2D Pose SLAM in GTSAM

Frank Dellaert

April 5, 2013

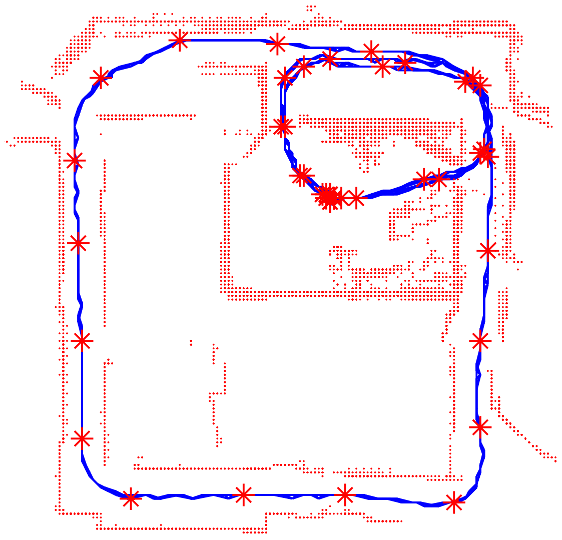

Figure 1: Pose SLAM applied to a Quadrotor Dataset, data courtesy of Nathan Michael.

PoseSLAM is a SLAM method that is especially well suited to working with odometry and laser-range finders. For example, the ICP algorithm essentially yields a **relative pose measurement** $\Delta\xi = (\Delta x, \Delta y, \Delta\theta)$ on the difference between two poses $\xi_1 = (x_1, y_1, \theta_1)^T$ and $\xi_2 = (x_2, y_2, \theta_2)^T$. Both odometry and matching successive laser scans will eventually accumulate so much error, however, that the robot will no longer know where it is with any degree of accuracy. However, if you can establish **loop closures**, e.g., pose constraints imposed by running ICP on laser-scans separated in time, but in the same location, you "correct" the entire map to something that will much more closely resemble the actual environment. Figure 1 above shows an example.

1 The Measurement Function

Let us first figure out what a measurement $\Delta\xi$ would be if we knew the poses ξ_1 and ξ_2 exactly. This is easier to do by embedding poses in $GL(3)$, as before, as 3×3 matrices:

$$T_1^w = \begin{bmatrix} R_1 & t_1 \\ 0 & 1 \end{bmatrix}$$

$$T_2^w = \begin{bmatrix} R_2 & t_2 \\ 0 & 1 \end{bmatrix}$$

Remember that the translation $t_1 = (x_1, y_1)$ and the rotation matrix R_1 is given by

$$R_1 = Rot(\theta_1) \triangleq \begin{bmatrix} \cos(\theta_1) & -\sin(\theta_1) \\ \sin(\theta_1) & \cos(\theta_1) \end{bmatrix},$$

and similarly for ξ_2. We can now find the relative pose between ξ_1 and ξ_2 by matrix operations:

$$\begin{aligned} T_2^1 &= (T_1^w)^{-1} T_2^w \\ &= \begin{bmatrix} R_1 & t_1 \\ 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} R_2 & t_2 \\ 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} R_1^T & -R_1^T t_1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R_2 & t_2 \\ 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} R_1^T R_2 & R_1^T(t_2 - t_1) \\ 0 & 1 \end{bmatrix} \end{aligned}$$

Because $R_1^T = Rot(-\theta_1)$ and $R_1^T R_2 = Rot(\theta_2 - \theta_1)$, we then have the following **measurement function**, which predicts the relative pose:

$$\widehat{\Delta\xi} = h(\xi_1, \xi_2) = \begin{bmatrix} Rot(-\theta_1)(t_2 - t_1) \\ \theta_2 - \theta_1 \end{bmatrix}$$

The predicted relative translation is just the translation between the two poses, rotated into the first coordinate frame, because that is where we measure the relative pose.

2 SLAM as Optimization

If we have many such relative pose measurements $\Delta\xi_i$, we would like to find the optimal set of poses $X = \{\xi_j\}_{j=1}^n$ compatible with the measurements. We can do this by minimizing the following sum-of-square errors criterion, which measures the difference between the predicted and measured relative poses:

$$E(X) = \frac{1}{2} \sum_i \| h(\xi_{j1}, \xi_{j2}) - \Delta\xi_i \|^2$$

where $\Delta\xi_i$ is a relative pose measurement between poses ξ_{j1} and ξ_{j2}. This corresponds to assuming independent and identical Gaussian measurement noise on the three components of the $\Delta\xi$ pose measurements. Other noise models are easily accommodated.

3 Linearization

How do we optimize something like this? Because $h(.,.)$ is a non-linear function, the first step is to linearize this measurement function. We can do this using a generalized Taylor expansion,

$$h(\xi_1 \oplus \delta_1, \xi_2 \oplus \delta_2) = h(\xi_1, \xi_2) \oplus \{H_1\delta_1 + H_2\delta_2\}$$

where $\delta_1\delta_2 \in \mathbb{R}^3$ are **pose updates**, and H_1 and H_2 are 3×3 **Jacobian matrices**, and where we define an update $\xi \oplus \delta$ *in the coordinate frame of* ξ, i.e.:

$$\xi \oplus \delta = \begin{bmatrix} t + Rot(\theta)\delta t \\ \theta + \delta\theta \end{bmatrix}$$

The reason we have to use the $\oplus$ operator is because $\xi \in SE(2)$, for which vector addition is not defined. The $\oplus$ operator above makes the math well behaved and in line with differential geometry, and is equivalent to composing with a small "delta pose":

$$\begin{bmatrix} Rot(\theta + \delta\theta) & t + Rot(\theta)\delta_t \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} Rot(\theta) & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} Rot(\delta\theta) & \delta t \\ 0 & 1 \end{bmatrix}$$

As shown in the Appendix, the Jacobians H_1 and H_2 can be calculated as

$$H_1 = -\begin{bmatrix} R_2^T R_1 & Rot(-\pi/2)R_2^T(t_1 - t_2) \\ 0 & 1 \end{bmatrix} \tag{1}$$

$$H_2 = \begin{bmatrix} I & 0 \\ 0 & 1 \end{bmatrix} \tag{2}$$

4 Solving

After linearization we have a new objective function, **linear** in the updates δ :

$$E(\delta) = \frac{1}{2}\sum_i \|h(\xi_{j1}, \xi_{j2}) + H_{j1}\delta_{j1} + H_{j2}\delta_{j2} - \Delta\xi_i\|^2 = \frac{1}{2}\sum_i \|A_i\delta - b_i\|^2 = \frac{1}{2}\|A\delta - b\|^2$$

where the large $3m \times 3n$ measurement Jacobian matrix A is composed out of the block rows

$$A_i = \begin{bmatrix} \dots & H_{j1} & \dots & H_{j2} & \dots \end{bmatrix}$$

and the $3m \times 1$ right-hand side b is composed out of the 3×1 prediction errors

$$b_i = h(\xi_{j1}, \xi_{j2}) - \Delta\xi_i$$

The optimal update δ is then given simply by setting the derivative of $\frac{1}{2}\|A\delta - b\|^2$ to zero:

$$A^T(A\delta^* - b) = 0$$
$$\left(A^TA\right)\delta^* = A^Tb$$
$$\delta^* = \left(A^TA\right)^{-1}A^Tb$$

After solving this *linear* system, we can simply update our estimate by doing

$$\xi_i^{t+1} = \xi_i^t \oplus \delta_i$$

for all poses ξ_i, and where t indexes over iterations. We might have to iterate this process of linearization/optimization several times until the solution converges. We can check this, for example, by checking the magnitude of the updates δ_i. This entire process as described above is called **Gauss-Newton non-linear minimization.**

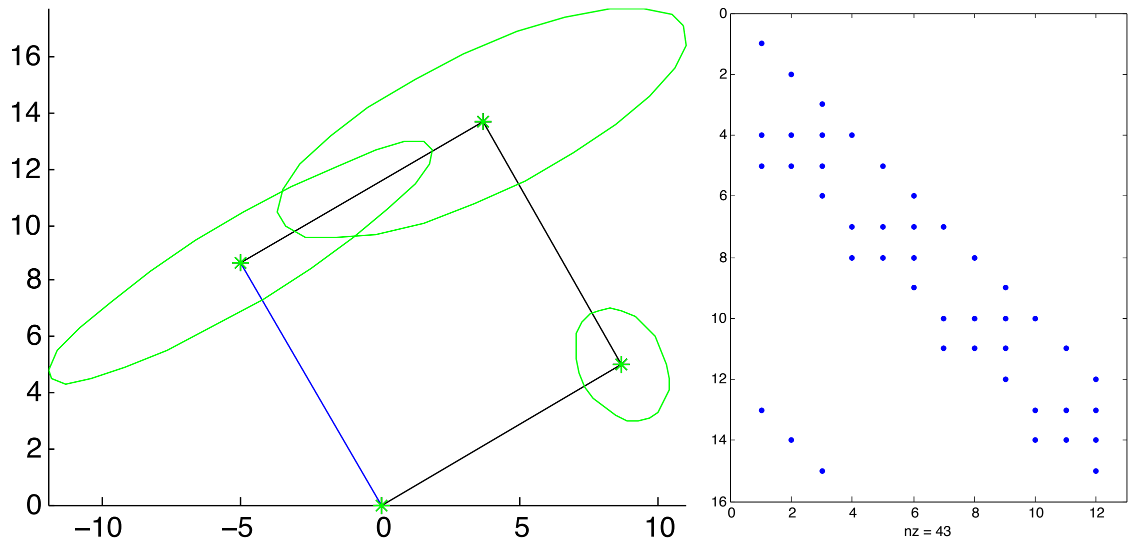

Figure 2: Pose SLAM example and corresponding Jacobian matrix A (after the last linearization).

5 GTSAM Example

Just to give you a flavor for the GTSAM (Georgia Tech Smoothing And Mapping) toolbox, I give an example below where there are just 4 poses involved and 4 relative constraints between them. The code below will run in MATLAB.

Create a factor graph and add three pose constraints:

```
graph = NonlinearFactorGraph;
graph.add(PriorFactorPose2(1, Pose2(0, 0, pi/6), noiseModel));
graph.add(BetweenFactorPose2(1, 2, Pose2(10, 0, pi/2), noiseModel));
graph.add(BetweenFactorPose2(2, 3, Pose2(10, 0, pi/2), noiseModel));
graph.add(BetweenFactorPose2(3, 4, Pose2(10, 0, pi/2), noiseModel));
```

Add a loop closure:

```
graph.add(BetweenFactorPose2(4, 1, Pose2(10, 0, pi/2), noiseModel));
```

Create an initial estimate:

```
initialEstimate = Values;
initialEstimate.insert(1, Pose2(0.5, 0.0, 0.2 ));
initialEstimate.insert(2, Pose2(20.3, 0.1, pi/2 ));
initialEstimate.insert(3, Pose2(20.1, 20.1, pi ));
initialEstimate.insert(4, Pose2(0.1, 20.0, -pi/2));
```

Optimize using a Gauss-Newton non-linear optimizer:

```
optimizer = GaussNewtonOptimizer(graph, initialEstimate);
result = optimizer.optimizeSafely();
```

This yields the result in Figure (2), shown with the sparse block-matrix A. Note that the block structure of the matrix A is clearly visible. In this case, we have a 5×4 block structure

$$A = \begin{bmatrix} H_{11} & & & \\ H_{21} & H_{22} & & \\ & H_{32} & H_{33} & \\ & & H_{43} & H_{44} \\ H_{51} & & & H_{54} \end{bmatrix}$$

corresponding to 5 factors (1 prior, 3 relative pose constraints, and a loop closure) and 4 poses.

6 Practicalities

In a practical implementation we do not actually compute the inverse of $A^T A$, because this would convert the very sparse $3n \times 3n$ matrix $A^T A$ into a dense matrix, its inverse $\left(A^T A\right)^{-1}$. Instead, all state of the art methods employ sparse Cholesky factorization,

$$A^T A = R^T R$$

where R is a *sparse* upper-triangular matrix. The optimal update δ^* can then be recovered using two, very efficient, back-substitutions $R^T y = A^T b$ and $y = R\delta^*$.

Gauss-Newton can get stuck in local minima, because the measurement functions are non-linear. However, there are two tricks that you can apply:

1. If you have odometry, or successive scan matches from ICP, then you can simply compose them to get a good initial estimate to start from. That will reduce the amount of computation because we will have to iterate less, and will also reduce the risk of getting stuck in a local minimum.

2. If you look carefully, in the 2D case the measurement function h is actually linear in the rotations:
$$h_\theta(\xi_1, \xi_2) = \theta_2 - \theta_1$$
Hence, a common trick is to first linearly solve for the rotation angles θ. After that, we can solve a second linear system associated with the translation measurements,
$$h_\theta(\xi_1, \xi_2) = Rot(-\theta_1)(t_2 - t_1)$$
where now $Rot(-\theta_1)$ is treated as a *constant* matrix.

Note that in 3D (the 6 DOF case) the translation is also linear once the rotations are known, but unfortunately the rotation part is non-linear. Still, if you have an accelerometer (which gives you pitch and roll with respect to gravity) and a compass (which gives you yaw), you can have good estimates for the rotations and quickly solve for them in the 3D case, as well.

A Jacobians

To calculate the Jacobians you express the measurement function in terms of the updates:

$$\begin{aligned} h(\xi_1 \oplus \delta_1, \xi_2 \oplus \delta_2) &= \left[\begin{array}{c} Rot(-\theta_1 - \delta\theta_1)(t_2 + R_2\delta t_2 - t_1 - R_1\delta t_1) \\ \theta_2 + \delta\theta_2 - \theta_1 - \delta\theta_1 \end{array} \right] \\ &= \left[\begin{array}{c} R_1^T(I + Rot(-\pi/2)\delta\theta_1)(t_2 + R_2\delta t_2 - t_1 - R_1\delta t_1) \\ \theta_2 + \delta\theta_2 - \theta_1 - \delta\theta_1 \end{array} \right] \end{aligned} \tag{3}$$

where we made use of the approximation, valid for small $\delta\theta_1$,

$$Rot(-\delta\theta_1) = \left[\begin{array}{cc} \cos(-\delta\theta_1) & -\sin(-\delta\theta_1) \\ \sin(-\delta\theta_1) & \cos(-\delta\theta_1) \end{array} \right] \approx \left[\begin{array}{cc} 1 & \delta\theta_1 \\ -\delta\theta_1 & 1 \end{array} \right] = I - Rot(\pi/2)\delta\theta_1$$

By equating (3) to

$$h(\xi_1, \xi_2) \oplus \delta = \left[\begin{array}{c} R_1^T(t_2 - t_1) + R_1^T R_2 \delta t \\ \theta_2 - \theta_1 + \delta\theta \end{array} \right]$$

we obtain the following identities:

$$R_1^T(t_2 - t_1) + R_1^T R_2 \delta t = R_1^T(I - Rot(\pi/2)\delta\theta_1)(t_2 + R_2\delta t_2 - t_1 - R_1\delta t_1)$$

$$\theta_2 - \theta_1 + \delta\theta = \theta_2 + \delta\theta_2 - \theta_1 - \delta\theta_1$$

After some algebra, we obtain the following for δt and $\delta\theta$:

$$\delta t = -R_2^T R_1 \delta t_1 - Rot(-\pi/2) R_2^T (t_1 - t_2)\delta\theta_1 + \delta t_2 \text{ and } \delta\theta = \delta\theta_2 - \delta\theta_1$$

When we separate the two delta pose updates and write in this in matrix form this becomes

$$\delta = \left[\begin{array}{c} \delta t \\ \delta\theta \end{array} \right] = -\left[\begin{array}{cc} R_2^T R_1 & Rot(-\pi/2) R_2^T (t_1 - t_2) \\ 0 & 1 \end{array} \right] \left[\begin{array}{c} \delta t_1 \\ \delta\theta_1 \end{array} \right] + \left[\begin{array}{cc} I & 0 \\ 0 & 1 \end{array} \right] \left[\begin{array}{c} \delta t_2 \\ \delta\theta_2 \end{array} \right]$$

and hence we recover the Jacobians (1) and (2).

Note that the above is an introductory way to explain the concept of **differentials**, which describe the differential geometry of the mappings between manifolds, in this case the measurement function $h : SE(2) \times SE(2) \rightarrow SE(2)$. A similar story can be told for $SE(3)$, with only a slight increase of complexity. In particular, an update to a pose $\xi \in SE(3)$ is now given as

$$\xi \oplus \delta = (R, t) \oplus \left[\begin{array}{c} \omega \\ v \end{array} \right] = (R(I + [\omega]_\times), t + Rv)$$

Note that in 3D the GTSAM convention is to put rotation first, then translation. The Jacobians corresponding to the measurement function are then

$$H_1 = -\left[\begin{array}{cc} R_2^T R_1 & 0 \\ \left[R_2^T (t_1 - t_2)\right]_\times R_2^T R_1 & R_2^T R_1 \end{array} \right] \text{ and } H_2 = \left[\begin{array}{cc} I & 0 \\ 0 & I \end{array} \right]$$

As you can see, the 3D and 2D Jacobians look quite similar, modulo the change in order. The rotation by $Rot(-\pi/2)$ in the 2D case plays the same role as the cross product in 3D.