Graph-like representations have also been applied in the context of SLAM filtering algorithms. In 1997, Csorba developed an information filter that maintained relative information between triplets of three landmarks. He was possibly the first to observe that such information links maintained global correlation information implicitly, paving the way from algorithms with quadratic to linear memory requirements. Newmann (2000) and Newman and Durrant-Whyte (2001) developed a similar information filter, but left open the question of how the landmark-landmark information links are actually acquired. Under the ambitious name '*consistent, convergent, and constant-time SLAM,*' Leonard and Newman further developed this approach into an efficient alignment algorithm, which was successfully applied to an autonomous underwater vehicle using synthetic aperture sonar (Newman and Rikoski 2003). Another seminal algorithm in the field is Paskin's (2003) *thin junction filter* algorithm, which represents the SLAM posterior in a sparse network known as thin junction trees (Pearl 1988; Cowell et al. 1999). The same idea was exploited by Frese (2004), who developed a similar tree factorization of the information matrix for efficient inference. Julier and Uhlmann (2000) developed a scalable technique called *covariance intersection*, which sparsely approximates the posterior in a way that provable prevents overconfidence. Their algorithm was successfully implemented on NASA's MARS Rover fleet (Uhlmann, Lanzagorta, and Julier 1999). The information filter perspective is also related to early work by Bulata and Devy (1996), whose approach acquired landmark models first in local landmark-centric reference frames, and only later assembles a consistent global map by resolving the relative information between landmarks. Another online filter related to this work is the SEIF algorithm, which was developed by Thrun et al. (2002). A greedy data association algorithm for SEIFs was developed by Liu and Thrun (2003), which was subsequently extended to multi-robot SLAM by Thrun and Liu (2003). A branch-and-bound data association search is due to Hähnel et al. (2003), based on earlier branch-and-bound methods by Lawler and Wood (1966) and Narendra and Fukunaga (1977). It parallels work by Kuipers et al. (2004), who developed a similar data association technique, albeit not in the context of an information theoretic concepts. Finally, certain 'offline' SLAM algorithms that solve the full SLAM problem, such as the ones by Bosse et al. (2004), Gutmann and Konolige (2000), and Frese (2004), have been shown to be fast enough to run online on limited-sized data sets. None of these approaches address how to incorporate occasional GPS measurements into SLAM.

3. Mapping SLAM Problems into Graphs

3.1. The Offline SLAM Problem

We begin our technical exposition with the basic notation used throughout this article. In SLAM, time is usually discrete, and t labels the time index. The robot pose at time t is denoted x_t; we will use $x_{1:t}$ to denote the set of poses from time 1 all the way to time t. The world itself is denoted m, where m is short for map. The map is assumed to be time-invariant, hence we do not use a time index to denote the map. In this paper, we think of the map of a (large) set of features m_j.

To acquire an environment map, the robot is able to sense. The measurement at time t is denoted z_t. Usually, the robot can sense multiple features at each point in time; hence each individual measurement beam is denoted z_t^i. Commonly, one assumes that z_t^i is a range measurement. The measurement function h describes how such a measurement comes into being:

$$z_t^i = h(x_t, m_j, i) + \varepsilon_t^i \tag{1}$$

here ε_t^i is a Gaussian random variable modeling the measurement noise, with zero mean and covariance Q_t, and m_j is the map feature sensed by the i-th measurement beam at time t. Put differently, we have

$$p(z_t^i \mid x_t, m) = \text{const.} \ \exp -\frac{1}{2} (z_t^i - h(x_t, m_j, i))^T Q_t^{-1} (z_t^i - h(x_t, m_j, i)) \tag{2}$$

Some robotic systems are also are provided with a GPS system. Then the measurement is of the form

$$z_t^i = h(x_t, i) + \varepsilon_t^i \tag{3}$$

where z_t^i is a noisy estimate of the pose x_t, and ε_t^i is once again a Gaussian noise variable The mathematics for such measurements are analogous to those of nearby features; and GraphSLAM admits for arbitrary measurement functions h.

Finally, the robot changes its pose in SLAM by virtue of issuing control commands. The control asserted between time $t-1$ and time t is denoted u_t. The state transition is governed by the function g:

$$x_t = g(u_t, x_{t-1}) + \delta_t \tag{4}$$

where $\delta_t \sim \mathcal{N}(0, R_t)$ models the noise in the control command. The function g can be thought of as the kinematic model of the robot. Equation (4) induces the state transition probability

$$p(x_t \mid u_t, x_{t-1}) = \text{const.} \ \exp -\frac{1}{2} (x_t - g(u_t, x_{t-1}))^T R_t^{-1} (x_t - g(u_t, x_{t-1})) \tag{5}$$

The offline SLAM posterior is now given by the following posterior probability over the robot path $x_{1:t}$ and the map m:

$$p(x_{1:t}, m \mid z_{1:t}, u_{1:t}) \tag{6}$$

This is the posterior probability over the entire path $x_{1:t}$ along with the map, instead of just the current pose x_t. We note that

in many SLAM problems, it suffices to determine the mode of this posterior. The actual posterior is usually too difficult to express for high-dimensional maps m, since it contains dependencies between any pair of features in m.

We note that a key assumption in our problem formulation is the assumption of independent Gaussian noise. GraphSLAM shares this assumption with the vast majority of published papers in the field of SLAM. The Gaussian noise assumption proves convenient in that it leads to a nice set of quadratic equations which can be solved efficiently. Other SLAM approaches have relaxed this assumption (Montemerlo et al. 2002) or made special provisions for incorporating non-Gaussian noise into Gaussian SLAM (Guivant and Masson 2005).

3.2. GraphSLAM: Basic Idea

Figure 1 illustrates the GraphSLAM algorithm. Shown there is the graph that GraphSLAM extracts from four poses labeled $x_1, \ldots, x_4$, and two map features m_1, m_2. Arcs in this graph come in two types: motion arcs and measurement arcs. Motion arcs link any two consecutive robot poses, and measurement arcs link poses to features that were measured there. Each edge in the graph corresponds to a nonlinear constraint. As we shall see later, these constraints represent the negative log likelihood of the measurement and the motion models, hence are best thought of as *information constraints*. Adding such a constraint to the graph is trivial for GraphSLAM; it involves no significant computation. The sum of all constraints results in a nonlinear *least squares problem*, as stated in Figure 1.

To compute a map posterior, GraphSLAM linearizes the set of constraints. The result of linearization is a sparse information matrix and an information vector. The sparseness of this matrix enables GraphSLAM to apply the variable elimination algorithm, thereby transforming the graph into a much smaller one only defined over robot poses. The path posterior map is then calculated using standard inference techniques. GraphSLAM also computes a map and certain marginal posteriors over the map; the full map posterior is of course quadratic in the size of the map and hence is usually not recovered.

3.3. Building Up the Graph

Suppose we are given a set of measurements $z_{1:t}$ with associated correspondence variables $c_{1:t}$, and a set of controls $u_{1:t}$. GraphSLAM turns this data into a graph. The nodes of this graph are the robot poses $x_{1:t}$ and the features in the map $m = \{m_j\}$. Each edge in the graph corresponds to an event: a motion event generates an edge between two robot poses, and a measurement event creates a link between a pose and a feature in the map. Edges represent soft constraints between poses and features in GraphSLAM.

For a linear system, these constraints are equivalent to entries in an information matrix and an information vector of a large system of equations. As usual, we will denote the information matrix by Ω and the information vector by ξ. As we shall see below, each measurement and each control leads to a *local* update of Ω and ξ, which corresponds to a local addition of an edge to the graph in GraphSLAM. In fact, the rule for incorporating a control or a measurement into Ω and ξ is a local addition, paying tribute to the important fact that information is an additive quantity.

Figure 2 illustrates the process of constructing the graph along with the corresponding information matrix. First consider a measurement z_t^i. This measurement provides information between the location of the feature $j = c_t^i$ and the robot pose x_t at time t. In GraphSLAM, this information is mapped into a constraint between x_t and m_j. We can think of this edge as a (possibly degenerate) "spring" in a spring-mass model. As we shall see below, the constraint is of the type:

$$(z_t^i - h(x_t, m_j, i))^T \; Q_t^{-1} \; (z_t^i - h(x_t, m_j, i)) \qquad (7)$$

Here h is the measurement function, and Q_t is the covariance of the measurement noise. Figure 2(a) shows the addition of such a link into the graph maintained by GraphSLAM. Note that the constraint may be *degenerate*, that is, it may not constrain all dimensions of the robot pose x_t. This will be of no concern for the material yet to come.

In information form, the constraint is incorporated into Ω and ξ by adding values between the rows and columns connecting x_{t-1} and x_t. The magnitude of these values corresponds to the stiffness of the constraint, as governed by the uncertainty covariance Q_t of the motion model. This is illustrated in Figure 2(b), which shows the link between two robot poses along with the corresponding element in the information matrix.

Now consider robot motion. The control u_t provides information about the relative value of the robot pose at time $t-1$ and the pose at time t. Again, this information induces a constraint in the graph, which will be of the form:

$$(x_t - g(u_t, x_{t-1}))^T \; R_t^{-1} \; (x_t - g(u_t, x_{t-1})) \qquad (8)$$

Here g is the kinematic motion model of the robot, and R_t is the covariance of the motion noise. Figure 2(b) illustrates the addition of such a link in the graph. It also shows the addition of a new element in the information matrix, between the pose x_t and the measurement z_t^i. This update is again additive. As before, the magnitude of these values reflects the residual uncertainty R_t due to the measurement noise; the less noisy the sensor, the larger the value added to Ω and ξ.

After incorporating all measurements $z_{1:t}$ and controls $u_{1:t}$, we obtain a sparse graph of soft constraints. The number of constraints in the graph is linear in the time elapsed, hence the graph is sparse. The sum of all constraints in the graph will be of the form

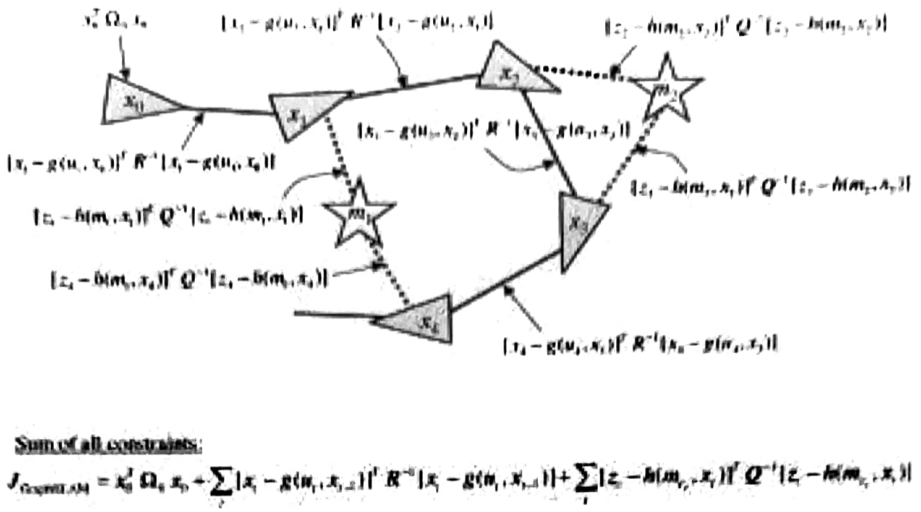


Fig. 1. GraphSLAM illustration, with 4 poses and two map features. Nodes in the graphs are robot poses and feature locations. The graph is populated by two types of edges: Solid edges which link consecutive robot poses, and dashed edges, which link poses with features sensed while the robot assumes that pose. Each link in GraphSLAM is a non-linear quadratic constraint. Motion constraints integrate the motion model; measurement constraints the measurement model. The target function of GraphSLAM is sum of these constraints. Minimizing it yields the most likely map and the most likely robot path.

$$
\begin{aligned}
J_{\text{GraphSLAM}} = x_0^T \, \Omega_0 \, x_0 \; + \; & \sum_t (x_t - g(u_t, x_{t-1}))^T \\
& R_t^{-1} \; (x_t - g(u_t, x_{t-1})) \\
+ \; & \sum_t \sum_i (z_t^i - h(y_t, c_t^i, i))^T \\
& Q_t^{-1} \; (z_t^i - h(y_t, c_t^i, i))
\end{aligned}
\tag{9}
$$

It is a function defined over pose variables $x_{1:t}$ and all feature locations in the map m. Notice that this expression also features an *anchoring constraint* of the form $x_0^T \; \Omega_0 \; x_0$. This constraint anchors the absolute coordinates of the map by initializing the very first pose of the robot as $(0 \;\; 0 \;\; 0)^T$.

In the associated information matrix Ω, the off-diagonal elements are all zero with two exceptions: between any two consecutive poses x_{t-1} and x_t will be a non-zero value that represents the information link introduced by the control u_t. Also non-zero will be any element between a map feature m_j and a pose x_t, if m_j was observed when the robot was at x_t. All elements between pairs of different features remain zero. This reflects the fact that we never receive information pertaining to their relative location—all we receive in SLAM are measurements that constrain the location of a feature relative to a robot pose. Thus, the information matrix is equally sparse; all but a linear number of its elements are zero.

3.4. Inference

Of course, neither the graph representation nor the information matrix representation gives us what we want: the map and the path. In GraphSLAM, the map and the path are obtained from the linearized information matrix Ω and the information vector ξ, via the equations $\Sigma = \Omega^{-1}$ and $\mu = \Sigma \; \xi$. This operation requires us to solve a system of linear equations. This raises the question on how efficiently we can recover the map estimate μ.

The answer to the complexity question depends on the topology of the world. If each feature is seen only locally in time, the graph represented by the constraints is linear. Thus, Ω can be reordered so that it becomes a band-diagonal matrix, that is, all non-zero values occur near its diagonal. The equation $\mu = \Omega^{-1}\xi$ can then be computed in linear time. This intuition carries over to a cycle-free world that is traversed once, so that each feature is seen for a short, consecutive period of time.

The more common case, however, involves features that are observed multiple times, with large time delays in between. This might be the case because the robot goes back and forth through a corridor, or because the world possesses *cycles*. In either situation, there will exist features m_j that are seen at drastically different time steps x_{t_1} and x_{t_2}, with $t_2 \gg t_1$. In our constraint graph, this introduces a cyclic dependence: x_{t_1} and x_{t_2} are linked through the sequence of controls $u_{t_1+1}, u_{t_1+2}, \ldots, u_{t_2}$ and through the joint observation links between x_{t_1} and m_j, and x_{t_2} and m_j, respectively. Such links make our variable reordering trick inapplicable, and recovering the map becomes more complex. In fact, since the inverse of Ω is multiplied with a vector, the result can be computed with optimization techniques such as conjugate gradient, without explicitly computing the full inverse matrix. Since most worlds possess cycles, this is the case of interest.

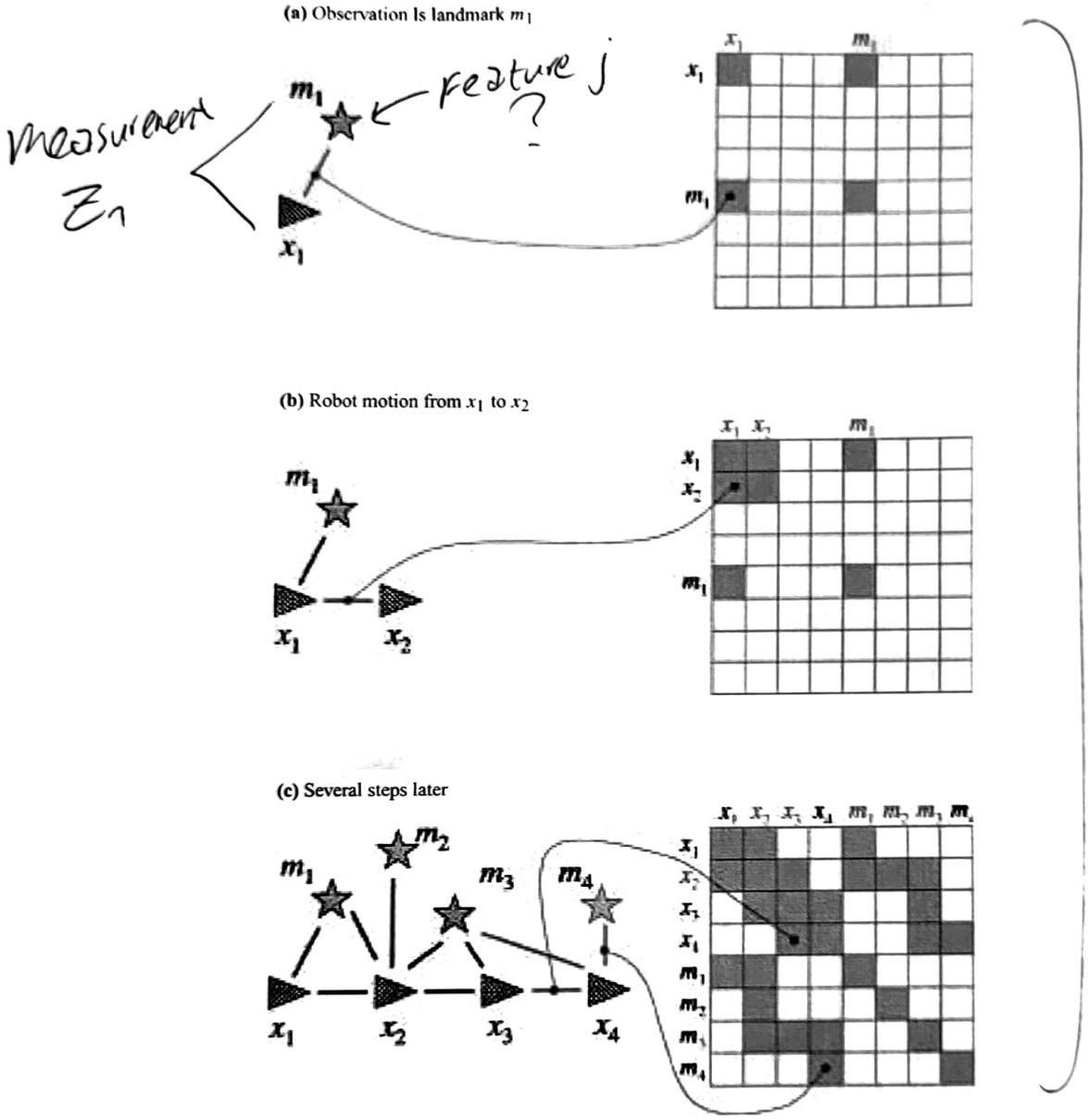


Fig. 2. Illustration of the acquisition of the information matrix in GraphSLAM. The left diagram shows the dependence graph, the right the information matrix.

The GraphSLAM algorithm now employs an important *factorization trick*, which we can think of as propagating information trough the information matrix (in fact, it is a generalization of the well-known *variable elimination algorithm* for matrix inversion). Suppose we would like to remove a feature m_j from the information matrix Ω and the information state ξ. In our spring mass model, this is equivalent to removing the node and all springs attached to this node. As we shall see below, this is possible by a remarkably simple operation: we can remove all those springs between m_j and the poses at which m_j was observed, by introducing new springs between any pair of such poses.

This process is illustrated in Figure 3, which shows the removal of two map features, m_1 and m_3 (the removal of m_2 and m_4 is trivial in this example). In both cases, the feature removal modifies the link between any pair of poses from which a feature was originally observed. As illustrated in Figure 3(b), this operation may lead to the introduction of new

links in the graph. In the example shown there, the removal of m_3 leads to a new link between x_2 and x_4.

Let $\tau(j)$ be the set of poses at which m_j was observed (that is, $x_i \in \tau(j) \Longleftrightarrow \exists i : c_i^t = j$). Then we already know that the feature m_j is only linked to poses x_i in $\tau(j)$; by construction, m_j is *not* linked to any other pose, or to any feature in the map. We can now set all links between m_j and the poses $\tau(j)$ to zero by introducing a new link between any two poses $x_i, x_{i'} \in \tau(j)$. Similarly, the information vector values for all poses $\tau(j)$ are also updated. An important characteristic of this operation is that it is local: It only involves a small number of constraints. After removing all links to m_j, we can safely remove m_j from the information matrix and vector. The resulting information matrix is smaller—it lacks an entry for m_j. However, it is equivalent for the remaining variables, in the sense that the posterior defined by this information matrix is mathematically equivalent to the original posterior before removing m_j. This equivalence is intuitive: We simply have replaced springs connecting m_j to various poses in our spring mass model by a set of springs directly linking these poses. In doing so, the total force asserted by these springs remains equivalent, with the only exception that m_j is now disconnected.

The virtue of this reduction step is that we can gradually transform our inference problem into a smaller one. By removing each feature m_j from Ω and ξ, we ultimately arrive at a much smaller information form $\tilde{\Omega}$ and $\tilde{\xi}$ defined only over the robot path variables. The reduction can be carried out in time linear in the size of the map; in fact, it generalizes the variable elimination technique for matrix inversion to the information form, in which we also maintain an information state. The posterior over the robot path is now recovered as $\tilde{\Sigma} = \tilde{\Omega}^{-1}$ and $\tilde{\mu} = \tilde{\Sigma}\tilde{\xi}$. Unfortunately, our reduction step does not eliminate cycles in the posterior. The remaining inference problem may still require more than linear time.

As a last step, GraphSLAM recovers the feature locations. Conceptually, this is achieved by building a new information matrix Ω_j and information vector ξ_j for each m_j. Both are defined over the variable m_j and the poses $\tau(j)$ at which m_j were observed. It contains the original links between m_j and $\tau(j)$, but the poses $\tau(j)$ are set to the values in $\tilde{\mu}$, without uncertainty. From this information form, it is now simple to calculate the location of m_j, using the common matrix inversion trick. Clearly, Ω_j contains only elements that connect to m_j; hence the inversion takes time linear in the number of poses in $\tau(j)$.

It should be apparent why the graph representation is such a natural representation. The full SLAM problem is solved by locally adding information into a large information graph, one edge at a time for each measurement z_t^i and each control u_t. To turn such information into an estimate of the map and the robot path, it is first linearized, then information between poses and features is gradually shifted to information between pairs of poses. The resulting structure only constrains the robot poses, which are then calculated using matrix inversion. Once the poses are recovered, the feature locations are calculated one after another, based on the original feature-to-pose information.

4. The GraphSLAM Algorithm

We will now make the various computational steps of the GraphSLAM precise. The full GraphSLAM algorithm will be described in a number of steps. The main difficulty in implementing the simple additive information algorithm pertains to the conversion of a conditional probability of the form $p(z_t^i \mid x_t, m)$ and $p(x_t \mid u_t, x_{t-1})$ into a link in the information matrix. The information matrix elements are all linear; hence this step involves linearizing $p(z_t^i \mid x_t, m)$ and $p(x_t \mid u_t, x_{t-1})$. To perform this linearization, we need an initial estimate $\mu_{0:t}$ for all poses $x_{0:t}$.

There exist a number of solutions to the problem of finding an initial mean μ suitable for linearization. For example, we can run an EKF SLAM and use its estimate for linearization (Dissanayake et al. 2001). GraphSLAM uses an even simpler technique: our initial estimate will simply be provided by chaining together the motion model $p(x_t \mid u_t, x_{t-1})$. Such an algorithm is outlined in Table 1, and called there **GraphSLAM_initialize**. This algorithm takes the controls $u_{1:t}$ as input, and outputs sequence of pose estimates $\mu_{0:t}$. It initializes the first pose by zero, and then calculates subsequent poses by recursively applying the velocity motion model. Since we are only interested in the mean poses vector $\mu_{0:t}$, **GraphSLAM_initialize** only uses the deterministic part of the motion model. It also does not consider any measurement in its estimation.

Once an initial $\mu_{0:t}$ is available, the GraphSLAM algorithm constructs the full SLAM information matrix Ω and the corresponding information vector ξ. This is achieved by linearizing the links in the graph. The algorithm **GraphSLAM_linearize** is depicted in Table 2. This algorithm contains a good amount of mathematical notation, much of which will become clear in our derivation of the algorithm further below. **GraphSLAM_linearize** accepts as an input the set of controls, $u_{1:t}$, the measurements $z_{1:t}$ and associated correspondence variables $c_{1:t}$, and the mean pose estimates $\mu_{0:t}$. It then gradually constructs the information matrix Ω and the information vector ξ through linearization, by locally adding sub-matrices in accordance with the information obtained from each measurement and each control.

In particular, line 2 in **GraphSLAM_linearize** initializes the information elements. The "infinite" information entry in line 3 fixes the initial pose x_0 to $(0\ 0\ 0)^T$. It is necessary, since otherwise the resulting matrix becomes singular, reflecting the fact that from relative information alone we cannot recover absolute estimates.

Controls are integrated in lines 4 through 9 of **GraphSLAM_linearize**. The pose $\hat{x}$ and the Jacobian G_t calculated

Table 1. Initialization of the Mean Pose Vector $\mu_{1:t}$ in the GraphSLAM Algorithm

Algorithm GraphSLAM_initialize($u_{1:t}$):

$$\begin{pmatrix} \mu_{0,x} \\ \mu_{0,y} \\ \mu_{0,\theta} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

for all controls $u_t = (v_t \;\; \omega_t)^T$ *do*

$$\begin{pmatrix} \mu_{t,x} \\ \mu_{t,y} \\ \mu_{t,\theta} \end{pmatrix} = \begin{pmatrix} \mu_{t-1,x} \\ \mu_{t-1,y} \\ \mu_{t-1,\theta} \end{pmatrix} + \begin{pmatrix} -\frac{v_t}{\omega_t}\sin\mu_{t-1,\theta} + \frac{v_t}{\omega_t}\sin(\mu_{t-1,\theta} + \omega_t \Delta t) \\ \frac{v_t}{\omega_t}\cos\mu_{t-1,\theta} - \frac{v_t}{\omega_t}\cos(\mu_{t-1,\theta} + \omega_t \Delta t) \\ \omega_t \Delta t \end{pmatrix}$$

endfor

return $\mu_{0:t}$

Table 2. Calculation of Ω and ξ in GraphSLAM

Algorithm GraphSLAM_linearize($u_{1:t}$, $z_{1:t}$, $c_{1:t}$, $\mu_{0:t}$):

set $\Omega = 0, \xi = 0$

add $\begin{pmatrix} \infty & 0 & 0 \\ 0 & \infty & 0 \\ 0 & 0 & \infty \end{pmatrix}$ *to* Ω *at* x_0

for all controls $u_t = (v_t \;\; \omega_t)^T$ *do*

$$\hat{x}_t = \mu_{t-1} + \begin{pmatrix} -\frac{v_t}{\omega_t}\sin\mu_{t-1,\theta} + \frac{v_t}{\omega_t}\sin(\mu_{t-1,\theta} + \omega_t \Delta t) \\ \frac{v_t}{\omega_t}\cos\mu_{t-1,\theta} - \frac{v_t}{\omega_t}\cos(\mu_{t-1,\theta} + \omega_t \Delta t) \\ \omega_t \Delta t \end{pmatrix}$$

$$G_t = \begin{pmatrix} 1 & 0 & \frac{v_t}{\omega_t}\cos\mu_{t-1,\theta} - \frac{v_t}{\omega_t}\cos(\mu_{t-1,\theta} + \omega_t \Delta t) \\ 0 & 1 & \frac{v_t}{\omega_t}\sin\mu_{t-1,\theta} - \frac{v_t}{\omega_t}\sin(\mu_{t-1,\theta} + \omega_t \Delta t) \\ 0 & 0 & 1 \end{pmatrix}$$

see next page for continuation

continued from the previous page

7: $\quad add \begin{pmatrix} 1 \\ -G_t \end{pmatrix} R_t^{-1} \begin{pmatrix} 1 & -G_t \end{pmatrix}$ *to* Ω *at* x_t *and* x_{t-1}

8: $\quad add \begin{pmatrix} 1 \\ -G_t \end{pmatrix} R_t^{-1} [\hat{x}_t + G_t\, \mu_{t-1}]$ *to* ξ *at* x_t *and* x_{t-1}

9: *endfor*

10: *for all measurements* z_t *do*

11: $Q_t = \begin{pmatrix} \sigma_r & 0 & 0 \\ 0 & \sigma_\phi & 0 \\ 0 & 0 & \sigma_s \end{pmatrix}$

12: *for all observed features* $z_t^i = (r_t^i \;\; \phi_t^i)^T$ *do*

13: $j = c_t^i$

14: $\delta = \begin{pmatrix} \delta_x \\ \delta_y \end{pmatrix} = \begin{pmatrix} \mu_{j,x} - \mu_{t,x} \\ \mu_{j,y} - \mu_{t,y} \end{pmatrix}$

15: $q = \delta^T \delta$

16: $\hat{z}_t^i = \begin{pmatrix} \sqrt{q} \\ \text{atan2}(\delta_y, \delta_x) - \mu_{t,\theta} \end{pmatrix}$

17: $H_t^i = \frac{1}{q} \begin{pmatrix} \sqrt{q}\delta_x & -\sqrt{q}\delta_y & 0 & -\sqrt{q}\delta_x & \sqrt{q}\delta_y \\ \delta_y & \delta_x & -1 & -\delta_y & -\delta_x \end{pmatrix}$

18: *add* $H_t^{iT}\; Q_t^{-1}\; H_t^i$ *to* Ω *at* x_t *and* m_j

19: *add* $H_t^{iT}\; Q_t^{-1}\; [\; z_t^i - \hat{z}_t^i - H_t^i \begin{pmatrix} \mu_{t,x} \\ \mu_{t,y} \\ \mu_{t,\theta} \\ \mu_{j,x} \\ \mu_{j,y} \end{pmatrix} \;]$ *to* ξ *at* x_t *and* m_j

20: *endfor*

21: *endfor*

22: *return* Ω, ξ

in lines 5 and 6 represent the linear approximation of the non-linear measurement function g. As is obvious from these equations, this linearization step utilizes the pose estimates $\mu_{0:t-1}$, with $\mu_0 = (0\;\;0\;\;0)^T$. This leads to the updates for Ω, and ξ, calculated in lines 7, and 8, respectively. Both terms are added into the corresponding rows and columns of Ω and ξ. This addition realizes the inclusion of a new constraint into the SLAM posterior, very much along the lines of the intuitive description in the previous section.

Measurements are integrated in lines 10 through 21 of **GraphSLAM_linearize**. The matrix Q_t calculated in line 11 is the familiar measurement noise covariance. Lines 13 through 17 compute the Taylor expansion of the measurement function. This calculation assumes *known* correspondence between observed features and features in the map (line 13). Attention has to be paid to the implementation of line 16, since the angular expressions can be shifted arbitrarily by 2π. This calculation culminates in the computation of the measurement update in lines 18 and 19. The matrix that is being added to Ω in line 18 is of dimension 5×5. To add it, we decompose it into a matrix of dimension 3×3 for the pose x_t, a matrix of dimension 2×2 for the feature m_j, and two matrices of dimension 3×2 and 2×3 for the link between x_t and m_j. Those are added to Ω at the corresponding rows and columns. Similarly, the vector added to the information vector ξ is of vertical dimension 5. It is also chopped into two vectors of size 3 and 2, and added to the elements corresponding to x_t and m_j, respectively. The result of GraphSLAM_linearize

is an information vector ξ and a matrix Ω. We already noted that Ω is sparse. It contains only non-zero sub-matrices along the main diagonal, between subsequent poses, and between poses and features in the map. The running time of this algorithm is linear in t, the number of time steps at which data was accrued.

The next step of the GraphSLAM algorithm pertains to reducing the dimensionality of the information matrix/vector. This is achieved through the algorithm **GraphSLAM_reduce** in Table 3. This algorithm takes as input Ω and ξ defined over the full space of map features and poses, and outputs a reduced matrix $\tilde{\Omega}$ and vectors $\tilde{\xi}$ defined over the space of all poses (but not the map!). This transformation is achieved by removing features m_j one at a time, in lines 4 through 9 of **GraphSLAM_reduce**. The bookkeeping of the exact indexes of each item in $\tilde{\Omega}$ and $\tilde{\xi}$ is a bit tedious, hence Table 3 only provides an intuitive account.

Line 5 calculates the set of poses $\tau(j)$ at which the robot observed feature j. It then extracts two sub-matrices from the present $\tilde{\Omega}$: $\tilde{\Omega}_{j,j}$ and $\tilde{\Omega}_{\tau(j),j}$. $\tilde{\Omega}_{j,j}$ is the quadratic sub-matrix between m_j and m_j, and $\tilde{\Omega}_{\tau(j),j}$ is composed of the off-diagonal elements between m_j and the pose variables $\tau(j)$. It also extracts from the information state vector $\tilde{\xi}$ the elements corresponding to the j-th feature, denoted here as ξ_j. It then subtracts information from $\tilde{\Omega}$ and $\tilde{\xi}$ as stated in lines 6 and 7. After this operation, the rows and columns for the feature m_j are zero. These rows and columns are then removed, reducing the dimension on $\tilde{\Omega}$ and $\tilde{\xi}$ accordingly. This process is iterated until all features have been removed, and only pose variables remain in $\tilde{\Omega}$ and $\tilde{\xi}$. The complexity of **GraphSLAM_reduce** is once again linear in t.

The last step in the GraphSLAM algorithm computes the mean and covariance for all poses in the robot path, and a mean location estimate for all features in the map. This is achieved through **GraphSLAM_solve** in Table 4. Line 3 computes the path estimates $\mu_{0:t}$. This can be achieved by inverting the reduced information matrix $\tilde{\Omega}$ and multiplying the resulting covariance with the information vector, or by optimization techniques such as conjugate gradient descent. Subsequently, **GraphSLAM_solve** computes the location of each feature in lines 4 through 7. The return value of **GraphSLAM_solve** contains the mean for the robot path and all features in the map, but only the covariance for the robot path.

The quality of the solution calculated by the GraphSLAM algorithm depends on the goodness of the initial mean estimates, calculated by **GraphSLAM_initialize**. The x- and y-components of these estimates affect the respective models in a linear way, hence the linearization does not depend on these values. Not so for the orientation variables in $\mu_{0:t}$. Errors in these initial estimates affect the accuracy of the Taylor approximation, which in turn affects the result.

To reduce potential errors due to the Taylor approximation in the linearization, the procedures **GraphSLAM_linearize**, **GraphSLAM_reduce**, and **GraphSLAM_solve** are run multiple times over the same data set. Each iteration takes as an input an estimated mean vector $\mu_{0:t}$ from the previous iteration, and outputs a new, improved estimate. The iterations of the GraphSLAM optimization are only necessary when the initial pose estimates have high error (e.g. more than 20 degrees orientation error). A small number of iterations (e.g. 3) is usually sufficient.

Table 5 summarizes the resulting algorithm. It initializes the means, then repeats the construction step, the reduction step, and the solution step. Typically, two or three iterations suffice for convergence. The resulting mean μ is our best guess of the robot's path and the map.

5. Mathematical Derivation of GraphSLAM

The derivation of the GraphSLAM algorithm begins with a derivation of a recursive formula for calculating the full SLAM posterior, represented in information form. We then investigate each term in this posterior, and derive from them the additive SLAM updates through Taylor expansions. From that, we will derive the necessary equations for recovering the path and the map.

5.1. The Full SLAM Posterior

It will be beneficial to introduce a variable for the augmented state of the full SLAM problem. We will use y to denote state variables that combine one or more poses x with the map m. In particular, we define $y_{0:t}$ to be a vector composed of the path $x_{0:t}$ and the map m, whereas y_t is composed of the momentary pose at time t and the map m:

$$y_{0:t} = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_t \\ m \end{pmatrix} \quad \text{and} \quad y_t = \begin{pmatrix} x_t \\ m \end{pmatrix} \tag{10}$$

The posterior in the full SLAM problem is $p(y_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t})$, where $z_{1:t}$ are the familiar measurements with correspondences $c_{1:t}$, and $u_{1:t}$ are the controls. Bayes rule enables us to factor this posterior:

$$\begin{aligned} & p(y_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) \\ &= \eta\, p(z_t \mid y_{0:t}, z_{1:t-1}, u_{1:t}, c_{1:t})\; p(y_{0:t} \mid z_{1:t-1}, u_{1:t}, c_{1:t}) \end{aligned} \tag{11}$$

where η is the familiar normalizer. The first probability on the right-hand side can be reduced by dropping irrelevant conditioning variables:

$$p(z_t \mid y_{0:t}, z_{1:t-1}, u_{1:t}, c_{1:t}) = p(z_t \mid y_t, c_t) \tag{12}$$

Table 3. Algorithm for Reducing the Size of the Information Representation of the Posterior in GraphSLAM

Algorithm GraphSLAM_reduce(Ω, ξ):

$\tilde{\Omega} = \Omega$

$\tilde{\xi} = \xi$

for each feature j do

  let $\tau(j)$ be the set of all poses x_t at which j was observed

  subtract $\tilde{\Omega}_{\tau(j),j}\ \tilde{\Omega}_{j,j}^{-1}\ \xi_j$ from $\tilde{\xi}$ at $x_{\tau(j)}$ and m_j

  subtract $\tilde{\Omega}_{\tau(j),j}\ \tilde{\Omega}_{j,j}^{-1}\ \tilde{\Omega}_{j,\tau(j)}$ from $\tilde{\Omega}$ at $x_{\tau(j)}$ and m_j

  remove from $\tilde{\Omega}$ and $\tilde{\xi}$ all rows/columns corresponding to j

endfor

return $\tilde{\Omega}, \tilde{\xi}$

Table 4. Algorithm for Updating the Posterior μ

Algorithm GraphSLAM_solve($\tilde{\Omega}, \tilde{\xi}, \Omega, \xi$):

$\Sigma_{0:t} = \tilde{\Omega}^{-1}$

$\mu_{0:t} = \Sigma_{0:t}\ \tilde{\xi}$

for each feature j do

  set $\tau(j)$ to the set of all poses x_t at which j was observed

  $\mu_j = \Omega_{j,j}^{-1}\ (\xi_j + \Omega_{j,\tau(j)}\ \tilde{\mu}_{\tau(j)})$

endfor

return μ, $\Sigma_{0:t}$

Table 5. The GraphSLAM Algorithm for the Full SLAM Problem with Known Correspondence

Algorithm GraphSLAM_known_correspondence($u_{1:t}, z_{1:t}, c_{1:t}$):

$\mu_{0:t}$ = **GraphSLAM_initialize**($u_{1:t}$)

repeat

  Ω, ξ = **GraphSLAM_linearize**($u_{1:t}, z_{1:t}, c_{1:t}, \mu_{0:t}$)

  $\tilde{\Omega}, \tilde{\xi}$ = **GraphSLAM_reduce**(Ω, ξ)

  $\mu, \Sigma_{0:t}$ = **GraphSLAM_solve**($\tilde{\Omega}, \tilde{\xi}, \Omega, \xi$)

until convergence

return μ

#1

x_i : Robot pose @ time t

$x_{1:i}$: Robot poses from time $1 \rightarrow t$

m: map

z_t : Measurement @ time t

z_t^i : Individual ith measurement beam

m_j : Map feature sensed by z_t^i

#2

$$z_t^i = h(x, m_j, i) + \varepsilon_t^i$$

#3

ε_t^i = Gaussian random variable modeling measurement noise

With $0 = \bar{x}$ & covariance Q_t

#12

$$p(x_t | u_t, x_{t-1}) = \text{cons.exp} -\frac{1}{2}(x_t - g(u_t, x_t))^T \times R_t^{-1}(x_t - g(u_t, x_{t-1}))$$

#13

$$p(x_{1:t}, m | z_{1:t}, u_{1:t})$$ #13

Probability that a path is $x_{1:t}$ & a map is m, given that measurements where $z_{1:t}$ & ~~(robot st)~~ controls asserted where $u_{1:t}$

#14

$$x_t = g(u_t, x_{t-1}) + \delta_t$$

u_t: Control asserted between $t-1 \to t$

g: State transition is governed by

$$\delta \sim N(0, R_t)$$

#10

$$P(z_t^i \mid x_t, m) = \text{const} \exp.-\frac{1}{2} \left(z_t^i - h(x_t, m, i)\right)^T \times Q_t^{-1} \left(z_t^i - h(x, m, i)\right)$$

Transpose

Not a matrix but a vector
↳ The vector is a LIDAR measurement so assume it has 360 values

Flips vector from column vector into row vector

Inverse covariance matrix

Probability of measurement being z_t^i given we are @ x_t & we know m is ...

#11

Difference between actual measurement & predicted measurement

#14

2 TYPES OF ARCS:

- MOTION ARCS:
  Link x_i & x_{i-1}
- MEASUREMENT ARCS:
  Link x_i to m_j

#15

Ω : Information Matrix
ξ : Information Vector
} ???

Simple example:

$$\Omega^{-1}\xi = \mu$$

Imagine the following robot poses

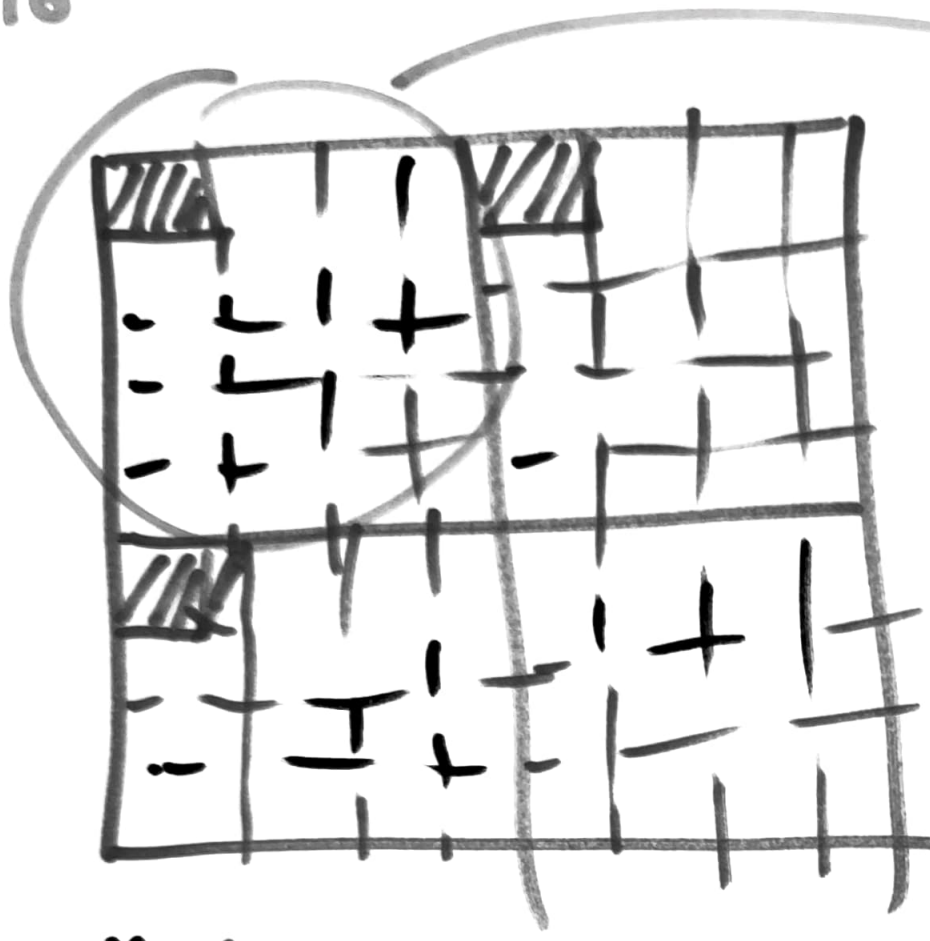
$x_0 = -3$, $x_1 = x_0 + 5$, $x_2 = x_1 + 3$

then $\mu = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$ & $\xi = \begin{bmatrix} -3 \\ 5 \\ 3 \end{bmatrix}$

So assuming perfect sensors & movement

$$\Omega = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix}$$

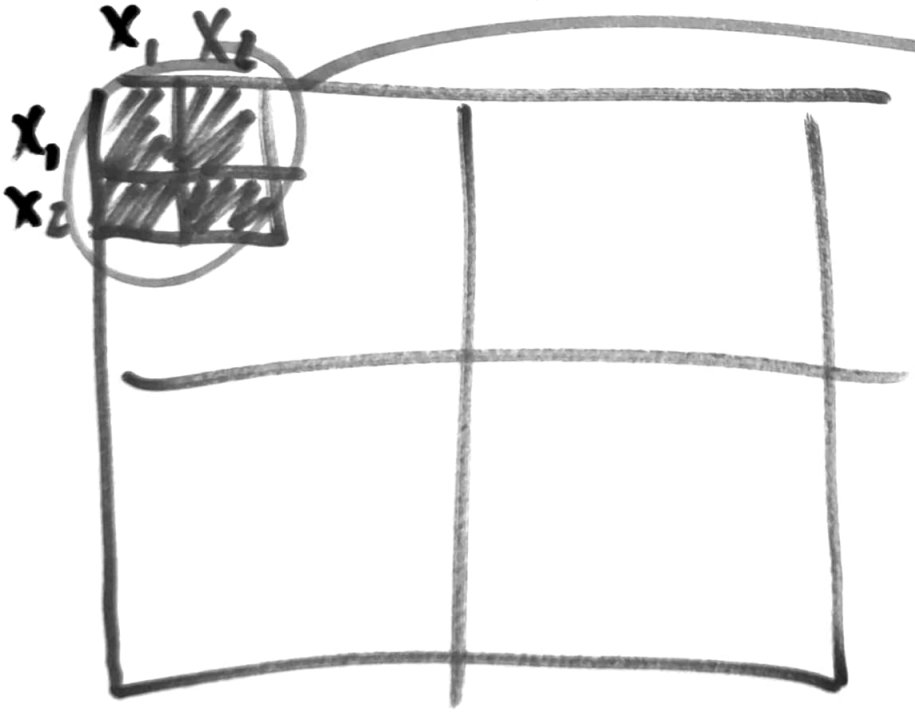
#16



Is this Ω ???

~~NO~~

NO, this is all Omega



Does this mean x_1 & x_2 are linked in 3 directions ???

Ω is split up to contain robot poses & landmarks

#29

Graph SLAM initialization requires full history of u_t (controls)

↳ Does graph SLAM require to run initialization @ every new t? ⇒ #37

#30

u_t a.k.a. the control @ t,
contains $(v_t \ \omega_t)$

So basically each control contains information about a robot's linear & angular velocity @ ~~point~~ time t

#31

GrapSLAM initial outputs an initial estimate of μ

Where:

$$\mu_0 = (0, 0, 0)$$

And each μ after is the previous μ plus this:

$$\begin{bmatrix} -\frac{v_t}{\omega_t}\sin(\text{prev angle}) + \frac{v_t}{\omega_t}\sin(\text{prev angle} + \Delta t\,\omega_t) \\ \frac{v_t}{\omega_t}\cos(\text{prev angle}) - \frac{v_t}{\omega_t}\cos(\text{prev angle} + \Delta t\,\omega_t) \\ \omega_t \Delta t \end{bmatrix}$$

Which is basically Figuring out the new position based on prev position & taking into account current velocity & angular velocity (beautifully simple ODOMETRY)

#35

In GRAPH SLAM Linearize:

Lines 4→9: use control data (u_t) & initial μ_{t-1} to generate initial Ω & ξ

Lines 10→21: Uses measurement data & initial μ_{t-1} to refine Ω & ξ

#36

After GraphSLAM Linearize, we have GraphSLAMsolve & graphSLAMreduce

↳ GRAPHSLAM reduce
removes redundant information

↳ GRAPH SLAM solve
~~reduces~~ solves the system of equations:

$$\Omega \times \xi = \mu$$

Returns final μ (robot trajectory)

#37

THE FULL GRAPH SLAM ALGORITHM:

```
GraphSLAM_Initialize(U_t)

Repeat until converge
    GraphSLAM_Linearize()
    GraphSLAM_reduce()
    GraphSLAM_solve()

return M
```

It basically runs linearize, reduce & solve until M converges at a location

To answer question #29, yes ✓ each new t this specific version of the algorithm requires a new initialization & each new t