(\mathbf{x_p}))\right\} \tag{2}$$

where $\eta = 1/\sqrt{(2\pi)^{|\mathbf{x_p}|}\det(\mathbf{Q})}$ is a normalizing constant and $|\mathbf{x_p}|$ is the dimension of $\mathbf{x_p}$. In practice, one usually extends this basic model to also estimate other quantities, such as linear and angular velocities; for clarity, we will stick with this basic formulation.

2.3 Sensor Model

A measurement of the i^{th} landmark taken from the j^{th} pose is related to the state vector by the sensor model $h_{ij} : \mathbb{R}^{|\mathbf{x}_{\mathbf{m}_i}|+|\mathbf{x}_{\mathbf{p}_j}|} \to \mathbb{R}^{|\mathbf{z}_{ij}|}$

$$\mathbf{z}_{ij} = h_{ij}(\mathbf{x}_{\mathbf{m}_i}, \mathbf{x}_{\mathbf{p}_j}) + \mathbf{v}_{ij} \tag{3}$$

which generates the expected value the sensor will return when landmark i is observed from pose j. We assume $\mathbf{v}_{ij} \sim N(0, \mathbf{R}_{ij})$ so that $\mathbf{z}_{ij} \sim N(h_{ij}, \mathbf{R}_{ij})$, where $\mathbf{R}_{ij}$ is the observation error covariance matrix.

Concatenating all the observations, measurement functions and measurement covariances together we write $\mathbf{z}$, h, and $\mathbf{R}$ as

$$\mathbf{z} = \begin{bmatrix} \mathbf{z}_{11} \\ \vdots \\ \mathbf{z}_{1m} \\ \vdots \\ \mathbf{z}_{nm} \end{bmatrix},\ h(\mathbf{x}) = \begin{bmatrix} h_{11} \\ \vdots \\ h_{1m} \\ \vdots \\ h_{nm} \end{bmatrix},\ \mathbf{R} = \begin{bmatrix} \mathbf{R}_{11} & 0 & & \dots & 0 \\ 0 & \ddots & & & \vdots \\ & & \mathbf{R}_{1m} & & \\ \vdots & & & \ddots & \\ 0 & \dots & & & \mathbf{R}_{nm} \end{bmatrix}$$

Thus the probability density describing the measurement likelihood is simply

$$P(\mathbf{z}|\mathbf{x}) = \eta_{\mathbf{z}} \exp\left\{-\frac{1}{2}(\mathbf{z} - h(\mathbf{x}))^T \mathbf{R}^{-1}(\mathbf{z} - h(\mathbf{x}))\right\} \tag{4}$$

where $\eta_z = 1/\sqrt{(2\pi)^{|\mathbf{z}|}\det(\mathbf{R})}$ is again a normalizing term.

2.4 Point Estimation

In the beginning let us say we have some prior information about the state represented by a prior distribution $\hat{\mathbf{x}} \sim N\left(\mathbf{x}, \mathbf{\Pi}^{-1}\right)$ which encodes information about a single starting pose, about some previously known map of n landmarks, and about the relationships between the starting pose and the map. The inverse covariance Π can be broken into four distinct parts

$$\hat{\mathbf{x}} \sim N\left(\begin{bmatrix} \mathbf{x}_{\mathbf{p}1} \\ \mathbf{x}_{\mathbf{m}1:n} \end{bmatrix}, \begin{bmatrix} \Pi_{\mathbf{p}} & \Pi_{\mathbf{pm}} \\ \Pi_{\mathbf{pm}}^T & \Pi_{\mathbf{m}} \end{bmatrix}^{-1}\right)$$

where here $\Pi_{\mathbf{p}}$ is the 6×6 initial pose information matrix, $\Pi_{\mathbf{m}}$ is $3n \times 3n$ map prior information matrix, and $\Pi_{\mathbf{pm}}$ is the $6 \times 3n$ pose-map information matrix. The probability density function representing this prior knowledge is

$$P(\hat{\mathbf{x}}) = \eta_{\hat{\mathbf{x}}} \exp\left\{-\frac{1}{2}(\hat{\mathbf{x}} - \mathbf{x})^T \Pi (\hat{\mathbf{x}} - \mathbf{x})\right\}, \tag{5}$$

where $\eta_{\hat{\mathbf{x}}} = 1/\sqrt{(2\pi)^{|\hat{\mathbf{x}}|} \det(\Pi^{-1})}$ is the normalizing factor.

Using (2), (5) and (4) we can now write the posterior probability of the state

$$P(\mathbf{x}|\mathbf{z}, \hat{\mathbf{x}}) = P(\mathbf{z}|\mathbf{x}) P(\mathbf{x}_{\mathbf{p}}) P(\hat{\mathbf{x}}) \tag{6}$$

We wish to compute the *maximum a posteriori* estimate of $\mathbf{x}$ which maximizes this density. It helps if we lump the sensor model, process model, and prior information terms together by defining the function $g(x)$ and matrix $\mathbf{C}$ as

$$g(\mathbf{x}) = \begin{bmatrix} g_z(\mathbf{x}) \\ g_f(\mathbf{x}) \\ g_\pi(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} \mathbf{z} - h(\mathbf{x}) \\ \mathbf{x}_{\mathbf{p}} - f(\mathbf{x}_{\mathbf{p}}) \\ \hat{\mathbf{x}} - \begin{bmatrix} \mathbf{x}_{\mathbf{p}_1} \\ \mathbf{x}_{\mathbf{m}} \end{bmatrix} \end{bmatrix}, \qquad \mathbf{C}^{-1} = \begin{bmatrix} \mathbf{R}^{-1} & 0 & 0 \\ 0 & \mathbf{Q}^{-1} & 0 \\ 0 & 0 & \mathbf{\Pi} \end{bmatrix};$$

then by taking the negative logarithm of (6) we get a proportional non-linear least squares problem

$$\begin{aligned} \ell(x) &= \frac{1}{2}\left(g_z(\mathbf{x})^T \mathbf{R}^{-1} g_z(\mathbf{x}) + g_f(\mathbf{x})^T \mathbf{Q}^{-1} g_f(\mathbf{x}) + g_\pi(\mathbf{x})^T \mathbf{\Pi} g_\pi(\mathbf{x})\right) \\ &= \frac{1}{2}\left(g(\mathbf{x})^{\mathbf{T}} \mathbf{C}^{-\mathbf{1}} \mathbf{g}(\mathbf{x})\right). \end{aligned} \tag{7}$$

If we let $\mathbf{S}^T\mathbf{S} = \mathbf{C}^{-1}$ and $r(\mathbf{x}) = \mathbf{S}g(\mathbf{x})$ then (7) is clearly a non-linear least squares problem of the form

$$\ell(\mathbf{x}) = \frac{1}{2}||r(\mathbf{x})||^2. \tag{8}$$

Newton's solution to such optimization problems is the iterative sequence

$$\mathbf{x}_{i+1} = \mathbf{x}_i - (\nabla^2 \ell(\mathbf{x}_i))^{-1} \nabla \ell(\mathbf{x}_i). \tag{9}$$

For small residual problems a useful approximation to (9) is the Gauss-Newton method, which approximates the Hessian $\nabla^2 \ell(\mathbf{x}_i)$ by $r'(\mathbf{x}_i)^T r'(\mathbf{x}_i)$. Thus, since

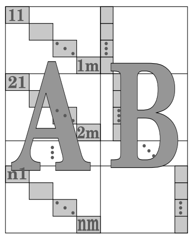


Figure 1: Basic structure of the sensor model Jacobian, $\mathbf{H}$.

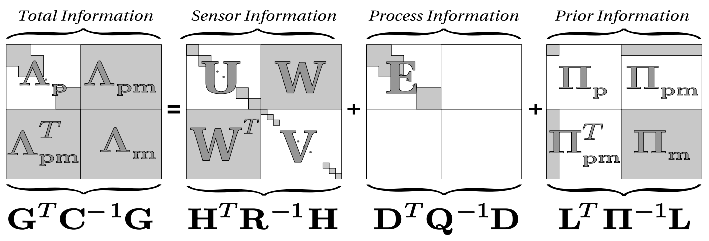


Figure 2: The sparse structure of least squares SLAM system matrix is due to contributions from three components: the measurement block $\mathbf{H}^T\mathbf{R}^{-1}\mathbf{H}$, the process block $\mathbf{D}^T\mathbf{Q}^{-1}\mathbf{D}$, and the prior information block $\mathbf{L}^T\mathbf{\Pi}\mathbf{L}$.

the gradient of (8) is $\nabla\ell(\mathbf{x}_i) = r'(\mathbf{x}_i)^T r(\mathbf{x}_i)$, the Gauss-Newton method defines the sequence of iterates[6]

$$\mathbf{x}_{i+1} = \mathbf{x}_i - (r'(\mathbf{x}_i)^T r'(\mathbf{x}_i))^{-1} r'(\mathbf{x}_i)^T r(\mathbf{x}_i) \qquad (10)$$

Noting that $r'(\mathbf{x}_i) = \mathbf{S}\mathbf{G}_i$ where $\mathbf{G}_i$ is the Jacobian of $g(\mathbf{x}_i)$, (10) becomes

$$\delta\mathbf{x}_i = (\mathbf{G}_i^T\mathbf{C}^{-1}\mathbf{G}_i)^{-1}\mathbf{G}_i^T\mathbf{C}^{-1}g(\mathbf{x}_i). \qquad (11)$$

such that $\mathbf{x}_{i+1} = \mathbf{x}_i + \delta\mathbf{x}_i$. When iterated, this sequence is locally q-quadratically convergent to the MAP estimate for near zero-residual problems[6]. The system of linear equations

$$\mathbf{G}_i^T\mathbf{C}^{-1}\mathbf{G}_i\delta\mathbf{x}_i = \mathbf{G}_i^T\mathbf{C}^{-1}g(\mathbf{x}_i) \qquad (12)$$

is the essential least squares form of the SLAM problem. The difference between many SLAM algorithms can be boiled down to differences in how these equations are solved. It is also interesting to note here that for many problems the Gauss-Newton method is algebraically identical to the Iterated Extended Kalman Filter. In fact, as will become clear later, the Sliding Window Filter over single time step and all landmarks is exactly the IEKF SLAM solution.

As $\delta\mathbf{x}_i \to 0$ the term $(\mathbf{G}_i^T\mathbf{C}^{-1}\mathbf{G}_i)$ converges to the Hessian of the likelihood function (this is the approximation the Gauss-Newton method makes over the full second order Newton method[6]). It turns out that for maximum-likelihood estimation with normal distributions (or MAP estimation when prior information is included like in our formulation) the term $(\mathbf{G}_i^T\mathbf{C}^{-1}\mathbf{G}_i)$ is also the Fisher Information matrix, and it's inverse approximates the system covariance.

3 Sparsity in the System Equations

Before describing the Sliding Window Filter it is useful to take a look at the overall structure of the SLAM least squares equations, and to study how this structure lends itself to various algebraic solutions.

Expanding the Jacobian $\mathbf{G}$,

$$\mathbf{G} = \begin{bmatrix} \frac{\partial g_z}{\partial \mathbf{x}} \\ \frac{\partial g_f}{\partial \mathbf{x}} \\ \frac{\partial g_\pi}{\partial \mathbf{x}} \end{bmatrix} = -\begin{bmatrix} \mathbf{H} \\ \mathbf{D} \\ \mathbf{L} \end{bmatrix},$$

we see that the system matrix, $\mathbf{G}^T\mathbf{C}^{-1}\mathbf{G} = \mathbf{H}^T\mathbf{R}^{-1}\mathbf{H}+\mathbf{D}^T\mathbf{Q}^{-1}\mathbf{D}+\mathbf{L}^T\mathbf{\Pi}\mathbf{L}$, has a sparse structure due to the form of $\mathbf{D}$, $\mathbf{L}$ and especially $\mathbf{H}$. The structure of $\mathbf{H}$ is shown in Fig. 1. The sparsity pattern of least squares SLAM system matrix is due to contributions from the three components

$$\mathbf{H}^T\mathbf{R}^{-1}\mathbf{H}=\begin{bmatrix} \mathbf{U} & \mathbf{W} \\ \mathbf{W}^T & \mathbf{V} \end{bmatrix}, \quad \mathbf{D}^T\mathbf{Q}^{-1}\mathbf{D}=\begin{bmatrix} \mathbf{E} & 0 \\ 0 & 0 \end{bmatrix}, \quad \text{and } \mathbf{L}^T\mathbf{\Pi}\mathbf{L}=\begin{bmatrix} \mathbf{\Pi_p} & 0 & \cdots & \mathbf{\Pi_{pm}} \\ 0 & 0 & & 0 \\ \vdots & 0 & \ddots & \vdots \\ \mathbf{\Pi_{pm}}^T & 0 & \cdots & \mathbf{\Pi_m} \end{bmatrix}$$

where $\mathbf{U} = \mathbf{A}^T\mathbf{R}^{-1}\mathbf{A}$, $\mathbf{W} = \mathbf{A}^T\mathbf{R}^{-1}\mathbf{B}$ and $\mathbf{V} = \mathbf{B}^T\mathbf{R}^{-1}\mathbf{B}$. This structure is also depicted graphically in Fig. 2. The task is to solve the system of normal equations 10 which expand to

$$\begin{bmatrix} \mathbf{\Lambda_p} & \mathbf{\Lambda_{pm}} \\ \mathbf{\Lambda_{pm}}^T & \mathbf{\Lambda_m} \end{bmatrix} \begin{bmatrix} \delta\mathbf{x_p} \\ \delta\mathbf{x_m} \end{bmatrix} = \begin{bmatrix} \mathbf{g_p} \\ \mathbf{g_m} \end{bmatrix}$$

where $\mathbf{g_p}$ and $\mathbf{g_m}$ are the least squares RHS vector corresponding to the robot path and map, respectively. We solve this system of equations using elementary matrix operations – e.g. the Schur complement (Appendix A) - to *reduce* the lower right map block $\mathbf{\Lambda_m}$ onto the upper left process block $\mathbf{\Lambda_p}$

$$\begin{bmatrix} \mathbf{\Lambda_p} - \mathbf{\Lambda_{pm}}(\mathbf{\Lambda_m})^{-\mathbf{1}}\mathbf{\Lambda_{pm}}^{\mathbf{T}} & 0 \\ \mathbf{\Lambda_{pm}}^T & \mathbf{\Lambda_m} \end{bmatrix} \begin{bmatrix} \delta\mathbf{x_p} \\ \delta\mathbf{x_m} \end{bmatrix} = \begin{bmatrix} \mathbf{g_p} - \mathbf{\Lambda_{pm}}(\mathbf{\Lambda_m})^{-\mathbf{1}}\mathbf{g_m} \\ \mathbf{g_m} \end{bmatrix}$$

which is solved directly for $\delta\mathbf{x_p}$ and then for $\delta\mathbf{x_m}$ by back-substitution:

$$\begin{aligned} \delta\mathbf{x_p} &= (\Lambda_p - \mathbf{\Lambda_{pm}}(\mathbf{\Lambda_m})^{-\mathbf{1}}\mathbf{\Lambda_{pm}}^{\mathbf{T}})^{-\mathbf{1}}(\mathbf{g_p} - \mathbf{\Lambda_{pm}}(\mathbf{\Lambda_m})^{-\mathbf{1}}\mathbf{g_m}) \\ \delta\mathbf{x_m} &= (\mathbf{\Lambda_m})^{-\mathbf{1}}(\mathbf{g_m} - \mathbf{\Lambda_{pm}}^{\mathbf{T}}\delta\mathbf{x_p}) \end{aligned}$$

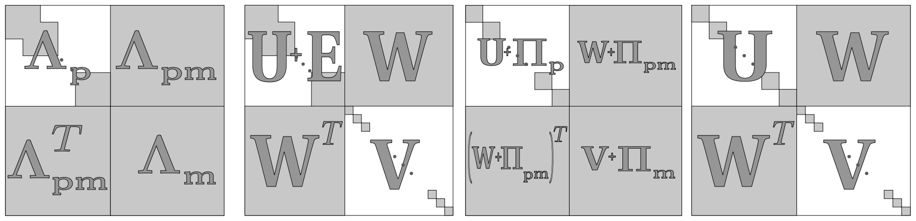
Alternately, we can also reduce the upper left process block $\mathbf{\Lambda_p}$ onto the lower right map block $\mathbf{\Lambda_m}$

$$\begin{bmatrix} \mathbf{\Lambda_p} & \mathbf{\Lambda_{pm}} \\ 0 & \mathbf{\Lambda_m} - \mathbf{\Lambda_{pm}}^{\mathbf{T}}(\mathbf{\Lambda_p})^{-1}\mathbf{\Lambda_{pm}} \end{bmatrix} \begin{bmatrix} \delta\mathbf{x_p} \\ \delta\mathbf{x_m} \end{bmatrix} = \begin{bmatrix} \mathbf{g_p} \\ \mathbf{g_m} - \mathbf{\Lambda_{pm}}^T(\mathbf{\Lambda_p})^{-\mathbf{1}}\mathbf{g_p} \end{bmatrix}$$

giving the solution

$$\begin{aligned} \delta\mathbf{x_m} &= (\mathbf{\Lambda_m} - \mathbf{\Lambda_{pm}}^{\mathbf{T}}(\mathbf{\Lambda_p})^{-1}\mathbf{\Lambda_{pm}})^{-\mathbf{1}}(\mathbf{g_m} - \mathbf{\Lambda_{pm}}^{\mathbf{T}}(\mathbf{\Lambda_p})^{-1}\mathbf{g_p}) \\ \delta\mathbf{x_p} &= (\mathbf{\Lambda_{pm}})^{-\mathbf{1}}(\mathbf{g_p} - \mathbf{\Lambda_{pm}}\delta\mathbf{x_m}) \end{aligned}$$

Depending on the process noise and the prior, the system matrix $\mathbf{G}^T\mathbf{C}^{-1}\mathbf{G}$ can take on different sparsity patterns that affect the complexity of finding a solution. The possible sparsity patterns are shown in figure 3. In the field, the problem at hand will define the sparsity pattern, which will influence the choice of which algorithm to use. For instance, an infinite process noise covariance



(a) With motion model; with prior information. This is the structure of the full SLAM problem in this paper. (b) With motion model; Without prior information. This is the structure of GraphSLAM[19]. (c) No motion information; with prior information. (d) No motion information; No prior information. This is the structure of Photogrammetric Bundle Adjustment[20].

Figure 3: The pattern of the system matrix in the SLAM problem depends on information contributed (or not) from the dynamical process model and any prior information. With no process model information and no prior information the problem is equivalent to Photogrammetric Bundle Adjustment (the structure of this problem is visible on the right). Including a process model makes the upper left $m \times m$ "process block" tridiagonal. Including a prior can potentially cause complete fill-in of the lower right $n \times n$ "map-block".

would mean the motion model does not contribute information to the system ($\mathbf{Q}^{-1} = 0 \implies \mathbf{E} = 0$), which would reduce the upper left block of the system matrix to block diagonal, which is $O(m + n^3)$ to solve. Similarly, without prior information ($\mathbf{\Pi} = 0$) the system matrix lower right block is also block diagonal, which is $O(m^3 + n)$ to solve. Without information from the motion model and without prior information the problem is equivalent to the Bundle Adjustment problem in Photogrammetry[3], which can be solved in either $O(m^3 + n)$ or $O(m + n^3)$. It is interesting to note that in this form (no motion model, no prior), the first optimal solution to the SLAM problem using cameras appears to have been developed by Brown circa 1958[2]. Brown was also the originator of what has come to be known as the Tsai camera model[3, 21].

4 The Sliding Window Filter

For SLAM to operate usefully over long periods of time, perhaps over the course of a robots life time, its computational complexity must be $O(1)$ - hence the size of parameter vector cannot grow without bound.

The simplest way to bound computational complexity is to reduce the size of the state vector by, say, removing the oldest pose parameters or distant landmark parameters. If we directly remove parameters from the system equation however, we can lose information about how the parameters interact. The right way to remove parameters from a multi-dimensional normal distribution is to marginalize them out.

4.1 Downdating

Analogously to the how the sparse SLAM equations are solved via reduction and back-substitution, downdating uses the Schur Compliment to reduce the parameters of interest onto a smaller system, which is then solved for the remaining parameters. This is equivalent to marginalizing out the parameters we wish to remove[10]. Considering the effects of the Schur Compliment, it is not surprising that marginalizing out parameters induces conditional dependencies between all the remaining parameters that the removed parameters were conditionally dependent on.

Thus, at a minimum, downdating a set of pose parameters will add cross-information terms between all the landmarks that were visible from that pose. This is depicted graphically in Fig. 4 for a system that starts *without* any prior information. From this we see that it is important to consider how downdating affects the system of equations. It is well known that the information matrix can be interpreted as an adjacency matrix of an undirected graph where non-zero off-diagonal entries encode conditional dependencies between parameters. The i^{th}, j^{th} element of the $\mathbf{\Lambda_{pm}}$ matrix encodes pose-to-map conditional dependencies and is non-zero only if the i^{th} map landmark was visible from the j^{th} pose.

Studying this structure we see that downdating the oldest pose causes fill-in in three places: 1) between any landmarks that were visible from the downdated pose, 2) between the parameters of the next-oldest-pose (the pose one time step after the pose being downdated), and 3) between the next-oldest-pose and all landmarks seen by the downdated pose. *Notice that only* $\mathbf{\Pi}$ *experiences additional fill-in.*

This is important because it means that the upper left block of the full SLAM problem is still block tri-diagonal, and the $\mathbf{\Lambda_{pm}}$ block is only changed across the top 6 rows - exactly where it overlaps with $\mathbf{\Pi_{pm}}$. Hence, when solving we can still take advantage of any sparsity patterns that may exist in $\mathbf{\Lambda_{pm}}$, just like as in Bundle Adjustment (see the Appendix in[9] for more on this exploit). Note that the patterns in $\mathbf{\Lambda_{pm}}$ are problem specific, depending on how the robot went about observing the environment. For instance, a robot sentry that always sees the same landmarks will have a completely dense $\mathbf{\Lambda_{pm}}$ block, while for a robot exploring and sensing new landmarks, it will have a sparse banded pattern. This fact can hamper methods that rely on predictability in $\mathbf{\Lambda_{pm}}$ in order to achieve computational efficiency[5].

When landmarks are observed from a pose it adds pose-to-landmarks conditional dependence information to the system matrix. This information encodes a soft spatial rigidity between the parameters. Downdating a pose preserves this soft spatial rigidity by transferring its structure into a map of conditionally dependent landmarks.

By downdating poses we have succeeded in removing the $O(m^3)$ cost of carrying the complete robot path in the state estimate. Interestingly, this process is in some sense the opposite of sparsification in Sparse Extended Information Filters[18] - one might call it map “densification” - it is also the opposite of what is done in Delayed State Filters[7].

The next thing is to bound the growth of the map by a constant, which we will do by downdating landmarks that are no longer visible from any pose currently in the state vector. To downdate landmark parameters the parameters to be removed are first manipulated into the upper-left block of the system equations via elementary matrix manipulations. The fact that the landmark we are downdating is no longer visible is crucial because except for the oldest pose-to-map terms, all the cross-information terms in $\mathbf{\Lambda_{pm}}$ will be zero, which means that the Schur complement onto the remaining system will once again

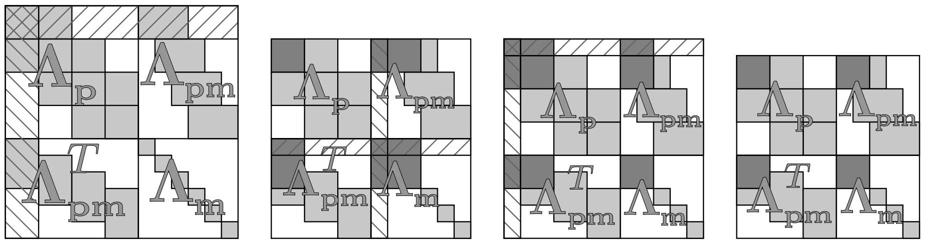


Figure 4: Evolution of the system matrix for a toy problem with 4 poses and 6 landmarks. On the left is the system matrix after measuring landmarks 1, 2, 3 from pose 1, landmarks 2, 3, and 4 from pose 2, landmarks 3, 4, and, 5 from pose 3 and 4, 5, and 6 at pose 4. This is depicted graphically in Fig. 5. Marginalizing out pose 1 via downdating induces conditional dependencies (fill-in) in three places: 1) the top left 6×6 of the process-block, 2) the prior map-block $\mathbf{\Pi_m}$ between landmarks that were visible from pose 1, and 3) the prior pose-to-map block $\mathbf{\Pi_{pm}}$ between landmarks that were visible from pose 1. These places are shaded in darker grey. At this point downdating landmark 1, which is not visible from any of the remaining poses, will induce no extra fill-in in $\mathbf{\Pi}$.

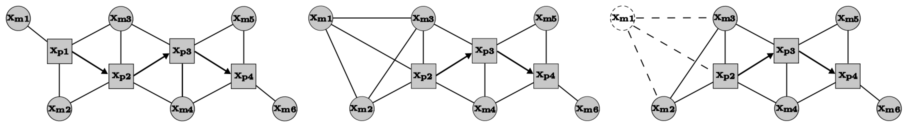
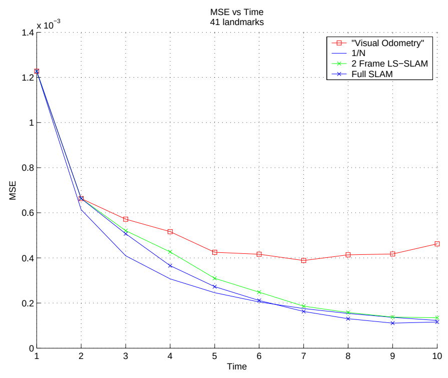
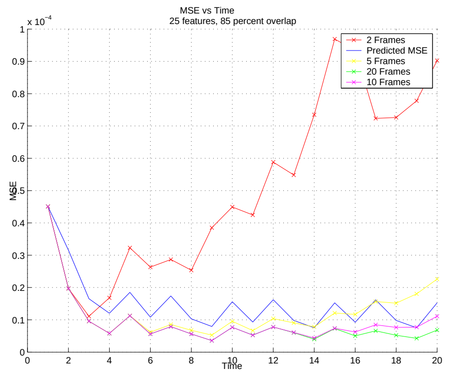


Figure 5: Graph interpretation of the downdating example described in Fig. 4. On the left is the initial system, the middle image shows the result of downdating the first pose, the right image shows the result of downdating the first landmark - downdating these "invisible" landmarks does not cause additional fill-in. The dashed lines indicate parameters that are stored but no longer in the estimator. Saving the map graph structure for later use, say to aid in closing loops, requires only $O(n)$ storage.



(a) Sliding Window Filter comes close to the Full SLAM solution. Sliding Window of 2 is close to optimal $1/k$ full batch curve. Each curve is a trial for different size time window, averaged over 50 Monte-Carlo trials, with 0.1 pixel std.dev measurement noise. 1.0m std.dev process noise. Because VO does not combine information over time, it does not reduce uncertainty as time passes.



(b) MSE vs Time with the cameras in forward motion over 1m. Each frame overlaps ∼85 with ∼25 features per frame. The saw-tooth pattern stems from the fact that new features will cause the MSE to jump up. The 2 frame SWF suffers from marginalizing information out too quickly.

only affect the prior, $\mathbf{\Pi}$.

With a sensor that has a limited range and under the reasonable condition that the spatial density of landmarks in the world is roughly uniform, then the *number of pose-to-landmark information terms is constant*. Downdating poses at a fixed rate and landmarks when they become "invisible" results in a constant time complexity SLAM estimation algorithm. Unlike other techniques this is true *regardless* of how the robot explores the environment[8, 19].

The key points here are that 1) downdating can preserve information by transforming the way in which the system probability distribution is represented; 2) downdating both pose and "invisible" landmark parameters only ever affects the system prior, and not the general sparsity pattern of the system equations - the prior terms $\hat{\mathbf{x}}$ and $\mathbf{\Pi}$ "catch" all the information we marginalize out.

Note that just by choosing when to downdate poses and landmarks the Least Squares SLAM algorithm can scale from the full Batch solution, to the Extended Kalman Filter solution, to the incremental Visual Odometry solution. That all these algorithms are subsumed within one framework testifies to the generality of the simple least squares approach.

In theory, Sliding Window Filters offer a method to achieve the optimal all-time MAP estimate at reduced complexity, potentially even at constant complexity. A key question that remains to be answered is how much information is lost when we transform the system p.d.f. via downdating? If we can quantify this information loss then we can possibly show a bounded equivalence between the full SLAM solution and a constant time approximation based on downdating. For now we show simulation and empirical evidence that the information loss is negligible.

It is interesting to note what happens if we simply delete parameters from the estimator instead of marginalizing them out. For a sliding window of size k, the error converges like $1/k$ just as we would expect the batch estimator to do. However, after k steps, the error stops converging as we delete information from the back of the filter. With such deleting and a sliding window of $k = 1$ it is interesting to note that we end up with a solution that is nearly identical to previous forms of Visual Odometry[12, 14, 15] (though we should expect slightly better results since the state estimate of the landmarks is being improved with each observation, that is, there is some data fusion, even with "deleting"). The graph in Fig. 4.1 shows the MSE performance for this type of Visual Odometry compared to the batch solution, as well as the Sliding Window Filter solution.

5 Optimality and Efficiency

If (10) converges to the global minimum of $\ell(\mathbf{x})$ then the resultant $\mathbf{x}$ is the minimal variance *MAP* estimate [6]. This is the "best" or "optimal" estimate possible in the non-linear least squares sense with normally distributed measurements. Unlike the Extended Kalman Filter, the Gauss-Newton method has a known convergence theory, which, for near zero-residual problems like SLAM, is locally q-quadratically convergent[6]. The signature of an optimal estimator is a $1/k$ reduction in mean squared error of the state estimate vs. ground truth, where here k is the number of time steps.

Another important factor to address is estimator efficiency, that is, how well the estimator approximates a minimal variance estimate of the parameters. The information inequality, $\mathrm{cov}_{\mathbf{x}}(\mathbf{x}) \geq \mathcal{I}(\mathbf{x})^{-1}$, defines the minimal variance bound, which is called Cramer-Rao lower bound[4]. Here the matrix $\mathcal{I}(\mathbf{x})$ is the Fisher information matrix defined by the symmetric matrix whose i^{th}, j^{th} element is the covariance between first partial derivatives of the log-likelihood function (7)

$$\mathcal{I}(\mathbf{x})_{,i,j} = \mathrm{cov}_{\mathbf{x}}\left(\frac{\partial \ell}{\partial \mathbf{x}_i}, \frac{\partial \ell}{\partial \mathbf{x}_j}\right) \tag{13}$$

For a multivariate normal distribution (13) reduces to $\mathcal{I}_{i,j}(\mathbf{x}) = \mathbf{G}^T\mathbf{C}^{-1}\mathbf{G}$, which is equivalent to the Hessian matrix in the Gauss-Newton method. This is why in least squares problems like SLAM the Hessian (and inverse covariance) is the Information Matrix[17]. Iterative batch solutions to non-linear problems generally give better parameter estimates than both non-iterative methods and first order recursive methods. For example, because they do not iterate, methods like the Extended Kalman Filter have no guarantee of converging to the local cost minimum of the objective function, while iterative techniques like the Gauss-Newton method have a well established convergence theory[6]. For convergence, one would have to use the iterative Extended Kalman Filter (which is algebraically equivalent to the Gauss-Newton method[1]). Iteration is fundamental.

In SLAM, batch estimators are beneficial because new observations can often improve *old* pose estimates, which can in turn lead to better estimate for the current pose. For instance, new landmark measurements can help estimate not only where the current pose, but also previous poses. Recursive methods marginalize out old poses and hence cannot improve previous pose estimates. On the other hand, batch methods maintain the old pose information necessary to re-evaluate and re-linearize the objective function using all measurements, which improves a significant portion of the robot path estimate. Estimating the robot path and re-considering past measurements both lead to better (smaller residual, less variance) parameter estimates. Batch methods also produce smooth trajectories (hence techniques that optimize over more than one time step are called *smoothers*).

The covariance estimate in Recursive Least Squares like the EKF is based on the assumption that second derivative terms in the true Hessian are negligible. The closer the parameter estimate to the true value the better this covariance approximation will be. Since iterative batch least squares solutions give better parameter estimates, the approximated covariance matrix (and information matrix) is also more accurate. In the non-linear least squares framework, estimating over all the parameters in the system, considering the entire measurement history, and iterating the update are crucial for attaining an optimal, efficient and consistent estimate.

6 Related Work

This work was inspired by the results from the Photogrammetry community, dating back to the late 1950's[2], and later derivatives like the Variable State Dimension Filter[13], Visual Odometry[12], and of course SLAM[16].

A Schur Complement, Gaussian Marginals, and the Matrix Inverse Lemma

In this section we look at some properties of block matrices and how they are used in recursive filtering [11]. First we look at the problem of computing the inverse of a matrix in terms its sub-matrices. This derivation leads to the Matrix Inversion Lemma, and makes use of an operation called the Schur Complement, which for normal distributions in Canonical form turns out to be equivalent to marginalization.

The Matrix Inverse Lemma and Marginalization via the Schur Complement are useful for understanding recursive linear estimation theory. For instance, deriving the Kalman Filter from Bayes rule is greatly simplified by reference to the Matrix Inversion Lemma, and the update step in state estimation is an application of marginalization.

Let us say $\mathbf{M}$ is a large matrix

$$\mathbf{M} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}$$

that we wish to invert, and we know that $\mathbf{A}$ and $\mathbf{D}$ are square and invertable. The first thing is to notice the two following simple matrix multiplications that allow us to triangularize $\mathbf{M}$: first, the following left multiplication creates an upper-right triangular system,

$$\begin{bmatrix} \mathbf{I} & \mathbf{0} \\ -\mathbf{C}\mathbf{A}^{-1} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{0} & \Delta_{\mathbf{A}} \end{bmatrix}$$

and the following right multiplication creates a lower-left triangular system,

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} \begin{bmatrix} \mathbf{I} & -\mathbf{A}^{-1}\mathbf{B} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} = \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{C} & \Delta_{\mathbf{A}} \end{bmatrix}$$

The term $\Delta_{\mathbf{A}} = \mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B}$ is called the *Schur Complement of* $\mathbf{A}$ *in* $\mathbf{M}$. Similarly, we can complement $\mathbf{D}$ instead of $\mathbf{A}$,

$$\begin{bmatrix} \mathbf{I} & -\mathbf{B}\mathbf{D}^{-1} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} = \begin{bmatrix} \Delta_{\mathbf{D}} & \mathbf{0} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}$$

and

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ -\mathbf{D}^{-1}\mathbf{C} & \mathbf{I} \end{bmatrix} = \begin{bmatrix} \Delta_{\mathbf{D}} & \mathbf{B} \\ \mathbf{0} & \mathbf{D} \end{bmatrix}$$

where $\Delta_{\mathbf{D}} = \mathbf{A} - \mathbf{B}\mathbf{D}^{-1}\mathbf{C}$ is the Schur Complement of $\mathbf{D}$ in $\mathbf{M}$.

Combining the above gives two different ways to block diagonalize $\mathbf{M}$:

$$\begin{bmatrix} \mathbf{I} & \mathbf{0} \\ -\mathbf{C}\mathbf{A}^{-1} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} \begin{bmatrix} \mathbf{I} & -\mathbf{A}^{-1}\mathbf{B} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} = \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \Delta_{\mathbf{A}} \end{bmatrix}$$

and

$$\begin{bmatrix} \mathbf{I} & -\mathbf{B}\mathbf{D}^{-1} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ -\mathbf{D}^{-1}\mathbf{C} & \mathbf{I} \end{bmatrix} = \begin{bmatrix} \mathbf{D} & \mathbf{0} \\ \mathbf{0} & \Delta_{\mathbf{D}} \end{bmatrix}.$$

Using these we can re-express the original matrix $\mathbf{M}$ in terms of a lower-left block triangular component, a block diagonal component, and an upper-right block triangular component. That is,

$$\begin{aligned} \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} &= \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{C}\mathbf{A}^{-1} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \Delta_{\mathbf{A}} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{A}^{-1}\mathbf{B} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{I} & \mathbf{B}\mathbf{D}^{-1} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{D} & \mathbf{0} \\ \mathbf{0} & \Delta_{\mathbf{D}} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{D}^{-1}\mathbf{C} & \mathbf{I} \end{bmatrix}. \end{aligned}$$

which greatly simplifies computing the inverse since the middle term is block diagonal. For instance,

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}^{-1} = \begin{bmatrix} \mathbf{I} & -\mathbf{A}^{-1}\mathbf{B} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{A}^{-1} & \mathbf{0} \\ \mathbf{0} & \Delta_{\mathbf{A}}^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ -\mathbf{C}\mathbf{A}^{-1} & \mathbf{I} \end{bmatrix} \tag{A.2}$$

$$= \begin{bmatrix} \mathbf{A}^{-1} + \mathbf{A}^{-1}\mathbf{B}\Delta_{\mathbf{A}}^{-1}\mathbf{C}\mathbf{A}^{-1} & -\mathbf{A}^{-1}\mathbf{B}\Delta_{\mathbf{A}}^{-1} \\ -\Delta_{\mathbf{A}}^{-1}\mathbf{C}\mathbf{A}^{-1} & \Delta_{\mathbf{A}}^{-1} \end{bmatrix} \tag{A.3}$$

and equivalently,

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}^{-1} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ -\mathbf{D}^{-1}\mathbf{C} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \Delta_{\mathbf{D}}^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{D}^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{I} & -\mathbf{B}\mathbf{D}^{-1} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \tag{A.4}$$

$$= \begin{bmatrix} \Delta_{\mathbf{D}}^{-1} & -\Delta_{\mathbf{D}}^{-1}\mathbf{B}\mathbf{D}^{-1} \\ -\mathbf{D}^{-1}\mathbf{C}\Delta_{\mathbf{D}}^{-1} & \mathbf{D}^{-1} + \mathbf{D}^{-1}\mathbf{C}\Delta_{\mathbf{D}}^{-1}\mathbf{B}\mathbf{D}^{-1} \end{bmatrix} \tag{A.5}$$

Equating various terms of (A.3) and (A.5) yeild the differnt forms of the Matrix Inverse Lemma, one of which is

$$(\mathbf{A} - \mathbf{B}\mathbf{D}^{-1}\mathbf{C})^{-1} = \mathbf{A}^{-1} + \mathbf{A}^{-1}\mathbf{B}(\mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B})^{-1}\mathbf{C}\mathbf{A}^{-1}.$$

This lemma is one of the primary tricks used for manipulating linear least squares equations (for instance, it is used to get betwen the Gauss-Newton method and the Kalman filter). The Schur Complement is the same as downdating, when applied to an inverse covariance matrix it is equivalent to Marginalizing a normal distribution [10].

B Spatial Operations

This section discusses two tools for reasoning about spatial relationships, namely the *compound* and *inverse* operators. Generally, objects are represented by a pose vector which is an entity comprised of a 3D position and an orientation. A pose always has 6 degrees of freedom, though it may have more elements (i.e. a quaternion has 4 parameters to represent 3 degrees of rotational freedom, giving a pose vector with 7 elements). This detail has consequences. For example, it is worth remembering that all 3-parameter representations of orientation will have a singularity associated with them (e.g. gimbal-lock for Euler angles), and all 4 or more parameter representations are redundant, leading to other problems, such as rank deficient covariance matrices.

In general there is no one "best" representation of orientation, and it is often up to the practitioner to decide what is the right one for the application. One approach is to use quaternions to represent pose globally, and then use Euler angles to represent small local changes. This gets the best of both worlds: a non-singular global representation and a minimal local representation.

Note that using local parameterizations like this buys us some of the benefits enjoyed by so-called error state Kalman filters, namely that the least squares optimization takes place around a state vector whose optimal solution will be near zero. In our case this has the effect of keeping the optimization machinery far away from the singularities of SO^3.

Therefore *local* pose, $\mathbf{x}_{ij}$ is represented by a six parameter column vector comprised of a 3D point, $\mathbf{p}_{ij} = [x_{ij}\ y_{ij}\ z_{ij}]^T$ and a 3 element Euler Roll-Pitch-Yaw angle $\theta_{ij} = [r_{ij}\ p_{ij}\ q_{ij}]^T$. The $+x$-axis is forward, $+y$-axis is right and $+z$-axis is down; roll is about x-axis, pitch about y-axis and yaw is about z-axis. Dropping the subscripts and abbreviating sin and cos, a rotation matrix is parameterized by θ such that $\mathbf{R}_\theta = \mathbf{RzRyRx}$. where

$$\mathbf{R_x} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \mathbf{c}r & -\mathbf{s}r \\ 0 & \mathbf{s}r & \mathbf{c}r \end{bmatrix}, \mathbf{R_y} = \begin{bmatrix} \mathbf{c}p & 0 & \mathbf{s}p \\ 0 & 1 & 0 \\ -\mathbf{s}p & 0 & \mathbf{c}p \end{bmatrix}, \mathbf{R_z} = \begin{bmatrix} \mathbf{c}q & -\mathbf{s}q & 0 \\ \mathbf{s}q & \mathbf{c}q & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Say, for example, that $\mathbf{x}_{wr}$ denotes the robot pose in the world coordinate frame and $\mathbf{x}_{rs}$ denotes a sensor in the robot coordinate frame. The compound operator, $\oplus$, can be used to find the sensor pose in the world coordinate frame, e.g. $\mathbf{x}_{ws} = \mathbf{x}_{wr} \oplus \mathbf{x}_{rs}$ (notice that the two r's are adjacent).

More generally, given two spatial relationships, $\mathbf{x}_{ij}$ and $\mathbf{x}_{jk}$ the compound operator, $\oplus$, gives the pose of object k in coordinate frame i,

$$\mathbf{x}_{jk} = \mathbf{x}_{ij} \oplus \mathbf{x}_{jk} = \begin{bmatrix} \mathbf{R}_{\theta_{ij}} \mathbf{p}_{jk} + \mathbf{p}_{ij} \\ \gamma\Big(\mathbf{R}_{\theta_{ij}} \mathbf{R}_{\theta_{jk}}\Big) \end{bmatrix}, \tag{A.6}$$

where $\mathbf{R}_{\theta_{ij}}$ and $\mathbf{R}_{\theta_{jk}}$ are the 3x3 rotation matrices defined by the Euler angle components of $\mathbf{x}_{ij}$ and $\mathbf{x}_{jk}$, respectively. The function γ maps from a rotation matrix to the appropriate Euler angle

$$\gamma(\mathbf{R}) = \begin{bmatrix} \arctan(\mathbf{R}_{32}/\mathbf{R}_{33}) \\ -\arcsin(\mathbf{R}_{31}) \\ \arctan(\mathbf{R}_{21}/\mathbf{R}_{11} \end{bmatrix}$$

which is well defined for attitude angles between -90^o and 90^o. The compounding operation is associative but not commutative.

The inverse operation, $\ominus$, inverts a spatial relation. For instance $\mathbf{x}_{ij}$ gives object j's pose in i's coordinate frame, and $\ominus\mathbf{x}_{ij}$ gives object i's pose in j's coordinate frame. The inverse operator is

$$\mathbf{x}_{ji} = \ominus\mathbf{x}_{ij} = \begin{bmatrix} -\mathbf{R}_{\theta_{ij}} \mathbf{p}_{ij} \\ \gamma(\mathbf{R}^T_{\theta_{ij}}) \end{bmatrix}$$

Spatial operations are useful in a variety of problems. For instance, in estimation, they simplify writing sensor models and process models. In manipulator robotics, spatial operations can be used for forward and inverse kinematics. The benefit of thinking in terms of operations on spatial relations is the intuitive appeal, and the fact that the underlying linear-algebra does not need to be considered over and over again.

Jacobians of spatial operations are needed in many places - e.g. linear error propagation, Kalman filtering applications, least squares normal equations, non-linear optimization, etc. These Jacobians are involved, but tractable; it is recommended to find them with the aid of symbolic solvers, such as Maple or Mathematica.

References

[1] F.W. Bell, B.M. Cathey. The iterated Kalman filter update as a Gauss-Newton method. *IEEE Transactions on Automatic Control*, 38(2):294–297, Feb 1993.

[2] D.C. Brown. A solution to the general problem of multiple station analytical stereotriangulation. Technical report, RCP-MTP Data Reduction Technical Report No. 43, Patrick Air Force Base, Flordia (also designated as AFMTC 58-8), 1958.

[3] D.C. Brown. The bundle adjustment - progress and prospects. In *XIIIth Congress of the International Society for Photogrammetry*, 1976.

[4] M. H. DeGroot and M. J. Schervish. *Probability and Statistics.* Addison Wesley, 2001.

[5] Frank Dellaert. Square root SAM. In *Proceedings of Robotics: Science and Systems*, 2005.

[6] Jr Dennis J.E. and R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Soc for Industrial & Applied Math, 1996.

[7] R. Eustice, H. Singh, and J. Leonard. Exactly sparse delayed-state filters. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, 2005.

[8] Ryan Eustice, Hanumant Singh, John Leonard, Matthew Walter, and Robert Ballard. Visually navigating the rms titanic with slam information filters. In *Robotics: Science and Systems*, 2005.

[9] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2000.

[10] M.I. Jordan. *An Introduction to Graphical Models*. unpublished, 2003.

[11] T. Kailath, A. H. Sayed, and B. Hassibi. *Linear Estimation*. Prentice Hall, 2000. Good reference for estimation theory. Good appendix on matrix algebra.

[12] L. Matthies and S. Shafer. Error modelling in stereo navigation. *IEEE Journal of Robotics and Automation*, 3(3):239–248, 1987.

[13] Philip F. McLauchlan. The variable state dimension filter applied to surface-based structure from motion. Technical report, University of Surrey, 1999.

[14] David Nister, Oleg Naroditsky, and James Bergen. Visual odometry. In *CVPR*, volume 1, 2004.

[15] Clark F. Olson, Larry H. Matthies, Marcel Schoppers, and Mark W. Maimone. Stereo egomotion improvements for robust rover navigation. In *In Proceedings IEEE Conference on Robotics and Automation (ICRA)*, volume 2, pages 1099– 1104, 2001.

[16] R. C. Smith, M. Self, and P. Cheeseman. Estimating uncertain spatial relationships in robotics. In I. J. Cox and G. T. Wilfong, editors, *Autonomous Robot Vehicles*, pages 167–193. Springer-Verlag, 1990.

[17] H. W. Sorenson. *Parameter Estimation: Principles and Problems*. Marcel Drekker, Inc., 1980.

[18] S. Thrun, D. Koller, Z. Ghahmarani, and H. Durrant-Whyte. Simultaneous mapping and localization with sparse extended information filters: Theory and initial results. In *Workshop on Algorithmic Foundations of Robotics*, 2002. URL `citeseer.nj.nec.com/article/thrun02simultaneous.html`.

[19] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. MIT Press, Cambridge, MA, 2005.

[20] Bill Triggs, Philip McLauchlan, Richard Hartley, and Andrew Fitzgibbon. Bundle adjustment – A modern synthesis. In W. Triggs, A. Zisserman, and R. Szeliski, editors, *Vision Algorithms: Theory and Practice*, LNCS, pages 298–375. Springer Verlag, 2000. URL `citeseer.ist.psu.edu/triggs00bundle.html`.

[21] R. Y. Tsai. A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses. *IEEE Journal of Robotics and Automation*, 3(4):pp 323–344, 1987.