AA 274: Principles of Robotic Autonomy Winter 2019

Lecture 14: SLAM II: graph-based SLAM and particle-filter SLAM

14.1 Introduction

During the last lecture, we covered an important class of SLAM algorithms based on EKFs. However, one main issue with EKF-SLAM is that it tends to scale quite poorly with the number of features. In general, EKF has squared complexity; in supposing we have on the scale of hundreds or more landmarks, this becomes computationally prohibitive. Furthermore, this method relies on a linearization process performed globally, which is only efficient when the estimate is close to the truth. If non-linearities are large, this can cause the divergence of our filter.

Also, a key issue is that we might assign "fake" landmarks and once it is done we keep them for granted. To fix this, an approach would consist in using a provisional list approach by waiting several iterations before putting a landmark in the state.

In this lecture, we cover two additional algorithms for SLAM that don't suffer from these limitations: graph-based SLAM and particle-filter SLAM. We give a very quick overview of graph SLAM before moving on to particle-filter SLAM, often referred to in literature as Fast-SLAM, an extension of Monte Carlo. This will give us a good overview of the different techniques used in localization.

Learning Objectives:

- Develop insight into solving full-scale SLAM problem, as opposed to the online-SLAM problem that was previously addressed. In full-scale SLAM, we estimate $p(x_{1:t}, m, c_t | z_{1:t}, u_{1:t})$
- Gain intuition in addressing full-scale SLAM problem via graph SLAM and fast SLAM
- Decompose these SLAM approaches into their fundamental mathematical formulation
- Understand the computational demands of these approaches, and distinguish scenarios regarding most appropriate SLAM implentation

14.2 Graph SLAM

While EKF's goal was to solve the SLAM problem online, the Graph SLAM algorithm aims at solving the full-scale SLAM problem. More precisely, we want to get an estimate of the map m and of the entire robot's path – i.e. the whole state history $x_{1:t}$ and not just an estimate of the state x_t at time t – conditioned on the measurement and control history:

$$p(x_{1:t}, m, c_t | z_{1:t}, u_{1:t}) \tag{14.1}$$

In full generality, we also want to get an estimate of the correspondence variable c_t that allows us to correlate a measurement with the features. We are not interested in c_t *per se*, but it is necessary in order to estimate the map and the state history.

In Graph SLAM, we interpret the SLAM problem as a graph optimization problem. We associate nodes of the graph with robot locations and feature locations, and we connect these nodes with constraints. The constraints between two robot locations have to obey the motion model, while the constraints between a robot location and a map feature obey the observation model.

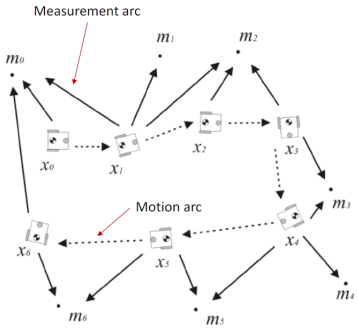

Figure 14.1: Graph representation of a simple SLAM problem.

In Figure 14.1 above, the robot moves from x_0 to x_6 while observing a set of features m_0 to m_6. As stated above, the nodes of the graph are the robot positions and the features. The edges can be either motion arcs representing relative distance between two consecutive robot poses subject to the motion model, or measurement arcs representing the distance to a feature, conditioned on the robot pose and subject to the measurement model.

The key idea is to interpret these edges – or constraints – as *soft probabilistic constraints*, that we can represent as springs (see Figure 14.2 below). The intuition is that a spring constant represents the uncertainty for a certain measurement model, i.e. the difference between the expected measurement and the actual measurement. The approach of the algorithm is then to find the best estimate of the robot poses and feature locations as the state of minimal energy for that spring network. In other words, the most probable joint estimate of robot poses and feature locations minimizes the energy of the network. Note that we usually linearize the constraints and represent them as quadratic constraints.

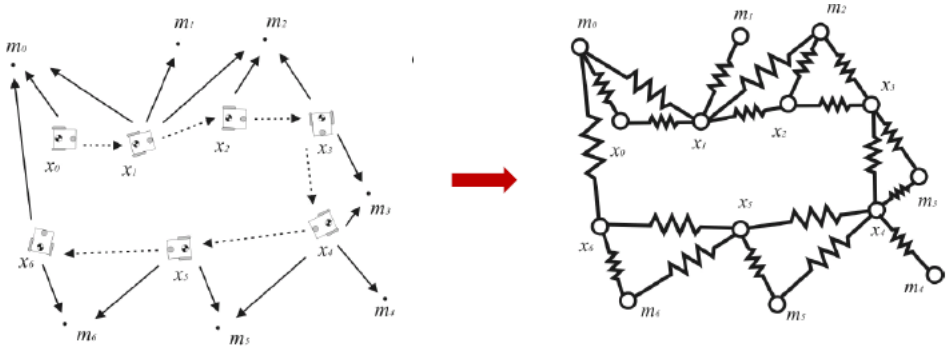

Figure 14.2: Interpretation of the SLAM graph as a springs network. The optimal estimate is obtained by minimizing the energy of the network.

An advantage of Graph SLAM is that we can solve the global problem directly in batch, which gives us a more robust estimate. However, Graph SLAM requires to solve a non-linear least-square optimization

problem, which can be a lot more complex to solve than EKF or Particle-filter SLAM, as we will see.

You can find more information on the Graph SLAM algorithm in chapter 11 of [2].

14.3 Particle Filter SLAM

The last class of SLAM algorithms we will focus on is Fast-SLAM. This is a powerful algorithm that has been used successfully in a variety of applications and it is a natural continuation of what we've seen thus far in particle filtering.

The idea here is to approximate our belief with a set of particles. The difference with particle localization is that we want to simultaneously estimate the robot pose as well as the map:

$$p(x_{1:t}, m, c_t | z_{1:t}, u_{1:t}) \tag{14.2}$$

If we apply particle filtering to SLAM in a naive way, however, we will be confronted to the curse of dimensionality, as the number of particles will grow too large for any practical implementation. More precisely, it can be shown that the number of particles will grow exponentially with the number of dimensions.

The key insight of Particle Filtering SLAM is that knowing the true robot's path makes features conditionally independent. As we will prove, one can then factor the SLAM posterior $p(y_{1:t}|z_{1:t}, u_{1:t}, c_{1:t})$ into the product of the posterior over the path, $p(x_{1:t}|z_{1:t}, u_{1:t}, c_{1:t})$, and the posteriors over each of the N map features, $p(m_n|x_{1:t}, z_{1:t}, c_{1:t})$:

$$p(y_{1:t}|z_{1:t}, u_{1:t}, c_{1:t}) = p(x_{1:t}|z_{1:t}, u_{1:t}, c_{1:t}) \prod_{n=1}^{N} p(m_n|x_{1:t}, z_{1:t}, c_{1:t}) \tag{14.3}$$

The intuition behind the conditional independence of features given the true path is made clearer in the context of a dynamic Bayesian network:

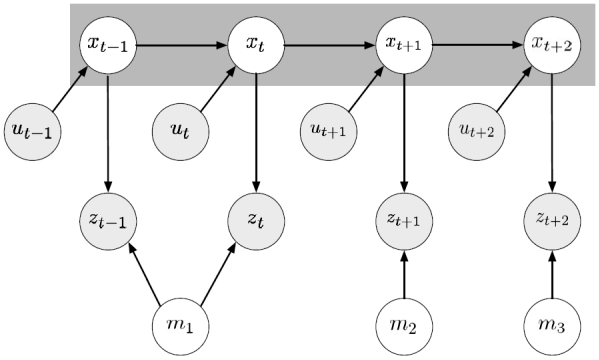

Figure 14.3: Interpretation of SLAM as a Bayesian network problem.

Each measurement z_t is a function of the state x_t (the robot pose), and – obviously – of the feature m_n that this measurement corresponds to. Note that for simplicity, we will restrict our analysis to the case where we only measure *one* feature m_n at a time. One can relax this assumption in the case of batch measurements by treating each measurement $z_{t,m}$ sequentially, assuming that the position x_t of the robot is kept constant.

We see form Figure 14.3 above that, given the position x_t, the measurement z_t is independent from the rest of the path and therefore the estimations of each feature location become independent problems.

14.3.1 Posterior Factorization

As mentioned above, we would like to express the following SLAM posterior $p(y_{1:t}|z_{1:t}, u_{1:t}, c_{1:t})$ as

$$p(y_{1:t}|z_{1:t}, u_{1:t}, c_{1:t}) = p(x_{1:t}|z_{1:t}, u_{1:t}, c_{1:t}) \prod_{n=1}^{N} p(m_n|x_{1:t}, z_{1:t}, c_{1:t}) \tag{14.4}$$

Step 1:

To start, we apply Bayes' rule to the posterior and split it into the posterior for the path times the posterior for map features:

$$p(y_{1:t}|z_{1:t}, u_{1:t}, c_{1:t}) = p(x_{1:t}|z_{1:t}, u_{1:t}, c_{1:t}) p(m|x_{1:t}, z_{1:t}, u_{1:t}, c_{1:t}) \tag{14.5}$$

Since the feautures do not depend on the control history, we can drop the condition on $u_{1:t}$ in the posterior for the map features. Our equation for the SLAM posterior becomes:

$$p(y_{1:t}|z_{1:t}, u_{1:t}, c_{1:t}) = p(x_{1:t}|z_{1:t}, u_{1:t}, c_{1:t}) p(m|x_{1:t}, z_{1:t}, c_{1:t}) \tag{14.6}$$

We are almost at our goal. Now we need to express $p(m|x_{1:t}, z_{1:t}, c_{1:t})$ as $\prod_{n=1}^{N} p(m_n|x_{1:t}, z_{1:t}, c_{1:t})$

In order to do so, we will go through two more steps.

Step 2.a

We have a total of n features. We will first consider the case where the correspondence at time t is not for feature n; i.e. $c_t \neq n$. In this case, the posterior of feature m_n is not dependent on c_t and we can write:

$$p(m_n|x_{1:t}, z_{1:t}, c_{1:t}) = p(m_n|x_{1:t}, z_{1:t}, c_{1:t-1}) \tag{14.7}$$

Step 2.b

In the case where the correspondence at time t is for feature n ($c_t = n$), we can apply Bayes' rule:

$$p(m_{c_t}|x_{1:t}, z_{1:t}, c_{1:t}) = \frac{p(z_t|m_{c_t}, x_{1:t}, z_{1:t-1}, c_{1:t}) p(m_{c_t}|x_{1:t}, z_{1:t-1}, c_{1:t})}{p(z_t|x_{1:t}, z_{1:t-1}, c_{1:t})} \tag{14.8}$$

Step 3

To factor $p(m|x_{1:t}, z_{1:t}, c_{1:t})$ into $\prod_{n=1}^{N} p(m_n|x_{1:t}, z_{1:t}, c_{1:t})$, we will use a proof by induction.

Induction Hypothesis:

assume that, at $t-1$,

$$p(m|x_{1:t-1}, z_{1:t-1}, c_{1:t-1}) = \prod_{n=1}^{N} p(m_n|x_{1:t-1}, z_{1:t-1}, c_{1:t-1}) \tag{14.9}$$

Induction Step:

at time t,

$$p(m|x_{1:t}, z_{1:t}, c_{1:t}) = \frac{p(z_t|m, x_{1:t}, z_{1:t-1}, c_{1:t})p(m|x_{1:t}, z_{t-1}, c_t)}{p(z_t|x_{1:t}, z_{1:t-1}, c_{1:t})} = \frac{p(z_t|m, x_{1:t}, c_{1:t})p(m|x_{1:t-1}, z_{t-1}, c_{t-1})}{p(z_t|x_{1:t}, z_{1:t-1}, c_{1:t})} \tag{14.10}$$

So far we just applied Bayes' rule and assumed the features are not dependent on measurements, nor are they dependent on the current state x_t or the current correspondence c_t. We now can apply our induction hypothesis to obtain:

$$p(m|x_{1:t}, z_{1:t}, c_{1:t}) = \frac{p(z_t|m, x_{1:t}, c_{1:t})}{p(z_t|x_{1:t}, z_{1:t-1}, c_{1:t})} \prod_{n=1}^{N} p(m_n|x_{1:t-1}, z_{1:t-1}, c_{1:t-1}) \tag{14.11}$$

Among the N features, there is one that is matched with the correspondence at time t.

$$p(m|x_{1:t}, z_{1:t}, c_{1:t}) = \frac{p(z_t|m, x_{1:t}, c_{1:t})}{p(z_t|x_{1:t}, z_{1:t-1}, c_{1:t})} p(m_{c_t}|x_{1:t-1}, z_{1:t-1}, c_{1:t-1}) \prod_{n \neq c_t} p(m_n|x_{1:t-1}, z_{1:t-1}, c_{1:t-1}) \tag{14.12}$$

In Step 2.a, we found an expression for $p(m_n|x_{1:t-1}, z_{1:t-1}, c_{1:t-1})$ when $n \neq c_t$. In Step 2.b, we found an expression for $p(m_{c_t}|x_{1:t-1}, z_{1:t-1}, c_{1:t-1})$. Plugging those expression into equation (14.12) above and simplifying, we get:

$$p(m|x_{1:t}, z_{1:t}, c_{1:t}) = p(m_{c_t}|x_{1:t}, z_{1:t}, c_{1:t}) \prod_{n=1}^{N} p(m_n|x_{1:t-1}, z_{1:t-1}, c_{1:t-1}) = \prod_{n=1}^{N} p(m_n|x_{1:t}, z_{1:t}, c_{1:t}) \tag{14.13}$$

Thus, assuming our factorization holds for $t-1$, we proved that it holds for t. Through induction, we proved that the factorization $p(m|x_{1:t}, z_{1:t}, c_{1:t}) = \prod_{n=1}^{N} p(m_n|x_{1:t}, z_{1:t}, c_{1:t})$ holds for all t.

Going back to the expression in Step 1, we can plug our induction results into (14.5) to obtain:

$$p(y_{1:t}|z_{1:t}, u_{1:t}, c_{1:t}) = p(x_{1:t}|z_{1:t}, u_{1:t}, c_{1:t}) \prod_{n=1}^{N} p(m_n|x_{1:t}, z_{1:t}, c_{1:t}) \tag{14.14}$$

14.3.2 Proof by Induction, a General Guide

This section is a more detailed guide on proof by induction. In general, when we want to use induction to prove a statement holds for all the elements of a sequence, we go through the following steps.

Base Case: We must check that our statement holds for the first value of the sequence. For example, if we want to prove that the sum of the first N integers is $\frac{N(N+1)}{2}$, we must check that "the sum of the first 1 integers" (i.e. 1) is indeed equivalent to that expression.

$$\frac{1(1+1)}{2} = 1 \tag{14.15}$$

Induction Hypothesis: Assume the statement holds for some k^{th} element in our sequence. For our example, assume the sum of the first k integers is equal to $\frac{k(k+1)}{2}$ for some integer k.

Induction Step: Use our induction step to prove that the statement holds for $(k+1)^{th}$ element in our sequence.

$$1+2+3+...+k+(k+1) = \frac{k(k+1)}{2} + (k+1) = (k+1)(\frac{k}{2}+1) = (k+1)(\frac{k+2}{2}) \tag{14.16}$$

We just proved the statement for the $(k+1)^{th}$ instance!

What does this mean? We showed that the statement is true for the first case (base case); we also showed that the statement being true for one case means it is true for the next (induction hypothesis and step). Thus, the statement is true for the second element of our sequence, which also means it holds for the third, and the fourth... It is therefore true for all elements of our sequence.

14.4 Fast SLAM

14.4.1 With Known Correspondences

Fast SLAM is a solution to the SLAM problem with unknown data association. The idea of Fast SLAM is to exploit the factorization of the posterior. This is particularly useful in how we keep track of filters. We treat each particle as a multi-dimensional vector. The first component $x_t^{[k]}$ corresponds a particle filter that shows our hypothesis about the robot pose. We then keep track of a set of EKF filters for each feature. Each filter is modeled as a Gaussian and thus, we keep track of a mean μ and a covariance Σ. Since there are m particles and n features, there are $m * n + 1$ filters. We are able to do this two step approach as a result of the conditional independence of each feature. In this sense, Fast SLAM is a hybrid combination between particle filtering for the robot pose and EKF filtering with respect to each feature conditional on robot hypotheses with respect to each robot pose. Each filter keeps track of one specific feature.

$$Y_t^{[k]} = < x_t^{[k]}, \mu_{1,t}^{[k]}, \Sigma_{1,t}^{[k]}, ..., \mu_{[N,t]}^{[k]}, \Sigma_{N,t}^{[k]} > \tag{14.17}$$

In order to keep track of the robot pose and all future locations, we must propagate forward a particle from time $t-1$ to t. This is done in the following three steps by incorporating a measurement z_t and a knowledge of the control input u_t.

Steps:
1. Extend path posterior
2. Update the observed feature estimates
3. Resample

Extend Path Posterior

Forward propagation is done by using the motion model. Since we sample probabilistically, the hypothesis about the robot pose at time t is related to the previous hypothesis about the robot pose and the control input. In this approach, x_{t-1}^k is the k-th hypothesis about the robot pose at time $t-1$.

$$x_t^k \propto p(x_t^k | x_{t-1}^k, u_t) \tag{14.18}$$

Update the Observed Feature Estimates

The updating of a feature estimate depends on whether that feature is observed at time t. The simplest case occurs when a feature is not observed. For this case, we keep our Gaussian estimate of the feature constant through the time step since we have no new information about the specific feature.

$$< \mu_{n,t}^{[k]}, \Sigma_{n,t}^{[k]} > = < \mu_{n,t-1}^{[k]}, \Sigma_{n,t-1}^{[k]} > \tag{14.19}$$

If we do observe a feature at time t, then that measurement will give us additional information that we can use to refine our estimate. The update is performed by exploiting Bayes' rule where the probability of the location of a feature given the history is equivalent to the probability of the measurement multiplied by the probability of the feature given the history up to the previous time step and a normalization factor η.

$$p(m_{c_t} | x_{1:t}, z_{1:t}, c_{1:t}) = \eta p(z_t | m_{c_t}, x_t, c_t) p(m_{c_t} | x_{1:t-1}, z_{1:t-1}, c_{1:t-1}) \tag{14.20}$$

The measurements up to this point are Gaussian and we must linearize the model to ensure that the new estimate is as well.

$$h(m_{c_t}, x_t^{[k]}) = h(\mu_{c_t,t-1}^{[k]}, x_t^{[k]}) + h'(\mu_{c_t,t-1}^{[k]}, x_t^{[k]})(m_{c_t} - \mu_{c_t,t-1}^{[k]}) \tag{14.21}$$

Once we linearize the measurement model, we must propagate forwards the mean and covariance through standard EKF update procedures.

$$K_t^{[k]} = \Sigma_{c_t,t-1}^{[k]} [H_t^{[k]}]^T (H_t^{[k]} \Sigma_{c_t,t-1}^{[k]} [H_t^{[k]}]^T + Q_t)^{-1} \tag{14.22}$$

$$\mu_{c_t,t}^{[k]} = \mu_{c_t,t-1}^{[k]} + K_t^{[k]} (z_t - \hat{z}_t^{[k]}) \tag{14.23}$$

$$\Sigma_{c_t,t}^{[k]} = (I - K_t^{[k]} H_t^{[k]}) \Sigma_{c_t,t-1}^{[k]} \tag{14.24}$$

Resample

Because step 1 generates pose x_t only in accordance with the most recent control u_t, paying no attention to the measurement z_t, we then need to resample the particles to correct for the inevitable mismatch.

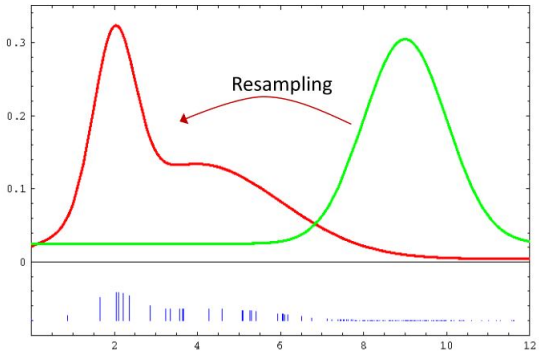

Figure 14.4: Example of Resampling

When resampling the particles, it is important to first determine the weights of each hypothesis, based on which ones are most favorable. At this stage, path particles are distributed according to the following probability function, which can be found using the definition of conditional probability:

$$p(x_{1:t}^{[k]}|z_{1:t-1}, u_{1:t}, c_{1:t-1}) = p(x_t|x_{t-1}^{[k]}, u_t)p(x_{1:t-1}^{[k]}|z_{1:t-1}, u_{1:t-1}, c_{1:t-1}) \tag{14.25}$$

In this equation, the first term to the right of the equal sign represents the sampling distribution, and the second term represents the distribution of path particles in $Y_{t-1}^{[k]}$. The target distribution takes into account both z_t and c_t, and is represented by the following:

$$p(x_{1:t}^{[k]}|z_{1:t}, u_{1:t}, c_{1:t}) \tag{14.26}$$

Now, one of the most crucial terms, the importance factor, is found by taking what is desired, and dividing by what is known, and by using Bayes' rule.

$$w_t^{[k]} = \frac{p(x_{1:t}^{[k]}|z_{1:t}, u_{1:t}, c_{1:t})}{p(x_{1:t}^{[k]}|z_{1:t-1}, u_{1:t}, c_{1:t-1})} = \eta p(z_t|x_t^{[k]}, c_t) \tag{14.27}$$

To derive an (approximate) close-form expression for $w_t^{[k]}$, one can then apply the total probability law along with a linearization of the measurement model to obtain the following:

$$w_t^{[k]} = \eta det(2\pi Q_t^{[k]})^{-1/2} exp\{-\frac{1}{2}(z_t - \hat{(z)}_t^{[k]})[Q_t^{[k]}]^-1(z_t - \hat{(z)}_t^{[k]})\} \tag{14.28}$$

Where:

$$Q_t^{[k]} = [H_t^{[k]}]^T \Sigma_{n}^{[k]}{}_{,t-1} H_t^{[k]} + Q_t \tag{14.29}$$

14.4.2 With Unknown Correspondences

The key advantage of particle filters is that each particle can rely on its own, local data association decisions, and the key idea is that per-particle data association generalizes the per-filter data association to

individual particles. In this method, each particle maintains a local set of data association variables, $\hat{c}_t^{[k]}$, which can be found through maximum likelihood estimation:

$$\hat{c}_t^{[k]} = argmax_{c_t} p(z_t | c_t, \hat{c}_{1:t-1}^{[k]}, x_{1:t}^{[k]}, z_{1:t-1}, u_{1:t}) \tag{14.30}$$

The probability term is computed, as usual, via total probability law and linearization.

14.5 Fast-SLAM Algorithm

When looking at this algorithm, the key idea to remember is that only the most recent pose is used in the process of generating a new particle at time t. On top of this, the complexity of an entire update requires $O(MlogN)$.

Algorithm 1: Fast-SLAM Algorithm

Data: Y_{t-1}, u_t, z_t, c_t
Result: Y_t
for $k = 1$ *to* M **do**
  $x_t^k \sim p(x_t | x_{t-1}^k, u_t)$;
  $j = c_t$;
  if *feature j never seen before* **then**
    initialize feature
  end
  else
    $\hat{z} = h(\mu_{j,t-1}^{[k]}, x_t^{[k]})$;
    calculate Jacobian H;
    $Q = H\Sigma_{j,t-1}^{[k]} H_t^T + Q_t$;
    $K = \Sigma_{j,t-1}^{[k]} H^T Q^{-1}$;
    $\mu_{j,t}^{[k]} = \mu_{j,t-1}^{[k]} + K(z_t - \hat{z})$;
    $\Sigma_{j,t}^{[k]} = (I - KH)\Sigma_{j,t-1}^{[k]}$;
    $w^{[k]} = \det(2\pi Q)^{-\frac{1}{2}} \exp\{-\frac{1}{2}(z_t - \hat{z}) Q_t^{-1} (z_t - \hat{z})\}$;
  end
  for *all other features* $n \neq j$ **do**
    $\langle \mu_{n,t}^{[k]}, \Sigma_{n,t}^{[k]} \rangle = \langle \mu_{n,t-1}^{[k]}, \Sigma_{n,t-1}^{[k]} \rangle$;
  end
  $Y_t = \emptyset$;
end
for $i = 1$ *to* M **do**
  Draw k with probability $\propto w^{[k]}$;
  Add $\langle x_t^{[k]} \mu_{1,t}^{[k]}, \Sigma_{1,t}^{[k]}, ..., \mu_{N,t}^{[k]}, \Sigma_{N,t}^{[k]} \rangle$ to Y_t;
end
Return Y_t

14.6 ROS packages with SLAM implementations

Below, we have compiled a list of the most popular ROS packages with SLAM implementations with short description based on the information available on wiki.ros.org and openslam-org.gihub.io.

1. Gmapping: Implementation of OpenSlam's Gmapping which provides a 2-D occupancy grid based on a laser scan and a pose data. It uses Rao-Blackwellized particle filer which allows to solve the simultaneous localization and mapping problem where each particle represents an individual map of the environment.
2. TinySlam: The simplest and lightweight SLAM implementation based on laser range data and odometry
3. MRPT (Mobile Robot Programming Toolkit, C++) based packages:
   (a) EKF SLAM 2D: EKF-based package. It uses range-bearing sensors, odometry, 2D robot pose and 2D landmarks.
   (b) EKF SLAM 3D: EKF-based package. It uses range-bearing sensors, odometry, 6D robot pose and 3D landmarks.
   (c) GRAPH SLAM 2D: single robot or multi-robot pose-only graphSLAM. It uses laser scans and odometry.
   (d) ICP SLAM 2D: ICP-based package (Iterative Closest Points). It uses 2D laser scans. Builds only small to midsized maps. Larger maps requires more advanced implementations
   (e) RBPF SLAM: Rao Blackwellized based package. Uses range-only.
4. Hector SLAM: LIDAR based SLAM that can be used without odometry. It can be used on robots with roll/pitch motion of the sensor, robot and both.
5. RGBD SLAM: uses RGB-D cameras (e.g. Microsoft Kinect).

14.7 EKF SLAM with unknown correspondences

We want to add discussion of EKF SLAM with unknown correspondences. We are still doing localization, but this time, the robot starts with a map of landmarks that does not change over the course of its run. So the difference is when the robot gets a range and bearing measurement, it will not know to which landmark that measurement corresponds.

Below we will explain firstly the process for extimating the correspondence between measurements and landmarks and present the correspondence algorithm on its own. Then we will show how the correspondence estimation algorithm can be integrated with EKF localization algorithm we have learned before.

14.7.1 Measurement Correspondence Intro

When our robot gets a measurement to a nearby landmark, it can use that information to refine its estimate of its pose. In this current problem, we are no longer directly telling the robot to which landmark each measurement corresponds. Without knowledge of this correspondence measurements are basically useless to our robot. So our robot needs a way to estimate the correspondence between its measurements and the landmarks in its map. One way it can do this is through maximum likelihood data association.

14.7.2 Maximum Likelihood Data Association

In maximum likelihood data association, the robot computes the probability that its current measurement corresponds to every landmark in its map. It then chooses the landmark with the highest probability of correspondence and assumes it is the correct association. This can be written as

$$\hat{c}_t^= argmax_c p(z_t|c_{1:t}, m, z_{1:t-1}, u_{1:t}) \tag{14.31}$$

In other words, given the robot's past measurement corresponcences, map, past measurements and past control inputs, we can get which landmark has the highest probability of producing measurement.

In practice, we find this by calculating the expected measurement and measurement covariance for every landmark in the map. We can then use these to calculate the measurement probability for every landmark and then select the landmark with the highest probability. The algorithm goes as follows:

for all measurements $z_t^i = [r_t^i \ \phi_t^i]^T$ do
  for all landmarks k in the map m do
    $q = (m_{k,x} - \bar{\mu}_{t,x})^2 + (m_{k,y} - \bar{\mu}_{t,y})^2$
    $\hat{z}_t^k = \begin{bmatrix} \sqrt{q} \\ \text{atan2}(m_{k,y} - \bar{\mu}_{t,y},\ m_{k,x} - \bar{\mu}_{t,x}) - \bar{\mu}_{t,\theta} \end{bmatrix}$
    $H_t^k = \begin{bmatrix} -\frac{m_{k,x} - \bar{\mu}_{t,x}}{\sqrt{q}} & -\frac{m_{k,y} - \bar{\mu}_{t,y}}{\sqrt{q}} & 0 \\ \frac{m_{k,y} - \bar{\mu}_{t,y}}{q} & -\frac{m_{k,x} - \bar{\mu}_{t,x}}{q} & -1 \end{bmatrix}$
    $S_t^k = H_t^k \bar{\Sigma}_t [H_t^k]^T + Q_t$
  endfor
  $j(i) = \text{argmax}_k \ \det(2\pi S_t^k)^{-\frac{1}{2}} \exp\{-\frac{1}{2}(z_t^i - \hat{z}_t^k)[S_t^k]^{-1}(z_t^i - \hat{z}_t^k)$
endfor

Figure 14.5: Algorithm

There are alternative methods that are more robust to incorrect associations. Multi-Hypothesis Tracking is one of those, but the maximum likelihood method is the simplest.

14.8 Summary

We have introduced 3 different implementations of SLAM, which are Gaussian SLAM (based on EKF and KF), Graph SLAM, and Fast SLAM (based on particle filter).

EKF SLAM assumes that all uncertainties follow a Gaussian distribution, and is therefore well understood and easy to compute. It is the best linear estimator that works especially well when uncertainty is low. However, because it is a unimodal estimate, it works poorly when uncertainties are high. When a point far from the mean is pushed through linearization, it will yield inaccurate estimates.

Graph SLAM provides the best possible estimate based on an optimization problem that models all uncertainties as spring stiffnesses, and finds the minimum energy of the network. Exploiting matrix sparsity leads to efficient solutions, but can still be computationally complex when solving a full SLAM problem, leading to difficult implementation.

Fast SLAM is the most natural representation of multimodal beliefs, able to handle any non-parametric distribution. It is the easiest to implement, relying on conditional independence to factorize the distribution to speed up computation. However, the problem does not scale well in large dimensions, and a large sample of particles is required for good convergence. Naive implementation is intractable due to the excessively large number of particles. A way to tackle this issue would be to use multiple filters; one estimator for trajectory and one for each landmark.

14.9 References

[1] R. Siegwart, I. R. Nourbakhsh, D. Scaramuzza.Introduction to Autonomous Mobile Robots. MIT Press, 2nd Edition, 2011

[2] S. Thrun, W. Burgard, and D. Fox. Probabilistic robotics. MIT press, 2005.

[3] ROS (Robot Operating System) Documentation wiki.ros.org

[4] OpenSLAM https://openslam-org.github.io/

[5] EKF Localization With Unknown correspondence http://andrewjkramer.net/intro-to-the-ekf-step-2/

Contributors

Winter 2019: Colin Shi, Bernard Lange, Paul Planeix, William Brannon, Sherry (Xuejiao) Li
Winter 2018: Bo Kim, Matt Subrahmanyam, Ianis Bougdal-Lambert, Michelle Zhang, Barrett Weiss, Eric Ballouz