1 Fast SLAM

Fast SLAM is an algorithm that runs a particle filter in robot location space using augmented particles. Each particle carries a history of its past locations $r_1...r_t$ and an individual Kalman filter (consisting of μ_{l_i} and Σ_{l_i}) for each landmark l_i it sees. This is visualized in Figure 1.
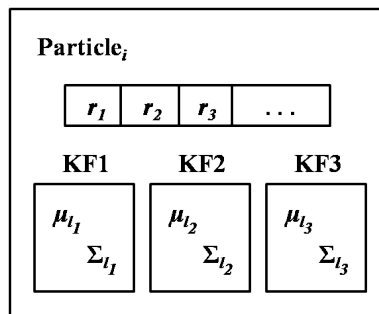

Figure 1: Fast SLAM particles include a history of their positions and individual Kalman filters for each landmark.

Note that landmarks are not special, and we could also use a grid based method.
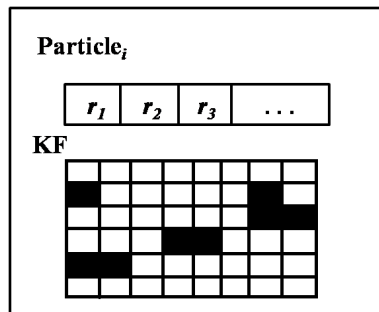
2 Occupancy Grid SLAM

Figure 2: DP SLAM particles include a history of their positions and an occupancy grid representing the map. Note that DP SLAM has a data structure that simulates each particle having a map but stores it much more efficiently in memory.

---

[1]Based on the scribe work of Dave Rollinson, Justin Haines, Neal Seegmiller, and Varun Ramakrishna

2.1 DP SLAM

This is an off the shelf 2D SLAM algorithm that is state of the art. Similar to FastSLAM, but instead of each particle keeping a landmark, each particle has a history of poses and a map, shown in Figure 2.

Murphy initially proposed Fast SLAM using occupancy grid maps. In the simplest implementation, a complete map is stored for each particle; however, this is too memory intensive (and computationally expensive to copy the maps). Researchers at Duke University proposed DP SLAM which uses dynamic programming to reduce the memory/computational requirements.

The basic idea: If two particles share a common ancestor, both their maps will agree with the observations of that common ancestor. A single occupancy grid map is stored with an observation tree at each cell (storing the observations of all particles that affect that cell).

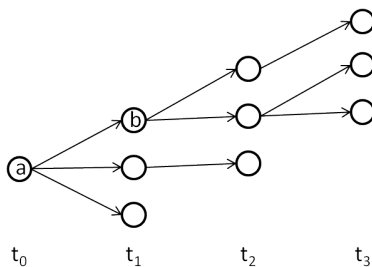Efficient maintenance of this data structure is non-trivial.

Figure 3: Example of particle ancestry for DP SLAM. Note that all particles at t_3 are descended from the root a as well as b at t_1. All particles at t_3 will share the observations of b and its predecessors, so these sections of the map need only be stored once.

2.1.1 Loss of diversity

Ideally, in order to close loops diversity must be maintained on the order of the largest loop that will ever be traversed. However, DP SLAM typically only maintains diversity on the order of 2-3 meters (i.e. all current particles are descended from a single particle 2-3 meters back). Despite this, DP SLAM still works due to high localization accuracy.

2.2 3D Fast SLAM

Nathaniel Fairfield's research on DEPTHX is an example of efficient occupancy grid SLAM in 3D. An octree data structure is used to maintain hundreds of maps required by the particle filter. The octree structure exploits spatial locality and temporal shared ancestry between particles to reduce the processing and storage requirements.

Kevin Murphy, "Bayesian Map Learning in Dynamic Environments," NIPS 1999. (Neural Info. Proc. Systems) http://www.cs.duke.edu/~parr/dpslam/

Nathaniel Fairfield, George Kantor, and David Wettergreen, "Real-Time SLAM with Octree Evidence Grids for Exploration in Underwater Tunnels," Journal of Field Robotics 2007.
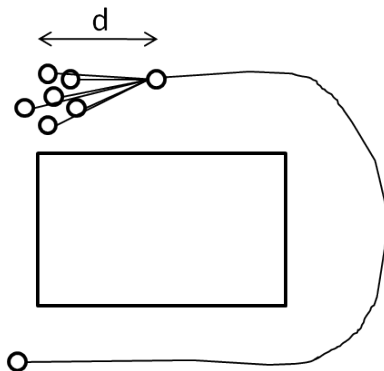
Figure 4: Illustration of loss of diversity in DP SLAM. In this example, correcting the map after closing the loop around the rectangular obstacle would require diversity of paths all the way back to the root; however, the map can only be corrected over length d because the maps of all current particles share the observations of the common ancestor.

Using an occupancy grid for 3D SLAM is nothing too complicated theoretically, but it is very tough to carry around large 3D maps, and this is really the special thing about Fairfield's research.

Note that FastSLAM makes mapping "easy" by breaking the chicken and egg. By having many samples of pose, we can condition the map on the pose, which turns SLAM into just a mapping problem, which we know how to do.

3 Stochastic Gradient Descent approach to SLAM

Until now, we've been viewing SLAM as a filtering problem: $bel(x) = p(x_t|z_{1...t})$. Using the Markov assumption allows us to throw away observations after we see them, and this allows us to maintain constant memory. This is probably a good approach if it's easy to represent $bel(x)$. If it's not easy, it may be better to have kept all of the observations around and frame SLAM as an optimization problem instead!

To do this, we optimize our states to fit all observations well, and then find the trajectory and map as well. We optimize over positions of the robot, and use a "bundle adjustment" (Like in Computer Vision). An optimizer may end up working even better than a filter!

Olson proposed a fast gradient descent approach to recovering a time sequence of robot poses. The steps are:

1. Marginalize out the landmarks

2. Use stochastic gradient descent to optimize pose graph

3. (if desired) Perform mapping using the optimized pose data

---

Edwin Olson, John Leonard, and Seth Teller, "Fast Iterative Alignment of Pose Graphs with Poor Initial Estimates," ICRA 2006

3.1 Marginalize out the landmarks

Observations of landmark location relative to robot pose are converted to constraints between poses from which the same landmark was observed. This results in fewer variables (no landmark variables) but more constraints. The resulting pose graph will have two types of constraints:

1. temporal (odometry) constraints between poses at adjacent timesteps

2. landmark constraints between poses from which the same landmark was observed
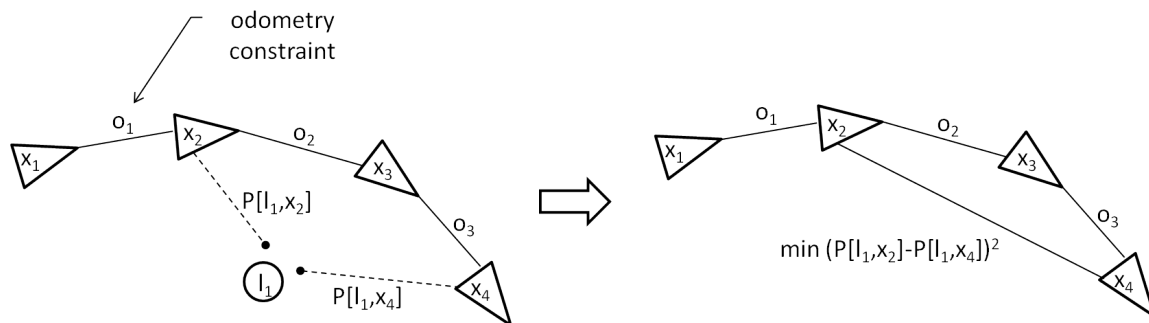

Figure 5: Illustration of marginalizing out a landmark. The observations of landmark l_1 at poses x_2 and x_4 (left) are converted to a single pose constraint (right).

3.2 Algorithm

Do forever:

- Pick Constraint

- Descend in direction of constraint's gradient

- Scale gradient magnitude by alpha/iteration

  − Slowly damps step size

- iteration++

Note that alpha/iteration $\rightarrow 0$ as $t \rightarrow \infty$. This also gives us a robustness to local concavities, since we can hop around in the state space, and stick in the best one. Note that this algorithm will give us a good solution quickly, and a really good solution as $t \rightarrow \infty$.

3.3 Critical trick: represent the state variables as pose deltas

At each iteration a single constraint is chosen at random for the update.

Instead of representing the state variables as robot poses, represent them as the difference in pose between time steps.

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \end{bmatrix} \rightarrow \begin{bmatrix} x_0 \\ \Delta x_1 \\ \Delta x_2 \\ \vdots \end{bmatrix}$$

Robot pose is calculated by summing the pose deltas, so a change in any one pose delta affects the robot pose at all subsequent time steps. By representing our poses this way, we only need to change one pose value, as opposed to all of them.

Alternatively, this change in representation can be thought of as a change in the distance metric used by gradient descent.
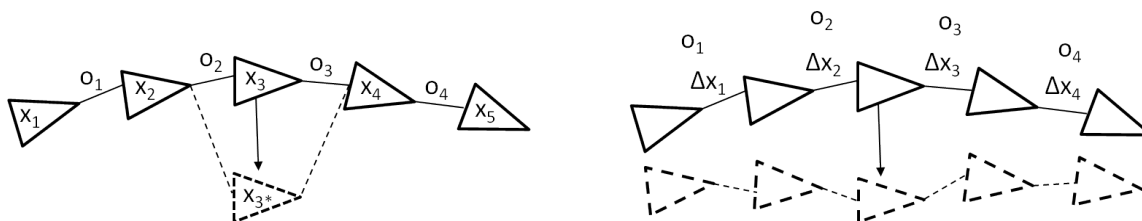

Figure 6: (left, original representation) Consider x_3 undergoes a large update due to a landmark constraint. Likely many iterations will pass before o_2 or o_3 are selected, finally updating the neighboring poses. (right, pose delta representation) Because a change in pose delta affects all subsequent poses, sequences of poses will update together.

3.4 Other Notes

- Never underestimate the power of a well tuned gradient descent algorithm
- Filtering isn't always awesome, especially if your belief is hard to represent
- it's magic! tends to jump over local minima
- be clever about the learning rate (increase for loop closure, etc.)
- this is a batch algorithm, but it can be implemented online using a sliding window/lag filter

4 Kernel Machines

4.1 Reproducing Kernel Hilbert Space Algorithm

A gaussian process can be thought of as a sum of bumps or some of kernels. A kernel machine can be thought of as the sum of a finite but arbitrarily large set of bumps.

We have seen how to use online convex programming to learn linear functions by optimizing costs of the following form:

$$\underbrace{|y_t - \mathbf{w}^T \mathbf{x}_t|}_{loss} + \underbrace{\lambda \mathbf{w}^T \mathbf{w}^2}_{regularization/prior}$$

5

We want generalize this to learn over a space of more general functions $f : \mathbb{R}^n \to \mathbb{R}$. The high-level idea is to learn non-linear models using the same gradient-based approach used to learn linear models.

$$|y_t - f(\mathbf{x}_t)| + \lambda ||f||^2$$

Up till now we have only considered functions of the form $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$, but we will now extend this to a more general space of functions.

5   Review of Kernels

We observed in Gaussian Processes that the mean function had the form $\mu_{f(\mathbf{x})|data} = K^T K_{data}^{-1} y$ which can be interpreted as a sum of Kernel functions on the observed data points:

$$f = \sum_i \alpha_i K(x_i, \cdot) \tag{1}$$

Kernels functions are a natural choice of non-linear functions for our task i.e to learn a function $f : \mathbb{R}^n \to \mathbb{R}$ that assigns a meaningful score given a data point.

E.g. in binary classification, we would like $f(\cdot)$ to return positive and negative values, given positive and negative samples, respectively.

A kernel $K : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ intuitively measures the *correlation* between $f(\mathbf{x_i})$ and $f(\mathbf{x_j})$. Considering a matrix $\mathbf{K}$ with entries $K_{ij} = K(\mathbf{x_i}, \mathbf{x_j})$, then matrix $\mathbf{K}$ must satisfy the properties:

- $\mathbf{K}$ is symmetric ($K_{ij} = K_{ji}$)

- $\mathbf{K}$ is positive-definite ($\forall \mathbf{x} \in \mathbb{R}^n : \mathbf{x} \neq \mathbf{0}, \mathbf{x^T}\mathbf{K}\mathbf{x} > 0$)

However to do gradient descent on the space of such functions, we need the notion of a distance, norm and an inner product. We formalize this by introducing the Reproducing Kernel Hilbert Space.

6   Reproducing Kernel Hilbert Space

The Reproducing Kernel Hilbert Space (RKHS), denoted by $\mathcal{H}_k$, is the space of functions $f(\cdot)$ that can be written as:

$$\mathcal{H}_k = \left\{ f | f = \sum \alpha_i k(x_i, \cdot) \right\}$$

where $k(\mathbf{x}_i, \mathbf{x}_j)$ satisfies certain properties described below.

We also define a *functional* as:

$$F[f] : f \in \mathcal{H}_k \mapsto \mathbb{R}$$

Example functionals:

- $min(f(x)), max(f(x))$

- $\int_{x_0}^{x_1} f(x)$

- Evaluation Functional: $F_x[f] = f(x)$

And define an *operator* as:
$$O[f] : f \in \mathcal{H}_k \mapsto g \in \mathcal{H}_k$$

Example operators:

- $O^-[f] : f \mapsto -f$

- $O^2[f] : f \mapsto f^2$

- $D[f] : f \mapsto \frac{df}{dt}$

To be able to manipulate objects in $\mathcal{H}_k$, we will look at some key properties:

The *inner product* of $f, g \in \mathcal{H}_k$ is defined as

$$\langle f, g \rangle \triangleq \sum_i \sum_j \alpha_i \beta_j k(\mathbf{x}_i, \mathbf{x}_j) = \boldsymbol{\alpha}^\top \mathbf{K} \boldsymbol{\beta}$$

where $f(\cdot) = \sum_i \alpha_i k(\mathbf{x}_i, \cdot)$, $g(\cdot) = \sum_j \beta_j k(\mathbf{x}_j, \cdot)$, $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are vectors comprising, respectively, α_i and β_i components, and $\mathbf{K}$ is n by m (where n is the number of $\mathbf{x}_i$ in f, and m those in g) with $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$.
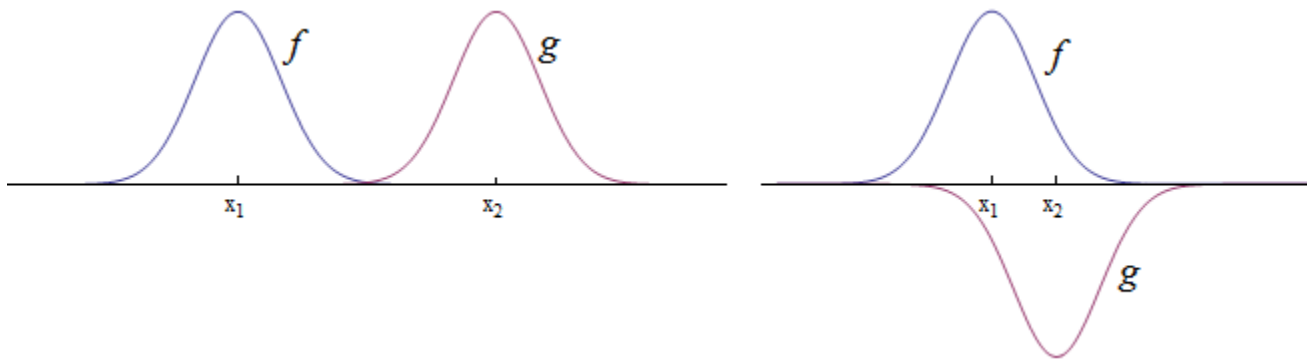

Figure 7: $< f, g > \approx 0$ in left graph, $< f, g > \approx$ large negative number in the right

Note that this will satisfy linearity (in both arguments):

- $\langle \lambda f, g \rangle = \lambda \langle f, g \rangle$

- $\langle f_1 + f_2, g \rangle = \langle f_1, g \rangle + \langle f_2, g \rangle$

With this inner product, the *norm* will be: $||f||^2 = \langle f, f \rangle = \boldsymbol{\alpha}^\top \mathbf{K} \boldsymbol{\alpha}$. The intuition for the norm is that smooth functions, small weights, and a small number of "bumps" will all lead to small norms.
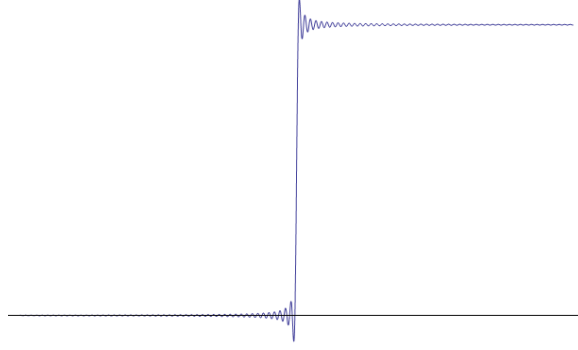
Figure 8: An example of a function with large norm

The *reproducing property* is observed by taking the inner-product of a function with a kernel

$$\langle f, K(\mathbf{x_j}, \cdot)\rangle = \langle \sum_{i=1}^{Q} \alpha_i K(\mathbf{x_i}, \cdot), K(\cdot, \mathbf{x_j})\rangle = \sum_{i=1}^{Q} \alpha_i \langle K(\mathbf{x_i}, \cdot), K(\cdot, \mathbf{x_j})\rangle = \sum_{i=1}^{Q} \alpha_i K(\mathbf{x_i}, \mathbf{x_j}) = f(\mathbf{x_j}) \quad (2)$$

An example of a valid kernel for $\mathbf{x} \in \mathbb{R}^n$ is the inner product: $k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$. Intuitively, the kernel measures the *correlation* between $\mathbf{x}_i$ and $\mathbf{x}_j$.

A very commonly used kernel is the RBF or Radial Basis Function kernel, which takes the form $k(\mathbf{x}_i, \mathbf{x}_j) = e^{-\frac{1}{\gamma}||\mathbf{x}_i - \mathbf{x}_j||^2}$. With this kernel in mind, a function can be considered as a weighted (by α_i) composition of bumps (the kernels) centered at the Q locations $\mathbf{x_i}$:

$$f(\cdot) = \sum_{i=1}^{Q} \alpha_i K(\mathbf{x}_i, \cdot),$$
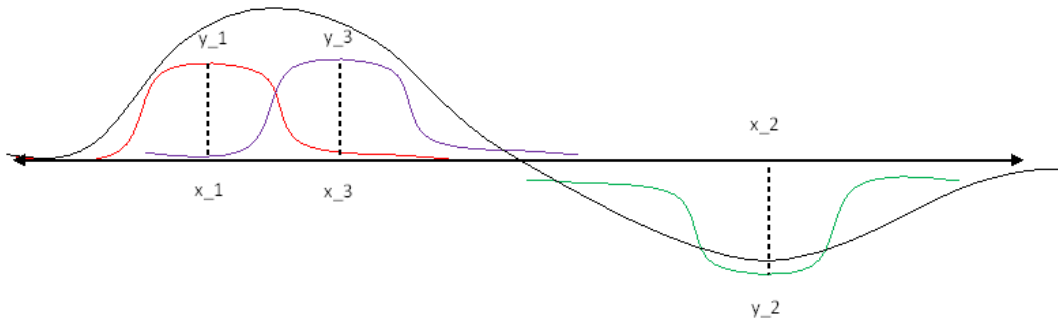
See Figure 9 for an illustration.

Figure 9: Illustration of function after 3 updates

7 Loss Minimization

Let us consider again our cost function defined over all functions f in our RKHS, as before our loss is :

$$|y_t - f(\mathbf{x}_t)| + \lambda ||f||^2$$

8

The purpose of $\langle f, f \rangle$ is to penalize the complexity of the solution f. Here it acts like the log of a gaussian prior over functions. Intuitively, the probability can be thought of as being distributed according to $P(f) = \frac{1}{Z}e^{-\frac{1}{2}\langle f,f \rangle}$ (in practice this expression doesn't work because Z becomes infinite).

We want to find the best function f in our RKHS so as to minimize this cost, and we will do this by moving in the direction of the negative gradient: $f - \alpha \nabla L$. To do this, we will first have to be able to express the gradient of a function of functions (ie. a *functional* such as $L[f]$).

7.1   Functional gradient

A gradient can be thought of as:

- Vector of partial derivatives

- Direction of steepest ascent

- Linear approximation of the function (or functional), ie. $f(x_0+\epsilon) = f(x_0)+\epsilon \cdot \underbrace{\nabla f(x_0)}_{gradient}+O(\epsilon^2)$.

We will use the third definition. A *functional* $F : f \to \mathbb{R}$ is a function of functions $f \in \mathcal{H}_K$. As an example let us write the terms of our loss function from abov as functionals:

- $F_1[f] = ||f||^2$

- $F_2[f] = (f(x) - y)^2$

- $F[f] = \frac{\lambda}{2}||f||^2 + \sum_i (f(x_i) - y_i)^2$

A functional gradient $\nabla F[f]$ is defined implictly as the linear term of the change in a function due to a small perturbation ϵ in its input: $F[f + \epsilon g] = F[f] + \epsilon \langle \nabla F[f], g \rangle + O(\epsilon^2)$.

Before computing the gradients for these functionals, let us look at a few tools that will help us derive the gradient of the loss functional

7.2   Chain rule for functional gradients

Consider *differentiable* functions $C : \mathbb{R} \to \mathbb{R}$ that are functions of functionals G, $C(G[f])$. Our cost function $L[f]$ from before was such a function, these are precisely the functions that we are interested in minimizing.

The derivative of these functions follows the chain rule:

$$\nabla C(G[f]) = \frac{\partial C(G[f])}{\partial \lambda}|_{G(f)} \nabla G[f] \tag{3}$$

Example: If $C = (||f||^2)^3$, then $\nabla C = 3(||f||^2)^2(2f)$

7.3   Some useful functional gradients

Another useful function that we come across often is the evaluation functional. The *evaluation functional* evaluates f at the specified x: $F_x[f] = f(x)$

- Gradient is $\nabla F_x = K(x, \cdot)$

$$
\begin{aligned}
F_x[f + \epsilon g] &= f(x) + \epsilon g(x) + 0 \\
&= f(x) + \epsilon \langle K(x, \cdot), g \rangle + 0 \\
&= F_x[f] + \epsilon \langle \nabla F_x, g \rangle + O(\epsilon^2)
\end{aligned}
$$

- It is called a *linear functional* due to the lack of a multiplier on perturbation ϵ.

Let's also look at the functional gradient of the norm of a function $(\nabla F[f] = \nabla ||f||^2)$:

- Expanding it out using a Taylor's series type expansion

$$
\begin{aligned}
F[f + \epsilon g] &= \langle f + \epsilon g, f + \epsilon g \rangle \\
&= ||f|| + 2\langle f, \epsilon g \rangle + \epsilon^2 ||g|| \\
&= ||f|| + \epsilon \langle 2f, g \rangle + O(\epsilon^2)
\end{aligned}
$$

- We observe that

$$
\nabla F[f] = \nabla ||f||^2 = 2f \tag{4}
$$