22.1 Introduction

For a robot to function well in a new environment, it needs to know 1. what its surroundings look like, 2. its own location in the surroundings.

The dilemma: we are able to easily obtain map from robot or vice versa, but in real world settings we do not know either. Hence, SLAM is the problem of jointly estimating the location of a robot in its surroundings along with a map of those surroundings.

The focus of the lecture is on feature-based visual SLAM.

22.1.1 Outline of feature/landmark-based SLAM algorithm

1. Build a map with reference to the current location.
2. Move and estimate the updated location.
3. Observe mapped landmarks, and initialize new landmarks.
4. Use observations to update the position estimate and landmarks’ positions.

22.1.2 Problem formulation

1. Assume we have a sequence of $n$ discrete timesteps. Denote $x_t$ the pose of the robot at time $t$. Equivalently, we have a sequence of $n$ images, which were taken from camera poses $x_0 \ldots x_n$. Constrain $x_0$ to be the global reference frame.
2. Assume that our entire scene is composed of $m$ stationary landmarks. Denote $f_i$ the configuration of the $i$th landmark or feature in the scene.
3. Denote $z_{ij}$ the measure of the $i$th landmark from the $j$th pose.
4. Denote $u_i$ the $i$th odometry measurement taken at time $i$.
5. Assume we know an underlying motion model $g$ that describes the discrete time evolution of the dynamic system

$$x_{t+1} = g(x_t, u_t) + w_t, w_t \sim N(0, \Sigma_w)$$

where $w_t$ is additive zero-mean Gaussian noise.
6. Assume we know the sensor model $h$ that can be used to predict the measurement $z_{ij}$ given the location of a landmark $f_i$ as viewed from a camera pose $x_j$

$$z_{ij} = h(f_i, x_j) + v_t, v_t \sim N(0, \Sigma_v)$$

where $v_t$ is additive zero-mean Gaussian noise.
Then, the SLAM problem can be stated as that of estimating the robot states \((x_1, \ldots, x_n)\) and the landmark states \((f_1, \ldots, f_m)\) given the landmark measurements \(\{z_{ij}\}\) and the odometry measurements \((u_0, \ldots, u_n)\).

### 22.1.3 Optimization approach

We can solve the above SLAM problem by solving the unconstrained optimization problem

\[
\arg\min_{X,F} \sum_{t=1}^{n} \|x_t - g(x_{t-1}, u_{t-1})\|_\Sigma^{-1} + \sum_{i,j} \|z_{ij} - h(f_i, x_j)\|_\Sigma^{-1}
\]

where \(X = \{x_0, \ldots, x_n\}\) and \(F = \{f_1, \ldots, f_m\}\). (Note: there was a typo in the slides where \(v\) and \(w\) are flipped.)

**Notation:** \(\|v\|^2_S := v^T S v\)

### 22.1.4 Anatomy of a modern SLAM system

![Components of a modern SLAM system](image)

**Figure 22.1:** Components of a modern SLAM system

### 22.2 Front End

Front end is responsible for processing raw sensor measurements into abstracted data (landmark measurements, estimated landmark positions, estimated robot pose etc.)

#### 22.2.1 Feature extraction

The most popular choice of landmarks for visual SLAM are *keypoints* (repeatably detectable points from raw images from multiple views of the scene). Each keypoint is assumed to correspond to a point feature in the 3D environment. We detect them by looking for distinctive "image patches".

Visual corner points work well for our purpose. Commonly used corner detectors include FAST, HARRIS, DoG.
22.2.2 Data association

We need to associate newly extracted keypoints to a feature we have already seen before (i.e. detect the same point from multiple views).

Descriptor is a vector that explains the distinctive features of an image patch around a feature point in a way that is comparable, informative, and invariant to camera orientation. Commonly used methods are BRISK, SURF, SIFT, BRIEF, ORB. We can use nearest-neighbor to establish correspondences between descriptors of two images.

22.2.2.1 Feature tracks

We need to keep track of features throughout the image sequence by matching descriptors between consecutive frames. Given an estimate of the camera pose, we can get an estimate of feature location by triangulating from multiple views.

Tricks for improving data association:

1. Using binary descriptors can make comparison faster between hamming distance is much faster to compute than using float vector distance. (BRISK, ORB)
2. Pre-rotate image patches before describing them to achieve rotation invariance. (ORB)
3. When IMU is available, can also pre-rotate features to be aligned with gravity vector (OK-Vis).
22.2.3 Outlier rejection

Nearest neighbor leads to plenty of erroneous matches because many corners look "alike" in real-world data. Hence, we need to reject outlier matches. The two common methods:

1. RANSAC (Random sampling and consensus)
   
   (a) Sample 8 points. Use the 8 point algorithm to estimate fundamental matrix, and count the number of correspondences that violate the epipolar constraints. Repeat the process several times, and choose the fundamental matrix that have the least number of violations. Then reject the points that violate epipolar constraint imposed by our fundamental matrix.
   
   (b) Estimate Perspective-N-Point solution, reject matches with high reprojection error.

2. Mahalanobis distance test ("chi-squared rejection test"): When a known feature is detected, accept the match only if the location of the new image feature is within 3 standard deviations of the expected location given the current best estimate.

\[
d(x, \mu) = \|x - \mu\|_{S^{-1}} = \sqrt{(x - \mu)^T S^{-1} (x - \mu)}
\]

22.2.4 Landmark initialization

When a feature point is detected by not matched to an existing feature track, it is treated as a new landmark. The front end is responsible for coming up with an initial guess for its 3D location. Some methods for initialization:

1. If a stereo camera or depth camera is available, then triangulation or depth measurement can used to directly estimate the landmark location.

2. Delay adding the feature until it is observed from another view, then we can perform triangulation.


22.3 Back End

With the abstracted data produced by the front-end, the back-end is responsible for constructing and solving the SLAM optimization problem.

22.3.1 Optimization Problem

To solve for the desired information at time \(t=n\), we can formulate the problem as a minimization problem over a cost function in the following form,

\[
c(x) = \|x - \mu_n\|_{\Sigma_n}^2 + \sum_{t=1}^{n} \|x_t - g(x_{t-1}, u_{t-1})\|_{\Sigma_v}^2 + \sum_{i,j} \|z_{ij} - h(f_i, x_j)\|_{\Sigma_w}^2
\]
in which we denote the variable \( x_i \) to be the optimal estimation of robot’s egomotion at timestep \( i \), and the variable \( f_j \) as the environment map that optimize the given optimization problem. And the term \( x \) present the concatenation of all the optimization variables in problem. In this optimization cost problem, the first term is the prior over the entire vector \( x \), where \( \mu_n \) and \( \Sigma_n \) denoted the mean and covariance of our estimation of the prior.

Something worth noticing about is that although our measurements don’t hold any information about the global reference frame, the optimization problem is still well-constrained because we’re using a prior over the initial distribution.

Because the optimization problem is based on function \( g \) and \( h \), it is highly non-linear possibly non-convex and we’re not guaranteed to find global optimal. However we can iteratively optimize our guess with gradient of the function. Since our initial guess is provided by the front end, the more accurate it is, the easier for us to obtain the optimal solution within reasonable steps.

### 22.3.2 Nonlinear Least Square Problem

A nonlinear least squares (NLLS) problem is an optimization problem of the form

\[
\min_x \| f(x) \|^2
\]

where \( f \) is called the cost vector and is assumed to be smooth.

And in the special case where \( f \) is affine, the NLLS problem becomes a standard linear least square problem. If \( f \) is in the form \( f(x) = Ax + b \), this becomes the linear least square problem

\[
\min_x \| Ax + b \|^2
\]

and one can find the unique optimal solution, assuming that \( A \) is full rank, by solving the equation

\[
(A^\top A)x^* = -A^\top b
\]

In order to solve a NLLS problem, one can locally approximate function \( f \) by an affine function using a Taylor expansion, then solve the resulting linear least squares system. However we need a linearization point to start the optimization and such \( x_0 \) is provided by the front-end as the initial guess.

We can linearize about \( x_0 \) to get an affine Taylor approximation for \( f \). In particular, compute the Jacobian

\[
J = \frac{\partial f}{\partial x} \bigg|_{x_0^*}
\]

and then write

\[
f(x_0^* + \Delta x) \approx f(x_0^*) + J\Delta x
\]

Then we can solve for the optimal displacement of \( x \) by solving the linear least square optimization problem

\[
\min_{\Delta x} \| f(x_0^*) + J\Delta x \|^2
\]

And same as before, such LLS problem can be solved by the normal equation

\[
(J^\top J)\Delta x = -J^\top f(x_0^*)
\]

And then we can update our guess of state \( x \) by

\[
x^* \leftarrow x_0^* + \Delta x
\]
and $x^*$ would serve as our new guess or possible linearization point to a new iteration of optimization. And the optimization iteration should end if converged or reached maximum iteration. The figure below illustrates linearization and optimization within one iteration.

With the techniques above, we can solve the SLAM optimization problem by writing it as a NLLS optimization problem

$$C(x) = \begin{bmatrix} 
\Sigma_n^{-1/2} (x - \mu_n) \\
\Sigma_w^{-1/2} (x_1 - g(x_0, u_0)) \\
\vdots \\
\Sigma_w^{-1/2} (x_n - g(x_{n-1}, u_{n-1})) \\
\vdots \\
\Sigma_v^{-1/2} (z_{1i} - h(f_1, x_i)) \\
\vdots \\
\Sigma_v^{-1/2} (z_{mj} - h(f_m, x_j)) 
\end{bmatrix}$$

then we can rewrite $c(x)$ to be

$$c(x) = C^T C = \|C(x)\|_2^2$$

Then given a linearization point $\mu$, a simple transformation shows that we can also write

$$c(x) \approx \|c(\mu) + J(x - \mu)\|_2^2 \approx (x - \hat{\mu})^T \hat{\Sigma}^{-1}(x - \hat{\mu}) = \|x - \hat{\mu}\|_{\hat{\Sigma}^{-1}}^2$$
where
\[ \hat{\Sigma} = (J^\top J)^{-1}, \quad \hat{\mu} = \mu - (J^\top J)^{-1} J^\top C(\mu) \]

And we call this the linearization cost since it has a nice interpretation as a prior with mean \( \hat{\mu} \) and covariance \( \hat{\Sigma} \). Notice that \( \simeq \) means the two expressions are equivalent up to a constant offset; so minimizing one is the same as minimizing the other.

### 22.3.3 Marginalization

In SLAM, we want estimates in real time. We don’t want to wait until we have all the measurements and then perform one round of inference until convergence. Rather, we want to build up the optimization problem incrementally, and take Gauss-Newton steps at each iteration to improve our estimate. However, the size of the optimization problem grows linearly with time as additional measurements are registered every time step. This means the complexity of taking Gauss-Newton steps also grows, and very soon it becomes infeasible to perform inference in real-time.

One way to do this is to simply drop terms corresponding to old robot poses. However, this throws away all information related to those poses, and thus makes subsequent estimates sub-optimal. This motivates us to find a way to eliminate variables from the cost function while still maintaining information about how those variables affect the remaining variables. This operation is called marginalization.

As above, let \( x \) be the vector of all optimization variables. Let \( x_M \) be the set of variables we wish to remove, or marginalize. Let \( x_K \) be the set of remaining variables (that we wish to keep). And our objective is to remove variables \( x_M \) from the cost function while maintaining some information about how they affect \( x_K \). We will do this by introducing a new linearized prior term over \( x_K \) which encapsulates the relationships between the variables in \( x_K \) induced through the variables in \( x_M \).

Our original cost function \( c \) and cost vector \( C \) are functions of both \( x_K \) and \( x_M \).
\[
c(x_K, x_M) = \|C(x_K, x_M)\|_2^2
\]

And we can partition \( c \) into terms that depend on \( x_M \), and those that depend only on \( x_K \).
\[
c(x_K, x_M) = c_1(x_K) + c_2(x_K, x_M) = \|c_1(x_K)\|_2^2 + \|C_2(x_K, x_M)\|_2^2
\]

And split the minimization as
\[
\min_{x_K, x_M} c(x_K, x_M) = \min_{x_K} \left( c_1(x_K) + \min_{x_M} c_2(x_K, x_M) \right)
\]
\[
= \min_{x_K} \left( \|C_1(x_K)\|_2^2 + \min_{x_M} \|C_2(x_K, x_M)\|_2^2 \right)
\]

We can use Taylor expansion and algebra to prove that up to first order, we have
\[
\min_{x_K} \left( c_1(x_K) + \min_{x_M} \|C_2(x_K, x_M)\|_2^2 \right) = \min_{x_K} \left( c_1(x_K) + \|x_K - \hat{\mu}_K\|_{\hat{\Sigma}_K^{-1}}^2 \right)
\]
for
\[
\hat{\Sigma}_K^{-1} = J_K^\top \left[ I - J_M \left( J_M^\top J_M \right)^{-1} J_M^\top \right] J_K
\]
\[
\hat{\mu}_K = \mu_K - \hat{\Sigma}_K^\top J_K \left[ I - J_M \left( J_M^\top J_M \right)^{-1} J_M^\top \right] C_2(\mu_K, \mu_M)
\]

where \( J_K \) and \( J_M \) are the Jacobians of \( C_2(x_K, x_M) \) with respect to \( x_K \) and \( x_M \) respectively, evaluated at \( \mu \), and \( \mu = (\mu_K, \mu_M) \) be a given linearization point.
22.3.4 SLAM Algorithm Template

Start off with an initial guess $\mu_0$ and a prior covariance $\Sigma_0$ over the initial pose $x_0$. Set the cost function to $c(x) = \|x - \mu_0\|^2_{\Sigma_0^{-1}}$. Set $t = 0$.

1. Receive a new image measurement for time $t+1$.
2. Receive a new odometry measurement $\mu_t$. Add term $\|x_{t+1} - g(x_t, u_t)\|^2_{\Sigma_w^{-1}}$ to $c$. Add an initial guess $g(x_t, u_t)$ for $x_{t+1}$ to the estimate $\mu_t$.
3. If needed, perform 1 or more Gauss-Newton steps to update $\mu_t$ or linearly approximate the cost function.
4. Receive landmark measurements. Add terms of the form $\|z - h(f, x)\|^2_{\Sigma_v^{-1}}$ to $c$. Add initial guesses for newly detected landmarks to $\mu_t$ supplied by the front-end.
5. If needed, perform 1 or more Gauss-Newton steps to update $\mu_t$ or linearly approximate the cost function.
6. If needed, marginalize away some subset of variables. This will update both $\mu_t$ and $\Sigma_t$ (the current prior term in the cost function). Drop some other subset of variables.
7. Set $\mu_{t+1} \leftarrow \mu_t$, $\Sigma_{t+1} \leftarrow \Sigma_t$, $t \leftarrow t + 1$ and repeat.

22.3.5 Extension

Many state-of-the-art algorithms can be described in terms of the algorithm template above:

1. Extended Kalman Filter (EKF): Perform 1 Gauss-Newton step after introducing measurement terms. Marginalize all poses except the most recent one. Do not marginalize any features.
2. Iterative EKF: Same as EKF, except perform multiple Gauss-Newton steps.
3. Multi-State Constraint Kalman Filter (MSCKF): Performs 1 Gauss-Newton step, marginalizes all poses except the most recent $k$ poses, marginalizes features as soon as they are no longer seen by the current pose.
4. Fixed-lag smoother: Same as MSCKF, but performs multiple Gauss-Newton steps.
5. Open Keyframe Visual-Inertial SLAM (OK-Vis): Same as fixed-lag smoothing, but drops some intermediate poses.
6. Graph SLAM: Performs multiple Gauss-Newton steps when all information has been collected, does not marginalize any variables.
7. Pose Graph SLAM: Same as Graph SLAM, but measurements are pose constraints.
8. Bundle adjustment: Same as Graph SLAM, but without motion constraints. May introduce additional constraints to deal with pose ambiguity.

22.3.6 Optimization on Manifolds

In reality, our optimization variables lie on some smooth manifold $M$, not in regular Euclidean space. Ex: rotations in $SO(3)$ or poses in $SE(3)$.
22.3.6.1 Local coordinates

In order to rigorously define Taylor expansion, we need to be able to say what it means to "add a small $\Delta x"$ for a manifold-valued variable $x$, like a rotation matrix. We need a map $\varphi_x : U_x \to \mathbb{R}^n$ that will smoothly (and invertibly) map us from a neighbourhood $U_x$ of $x$ in $M$ into a neighbourhood of 0 in $\mathbb{R}^n$. $\varphi_x^{-1}$ will then map small vector quantities to small deviations from $x$ on the correct $x$ manifold. Such a map is called a local coordinate map centered at $x$.  

Figure 22.5: local coordinate map centered at $x$