EECS C106B / 206B Robotic Manipulation and Interaction **Spring 2022**

Lecture 23: SLAM, Part I

Scribes: Arul Gupta, Svein Jostein Husa, Leonard Wei

23.1 Introduction

In this lecture, we will get an introduction to SLAM (Simultaneous Localisation and Mapping). First, the problem setup and terminology are introduced. Then, the different methods used in the Front End and Back End of SLAM are explained in more detail. Finally, we get to hear some of the next steps that are worked on in SLAM.

23.1.1 What is SLAM?

Simultaneous Localization and Mapping (SLAM) is about doing localization and mapping at the same time. Given a perfect map, localization is solving where your location is in that map. Mapping is about generating a map by using the position of the objects surroundings you, given a egocentric coordinate system.

Localization - Estimate the robot state (pose, velocity etc.)
Mapping - Construct a map (landmarks, features, etc.) of its surroundings.

23.1.2 Why is SLAM useful

Why should we care about SLAM? Some of the reasons are:

- Research on SLAM has produced the visual-inertial odometry algorithms used today.
- It is capable of establishing **loop closures**. If you traverse a unknown environment and come back to your original position and have accumulated errors along the way, loop closure allows for a global consistency check. You can have good local mapping throughout a loop, but globally it can get very poor. Loop closure allows the map to "snap back" to the correct configuration.
- SLAM is necessary for many applications that require a globally consistent map

It could also be unnecessary in in some situations. For instance in scenarios where you know your localization sufficiently with GPS and LiDAR. Also unnecessary when you don't need a metric map for the navigation tasks.

23.1.3 Is SLAM Solved?

Is SLAM a solved problem? Well, it depends...
It depends on the sensors you are using, the more sensors, the more accurate estimates. It depends on the

environment you are in. Is it an indoor static environment or outdoor dynamic environment? It also depends on the performance requirements in the question.

Generally SLAM is solved for vision-based slow robotic systems. For instance if you want to map a 2D environment for a robot with encodes and lasers. However it is not solved for highly agile robots with rapidly moving environments. There are still many opens questions to explore with SLAM.

23.1.4 Problem Setup

In SLAM, you move around and based on your understanding of how you moved and how the orientation has changed, you observe the environment and try to make some sort of consistency between the current and the previous scene in front of you. You try to set the location of the objects that you observe with your sensors, and your location. The problem setup can be summed up in the following way:

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.

23.1.5 Terminology

- **Robot Poses**
  The position and orientation of the robot camera. Position is 2D or 3D depending on the environment. It is described by a transnational component and an rotational component. Quaternions are often used for the rotational component.
- **Feature**
  Positions of notable attributes in images. E.g. in a room corners, tables and chairs could be notable attributes. Then those attributes can help you to figure out the environment and where you are in the environment. Features are used to identify correspondence between different images of the same part of the environment.
- **Image measurement**
  2D measurements of the surroundings captured by sensors that provides metric information for robot poses and features.
- **State Vector**
  A collection of the physical quantities describing the robot. It is stored as a long vector. Can include for instance IMU state, poses and feature position estimates. When you decide which variables you want to include in your state vector it is a trade-off between speed and accuracy. More state variables lowers computational speed.

23.1.6 SLAM Front End and Back End

In SLAM the we separate between the front end and the back end. The front end extracts and processes features and converts signals from sensors into abstracted data. The back end does inference over abstracted data (e.g., MAP Estimation of robot states).

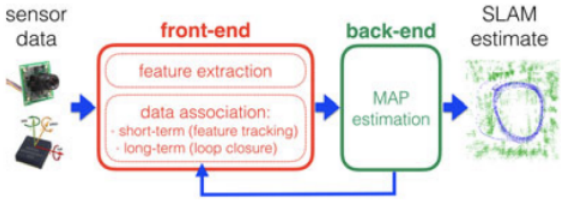


Figure 23.1: Front end and back end in a typical SLAM system

23.2 Front End

23.2.1 Feature Extraction

Goal Extract repeatably detected features from raw images

Approach To be able to consistently extract these features, the chosen feature should be highly detectable. Corners are a great example of this, since you can identify a significant change in the intensity of image pixels. Examples of robust corner detection algorithms include FAST, Harris, and DoG.

23.2.2 Data Association

Goal Reliably detect the same point in multiple views of a scene

Approach The image patch around a feature should be described in a way that is "comparable, informative, and invariant" to camera orientation. For example, the four corners of a glass panel should correspond to the same 4 points across images from multiple views. This is a simple task for human cognition, but far more complex for robots that do not understand the concept of doors, windows, etc.

Methods BRISK, SURF, SIFT, BRIEF, ORB

23.2.3 Outlier Rejection

Goal Take a closer look at the data association results and remove features that are likely incorrect correspondences.

Approach If you have outlier feature matches, they should be rejected so that they are not used to enforce consistency.

Method 1: RANSAC (Random Sampling and Consensus) Since we know how the camera moves, estimate the fundamental matrix of the camera, and use it to reject matches that do not meet the epipolar constraints. These will be points with high reprojection error.

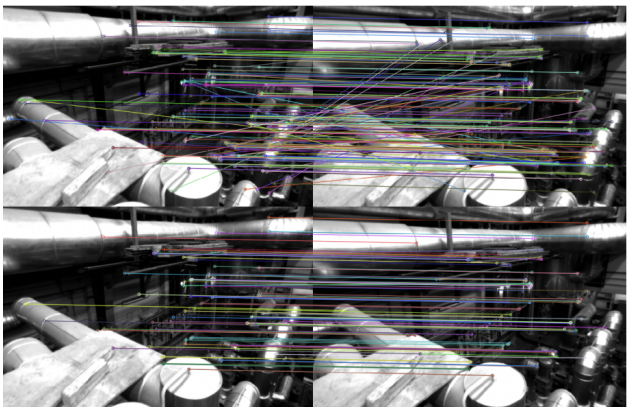
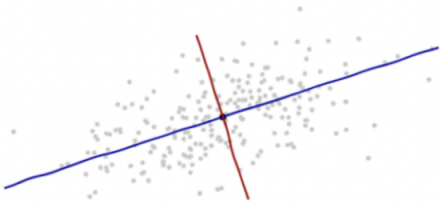


Figure 23.2: RANSAC. Visually, the diagonal lines that are not true matches are being removed from the processing (bottom is processed correspondences).

Method 2: Mahalanobis distance test (Chi-squared Rejection) Given a particular feature at one pose, keep the match only if the new location of the same feature in another pose is within 3SD of the expected location based on the current best estimate of how the camera moves (from the pose changes).



$$d(x) = \|x - \mu\|_{S^{-1}} = \sqrt{(x-\mu)^\top S^{-1}(x-\mu)}$$

Figure 23.3: Mahalanobis distance test. Points that exceed the general expectation of the points based on the camera movement will be removed, based on this component analysis

23.3 Back End

23.3.1 Setup

23.3.1.1 Dynamics Model

$$g : \mathbb{R}^{d_x} \to \mathbb{R}^{d_x} \tag{23.1}$$

$$\text{General form: } x_{t+1} = g(x_t) + w_t, \text{with } w_t \sim N(0, \Sigma_w), \forall t \geq 0 \tag{23.2}$$

$$\text{Associated residual: } w_{t+1} - g(x_t) \tag{23.3}$$

The dynamics model is generally defined as a deterministic function g with a noise term w_t where x_t is the pose at time t. One example of such as model is a input u_t and IMU measurements.

23.3.1.2 Measurement model

$$h : \mathbb{R}^{d_x} \times \mathbb{R}^{d_f} \to \mathbb{R}^{d_x} \tag{23.4}$$

$$\text{Image measurement of feature } j \text{ at time } i\text{: } z_{t,j} \in \mathbb{R}_{d_z}, \forall t, j \geq 0 \tag{23.5}$$

$$\text{General form: } z_{t,j} = h(x_t, f_{t,j}) + v_{t,j}, \text{with } w_t \sim N(0, \Sigma_v), \forall t \geq 0 \tag{23.6}$$

$$\text{Associated residual: } z_{t,j} - h(x_t, f_{t,j}) \tag{23.7}$$

Similar to the dynamics model, there is a deterministic term g and noise term $v_{t,j}$. However, this term, defined as h, is the takes in the pose x_t and position estimate of a feature $f_{t,j}$ and returns where a feature is best represented the image measurement given the robot pose. An example is the pinhole camera model.

23.3.1.3 State Vector

$$\text{Camera pose, at time } t\text{: } x_t \in \mathbb{R}^{d_x}, \forall t \geq 0 \tag{23.8}$$

$$\text{Position of feature } j \text{ at time } t \text{ in global frame: } f_{t,j} \in \mathbb{R}^{d_f}, \forall t \geq 0, j \geq 1 \tag{23.9}$$

$$\text{Full state, at time } t\text{: } \tilde{x}_t \in , \forall t \geq 0, \text{with prior } N(\mu_t, \Sigma_t) \tag{23.10}$$

A state vector is simply a list of variables to track over time. The full state is the concatenation of multiple poses and features. An example of a state vector can be found in the Extended Kalman Filter(EKF). In EKF, the state vector contains all features detected and the current pose: $x_t = (x_t, f_{t,1}, \cdots, f_{t,p}) \in \mathbb{R}^{d_x + pd_f} := \mathbb{R}^d$.

23.3.2 Unifying Framework

In scientific literature, different SLAM algorithms are described differently due to different design choices. However, SLAM algorithms can be simply defined as 3 modules: Cost Construction, Gauss-Newton Steps, and Marginalization. These steps are similar to how humans process information.

23.3.2.1 Cost Construction

Q: What variables and measurement data should we take into consideration.

In our SLAM algorithm, we can include more measurements and variables to track in the models we previously defined to generate more residuals. More residuals allows use to enforce consistency in our variables.
If our model and measurement are consistent with each other, then all residuals should be small. With this idea, we can compute the cost $c(x_t$ which we define as the sum of weighted 2-norm squared of the residuals. Then we can find a x_t that minimizes $c(x_t$. This minimization is a nonlinear least squares problem.

23.3.2.2 Gauss-Newton Update

Q: How do we compute a minimizer of the cost quickly and accurately.

One method is to compute the Jacobians of the residuals term, locally approximate the cost as a convex quadratic, and find the minimum of the quadratic. This is due to the cost being a nonlinear least squares problem. This is similar to gradient descent but instead of taking the gradient of the entire cost,

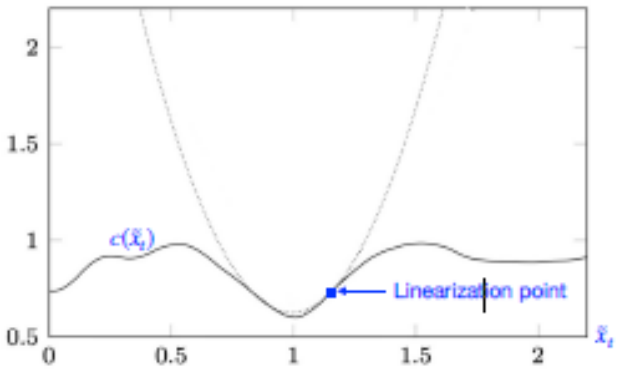


Figure 23.4: Cost being locally approximated into a convex quadratic.

Algorithm	Cost Construction	Gauss-Newton	Marginalization
EKF	Current pose + All current and past features	One	All past poses
iEKF	Current pose + All current and past features	Multiple	All past poses
MSCKF	Current IMU state + n ($\leq N_{\max}$) past poses, evenly spaced in time	One	All other poses
SWF	Current IMU state + n ($\leq N_{\max}$) most recent poses	One	All other poses
OKVis	Keyframe poses in sliding window + associated features	Multiple	Keyframe poses leaving sliding window + associated features
PGO	Current and all past poses, with no features	Multiple, until convergence	None
BA	Current and all past poses and features, with no pairwise pose costs	Multiple, until convergence	None

Figure 23.5: Several SLAM algorithms

we take the gradient of the residuals. This specfic method is called the Levenberg-Marquardt Update - $_2$-Regularized Gauss-Newton Update

23.3.2.3 Marginalization

Q: What variables should we disregard to increase to increase computation speed, and when?
An example of variables to disregard are past variables. We may not want to consider variables is the far past as they can be irrelevant to our cost.
To achieve this, we split our full state vector x_t into a vector to keep $x_{t,K}$ and a vector to marginalize $x_{t,M}$.

$$\min_{\tilde{x}_t} c(\tilde{x}_t) = \min_{\tilde{x}_{t,K}} (\min_{\tilde{x}_{t,M}} c(\tilde{x}_{t,K}, \tilde{x}_{t,M})) \tag{23.11}$$

Notice that the inner minimization in the RHS of the equation on depends on $x_{t,K}$ and is computed via Gauss-Newton steps.

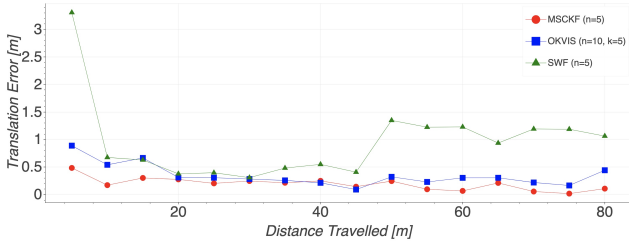


Figure 23.6: Translational Error

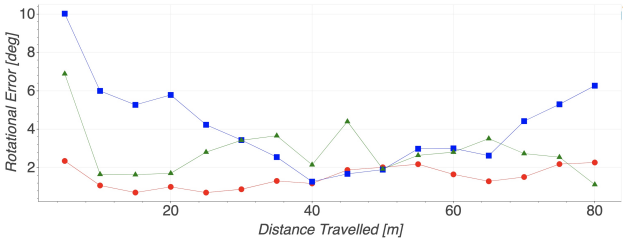


Figure 23.7: Rotational Error

23.3.3 Experiments

To compare SLAM algorithms (e.g. Multi-State Constrained Kalman Filter, Sliding Window Filter, Open-Keyframe Visual Inertial SLAM, etc.), the front end is standardized across all experiments. As shown in figures 23.4 and 23.5, the errors are computed and compared. In the experiment, MSCKF outperforms OK-VIS, and SWF. This is due to how MSCKF recovers more easily from localization errors because it maintains some poses in the far past. The experiment showed that older poses supply better localization information.

23.3.4 Global Optimization

23.4 Next Steps

23.4.1 Active Perception

When driving a car in a two-way stop, you want to do perception in a way that minimize the chance for collisions. In active perception you want to navigate a robot to move in an environment to collect more information to do salient tasks.

23.4.2 Semantic SLAM

Semantic SLAM is about incorporate semantic understanding in SLAM. Instead of identifying point or lines, identify objects to map an environment more similar to what we humans see. How do you integrate this with exiting SLAM algorithms?

23.4.3 Dynamic SLAM

Use SLAM to track moving features. This requires modelling of the how the features are moving in the environment. This is very challenging since you have to distinguish between to causes of a feature moving: estimation error or the feature actually moving. It is hard to determine which one is right.