SLAM

- State estimation: where is the robot
- Localization: Given known landmarks, where is the robot?
- Simultaneous Localization and Mapping (SLAM): Build a map while localizing with respect to it.
  - Add landmarks to state vector: jointly estimate features and robot pose.
Kalman Filter

- Kalman Filter
  - Optimal linear state estimation algorithm
  - Recursive
  - "Extended" Kalman Filter: handle nonlinear propagation/observation functions by linearizing
  - For many years, the classic mapping method.

- How can we address shortcomings?
  - Computational/Memory complexity
  - Linearization?

Non-Linear SLAM

- Robot Trajectory: a pose graph

\[ z_0 = f_0(x_0, x_1) \]
\[ z_1 = f_1(x_1, x_2) \]
\[ z_2 = f_2(x_2, x_3) \]
\[ z_3 = f_3(x_3, x_4) \]
\[ z_4 = f_4(x_0, x_5) \]
\[ z_5 = f_5(x_2, x_5) \]
\[ z_6 = f_6(x_3, x_5) \]
Non-Linear SLAM

- Non-linear SLAM
  - Reminder: consider observation equations
    \[ z_1(x) = x_1 \]
    \[ z_2(x) = x_2 \]
    \[ z_3(x) = \sqrt{x_1^2 + x_2^2} \]
  - Linearize and stack them.
    \[
    \begin{bmatrix}
    z_1(x) \\
    z_2(x) \\
    z_3(x)
    \end{bmatrix} \approx
    \begin{bmatrix}
    1 & 0 \\
    0 & 1 \\
    \frac{x_1}{d} & \frac{x_2}{d}
    \end{bmatrix}
    \begin{bmatrix}
    \Delta x_1 \\
    \Delta x_2
    \end{bmatrix} +
    \begin{bmatrix}
    x_1^- \\
    x_2^- \\
    d^-
    \end{bmatrix}
    \]

Non-Linear SLAM: Review

- Stacking observations
- Observations want \( Jd = r \)
  - Over-determined. Each observation associated to covariance
- Minimize the cost function:
  \[
  \chi^2 = (z(x) - \hat{z})^T \Sigma_z^{-1} (z(x) - \hat{z})
  \approx (J \Delta x + \tilde{z} - z)^T \Sigma_z^{-1} (J \Delta x + \tilde{z} - z)
  \]
- Manipulate a bit:
  \[
  (J^T \Sigma_z^{-1} J)d = J^T \Sigma_z^{-1} r
  \]
  \[
  \Sigma_z = \]
Non-Linear SLAM

- How do we solve for $d$?
  \[(J^T \Sigma_x^{-1} J)d = J^T \Sigma_x^{-1} r\]
- Easy answer: Matrix inversion
- What would covariance of $x$ be?
  \[\Sigma_x = (J^T \Sigma_x^{-1} J)^{-1}\]
- Let's think about this more carefully…
  - Specifically, what is the structure of the Jacobians and the information matrix? $J^T \Sigma_x^{-1} J$
  - Let's look at $J$, then $\Sigma_x^{-1}$, then the product.

We could stop here: this is a simple SLAM algorithm!

What is its time/memory complexity? (compare to EKF)

Structure of J

- Odometry constraint (1,2)

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Loop Closure between (4, 7)

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Structure of J

- Relationship to graph

\[
\begin{align*}
J^x_0 &= \begin{bmatrix}
* & * & 0 & 0 & 0 \\
0 & * & * & 0 & 0 \\
0 & 0 & * & * & 0 \\
0 & 0 & 0 & * & 0 \\
* & 0 & 0 & 0 & * \\
0 & 0 & * & 0 & 0 \\
0 & 0 & 0 & * & 0 
\end{bmatrix} \\
J^x_1 &= \begin{bmatrix}
* & 0 & 0 & 0 & 0 \\
0 & * & * & 0 & 0 \\
0 & 0 & * & * & 0 \\
0 & 0 & 0 & * & 0 \\
* & 0 & 0 & 0 & * \\
0 & 0 & * & 0 & 0 \\
0 & 0 & 0 & * & 0 
\end{bmatrix} \\
\end{align*}
\]

Lots of zeros

Structure of $\sum^{-1}_Z$

- Our observations are independent:
  - Observation covariance matrix is block diagonal
  - What about its inverse?
    - also block diagonal

- Is independence of observations a reasonable assumption?
Because $\Sigma^{-1}_z$ is block diagonal, we have:

$$J^T \Sigma^{-1}_z J = \sum_i J_i^T \Sigma^{-1}_i J_i$$

This is also evident from our original cost function: we're minimizing the sum of the squared errors of each observation.
Structure of $J^T \Sigma_z^{-1} J$

- Key observations:
  - The information matrix remains sparse
  - (Even though the covariance matrix becomes dense)
  - Directly encodes connectivity of pose graph
    - i.e., the adjacency matrix of the Bayes Net

- Sparsity is GOOD— we can exploit it!

Why is sparsity useful?

- Sparse matrix data structures
  - Make computation a function of # of non-zero elements
  - Use memory proportional to # of non-zero elements
**Sparse Matrix Representation**

- **CSR: Compressed Sparse Row**
- \( x = \begin{bmatrix} a & b & 0 & 0 & 0 & c & 0 & 0 & d & e & 0 & 0 & 0 & 0 & f \end{bmatrix} \)
  
  \( x = \{ \text{indices} = \{0, 1, 5, 8, 9, 15\}, \text{values} = \{a, b, c, d, e, f\} \} \)

**Sparse Matrix: Dot Product**

```cpp
double dotProduct(CSRVec a, CSRVec b) {
    int aidx = 0, bidx = 0;
    while (aidx < a.nz && bidx < b.nz) {
        int ai = a.indices[aidx], bi = b.indices[bidx];
        if (ai == bi) {
            acc += a.values[aidx] * b.values[bidx];
            aidx++;
            bidx++;
            continue;
        }
        if (ai < bi) aidx++;
        else bidx++;
    }
    return acc;
}
```

```
a = \begin{bmatrix} a & b & 0 & 0 & 0 & c & 0 & 0 & d & e & 0 & 0 & 0 & 0 & f \end{bmatrix}
a = \{ \text{indices} = \{0, 1, 5, 8, 9, 15\}, \text{values} = \{a, b, c, d, e, f\} \}
```

```
b = \begin{bmatrix} 0 & 0 & 0 & g & h & i & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}
b = \{ \text{indices} = \{3, 4, 5\}, \text{values} = \{g, h, i\} \}
```
Cholesky Decomposition

- We don't actually do Gaussian Elimination
  - G.E. computes the inverse
  - So we have to store the inverse!
  - Numerical stability issues

- We use Cholesky decomposition instead
  - Cholesky decomposition works for all symmetric + SPD

\[ A \Delta x = \Delta x \Delta = b \]

Cholesky Decomposition (Review)

\[
\begin{bmatrix}
16 & 4 & 8 \\
4 & 37 & 20 \\
8 & 20 & 14
\end{bmatrix}
= \begin{bmatrix}
4 & 0 & 0 \\
1 & 6 & 0 \\
2 & 3 & 1
\end{bmatrix}
= \begin{bmatrix}
4 & 1 & 2 \\
6 & 3 & \\
& 1
\end{bmatrix}
\]
Backsolve

\[
\begin{align*}
4 & \quad 1 & \quad 2 & \quad 48 \\
1 & \quad 6 & \quad & \quad 138 \\
2 & \quad 3 & \quad 1 & \quad 90 \\
\end{align*}
\]

\[
\begin{align*}
4 & \quad 1 & \quad 2 & \quad 48 \\
1 & \quad 6 & \quad & \quad 138 \\
2 & \quad 3 & \quad 1 & \quad 90 \\
\end{align*}
\]

\[
\begin{align*}
L & \quad v & \quad b \\
4 & \quad 12 & \quad 48 \\
1 & \quad 21 & \quad 138 \\
2 & \quad 3 & \quad 90 \\
\end{align*}
\]

Backsolve

\[
\begin{align*}
4 & \quad 1 & \quad 2 & \quad 48 \\
1 & \quad 6 & \quad & \quad 138 \\
2 & \quad 3 & \quad 1 & \quad 90 \\
\end{align*}
\]

\[
\begin{align*}
4 & \quad 1 & \quad 2 & \quad 48 \\
1 & \quad 6 & \quad & \quad 138 \\
2 & \quad 3 & \quad 1 & \quad 90 \\
\end{align*}
\]

\[
\begin{align*}
L^T & \quad x & \quad v \\
4 & \quad 1 & \quad 2 \\
6 & \quad 3 & \quad 1 \\
1 & \quad 2 & \quad 3 \\
\end{align*}
\]

\[
\begin{align*}
1 & \quad 21 \\
2 & \quad 3 \\
3 & \quad 3 \\
\end{align*}
\]
Cholesky Decomposition

- Strategy: incrementally triangularize A starting from the top.
- Which of these operations can we do efficiently on sparse matrices?
- Notice: if A is sparse, output will tend to be sparse
  - But there’s "fill-in" related to how each variable is connected to other variables.

Cholesky Decomposition: Backsolve

- Solving:
  - Let: $v = L^T x$. Solve $Lv = b$ for $v$.
    - How?
  - $L^T x = v$
    - Solve for $x$
Midterm 2

- Mean = 55
- StdDev = 13
- High = 77

- A-ish: 60+
- B-ish: 50+