2014 IEEE International Conference on Robotics & Automation (ICRA)
Hong Kong Convention and Exhibition Center
May 31 - June 7, 2014. Hong Kong, China

# Fast and Accurate PoseSLAM by Combining Relative and Global State Spaces

Brian Peasley[1,2] and Stan Birchfield[1,2]
[1]Electrical and Computer Engineering Dept.
Clemson University, Clemson, SC 29634
{bpeasle,stb}@clemson.edu
[2]Microsoft Robotics
Redmond, WA 98052
stanleyb@microsoft.com

***Abstract*— We revisit the question of state space in the context of performing loop closure. Although a relative state space has been previously discounted, we show that such a state space is actually extremely powerful, able to achieve recognizable results after just one iteration. The power behind the technique (called POReSS) is the coupling between parameters that causes the orientation of one node to affect the position and orientation of other nodes. At the same time, the approach is fast because, like the more popular incremental state space, the Jacobian never needs to be explicitly computed. Furthermore, we show that while POReSS is able to quickly compute a solution near the global optimum, it is not precise enough to perform the fine adjustments necessary to reach the global minimum. As a result, we augment POReSS with a fast variant of Gauss-Seidel (called Graph-Seidel) on a global state space to allow the solution to settle closer to the global minimum. We show that this combination of POReSS and Graph-Seidel converges more quickly and scales to very large graphs better than other techniques while at the same time computing a competitive residual.**

## I. INTRODUCTION

PoseSLAM is the problem of simultaneous localization and mapping (SLAM) using only constraints between robot poses, in which only the robot poses (as opposed to landmark positions) are estimated. To a large extent, once the robot poses have been determined, a map of the environment can be created by overlaying the sensor data obtained at the poses. Assuming a graph-based approach, the primary problem in PoseSLAM is to optimize the graph in the presence of loop closure.

The choice of state space has an enormous impact on the ability of an algorithm to solve for loop closure. In their influential work on PoseSLAM, Olson et al. [17] proposed the use of an incremental state space (ISS) with a variant of stochastic gradient descent. The advantage of this choice is that the Jacobian has a simple formulation and therefore does not need to be explicitly constructed. However, the coupling between the different parameters is lost, so that a change in orientation for one node does not directly affect the positions of the other nodes. Curiously, in the same paper the idea of using a relative state space (RSS) is dismissed with the argument that the resulting Jacobian is highly nonlinear and non-sparse, yielding a computationally expensive algorithm. The first two reasons are no doubt true, and as a result no one (to our knowledge) has attempted to use an RSS for loop closure.

In this paper we revisit the claim that using an RSS is computationally expensive. In fact we arrive at a surprising result, namely that the opposite conclusion is true. By formulating the loop closure problem using an RSS, we show that the same variation of stochastic gradient descent — which we call non-stochastic gradient descent (NGD) for clarity — is able to converge quickly, typically producing meaningful results in just one iteration. The key insight is that in an RSS the parameters are coupled so that changing the orientation of one node affects the global poses of all other downstream nodes. Like the ISS, the RSS leads to a formulation that is straightforward, leading to an implementation that requires less than 100 lines of C++ code, with no linear algebra required. We call this algorithm POReSS (Pose Optimization by a Relative State Space).

While POReSS is able to achieve recognizable results (meaning that the basic shape of the map is present) in just one iteration, the coarse movements of the algorithm prevent it from ever reaching the global minimum. Therefore, we use POReSS as a starting point for a fast variant of Gauss-Seidel, which we call Graph-Seidel, operating on a global state space (GSS) to make fine adjustments to the poses, thus enabling it to settle into a good solution. While Graph-Seidel requires many iterations, each iteration is extremely fast. We demonstrate that our combination of POReSS and Graph-Seidel is able to achieve competitive results compared with state-of-the-art, in less time and with fewer iterations. Moreover, the approach scales well, able to operate on graphs with tens of millions of nodes.

## II. RELATED WORK

Our approach falls within the framework of graph-based SLAM, which was pioneered by Lu and Milios [14]. Duckett et al. [3] optimize the map via relaxation, but this early work assumed knowledge of global orientation, which makes the problem linear. Frese et al.

[4] propose multi-level relaxation (MLR), a variant of Gauss-Seidel, to find the non-linear maximum likelihood solution. Howard et al. [8] showed that the general relaxation framework of Lu and Milios can be applied to a broad range of problems, including not only SLAM but also multi-robot SLAM and sensor network calibration.

To overcome the tendency of Gauss-Seidel to get trapped in local minima, Olson et al. [17] proposed two contributions: an alternative state space representation (incremental state space) so that a single iteration updates many poses, and a variant of stochastic gradient descent that is robust to local minima and converges more quickly than Gauss-Seidel. An extension of this work to incremental optimization of pose graphs was presented in [18]. Another extension is TORO (Tree-based netwORk Optimizer) [7], which uses a tree-based parameterization for describing the configuration of nodes in the graph, as well as slerp functions for handling 3D rotations [5].

Other researchers have investigated the problem of nonlinear least squares minimization using sparse linear algebra. Kummerle et al. [13] have developed g2o, a flexible open-source framework for 2D or 3D SLAM and bundle adjustment. Square Root SAM (smoothing and mapping) [2], and its incremental version iSAM [10], formulate the problem as a factor graph. SLAM++ [19] is an efficient approach to nonlinear least squares. Others use the preconditioned conjugate gradient (PCG) [11], [15] or exploit the sparse structure of the linear system [12]. Ranganathan et al. [20] show that loopy belief propagation (LBP) is equivalent to Gauss-Seidel relaxation but also recover the marginal covariances. Relative bundle adjustment has been proposed [21] as a way to avoid computing the solution in a single Euclidean coordinate system.

## III. Graph-based SLAM

The graph-based approach to SLAM attempts to find the maximum likelihood configuration given a set of measurements. In this section we briefly review this approach, loosely adopting the notation of [7]. The graph is given by $\mathcal{G} = (\mathcal{P}, \mathcal{E})$ consisting of a set of vertices $\mathcal{P} = \{\mathbf{p}_i\}_{i=0}^{n}$ representing robot poses, and a set of edges $\mathcal{E}$ between pairs of robot poses. Assuming a ground-based robot rolling on a horizontal floor plane, the $i$th pose is given by $\mathbf{p}_i = \begin{bmatrix} x_i & y_i & \theta_i \end{bmatrix}^T$, where $\begin{bmatrix} x_i & y_i \end{bmatrix}^T \in \mathbb{R}^2$, and $\theta_i \in SO(2)$. Typically the poses are traversed in a sequential manner, so for convenience we stack this sequence into the vector $\mathbf{p} = \begin{bmatrix} \mathbf{p}_1^T & \cdots & \mathbf{p}_n^T \end{bmatrix}^T$. These poses are in a global coordinate system fixed by convention at $\mathbf{p}_0 \equiv \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$.

Let $\mathbf{x} = \begin{bmatrix} \mathbf{x}_1^T & \cdots \mathbf{x}_n^T \end{bmatrix}^T$ be a state vector that is uniquely related to the sequence of poses through a bijective function $g$ such that $\mathbf{x} = g(\mathbf{p})$ and $\mathbf{p} = g^{-1}(\mathbf{x})$. In the simplest case $\mathbf{x}_i \equiv \mathbf{p}_i$, so that the states are equivalent to the global poses, but this is not required; as we shall see, the choice of state space can have significant impact upon the results.

Each edge $(a, b) \in \mathcal{E}$ captures a constraint $\delta_{ab}$ between poses $\mathbf{p}_a$ and $\mathbf{p}_b$ obtained by sensor measurements or by some other means. For ease of presentation we assume at most one edge between any two given poses, but the extension to a multigraph is straightforward. The uncertainty of the measurement is given by the *information matrix* $\Omega_{ab}$, which is the inverse of the covariance matrix. If we let $\mathbf{f}_{ab}(\mathbf{x})$ be the zero-noise observation between poses $\mathbf{p}_a$ and $\mathbf{p}_b$ given the current configuration $\mathbf{x}$, then the discrepancy between the predicted observation and the actual observation is the *residual*:

$$\mathbf{r}_{ab}(\mathbf{x}) \equiv \delta_{ab} - \mathbf{f}_{ab}(\mathbf{x}). \tag{1}$$

Assuming a Gaussian observation model, the negative log-likelihood of the observation is given by the squared Mahalanobis distance

$$\epsilon_{ab}(\mathbf{x}) \propto \mathbf{r}_{ab}^T(\mathbf{x}) \Omega_{ab} \mathbf{r}_{ab}(\mathbf{x}), \tag{2}$$

also known as the chi-squared error.

The goal of graph-based SLAM is to find the configuration $\mathbf{x}$ that minimizes the energy $\epsilon(\mathbf{x}) \equiv \sum_{(a,b)\in\mathcal{E}} \epsilon_{ab}$. This energy is a non-linear expression due to the orientation parameters, thus requiring an iterative approach. Let $\tilde{\mathbf{x}}$ be the current estimate for the state. The linearized energy about this current estimate is given by

$$\tilde{\epsilon}(\mathbf{x}) \equiv \sum_{(a,b)\in\mathcal{E}} \tilde{\mathbf{r}}_{ab}^T(\mathbf{x}) \Omega_{ab} \tilde{\mathbf{r}}_{ab}(\mathbf{x}), \tag{3}$$

where the linearized residual is given by the first-order Taylor expansion:

$$\tilde{\mathbf{r}}_{ab}(\mathbf{x}) \equiv \mathbf{r}_{ab}(\tilde{\mathbf{x}}) - J_{ab}(\tilde{\mathbf{x}}) \underbrace{(\mathbf{x} - \tilde{\mathbf{x}})}_{\Delta\mathbf{x}}, \tag{4}$$

where $J_{ab}(\tilde{\mathbf{x}})$ is the Jacobian of the *error* $\mathbf{e}_{ab}(\mathbf{x}) \equiv -\mathbf{r}_{ab}(\mathbf{x})$ evaluated at the current state.

Expanding the linear system, rearranging terms, differentiating $\partial\tilde{\epsilon}(\mathbf{x})/\partial\Delta\mathbf{x}$, and setting to zero yields

$$\sum_{(a,b)\in\mathcal{E}} \Omega_{ab} \left( \mathbf{r}_{ab}(\tilde{\mathbf{x}}) - J_{ab}(\tilde{\mathbf{x}}) \Delta\mathbf{x} \right) = 0. \tag{5}$$

If we define $K$ as the matrix obtained by concatenating the $\Omega_{ab}$ horizontally, $\mathbf{r}(\tilde{\mathbf{x}})$ as the vector obtained by stacking $\mathbf{r}_{ab}(\tilde{\mathbf{x}})$ vertically, and $J(\tilde{\mathbf{x}})$ as the matrix obtained by stacking $J_{ab}(\tilde{\mathbf{x}})$ vertically, we obtain the standard least squares system

$$K\mathbf{r}(\tilde{\mathbf{x}}) = KJ(\tilde{\mathbf{x}})\Delta\mathbf{x}. \tag{6}$$

Multiplying both sides by $(KJ)^T$ yields the so-called *normal equations*:

$$J^T(\tilde{\mathbf{x}})\Omega J(\tilde{\mathbf{x}})\Delta\mathbf{x} = J^T(\tilde{\mathbf{x}})\Omega\mathbf{r}(\tilde{\mathbf{x}}), \tag{7}$$

where $\Omega = K^T K$.

For reference let us consider the dimensions of these matrices. If we let $m$ be the number of elements in the state vector, then $\mathbf{x}_i$ is an $m \times 1$ vector; typically for 2D pose optimization we have $m = 3$ due to the translation and orientation parameters. The vectors $\mathbf{x}$, $\tilde{\mathbf{x}}$, and $\Delta\mathbf{x}$ are all $mn \times 1$. Let $m'$ be the number of elements in the observation $\delta_{ab}$; typically $m' = m$. Then $\delta_{ab}(\mathbf{x})$, $\mathbf{f}_{ab}(\mathbf{x})$, $\mathbf{r}_{ab}(\mathbf{x})$, and $\tilde{\mathbf{r}}_{ab}(\mathbf{x})$ are all $m' \times 1$ vectors, $\Omega_{ab}$ is $m' \times m'$, and $\epsilon_{ab}(\mathbf{x})$ and $\tilde{\epsilon}_{ab}(\mathbf{x})$ are scalars. The Jacobian $J_{ab}(\mathbf{x})$ is $m' \times mn$. If we let $n' = |\mathcal{E}|$ be the number of edges in the graph, then $\Omega$ is $m'n' \times m'n'$, $\mathbf{r}(\mathbf{x})$ is $m'n' \times 1$, and $J(\mathbf{x})$ is $m'n' \times mn$.

## IV. State spaces

As mentioned earlier, the choice of state space can have a significant impact upon the results. In this section we describe three different state spaces and outline their strengths and weaknesses. In all cases, $m' = m = 3$.

### A. Global state space (GSS)

The most natural choice for state space is the global pose, that is, the pose of the robot in a global coordinate system:

$$\mathbf{x}_i \equiv \mathbf{p}_i = \begin{bmatrix} x_i & y_i & \theta_i \end{bmatrix}^T. \tag{8}$$

The use of a global state space (GSS) leads to a simple formulation of the energy of the system and subsequently a sparse Jacobian. However, since the GSS representation directly solves for the global poses, each node is only affected by the nodes to which it is directly connected. This causes slow convergence since changes will be propagated slowly and can easily be trapped in a local minimum if the initial conditions are poor.

### B. Incremental state space (ISS)

Olson et al. [17] propose using the incremental state space, in which the state is the difference between consecutive poses:

$$\mathbf{x}_i \equiv \mathbf{p}_i - \mathbf{p}_{i-1} = \begin{bmatrix} x_i - x_{i-1} \\ y_i - y_{i-1} \\ \theta_i - \theta_{i-1} \end{bmatrix}, \qquad i = 1, \ldots, n, \tag{9}$$

with $\mathbf{x}_0 \equiv \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$.

With an incremental state space, the $i$th pose is given by the sum of all states up to and including $i$:

$$\mathbf{p}_i = \sum_{k=0}^{i} \mathbf{x}_k. \tag{10}$$

This state space allows changes to be propagated through the system quickly because changing one state affects the global pose of all nodes past it. However, the coupling between the different parameters has been lost, so that a change in orientation for one node does not directly affect the positions of the other nodes.

### C. Relative state space (RSS)

Another alternative is to use a relative state space:

$$\mathbf{x}_i \equiv \begin{bmatrix} x'_i & y'_i & \theta'_i \end{bmatrix}^T, \tag{11}$$

with $\mathbf{x}_0 \equiv \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$. The parameters $x'_i$, $y'_i$, and $\theta'_i$ describe the relative Euclidean transformation between the $(i-1)$th and $i$th poses, specifically the $i$th pose in the $(i-1)$th coordinate frame. Assuming a righthand coordinate system with positive angles describing counterclockwise rotation, we have:

$$\mathbf{p}_i = \mathbf{p}_{i-1} + \underbrace{\begin{bmatrix} \cos\theta_{i-1} & -\sin\theta_{i-1} & 0 \\ \sin\theta_{i-1} & \cos\theta_{i-1} & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{R(\theta_{i-1})} \underbrace{\begin{bmatrix} x'_i \\ y'_i \\ \theta'_i \end{bmatrix}}_{\mathbf{x}_i} \tag{12}$$

$$= \sum_{k=1}^{i} R(\theta_{k-1})\mathbf{x}_k, \tag{13}$$

where

$$\theta_b = \sum_{k=1}^{b} \theta'_k \tag{14}$$

is the global orientation, as mentioned earlier. If we define ${}^a\mathbf{p}_b$ as the relative pose between $a$ and $b$, $a < b$, that is, pose $b$ in coordinate frame $a$, it is not difficult to verify that

$${}^a\mathbf{p}_b = R^T(\theta_a)(\mathbf{p}_b - \mathbf{p}_a) = \sum_{k=a+1}^{b} R({}^a\theta_{k-1})\mathbf{x}_k \tag{15}$$

since ${}^0\mathbf{p}_b = \mathbf{p}_b$ and ${}^0\theta_b = \theta_b$, where

$${}^a\theta_b = \theta_b - \theta_a = \sum_{k=a+1}^{b} \theta'_k \tag{16}$$

is the angle of frame $b$ with respect to frame $a$. Note that $\theta_a = -{}^a\theta_0$, so that $R^T(\theta_a) = R({}^a\theta_0)$.

## V. Approach

In general, the solution to (7) is found by repeatedly computing $\mathbf{r}(\tilde{\mathbf{x}})$ and $J(\tilde{\mathbf{x}})$ for the current estimate, solving the equation for $\Delta\mathbf{x}$, then adding $\Delta\mathbf{x}$ to the current estimate to yield the estimate for the next iteration. The process is repeated until the system converges (i.e., $\|\Delta\mathbf{x}\| \leq \tau$, where $\tau$ is a threshold).

The standard Gauss-Newton approach is to solve the equation directly in each iteration, leading to

$$\Delta\mathbf{x} = M^{-1}J^T(\tilde{\mathbf{x}})\Omega\mathbf{r}(\tilde{\mathbf{x}}) \tag{17}$$

where $M = J^T(\tilde{\mathbf{x}})\Omega J(\tilde{\mathbf{x}})$ is a $3n \times 3n$ *preconditioning matrix*. Instead, we propose a two-step approach that first uses a variation of stochastic gradient descent in the relative state space (POReSS), followed by a variant of Gauss-Seidel in the global state space (Graph-Seidel). Although either of these is itself a standalone solution, we show in the results that the two exhibit complementary characteristics. The former is better at quickly getting near the global minimum even with poor initial conditions but can take many iterations to reach convergence, while the latter is better at performing detailed refinements of the estimate but requires a starting point near the global minimum.

### A. Non-stochastic gradient descent

Stochastic gradient descent (SGD) is a standard iterative method for finding the minimum of a function. SGD repeatedly updates the state based on a single constraint between nodes $a$ and $b$:

$$\Delta\mathbf{x} = \lambda_{ab} M^{-1} J_{ab}^T(\tilde{\mathbf{x}})\Omega_{ab}\mathbf{r}_{ab}(\tilde{\mathbf{x}}), \tag{18}$$

where $\lambda_{ab} \equiv \lambda/|b-a|$, and $\lambda$ is a scalar learning rate that follows an exponential decay. In contrast to traditional SGD, in which the order of the constraints is chosen randomly, we follow the approach of Olson et al. [17] in which the order is deterministic. For clarity, we refer to this approach as *non-stochastic gradient descent (NGD)*. Unlike Olson's method, our approach selects constraints in decreasing order of the number of nodes they affect.

Using a relative state space, the residual corresponding to a constraint between nodes $a$ and $b$ is given by

$$\mathbf{r}_{ab}(\mathbf{x}) = \delta_{ab} - {}^a\mathbf{p}_b. \tag{19}$$

The Jacobian is obtained by differentiating the right side of (15) with respect to the states:

$$J_{ab} = \begin{bmatrix} \cdots & \mathbf{0} & {}^a_bB_{a+1} & {}^a_bB_{a+2} & \cdots & {}^a_bB_b & \mathbf{0} & \cdots \end{bmatrix}, \tag{20}$$

where

$${}^a_bB_i \equiv \frac{\partial}{\partial \mathbf{x}_i}{}^a\mathbf{p}_b = \begin{bmatrix} \cos{}^a\theta_{i-1} & -\sin{}^a\theta_{i-1} & {}^a_b\alpha_i \\ \sin{}^a\theta_{i-1} & \cos{}^a\theta_{i-1} & {}^a_b\beta_i \\ 0 & 0 & 1 \end{bmatrix} \tag{21}$$

$$\begin{bmatrix} {}^a_b\alpha_i \\ {}^a_b\beta_i \end{bmatrix} \equiv \sum_{k=i}^{b} \begin{bmatrix} -x'_k \sin{}^a\theta_{k-1} - y'_k \cos{}^a\theta_{k-1} \\ x'_k \cos{}^a\theta_{k-1} - y'_k \sin{}^a\theta_{k-1} \end{bmatrix} \tag{22}$$

if $a+1 \leq i \leq b$, or ${}^a_bB_i \equiv \mathbf{0}$ otherwise, where $\mathbf{0}$ is a vector of zeros. As in the incremental state space approach of Olson et al. [17], we never need to compute the Jacobian explicitly.

In the general case the Jacobian is neither sparse nor linear. Plugging (20) into (18) yields a linear system that is difficult to compute due to the $M^{-1} = (J^T(\tilde{\mathbf{x}})\Omega J(\tilde{\mathbf{x}}))^{-1}$ term. Following Olson et al. [17], instead of explicitly computing this matrix we instead ignore all but the diagonal elements:

$$M \approx \text{diag}(J^T(\tilde{\mathbf{x}})\Omega J(\tilde{\mathbf{x}})) \tag{23}$$

$$= \text{diag}\underbrace{\left(\sum_{(a,b)\in\mathcal{E}} {}^a\mathbf{m}_b\right)}_{\mathbf{m}}, \tag{24}$$

where $\text{diag}(\mathbf{v}) \equiv \sum_i \mathbf{e}_i\mathbf{e}_i^T\mathbf{v}\mathbf{e}_i^T$ creates a diagonal matrix from a vector, $\mathbf{e}_i$ is a vector of all zeros except a 1 in the $i$th element, and

$${}^a\mathbf{m}_b \equiv \begin{bmatrix} \cdots & 0 & {}^a_b\gamma_{a+1}^T & {}^a_b\gamma_{a+2}^T & \cdots & {}^a_b\gamma_b^T & 0 & \cdots \end{bmatrix}^T, \tag{25}$$

where

$${}^a_b\gamma_i \equiv \mathit{diag}({}^a_bB_i^T\,\Omega_{ai}\,{}^a_bB_i), \tag{26}$$

and $\mathit{diag}(A) \equiv \sum_i \mathbf{e}_i^T A\mathbf{e}_i\mathbf{e}_i$ extracts the diagonal of a matrix.

When the constraint is between two consecutive nodes, $b = a+1$, the Jacobian reduces to a very simple form:

$$J_{ab} = \begin{bmatrix} \cdots & \mathbf{0} & I_{\{3\times3\}} & \mathbf{0} & \cdots \end{bmatrix}, \tag{27}$$

where $I_{\{3\times3\}}$ is the $3 \times 3$ identity matrix. Oftentimes the vast majority of constraints in a pose optimization problem are between consecutive nodes, in which case this simple form yields a tremendous speedup. Note also that as the preconditioned matrix is being constructed, the computation is simpler in the case of consecutive nodes:

$${}^a\mathbf{m}_b \equiv \begin{bmatrix} \cdots & \mathbf{0} & \mathit{diag}(\Omega_{ab}) & \mathbf{0} & \cdots \end{bmatrix}^T. \tag{28}$$

Plugging the simplified Jacobian of (27) into (18), approximating $M$ by its diagonal elements, and taking the Moore-Penrose pseudoinverse, we get

$$\Delta\mathbf{x} = \lambda_{ab} M^+ \begin{bmatrix} \ldots & \mathbf{0} & \mathit{diag}(\Omega_{ab}) & \mathbf{0} & \ldots \end{bmatrix}^T \mathbf{r}_{ab}(\tilde{\mathbf{x}}) \tag{29}$$

$$\approx \lambda_{ab} \begin{bmatrix} \ldots & \mathbf{0} & I_{\{3\times3\}} & \mathbf{0} & \ldots \end{bmatrix}^T \mathbf{r}_{ab}(\tilde{\mathbf{x}}) \tag{30}$$

$$= \lambda_{ab}\mathbf{r}_{ab}(\tilde{\mathbf{x}}), \tag{31}$$

where the second line is equal in the case of a diagonal $\Omega_{ab}$. Thus it can be seen that in the case of consecutive nodes, only one state needs to be modified, and the update is extremely simple. In the general case, the complexity of computing $J_{ab}^T\Omega_{ab}\mathbf{r}_{ab}$ is proportional to the number of nodes between $a$ and $b$. Pseudocode for POReSS can be seen in Algorithm 1, where $\mathbf{m} \equiv \sum {}^a\mathbf{m}_b \equiv \begin{bmatrix} m_1 & \cdots & m_n \end{bmatrix}^T$ is the vector such that $M \approx \text{diag}(\mathbf{m})$, $\mathbf{r} = \begin{bmatrix} r_x & r_y & r_\theta \end{bmatrix}^T$, $\Delta = \begin{bmatrix} \Delta_x & \Delta_y & \Delta_\theta \end{bmatrix}^T$, and $\text{mod}_\theta$ computes the modulo of the last element of the vector while leaving the other

**Algorithm 1** POReSS

**Input:** relative states $\mathbf{x}_{1:n}$, $\delta_{ab}$ and $\Omega_{ab}$ $\forall (a,b) \in \mathcal{E}$
**Output:** updated relative states $\mathbf{x}_{1:n}$
$\triangleright$ Precompute $M$
$\mathbf{m} \leftarrow \text{zeros}(3n, 1)$
**for** $(a,b)\ \epsilon\ \mathcal{E}$ **do** $\triangleright$Note: $a < b$
  $\mathbf{m} \leftarrow \mathbf{m} + {}^{a}\mathbf{m}_{b}$
**end for**
$\triangleright$ Minimize
**while** not converged **do**
  **for** $(a,b)\ \epsilon\ \mathcal{E}$ **do** $\triangleright$Note: $a < b$
    $\mathbf{r} \leftarrow (\delta_{ab} - {}^{a}\mathbf{p}_{b}) \bmod_{\theta} 2\pi$
    **if** $b == a + 1$ **then**
      $\mathbf{x}_b \leftarrow \mathbf{x}_b + \lambda \mathbf{r}$
    **else**
      $\mathbf{r} \leftarrow \Omega_{ab}\mathbf{r}$
      **for** $i \leftarrow a + 1$ **to** $b$ **do**

$$\Delta \leftarrow R({}^{a}\theta_{i-1}) \begin{bmatrix} r_x \\ r_y \\ 0 \end{bmatrix} + r_\theta \begin{bmatrix} {}^{a}_{b}\alpha_i \\ {}^{a}_{b}\beta_i \\ 1 \end{bmatrix}$$

$$\mathbf{x}_i \leftarrow \mathbf{x}_i + \tfrac{\lambda}{b-a} \begin{bmatrix} \Delta_x / m_{3i-2} \\ \Delta_y / m_{3i-1} \\ \Delta_\theta / m_{3i} \end{bmatrix}$$

      **end for**
    **end if**
  **end for**
  decrease $\lambda$
**end while**

elements unchanged. To improve readability the pseudocode does not include all the optimizations used in our implementation.

### *B. Graph-Seidel*

While the use of non-stochastic gradient descent (NGD) on a Relative State Space allows us to quickly optimize a pose graph, it does a poor job of converging to the correct solution. We address this in the second phase of our optimization process. In this phase we use an implementation of Gauss-Seidel that is optimized for a graph, which we refer to as *Graph-Seidel*. Graph-Seidel is better suited to finding exact solutions and can perform well given adequate initial conditions. In our Graph-Seidel optimization we do not use a relative state space but instead use a global state space. This change is made because the second phase provides a refinement to the original optimization, and we want to prevent small changes in one state from having large effects on the entire system.

The residual is the same as before, but now we use the left side of (15), which we combine with (19) to yield

$$\mathbf{r}_{ab}(\mathbf{x}) = \delta_{ab} - R^T(\theta_a)(\mathbf{p}_b - \mathbf{p}_a) \quad (32)$$

To simplify the math we define

$$\mathbf{r}'_{ab}(\mathbf{x}) \equiv R(\theta_a)\mathbf{r}_{ab}(\mathbf{x}) \quad (33)$$
$$= \mathbf{p}_a - \mathbf{p}_b + R(\theta_a)\delta_{ab} \quad (34)$$
$$\Omega'_{ab} \equiv R(\theta_a)\Omega_{ab}R^T(\theta_a) \quad (35)$$

and note that $\epsilon(\mathbf{x})$ does not change when substituting $\mathbf{r}'_{ab}$ for $\mathbf{r}_{ab}$, and $\Omega'_{ab}$ for $\Omega_{ab}$.

Graph-Seidel differs from Gauss-Seidel by assuming that $R(\theta_a)$ is constant when taking the derivative $\frac{\partial \epsilon(\mathbf{x})}{\partial \mathbf{p}_i}$. The key insight is that, if $R(\theta_a)$ is constant, then $\epsilon(\mathbf{x})$ is convex in the states, and we do not need to linearize the system at all. Instead we simply take derivatives to solve directly for the states, then iterate by updating $R(\theta_a)$. Employing this assumption and setting the derivative to zero yields

$$\sum_{(a,i)\in\mathcal{E}_i^{in}} \Omega'_{ai}\mathbf{r}'_{ai}(\mathbf{x}) = \sum_{(i,b)\in\mathcal{E}_i^{out}} \Omega'_{ib}\mathbf{r}'_{ib}(\mathbf{x}), \quad (36)$$

where

$$\mathcal{E}_i^{in} \equiv \{(a,b) : (a,b) \in \mathcal{E} \text{ and } b = i\} \quad (37)$$
$$\mathcal{E}_i^{out} \equiv \{(a,b) : (a,b) \in \mathcal{E} \text{ and } a = i\} \quad (38)$$

are the set of edges into and out of, respectively, node $i$.

Since we are using a GSS, $\mathbf{x} = \mathbf{p}$ and $\mathbf{x}_i = \mathbf{p}_i$ for $i = 1, \ldots, n$. Rearranging terms yields

$$\begin{bmatrix} \overline{\Omega}_1 & -\overline{\Omega}_{12} & \cdots & -\overline{\Omega}_{1n} \\ -\overline{\Omega}_{21} & \overline{\Omega}_2 & \cdots & -\overline{\Omega}_{2n} \\ \vdots & & \ddots & \vdots \\ -\overline{\Omega}_{n1} & -\overline{\Omega}_{n2} & \cdots & \overline{\Omega}_n \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_n \end{bmatrix} = \begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \vdots \\ \mathbf{v}_n \end{bmatrix}, \quad (39)$$

where, assuming we do not have a multigraph,

$$\overline{\Omega}_{ab} \equiv \overline{\Omega}_{ba} \equiv \begin{cases} \Omega'_{ab} & \text{if } \delta_{ab} \text{ exists} \\ \Omega'_{ba} & \text{if } \delta_{ba} \text{ exists} \\ \mathbf{0}_{\{3\times 3\}} & \text{otherwise} \end{cases} \quad (40)$$

$$\overline{\Omega}_a \equiv \sum_b \overline{\Omega}_{ab} \quad (41)$$

$$\mathbf{v}_i \equiv \underbrace{\sum_{(a,i)\in\mathcal{E}_i^{in}} R(\theta_a)\Omega_{ai}\delta_{ai}}_{\text{edges in}} - \underbrace{\sum_{(i,b)\in\mathcal{E}_i^{out}} R(\theta_i)\Omega_{ib}\delta_{ib}}_{\text{edges out}}. \quad (42)$$

This can be solved using Gauss-Seidel iterations of the form:

$$\mathbf{x}_i^{(k+1)} = \overline{\Omega}_i^{-1} \left( \mathbf{v}_i + \sum_{j<i} \overline{\Omega}_{ij}\mathbf{x}_j^{(k+1)} + \sum_{j>i} \overline{\Omega}_{ij}\mathbf{x}_j^{(k)} \right), \quad (43)$$

where $k$ is the iteration number; to improve convergence, we use successive over-relaxation (SOR). Note that, unlike Gauss-Seidel, Graph-Seidel does not actually

**Algorithm 2** Graph-Seidel

**Input:** global states $\mathbf{x}_{1:n}$, $\delta_{ab}$ and $\Omega_{ab}$ $\forall (a,b) \in \mathcal{E}$
**Output:** updated global states $\mathbf{x}_{1:n}$
**while** not converged **do**
  ▷ Compute $\mathbf{v}$
  **for** $i \leftarrow 1$ **to** $n$ **do**
    $\mathbf{v}_i \leftarrow \mathbf{0}_{\{3\times 1\}}$
    **for** $(a,b) \in \mathcal{E}$ **do**
      **if** $a == i$ **then**
        $\mathbf{v}_i \leftarrow \mathbf{v}_i + \overline{\Omega}_{ab} R(\theta_a) \delta_{ab}$
      **else if** $b == i$ **then**
        $\mathbf{v}_i \leftarrow \mathbf{v}_i - \overline{\Omega}_{ab} R(\theta_a) \delta_{ab}$
      **end if**
    **end for**
  **end for**
  ▷ Minimize
  **for** $i \leftarrow 1$ **to** $n$ **do**
    $\mathbf{w} \leftarrow \mathbf{0}_{\{3\times 1\}}$
    **for** $(a,b) \in \mathcal{E}$ **do**
      **if** $a == i$ **then**
        $\mathbf{w} \leftarrow \mathbf{w} + \overline{\Omega}_{ab} \mathbf{x}_b$
      **else if** $b == i$ **then**
        $\mathbf{w} \leftarrow \mathbf{w} + \overline{\Omega}_{ab} \mathbf{x}_a$
      **end if**
    **end for**
    $\mathbf{x}_i \leftarrow \overline{\Omega}_i^{-1} (\mathbf{v}_i + \mathbf{w})$
  **end for**
**end while**

minimize a linear system, because it recomputes the vector $\mathbf{v}$ each iteration. The algorithm is fast because the matrix remains constant. As before, the pseudocode in Algorithm 2 does not show any optimizations used in our implementation.

## VI. Experimental Results

For evaluation the approach was applied to two synthetic datasets and one real dataset, as well as to a simple, single-loop graph of varying sizes (up to 40 million nodes and constraints). All timings were performed on an Intel Core i7-3770 CPU at 3.4 GHz with 16 GB RAM.

### A. Synthetic datasets

Our approach of POReSS + Graph-Seidel was compared with two state-of-the-art algorithms, TORO [6], [5], [7] and g2o [13]. TORO is a tree-based implementation designed to increase the speed of the algorithm of Olson et al. [17]. Except for parameter differences, the solution found by TORO should be the same as that found by Olson et al.; we used the former because it is freely available online.

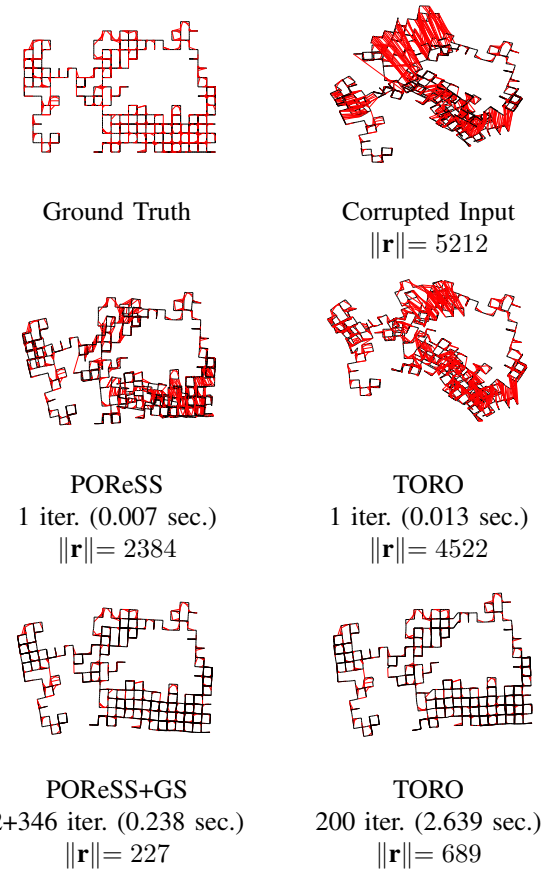



Fig. 1. The Manhattan World data set, containing 3500 nodes and 5600 constraints (top); output of POReSS and TORO after one iteration (middle); and final output of POReSS+GS and TORO (bottom). Not shown are the results for TORO+GS ($\|\mathbf{r}\| = 356$), g2o ($\|\mathbf{r}\| = 146$), and g2o+GS ($\|\mathbf{r}\| = 131$).

We ran the algorithms against two synthetic datasets. The first is the Manhattan World of [13], shown in Figure 1. POReSS achieves a recognizable result, cutting the residual in half in only one iteration.[1] Subsequent iterations of POReSS produce little change on this graph, but following POReSS with several hundred Graph-Seidel iterations reduces the residual by more than 95% from its original value. In contrast, one iteration of TORO barely changes the graph or residual, and even after 10 times more computation than POReSS+GS, TORO is only able to achieve a solution with much higher residual. Although at a glance it may be difficult to compare the solutions of POReSS+GS and TORO from the figure, close visual inspection reveals that indeed the former is more accurate in a number of places, thus validating the residual as a measure of accuracy. On this particular graph the best algorithm

[1]For brevity we use *residual* for the norm of the residual. This is equivalent to the chi-squared error since we use the identity matrix for the information matrix all experiments.

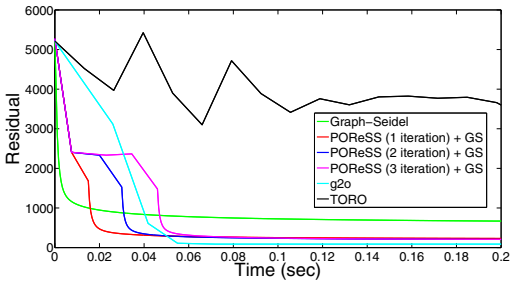



Fig. 2. The norm of the residual versus runtime for the different algorithms, on the Manhattan World dataset. The plot of TORO, which converges after more than 2 seconds, extends past what is shown here.

is g2o, which achieves a residual nearly half that of PORESS+GS with approximately the same computation. We do not show the output of g2o for lack of space.

Figure 2 plots the residual found by the algorithms versus time. Graph-Seidel converges the fastest but settles into a local minimum. PORESS makes significant progress in one iteration, but subsequent iterations have little effect. Graph-Seidel converges onto the same solution after approximately 0.06 seconds regardless of the number of PORESS iterations (assuming at least 1). This is the same amount of time that it takes g2o to converge. Note that due to TORO's relatively long running time, the scale of the plot does not reveal the fact that TORO does indeed reduce the residual considerably. Note also that each iteration of PORESS takes a noticeable amount of time, but Graph-Seidel iterations are extremely fast.

The second dataset, obtained from [9], is a larger graph with more interconnections, shown in Figure 3. On this dataset PORESS alone was not as successful as TORO at finding a solution, but PORESS+GS was able to a significantly better solution in less time than TORO. Note that the result shown is the final result of both algorithms after convergence. As with the Manhattan dataset, the best algorithm is g2o, which achieves a lower residual than either of the other approaches.

### B. Real dataset

The third dataset consists of real data collected from a vehicle driving in loops around a parking lot, from [1], shown in Figure 4. On this dataset PORESS+GS achieved the best results, even outperforming g2o.

### C. Large single loop

As a final experiment, we constructed graphs of varying sizes. Each graph consisted of a single square loop with one loop closing constraint between the first and last nodes, so that the number of nodes equaled the number of edges (constraints). Noise was added to the rotational component at the corners. The number of nodes per side of the square was varied from 1000 to

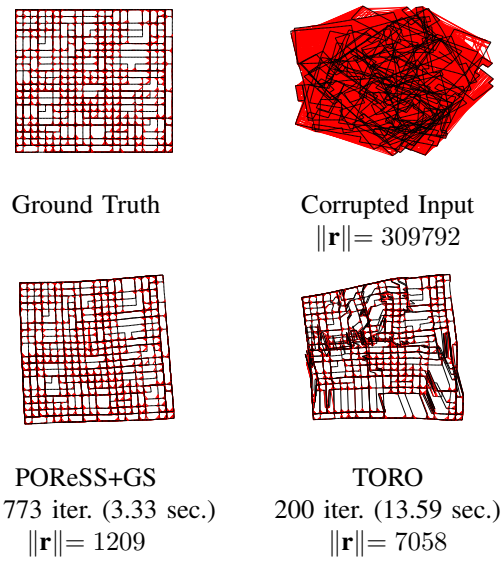



Fig. 3. The dataset of [9] containing 10,000 nodes and 30,000 constraints (top). The final result of PORESS+GS and TORO (bottom). Not shown are the results for TORO+GS ($\|\mathbf{r}\| = 3776$), g2o ($\|\mathbf{r}\| = 362$), and g2o+GS ($\|\mathbf{r}\| = 343$).

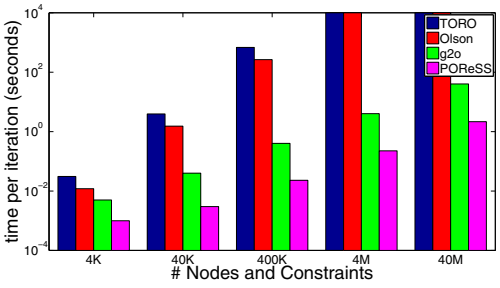
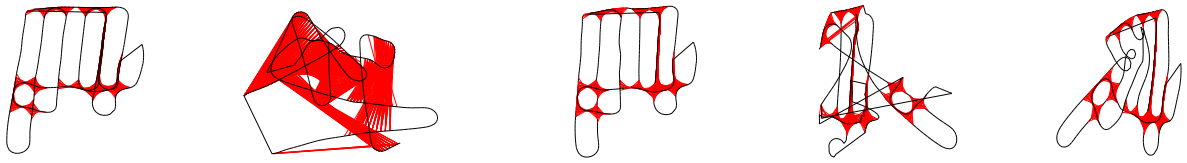



Fig. 5. The runtime for one iteration of each algorithm to optimize a single square loop with a certian number of nodes (equivalently constraints).

10 million, so that the total number of nodes (or edges) varied from 4000 to 40 million. Figure 5 shows the runtime of one iteration of PORESS, g2o, TORO, and our implementation of Olson's algorithm. Regardless of the size of the graph, PORESS required the least runtime. On a graph of 40 million nodes, PORESS required slightly more than two seconds per iteration.

## VII. Conclusion

We have presented a two-step optimization process for solving the PoseSLAM problem. The first step (PORESS) uses a relative state space and non-stochastic gradient descent (NGD) using a simple formulation for the Jacobian. The second step (Graph-Seidel) performs further relaxation in a global state space by executing extremely fast iterations. The idea of a two-step process is not new [16], but this particular combined approach converges more quickly than previous approaches. More-



| Ground Truth | Corrupted Input | PORelSS+GS | TORO | g2o |
|---|---|---|---|---|
| | | 12 + 330 iter. (0.06 s) | 100 iter. (0.37 s) | 10 iter. (0.02 s) |
| | $\|\mathbf{r}\| = 56308$ | $\|\mathbf{r}\| = 56$ | $\|\mathbf{r}\| = 1190$ | $\|\mathbf{r}\| = 408$ |

Fig. 4. Dataset of vehicle driving around parking lot, from [1], containing 407 nodes and 1625 constraints, along with the results of the algorithms. Not shown are the results for TORO+GS ($\|\mathbf{r}\| = 796$), and g2o+GS ($\|\mathbf{r}\| = 163$).

over, it is easy to implement, does not require linear algebra, and produces competitive results compared with state-of-the-art methods on several synthetic and real datasets. Alternatively, Graph-Seidel can be used on its own as a post-processing step to further improve the results of other algorithms.

There is plenty of room for future work. One bottleneck to using a relative state space is the same as that encountered using an incremental state space, namely constraints that affect multiple states. Others have shown that a tree representation of the graph allows for a quicker update of the states. While PORelSS does not require this update, it does require a composition of relative transformations between two nodes at either endpoint of a constraint. As a result, running PORelSS on a tree representation of the graph should speed up this composition. In addition, future work should focus on including landmarks into the system as well as extending the approach to 3D.

## VIII. Acknowledgment

This research was partially supported by the U.S. National Science Foundation under grant IIS-1017007.

## References

[1] J.-L. Blanco, F.-A. Moreno, and J. Gonzalez. A collection of outdoor robotic datasets with centimeter-accuracy ground truth. *Autonomous Robots*, 27(4):327–351, Nov. 2009.

[2] F. Dellaert and M. Kaess. Square root SAM: Simultaneous location and mapping via square root information smoothing. *International Journal of Robotics Research*, 25(12):1181–1204, Dec. 2006.

[3] T. Duckett, S. Marsland, and J. Shapiro. Learning globally consistent maps by relaxation. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, Apr. 2000.

[4] U. Frese, P. Larsson, and T. Duckett. A multilevel relaxation algorithm for simultaneous localisation and mapping. *IEEE Transactions on Robotics*, 21(2):196–207, Apr. 2005.

[5] G. Grisetti, S. Grzonka, C. Stachniss, P. Pfaff, and W. Burgard. Efficient estimation of accurate maximum likelihood maps in 3D. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3472–3478, Oct. 2007.

[6] G. Grisetti, C. Stachniss, and W. Burgard. Nonlinear constraint network optimization for efficient map learning. *IEEE Transactions on Intelligent Transportation Systems*, 10(3):428–439, Sept. 2009.

[7] G. Grisetti, C. Stachniss, S. Grzonka, and W. Burgard. A tree parameterization for efficiently computing maximum likelihood maps using gradient descent. In *Proceedings of Robotics: Science and Systems (RSS)*, June 2007.

[8] A. Howard, M. J. Matarić, and G. Sukhatme. Relaxation on a mesh: A formalism for generalized localization. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2001.

[9] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard, and F. Dellaert. iSAM2: Incremental smoothing and mapping using the bayes tree. *International Journal of Robotics Research*, 31:217–236, Feb. 2012.

[10] M. Kaess, A. Ranganathan, and F. Dellaert. iSAM: Incremental smoothing and mapping. In *IEEE Transactions on Robotics*, 2008.

[11] K. Konolige. Large-scale map-making. In *Proceedings of the National Conference on Artificial Intelligence*, July 2004.

[12] K. Konolige, G. Grisetti, R. Kummerle, W. Burgard, B. Limketkai, and R. Vincent. Sparse pose adjustment for 2D mapping. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010.

[13] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g2o: A general framework for graph optimization. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 3607–3613, May 2011.

[14] F. Lu and E. Milios. Globally consistent range scan alignment for environment mapping. *Autonomous Robots*, 4:333–349, 1997.

[15] M. Montemerlo and S. Thrun. Large-scale robotic 3-D mapping of urban structures. In *Proceedings of the International Symposium on Experimental Robotics (ISER)*, June 2004.

[16] E. Olson. *Robust and Efficient Robotic Mapping*. PhD thesis, Massachusetts Institute of Technology, June 2008.

[17] E. Olson, J. Leonard, and S. Teller. Fast iterative optimization of pose graphs with poor initial estimates. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 2262–2269, May 2006.

[18] E. Olson, J. Leonard, and S. Teller. Spatially-adaptive learning rates for online incremental SLAM. In *Robotics: Science and Systems*, June 2007.

[19] L. Polok, V. Ila, M. Solony, P. Smrz, and P. Zemcik. Incremental block Cholesky factorization for nonlinear least squares in robotics. In *Proceedings of Robotics: Science and Systems*, June 2013.

[20] A. Ranganathan, M. Kaess, and F. Dellaert. Loopy SAM. In *Proceedings of the International Joint Conference on Artificial Intelligence*, Jan. 2007.

[21] G. Sibley, C. Mei, I. Reid, and P. Newman. Adaptive relative bundle adjustment. In *Proceedings of Robotics: Science and Systems*, Aug. 2009.