Distributed SLAM with Unknown Measurement Noise using Alternating Direction Method of Multipliers

Gael Colas
Stanford University
Department of Aeronautics and Astronautics
colasg@stanford.edu

Michael Gobble
Stanford University
Department of Aeronautics and Astronautics
mjgobble@stanford.edu

Abstract

Simultaneous Localization and Mapping (SLAM) has become a ubiquitous tool in state estimation. SLAM finds at each time step the most likely position of a robot agent simultaneously with the most likely map of its environment using a set of measurements. First, the authors propose an extension of the Kalman Filter when the measurement noise covariance is unknown: estimating the noise covariance with an exponential moving average initialized from a prior yields robust results. For complicated systems, when the number of measurements grows, this process can be prohibitively expensive to compute. Many methods have been proposed that efficiently perform SLAM. This study describes an efficient method for performing SLAM when the measurement noise covariance matrix is unknown a priori, using the Alternating Direction Method of Multipliers (ADMM). ADMM is a convex optimization technique that decomposes optimization problems of a given form into simpler problems that can be solved more efficiently in parallel. The authors show that this technique can greatly increase the speed and efficiency of SLAM.

1. Introduction

1.1. Problem Statement

SLAM, for Simultaneous Localization and Mapping, refers to the problem of building the map of an unknown environment while estimating the position of a moving robot agent. This can be performed with one of two goals in mind: estimating the most likely series of states during a sequence of steps in the past, or estimating the most likely state at the current time step. The graphical conditional dependence structure of each of these is depicted in Figure 1 from [15] where z denotes measurements.

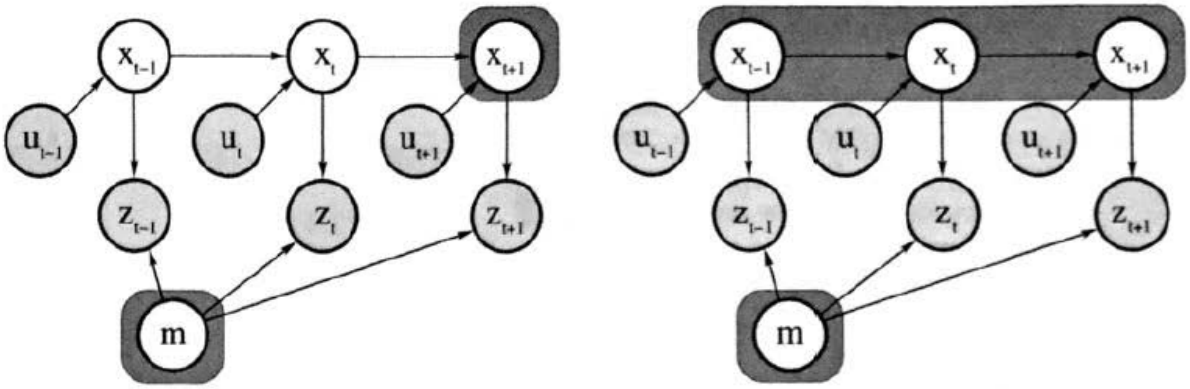

Figure 1. Incremental SLAM (left) and Full SLAM (right).

In class, we saw that the problem of Localization, given a series of measurements $z_{1:t}$ can be framed as a Recursive Bayesian estimation of the state x_t through the belief: $p(x_t|z_{1:t})$. For SLAM, the confidence about the state of the map m_t also needs to be estimated: $p(m_t, x_t|z_{1:t})$. This formulation considers the state of the map to be part of the state of the system, reformulating the SLAM technique as a straightforward application of the Kalman Filter, Extended Kalman Filter or any other Recursive Bayesian estimators [15].

This means that the effectiveness of SLAM can be improved by increasing the effectiveness of these algorithms.

1.2. Related Work

Generally, SLAM can be solved with different variants of Recursive Bayesian estimation. [2] provide a comparison of the filter algorithms' performance at this task. First, the Kalman Filter (KF), or the Extended Kalman Filter (EKF) if the dynamics are not linear, can be used to update the whole belief when the noise can be assumed to be Gaussian [3]. [9] propose an algorithm to estimate both the position of the robot and the map with EKF. Their method builds a map by keeping track of a list of features (or landmarks). These features are detected using the Harris corner detector on images generated with a stereo head. EKF filtering is used to update the belief position of the set of features. However, one of the major drawbacks of applying EKF to the mapping problem is the quadratic cost (in the number K of landmarks) when a landmark is observed to do an update step. This cost comes from the update of the covariance matrix which is of size $K \times K$.

The Particle Filter (PF) has been shown to be an efficient way to perform localization in the case of non-linear, non-Gaussian dynamics. The PF represents the belief state with a collection of particles. When doing mapping, the state space dimension increases each time a new feature is detected. Due to this characteristic, the PF growth in complexity at each new feature detection, making it not suited for map-building. [11] propose an algorithm to efficiently take advantage of the PF properties without its high computational cost when dealing with a large state space. The method, called FastSLAM, assumes the following Bayesian structure for the problem: first we have a localization problem, then the estimation of the landmarks is assumed to be conditionally independent given the position estimate. Their method uses a PF for updating the position belief and K independent KF to update the landmark location. Thus the computational cost is drastically decreased.

Algorithm 1: Kalman Filter

Result: Predict-Update step using incoming measurements
Input: Previous state estimates: mean $\mu_{t-1|t-1}$ and covariance $\Sigma_{t-1|t-1}$
Previous control u_{t-1} and current measurement z_t
Output: Current state estimates: mean $\mu_{t|t}$ and covariance $\Sigma_{t|t}$
Predict Step

$$\mu_{t|t-1} = A_{t-1}\mu_{t-1|t-1} + B_{t-1}u_{t-1}$$
$$\Sigma_{t|t-1} = A_{t-1}\Sigma_{t-1|t-1}A_{t-1}^T + Q_{t-1}$$

Update Step

$$K_t = \Sigma_{t|t-1}C_t^T(C_t\Sigma_{t|t-1}C_t^T + R_t)^{-1}$$
$$\mu_{t|t} = \mu_{t|t-1} + K_t(z_t - C_t\mu_{t|t-1})$$
$$\Sigma_{t|t} = (I - K_tC_t)\Sigma_{t|t-1}$$

return $\mu_{t|t}, \Sigma_{t|t}$

[6] provides an updated survey of more recent approaches. One of the new axis of interest is distributed SLAM algorithms. This idea arises in the case of multi-robot Cooperative Localization. Several robots are navigating a map (unknown in the framework of SLAM) and try to estimate their individual locations. A first approach would be to collect the measurements from all the agents into a fusion center and update the global position belief of all the robots at once. However, the computational cost of updating this large global state, as well as the complexity of ensuring the communication, prohibits the application of this technique for large teams of robots. Instead [12] proposes a method that enables the agents to simultaneously process and update their data, without the need of a fusion center. This is what they call "Distributed Maximum a Posteriori (MAP) Estimation". More recently [10], [1], and [7] all propose variants of this algorithm.

All of these distributed methods require to minimize a cost function to determine the MAP estimate of the robots' individual positions. [12] uses a linear approximation of the cost function. The problem is then solved using a distributed linear solver, ran in parallel by all the robots.

2. Methods

2.1. Kalman Filtering

The core algorithm used in this study is the Kalman Filter. A rigorous derivation of the Kalman Filter is provided by [15]. The Kalman filter algorithm is described in Algorithm 1

The Kalman Filter is used when one wants to compute the most likely state of a system given a prior mean vector, covariance matrix, and measurements vector. It is used when the system dynamics and measurements are linear and Markovian, and when they are perturbed by zero-mean Gaussian noise. Although this class of applicable problems is very general, a large set of practical problems are nonlinear. In this case, one can assume that if the time step is small enough, the system and measurement dynamics are approximately linear, and the dynamics can be found through linearization about the current state estimate. This linearization is performed by computing the Jacobian of the dynamics and measurement

Algorithm 2: Extended Kalman Filter

Result: Predict-Update step using incoming measurements
Input: Previous state estimates: mean $\mu_{t-1|t-1}$ and covariance $\Sigma_{t-1|t-1}$
Previous control u_{t-1} and current measurement z_t
Output: Current state estimates: mean $\mu_{t|t}$ and covariance $\Sigma_{t|t}$
Predict Step

$$\mu_{t|t-1} = f(\mu_{t-1|t-1}, u_{t-1})$$
$$\Sigma_{t|t-1} = A_{t-1}\Sigma_{t-1|t-1}A_{t-1}^T + Q_{t-1}$$

Update Step

$$K_t = \Sigma_{t|t-1}C_t^T(C_t\Sigma_{t|t-1}C_t^T + R_t)^{-1}$$
$$\mu_{t|t} = \mu_{t|t-1} + K_t(z_t - g(\mu_{t|t-1}))$$
$$\Sigma_{t|t} = (I - K_tC_t)\Sigma_{t|t-1}$$

return $\mu_{t|t}, \Sigma_{t|t}$

functions (f and g respectively), then performing the normal Kalman Filter update. This algorithm is called the Extended Kalman Filter and is presented in Algorithm 2 as derived in [15].

Note that if the Jacobian matrices are used to formulate linear dynamics and measurement equations, the covariance update step for the EKF and KF is the same. As this is the most computationally expensive step, techniques to increase the effectiveness of the basic Kalman Filter can also be used for the EKF on nonlinear systems.

As part of the KF derivation in [15], the Kalman Filter update step for the mean vector is defined as a minimization of the unconstrained objective function:

$$J_t = \frac{1}{2}(z_t - C_tx_t)^T R_t^{-1}(z_t - C_tx_t) + \frac{1}{2}(x_t - \mu_{t|t-1})^T\Sigma_{t|t-1}^{-1}(x_t - \mu_{t|t-1})$$

where z denotes measurements, x denotes the state, R denotes the measurement noise covariance matrix, $\mu_{t|t-1}$ denotes the predicted mean, and $\Sigma_{t|t-1}$ denotes the covariance matrix of the predicted state. This objective function is a sum of quadratic objective functions. The state covariance solution involves a matrix inverse, as reflected in Algorithm 1: at the step where the Kalman Gain K is computed. Note that more generally, this objective is a sum of matrix quadratic objective functions. The algorithm of algebraic steps is possible from the fact that there are only two terms in the formula, allowing the solution to be found in one matrix-inversion step.

There are extensions to the Kalman Filter such as Maximum A Posteriori (MAP) estimation, which find the most likely sequence of states for a given a time series of measurements and a prior estimate. The Kalman Filter can also be extended to cases where the noise is described by a multi-modal Gaussian distribution called a Gaussian Mixture Model. These processes are not strictly considered Kalman Filtering, but rather Kalman Filtering is considered a special case of these extensions. In these extensions, the solution is also found by minimizing a sum of matrix quadratic functions. Unfortunately, in many practical applications of these extensions, these sums of quadratic functions can grow large, making the computational scale required to minimize the sum of quadratic functions intractable to perform directly.

[15]

2.2. Convex Optimization

Optimization is the process of finding the input value x that minimizes an objective function $f(x)$. A function can have a global minimum, which is the lowest possible function value, one or more local minima, which are values that are lower than their immediate neighbors but are not global minima, and can be unbounded functions without minima.

A function is convex if the following inequality holds true for all points in the domain

$$\frac{f(a) + f(b)}{2} \geq f(\frac{a+b}{2})$$

Put more simply, the function has positive curvature everywhere. One implication of convexity is that provided the function is bounded and defined on a closed domain, it will have a single global minimum and no local minima. This fact has allowed researchers to find algorithms for efficiently solving convex optimization problems with convergence guarantees. Additionally, this property has enabled a large field of research that focuses on finding general algorithms and solvers that can be applied to any convex optimization problem. When solving convex problems, the issue is often not how to solve to problem, but deciding which technique will solve the problem as quickly and efficiently as possible. [5]

Matrix quadratic objective functions are convex if the matrix is positive definite and their unique solution can be found via matrix inversion.

$$x^T Ax + y^T By + C = \begin{bmatrix} x^T & y^T \end{bmatrix} \begin{bmatrix} A & 0 \\ 0 & B \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + C$$

$$\begin{bmatrix} x^* \\ y^* \end{bmatrix} = - \begin{bmatrix} A & 0 \\ 0 & B \end{bmatrix}^{-1} C$$

While this approach is simple to implement, it may not be the most efficient. If A and B are large and the matrix is inverted using a direct method, then the cost of this matrix inversion will scale with the cube of the size of A and B. Iterative matrix inversion techniques can take advantage of the sparsity structure of the matrix to solve this system of equations in linear time, depending on whether or not A and B are also sparse. These methods are often used with great performance benefits, but they do not necessarily provide parallel computation and fault tolerance.

2.3. Alternating Direction Method of Multipliers

Alternating Direction Method of Multipliers (ADMM) is a convex optimization algorithm that, in the general case, sacrifices efficiency by requiring multiple iterations to complete. However, it becomes possible to decompose an optimization objective function into separate optimization problems, each of which can be solved separately. For some problems, this decomposition allows other efficient algorithms to be used which more than compensate for the sacrifice of efficiency required by performing multiple iterations.

ADMM can be applied to objective functions of the form:

$$J(x) = \sum_i f_i(x)$$

where each $f_i(x)$ is a convex function. For simplicity, we will consider an objective that is the sum of two convex functions:

$$J(x) = f(x) + g(x)$$

This problem can be reformulated as a constrained optimization problem:

$$\begin{aligned} \text{minimize} \qquad & f(x) + g(z) \\ \text{subject to} \qquad & x - z = 0 \end{aligned}$$

Finally the ADMM algorithm to solve this problem is:

$$\begin{aligned} x^{k+1} &:= \underset{x}{\text{argmin}} \quad f(x) + (\rho/2)||x + z^k + u^k||_2^2 \\ z^{k+1} &:= \underset{z}{\text{argmin}} \quad g(z) + (\rho/2)||x^{k+1} + z + u^k||_2^2 \\ u^{k+1} &:= u^k + x^{k+1} + z^{k+1} \end{aligned}$$

where u is a dual variable. With a properly selected ρ, this algorithm can converge in approximately 20 iterations. If the problem is a sum of quadratic functions which, when combined, are solved by inverting a matrix of size N, then this algorithm changes the cost from $O(N^3)$ to approximately $O\left(k(N/k)^3\right)$ assuming the matrix was decomposed evenly into k equal sized matrices. As k approaches N, the complexity of the computation approaches $O(N)$.

This formulation is for a very general type of objective function, but ADMM can be reformulated for many different problem types. Another problem type that is useful in this study is the Consensus Problem. The consensus problem is formulated as:

$$\begin{aligned} \underset{x_i, z}{\text{minimize}} \quad & \sum_i f_i(x_i) \\ \text{subject to} \quad & x_i - z = 0 \end{aligned}$$

In order for this problem to be solved, each subproblem must agree on the minimizer that minimizes the objective overall. This problem can be formulated and solved using an ADMM algorithm of the form:

$$\begin{aligned} x^{k+1} &:= \underset{x_i}{\text{argmin}} \quad f_i(x_i) + y_i^k(x_i - \bar{x}^k) + (\rho/2)||x_i - \bar{x}^k||_2^2 \\ y_i^{k+1} &:= y_i^k + \rho(x_i^{k+1} - \bar{x}^{k+1}) \end{aligned}$$

where $\bar{x}^k$ is the average of the minimizers found by all the subproblems at step k. Again, with an appropriate value of ρ, this algorithm can converge in approximately 20 iterations. It also has notable features such as:

- the first step can be computed in parallel for each i,
- the only information that needs to be passed between the parallel processors is the value of the mean state.

For some problems, the ability to perform parallel computations with minimal communication provides tremendous increase in performance. [4]

2.4. Sparse Inverse Matrix Selection

Another situation where ADMM can provide significant performance increases is when an objective function is difficult to solve directly, but the separate components are easy to solve individually. This is the case with an essential step in this study's improved algorithm: Sparse Inverse Matrix Selection (SIMS). SIMS finds a sparse approximation to the inverse of a covariance matrix R using L_1 norm regularization. The general SIMS optimization problem is formulated as:

$$\text{minimize} \quad \mathbf{Tr}(RX) - \log\det(X) + \lambda||X||_1$$

Where R is the dense covariance matrix. When the L_1 term is included as part of the objective function, this optimization problem is expensive to solve. However, minimizing an L_1 norm by itself is a simple soft thresholding operation:

$$R_a = (f - a)_+ - (-f - a)_+$$

Therefore this procedure can be formulated using ADMM as:

$$X^{k+1} := \underset{X}{\text{argmin}} \quad \mathbf{Tr}(RX) - \log\det(X) + (\rho/2)||X - Z^k + U^k||_2^F$$

$$Z^{k+1} := R_{\lambda/\rho}(X^{k+1} + U^k)$$
$$U^{k+1} := U^k + (X^{k+1} - Z^{k+1})$$

and executed efficiently. λ controls how sparse the resulting covariance matrix inverse is. [13]

3. Motivation

3.1. The issue unknown Measurement Noise

In class, when deriving the Kalman Filter (and similarly the Extended Kalman Filter), we assumed the true measurement noise covariance R could be accessed. However that may not be the case or the measurement noise may evolve through time. What happens if the Kalman Filter algorithm 1 is applied with a bad estimate of the measurement noise covariance R?

The effect of updating our belief with a bad estimate of the measurement noise covariance was analyzed in the SLAM framework. The test case from Problem Set 6, Question 4 was used [14]. A non-holonomic robot moves in a 2D map. It receives range and bearing measurements from $m = 4$ beacons. The dynamics and measurement models are non-linear, so EKF-SLAM was used to estimate the pose of the robot as well as the location of the beacons (the map).

Three different cases were compared: applying the EKF Algorithm with a correct noise covariance $\tilde{R} = R$, with an over-confident noise covariance: $\tilde{R} = 0.01R$ and with an under-confident noise covariance: $\tilde{R} = 100R$. Figure 2 shows the resulting pose beliefs from the three cases for the same trajectory. The 95%-error ellipses were also plotted to evaluate the confidence of the algorithm.

From the second plot, we can observe that using a over-confident noise covariance estimate produces a very noisy trajectory estimate. Worse, it can even lead to fast divergence even though the confidence of the algorithm in its predicted state stays high: the error ellipses stay small. Using an under-confident noise covariance estimate produces a trajectory estimate more robust to divergence. The error ellipses are bigger and include the true trajectory. However, the resulting trajectory estimate is very smooth. The

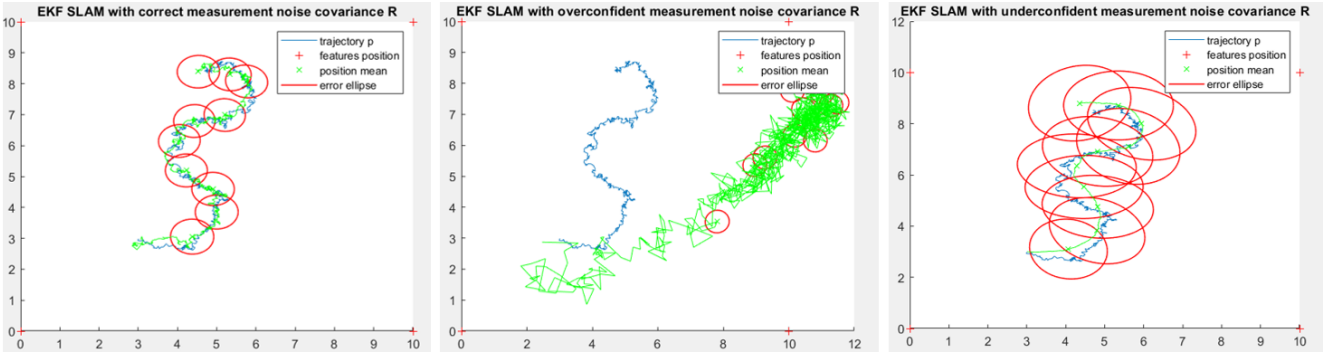

Figure 2. Evolution of the robot pose estimates (green). The true trajectory is represented in blue. The position of the beacons is indicated by red cross. The 95%-error ellipses are represented in red. From left to right: correct, over-confident and under-confident noise covariance estimates.

estimates are sometimes very far from the true trajectory compared to the trajectory estimates produced with the correct noise covariance.

The EKF performance was evaluated according to two metrics. The L1-distance between the true trajectory x_t and the estimated trajectory $\mu_{t|t}$, averaged over the time steps. The divergence of the estimated trajectory: the estimated trajectory is considered to have diverged if the last robot position estimate is outside the 95%-error ellipse. The results were averaged over $n_{sim} = 100$ random simulations. Table 1 summarizes the results. The previous analysis is confirmed: an over-confident measurement noise covariance very often leads to divergence (84% of the simulations), whereas an under-confident one yields smooth trajectories more robust to divergence (2% of the simulations) but far from the true trajectories (average L1-distance twice as big as the one generated from a correct noise covariance).

Table 1. EKF estimates' performance for correct, over-confident and under-confident measurement noise covariances

Measurement Noise	Noise Covariance	Average L1-distance	Proportion of Divergence
Correct covariance	$R = I_8$	0.0077	11%
Over-confident covariance	$\tilde{R} = 0.01R$	0.0336	84%
Under-confident covariance	$\tilde{R} = 100R$	0.0133	2%

3.2. Online Measurement Noise Estimation

To solve the issue of a bad measurement noise estimate, the incoming measurements z can be leveraged to correct our estimate online. More precisely, the measurement noise covariance R can be approximated by the empirical covariance R_t at time t:

$$R_t = \frac{1}{t}\sum_{i=1}^{t}(z_i - h(\mu_{i|i-1}))(z_i - h(\mu_{i|i-1}))^T$$

Two issues arise from this way of estimating the measurement noise. First when the number of time steps t is small, the empirical covariance will be a very noisy estimate of the true covariance ; worse, when the number of time steps is less than the number of measurements, this matrix is singular and

Algorithm 3: Extended Kalman Filter with Online Measurement Noise Estimation

Result: Predict-Update step using incoming measurements
Input: Previous state estimates: mean $\mu_{t-1|t-1}$ and covariance $\Sigma_{t-1|t-1}$
Previous control u_{t-1} and current measurement z_t
Output: Current state estimates: mean $\mu_{t|t}$ and covariance $\Sigma_{t|t}$
Predict Step

$$\mu_{t|t-1} = f(\mu_{t-1|t-1}, u_{t-1})$$
$$\Sigma_{t|t-1} = A_{t-1}\Sigma_{t-1|t-1}A_{t-1}^T + Q_{t-1}$$

Online Noise Estimation

$$R_t = \rho R_{t-1} + (1-\rho)(z_t - h(\mu_{t|t-1}))(z_t - h(\mu_{t|t-1}))^T$$

Update Step

$$K_t = \Sigma_{t|t-1}C_t^T(C_t\Sigma_{t|t-1}C_t^T + R_t)^{-1}$$
$$\mu_{t|t} = \mu_{t|t-1} + K_t(z_t - g(\mu_{t|t-1}))$$
$$\Sigma_{t|t} = (I - K_tC_t)\Sigma_{t|t-1}$$

return $\mu_{t|t}, \Sigma_{t|t}$

cannot be inverted. Second, at time t, t outer products have to be computed. This will explode as the number of time steps increases.

To solve this issue, a different approach was considered. The empirical covariance is computed as an exponential moving average of the estimated measurement error's outer products.

$$R_t = \rho R_{t-1} + (1-\rho)(z_t - h(\mu_{t|t-1}))(z_t - h(\mu_{t|t-1}))^T$$

Where ρ is a hyper-parameter determining the smoothness of the estimated variable: a larger ρ produces a very smooth evolution of the empirical covariance R_t, whereas a smaller ρ puts more weight on the last measurement z_t. In our simulations, $\rho = 0.9$ was found to yield robust results.

With the moving average, only one outer product is computed at each time step. Moreover, the average can be initialized from any prior R_0 and as long as the prior is a valid covariance matrix, R_t will be invertible too. Algorithm 3 presents the updated EKF algorithm incorporating the online measurement noise estimation.

Figure 3 presents the resulting trajectory estimates for both an over-confident (left plot) and an under-confident (right plot) priors. The plots show that the trajectory estimates are very close to the true trajectory and also very close to one another. Even though they were initialized from very different noise covariance priors, the estimated trajectories quickly converge to the same estimate. The error-ellipses also look identical.

The results were averaged over $n_{sim} = 100$ random simulations. Table 2 summarizes the results. The previous analysis is confirmed: the resulting EKF estimates are robust to the choice of prior. Initializing the noise estimate with an over or under-confident prior yields exactly the same performance after $T =$

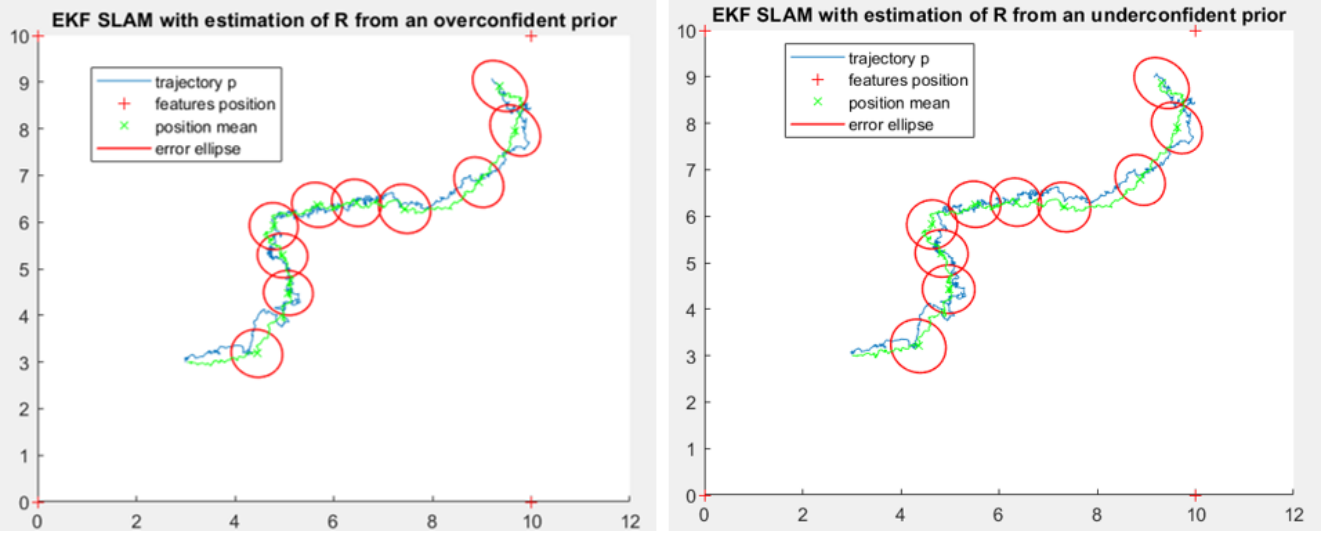

Figure 3. Evolution of the robot pose estimates (green). The true trajectory is represented in blue. The position of the beacons is indicated by red cross. The 95%-error ellipses are represented in red. From left to right: over-confident and under-confident noise covariance priors.

1000 time steps. Furthermore, using the EKF estimates with the correct measurement noise covariance does not yield better results.

Table 2. EKF estimates' performance for over-confident and under-confident measurement noise priors compared with the EKF estimates generated with the correct measurement noise

Measurement Noise	Noise Covariance	Average L1-distance	Proportion of Divergence
Correct covariance	$R = I_8$	0.0074	11%
Over-confident covariance Prior	$R_0 = 0.01R$	0.0065	5%
Under-confident covariance Prior	$R_0 = 100R$	0.0065	5%

Unfortunately, when the number of measurements m increases (for example when the number of map features increases in SLAM), this algorithm collapses. Indeed, at the update-step, the EKF algorithm requires to invert a square matrix of size $m \times m$. This operation is cubic in the number of measurements: $O(m^3)$. It becomes prohibitive as the number of measurements increases. In class, we saw that when the number of measurements is far greater than the state dimension, it becomes useful to solve the filtering process using the Information Filter formulation. However, the Information Filter also requires to invert R_t. This is very cheap when R is stationary as this only needs to be done once. Unfortunately, because our algorithm re-estimates R_t at every time step, it would have to be inverted at every time step. No computation would be saved in using the Information Filter.

This legitimates the idea of using a distributed algorithm that leverages the noise structure to increase the computational speed.

4. Proposed Algorithm

4.1. Sparse Measurement Noise Covariance Estimation

This study applies ADMM and SIMS to the Kalman Filter in order to formulate a computationally-efficient SLAM algorithm. In a SLAM scenario, the robot begins with a prior expectation and covariance of the map and its location. This study assumes that the noise covariance matrix is unknown, and so must be computed empirically:

$$R = \mathbb{E}[(z - g(x))(z - g(x))^T]$$

$$R \approx \frac{1}{T}\sum_{t}^{T}(z_t - g(\mu_{t|t-1}))(z_t - g(\mu_{t|t-1}))^T = \tilde{R}$$

If the noise affecting each measurement is entirely uncorrelated, then this matrix $\tilde{R}$ will converge to a diagonal matrix. However, in a real situation, this matrix will be dense and the off-diagonal entries will fluctuate around 0. If T is small, then it could be entirely unclear whether the true covariance matrix is diagonal or not. Because this matrix is an approximation of the underlying noise covariance matrix R, it is desirable for this matrix to be diagonal or sparse to minimize the computational cost of a Kalman filter update. This sparse approximation of R^{-1} can be found using SIMS.

4.2. Parallel Maximum Likelihood Estimation

Once a sparse approximation of the matrix R^{-1} is found, the objective function of the optimization form of the Kalman Filter update step can be optimized by decomposing the quadratic form of the matrix approximation of R^{-1} into the sum of smaller quadratic functions. To do so, it is necessary to express the approximation of R^{-1} in block-diagonal form.

Because the covariance matrix is symmetric positive definite, this problem is equivalent to separating the connected components of an undirected graph. The edge of this graph links the row-column pairs with non-zero matrix coefficients. These connected components can be found using the Cholesky factorization of R^{-1}. [8] For example, consider a sparse approximation of the matrix R^{-1} to be:

$$R^{-1} \approx \begin{bmatrix} R_1 & 0 \\ 0 & R_2 \end{bmatrix}$$

Its quadratic form can then be decomposed as follows:

$$\begin{bmatrix} x^T & y^T \end{bmatrix} \begin{bmatrix} R_1 & 0 \\ 0 & R_2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = x^T R_1 x + y^T R_2 y$$

Similarly, the quadratic objective function of a Kalman Filter update step can be decomposed into a larger sum of smaller quadratic functions. This sum of quadratic objective functions can then be optimized efficiently in parallel using ADMM.

4.3. Distributed Filtering Algorithm

1. Compute empirical covariance matrix R

$$\tilde{R} = \frac{1}{T}\sum_{t}^{T}(z_t - g(\mu_{t|t-1}))(z_t - g(\mu_{t|t-1}))^T$$

2. Compute X, the sparse approximation of R^{-1} using ADMM

$$X^{k+1} := \underset{X}{\text{argmin}} \quad \mathbf{Tr}(RX) - \log\det(X) + (\rho/2)||X - Z^k + U^k||_2^F$$
$$Z^{k+1} := R_{\lambda/\rho}(X^{k+1} + U^k)$$
$$U^{k+1} := U^k + (X^{k+1} - Z^{k+1})$$

3. Partition X into independent components

4. Decompose quadratic filter objective function into largest possible sum of simple quadratic terms

5. Compute maximum likelihood estimate in parallel of next state using ADMM

$$x^{k+1} := \underset{x_i}{\text{argmin}} \quad f_i(x_i) + y_i^k(x_i - \bar{x}^k) + (\rho/2)||x_i - \bar{x}^k||_2^2$$
$$y_i^{k+1} := y_i^k + \rho(x_i^{k+1} - \bar{x}^{k+1})$$

5. Experiment

5.1. Parallel Implementation

This algorithm was implemented in MATLAB in versions operating on both the CPU and a parallel GPU processor. The computation times of these implementations were compared. Although a serial computation would not be as fast as a parallel one, one should expect to see the computation time scale linearly for the distributed algorithm and exponentially for the classical algorithm.

5.2. Results

Unfortunately, all three implementations exhibited exponential scaling behavior and so we were unable to verify the expected efficiency of the proposed algorithm. The likely cause of this phenomenon is that not all parts of the implementation were analyzed to ensure optimal computational efficiency. For example, a matrix-matrix multiplication between two diagonal matrices not stored in a sparse format would be computed in $O(N^3)$ time. While the scaling behavior is important, the distributed implementations also exhibited the largest computational overhead compared to the fundamental KF algorithm. As can be seen in 4, the computation times of the distributed algorithm computed on a GPU and CPU grow approximately linearly on a log plot, so it would grow exponentially on a linear plot. The figure also shows the significant computational overhead required to perform the computation on the GPU. Figure 5 shows the comparison between the distributed algorithm computed on the CPU against the basic Kalman Filter. Figure 5 also shows the time difference when the SIMS calculation is removed. It is clear that the SIMS calculation adds negligible overhead to the overall computation time. It also shows that the CPU computation using the basic Kalman Filter algorithm has significantly less computational overhead compared to the distributed algorithm.

5.3. Analysis

These experimental results suggest that a naive implementation of this algorithm will not exhibit the expected speed increase and will introduce additional computational overhead. This does not disprove the effectiveness of the proposed technique. The results do show that the computational cost of the SIMS approximation is not significant and does not scale significantly compared to other computational costs.

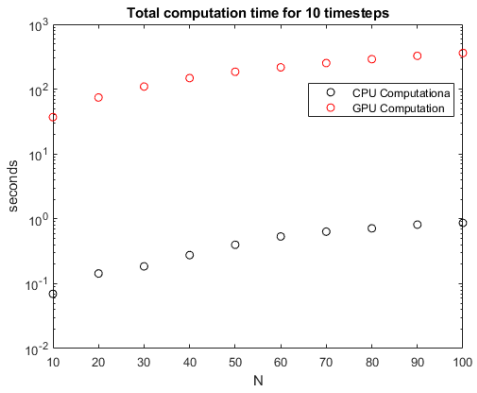

Figure 4. Computation Time Comparison between CPU and GPU implementations

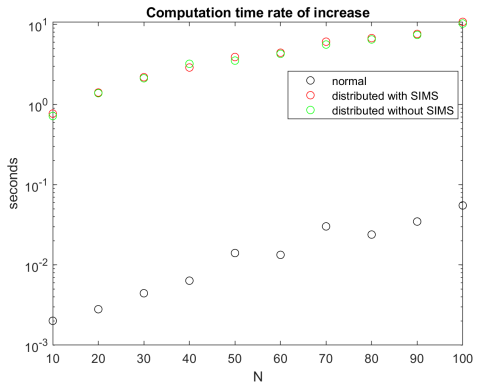

Figure 5. Computation Time Comparison between CPU and GPU implementations

6. Conclusion

This study first introduced an extension of the Kalman Filter to handle an unknown noise covariance matrix, when the number of measurements is reasonable. Then, when the number of measurements grows, this study presented an effective approach to distributed filtering using ADMM. Application areas include large-scale sensing and real-time embedded applications. Further research into this topic would include finding all the additional steps in the computation that would need to be optimized to achieve the desired speed. Additionally, comparison would also need to be made against other filtering approaches such as Particle Filters and Unscented Kalman Filters. In order to fully take advantage of this algorithm's effectiveness, a relatively low-level parallel implementation would have to be developed. Due to time and resource constraints, such a study and implementation were not pursued here.

7. Acknowledgements

This project was submitted for credit by Michael Gobble in EE364B simultaneously with AA273.

References

[1] R. Aragues, J. Cortes, and C. Sagues. Distributed consensus on robot networks for dynamically merging feature-based maps. *IEEE Transactions on Robotics*, 28(4).

[2] J. Aulinas, Y. Petillot, J. Salvi, and X. Llado. The slam problem: A survey. *Proceedings of the International Conference of the Catalan Association for Artificial Intelligence*, page 363371, 2008.

[3] T. Barfoot. State estimation for robotics. 2019.

[4] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1122, 2011.

[5] V. L. Boyd, S. Convex optimization. Cambridge University Press, 2004.

[6] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 32(6), 2016.

[7] A. Cunningham, V. Indelman, and F. Dellaert. Ddf-sam 2.0: Consistent distributed smoothing and mapping. *IEEE International Conference on Robotics and Automation*.

[8] W. M. Darve, E. Numerical linear algebra with julia. Unpublished.

[9] A. Davison and D. Murray. Simultaneous localization and map-building using active vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):865880, 2002.

[10] J. Dong, E. Nelson, V. Indelman, N. Michael, and F. Dellaert. Distributed real-time cooperative localization and mapping using an uncertainty-aware expectation maximization approach. *IEEE International Conference on Robotics and Automation*.

[11] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. Fastslam: A factored solution to the simultaneous localization and mapping problem. *Proceedings of the National Conference on Artificial Intelligence*, page 593598, 2002.

[12] R. S. M. A. Nerurkar, E. Distributed maximum a posteriori estimation for multi-robot cooperative localization. *IEEE International Conference on Robotics and Automation*, 2009.

[13] M. Pilanci and S. Boyd. ee364b course notes. Stanford University, 2019.

[14] M. Schwagger. aa273 problem set 6, question 4. Stanford University, 2019.

[15] S. Thrun, W. Burgard, and D. Fox. Probablistic robotics. MIT Press, 2005.