FastSLAM Design Document

Benned Hedegaard

February 8th, 2024

1 Introduction

This document proposes a design for a short-term implementation of the FastSLAM algorithm [1]. By implementing this particular SLAM solution, we'll learn quite a bit about how SLAM works, as well as develop experience (as a team) writing "research-quality" code.

Note: I *tried* to type this without LaTeX. It's just not a good time.

2 Intuitions of FastSLAM

FastSLAM was introduced by Montemerlo et al. (2002) as a new method to solve the *simultaneous localization and mapping (SLAM)* problem. [1]. Previously successful approaches to SLAM were based on the *Extended Kalman Filter (EKF)*, which models all uncertainty in the problem as a big multivariate Gaussian (i.e., a mean vector μ and covariance matrix Σ). EKF-SLAM approaches were therefore limited in how many landmarks they could represent at once because inverting the "big matrix" was quite computationally expensive.

To avoid this issue, FastSLAM doesn't use *exactly one* Gaussian to represent the robot's pose and *all* landmarks. Rather, FastSLAM doesn't represent the robot's pose estimate using a matrix at all! Instead, we'll make multiple "guesses" for the robot's current pose, forming a *proposal distribution*, then keep the "good" guesses. In FastSLAM, we call these guesses the *particles*, with particle m containing a robot pose estimate $s_t^{[m]} = (x, y, \theta)$ and a corresponding list of landmark locations $\theta_k^{[m]} = (x, y)$, each estimated using its own (small) EKF. Although we'll have many EKFs to update per timestep, each one will be much faster than the single EKF in EKF-SLAM.

So how do we select "good" particles? We'll calculate an *importance weight* $w_t^{[m]}$ for each particle, capturing that particle's likelihood based on the observed landmarks at timestep t. Imagine we've seen some landmark θ_k before: we can predict where it *should* be the next time we see it! By comparing what we expected with what we observed, we can find the "probably least wrong" particles and prioritize keeping those around. By sampling (with replacement) the particles with probability based on their importance weights, we create a new set of particles representing our "best guesses" based on the available sensor data.

So we've walked through a single step of FastSLAM. After we've accounted for the most recent motion command and landmark measurement(s) for a particular timestep, we can forget them: the core assumption of FastSLAM, as in any *filter-based* SLAM approach, is that our particle set captures the *maximum a posterior (MAP) estimate* (i.e., "most likely result") given the data. Under the *Markov assumption*, we only need to process new motion commands u_t or landmark observations z_t once, because our posterior estimate captures "everything we need to remember." You could say we "filter out" previous timesteps.

3 Math of FastSLAM

We're computing the posterior $p(s_{1:t}, \theta | z_{1:t}, u_{1:t})$, which we factor into (Eq. 4 of [1]):

$$p(s_{1:t}|z_{1:t}, u_{1:t}, n_{1:t}) \prod_k p(\theta_k | s_{1:t}, z_{1:t}, u_{1:t}, n_{1:t}) \tag{1}$$

This equation is decomposed into $K + 1$ estimation problems:

- Estimating posterior $p(s_{1:t}|z_{1:t}, u_{1:t}, n_{1:t})$ over robot paths $s_{1:t}$.
- Estimating K posteriors $p(\theta_k|s_{1:t}, z_{1:t}, u_{1:t}, n_{1:t})$, conditioned on the path $s_{1:t}$.

3.1 Path Estimation

We represent $p(s_{1:t}|z_{1:t}, u_{1:t}, n_{1:t})$ using a set of particles:

$$S_t = \{s_{1:t}^{[m]}\}_M, \tag{2}$$

where we incrementally calculate M particles using $s_t^{[m]} \sim p(s_t | s_{t-1}^{[m]}, u_t)$.

3.2 Landmark Location Estimation

We represent $p(\theta_k|s_{1:t}, z_{1:t}, u_{1:t}, n_{1:t})$ using an EKF for each landmark, within each particle. Therefore, the full posterior in FastSLAM is the sample set (Eq. 8 of [1]):

$$S_t = \{s_{1:t}^{[m]}, \mu_1^{[m]}, \Sigma_1^{[m]}, \ldots, \mu_K^{[m]}, \Sigma_K^{[m]}\}_M, \tag{3}$$

where each $\mu_k^{[m]} \in \mathbb{R}^2$ and each $\Sigma_k^{[m]} \in \mathbb{R}^{2\times 2}$. Each landmark estimate is updated:

$$p(\theta_k|s_{1:t}, z_{1:t}, u_{1:t}, n_{1:t}) \propto p(z_t|\theta_k, s_t, n_t) p(\theta_k|s_{1:t-1}, z_{1:t-1}, u_{1:t-1}, n_{1:t-1}) \tag{4}$$

using an EKF that linearizes $p(z_t|\theta_{n_t}^{[m]}, s_t^{[m]})$, leaving other θ_k with $n_t \neq k$ unchanged.

3.3 Importance Weighting

For each particle $s_t^{[m]}$ in the proposal distribution, we compute an importance weight $w_t^{[m]}$:

$$w_t^{[m]} \underset{\sim}{\propto} \int p(z_t|\theta_{n_t}^{[m]}, s_t^{[m]}) p(\theta_{n_t}^{[m]}) d\theta_{n_t}, \tag{5}$$

which can be directly calculated from $(\mu_{n_t}^{[m]}, \Sigma_{n_t}^{[m]})$, our representation of the posterior $p(\theta_{n_t}^{[m]})$.

3.4 Data Association

To associate each z_t with some θ_k for particle $s_t^{[m]}$, we can choose $n_t^{[m]}$ using MLE:

$$n_t^{[m]} = \underset{n_t}{\text{argmax}}\, p(z_t|s_t^{[m]}, n_t), \tag{6}$$

which we can compare to a threshold α to identify new, unseen landmarks.

4 General FastSLAM Algorithm

Given: Previous FastSLAM posterior S_{t-1}, motion command u_t, and measurement z_t.

1. Create the *proposal distribution* by sampling each $s_t^{[m]} \sim p(s_t|s_{t-1}^{[m]}, u_t)$.
   - Loop over the M particles in S_{t-1} and re-sample each $s_{t-1}^{[m]}$.
   - Probabilistic motion model: Given s_{t-1}, u_t, and Δt, samples s_t.
   - See *ProbRob* Ch. 5.3, Table 5.3 (pg. 124).
2. Compute per-particle *data association(s)* $n_t^{i,[m]}$ for each $z_t^i \in z_t$ using MLE.
   - Compute each $n_t^{i,[m]} = \underset{n_t^i}{\text{argmax}}\, p(z_t^i|s_t^{[m]}, \theta_{n_t^i})$ and compare with α.
   - Measurement likelihood model: Given z_t^i, s_t, and θ_k, compute $p \in [0, 1]$.
   - See *ProbRob* Ch. 6.6, Table 6.4 (pg. 179).
3. Per the data association(s) $n_t^{i,[m]}$, update *landmark posterior(s)* $p(\theta_{n_t^i}^{[m]})$ via EKF.
   - For each m, for each n_t^i, update $(\mu_{n_t^i}^{[m]}, \Sigma_{n_t^i}^{[m]})$ by linearizing $p(z_t^i|s_t^{[m]}, \theta_{n_t^i}^{[m]})$.
   - See *ProbRob* Ch. 13.3, Table 13.1, lines 12-17 (pg. 450).
4. Compute *importance weight* $w_t^{[m]}$ for each sampled particle $s_t^{[m]}$.
   - See *ProbRob* Ch. 13.3, Table 13.1, line 18 (pg. 450).
5. *Resample* (with replacement) new particle set S_t based on weights $w_t^{[m]}$.
   - See *ProbRob* Ch. 4.3, Table 4.4 (pg. 110).

Result: New FastSLAM posterior S_t.

5 Assumptions

We could implement FastSLAM in many ways. I assume the following simplifications:

1. Following the paper [1], we're modeling a 2D mobile robot. Therefore, $s_t = (x, y, \theta)$.
2. We're using the *velocity motion model* as described in *Probabilistic Robotics*, Chapter 5.3 [2]. Therefore, $u_t = (v_x, \omega_z)$. See the `VelocityCommand` struct.
3. We're using the *range-bearing measurement model* as described in *Probabilistic Robotics*, Chapter 6.6 [2]. As in the book, and unlike the paper [1], we permit observations of *multiple* landmarks in one timestep. Therefore, $z_t = \{z_t^1, z_t^2, \dots\}$, where $z_t^i = (r_t^i, \phi_t^i, n_t^i)$.
4. We're implementing FastSLAM for both the *known* and *unknown data association* (DA) cases. We can address both cases by wrapping n_t^i, the *correspondence variable* for individual landmark observation z_t^i, in a `std::optional` object. See the `LandmarkObservation` struct for a summary.
   (a) *Known* n_t^i - When $n_t^i = k$, we know that z_t^i observed the landmark θ_k.
   (b) *Unknown* n_t^i - Just leave the n_t^i field as `std::nullopt` within z_t^i.
5. We're implementing the "naive" version of FastSLAM, running in $O(MK)$, to avoid certain FastSLAM-specific implementation details involving a balanced binary tree.
6. Our EKF class will handle $\mathbb{R}^2$, not $\mathbb{R}^n$, so that we can use concrete dimensions (i.e., 2) throughout. Although the overall math should be identical, this may simplify things. The `Gaussian2D` and `EKF_2D` classes explicitly encode this assumption.
7. Unless otherwise stated, we use consistent units, namely meters for lengths and radians for angles. These units are clarified through variable suffixes (`_m` and `_rad`, respectively).

6 Software Design

These are the conceptual pieces involved with implementing FastSLAM. Each will probably become a class, struct, or namespace, depending on whether it needs an internal state.

6.1 Structs

- `Gaussian2D` - Represents a 2D Gaussian distribution with mean vector $\mu \in \mathbb{R}^2$ and covariance matrix $\Sigma \in \mathbb{R}^{2\times 2}$.
- `LandmarkObservation` - Represents a range-bearing-signature measurement of a single landmark: $z_t^i = (r_t^i, \phi_t^i, n_t^i)$ (units are meters, radians, and N/A). To represent both known and unknown data association, n_t^i is wrapped in a `std::optional` object.

- `Point2D` - Represents a point (i.e., just position) on the 2D plane: (x, y).
- `Pose2D` - Represents a pose (i.e., position and orientation) on the 2D plane: (x, y, θ).
- `VelocityCommand` - Represents a 2D planar robot's nominal velocity at a particular timestep as (v_x, ω_z), with *linear velocity* v_x (m/s) and *angular velocity* ω_z (rad/s).

6.2 Classes

- `EKF_2D` - Implements the Extended Kalman Filter over $\mu \in \mathbb{R}^2$ and $\Sigma \in \mathbb{R}^{2\times 2}$, as represented by the `Gaussian2D` type.
- `FastSLAM` - An abstract class defining the necessary components, over abstract types, to implement FastSLAM. These abstract components and types include:
  1. `RobotPose` - Abstract type defining the robot's pose s_t, and each $s_t^{[m]}$.
  2. `LandmarkLocation` - Abstract type defining the location of a landmark θ_k.
  3. `LandmarkGaussian` - Abstract type defining the posterior over a landmark θ_k.
  4. `Particle` - An abstract `std::pair` combining a `RobotPose` and a vector of `LandmarkGaussian` posteriors.
  5. `Posterior` - Full posterior of M particles: $S_t = \{s_t^{[m]}, \mu_1^{[m]}, \Sigma_1^{[m]}, \ldots, \mu_K^{[m]}, \Sigma_K^{[m]}\}_M$. This is simply a vector (with size M) of `Particle` objects.
  6. `MotionCommand` - Abstract type defining u_t, the robot's motion command between consecutive timesteps.
  7. `Observation` - Abstract type defining z_t^i, a single observation of a landmark θ_k.
  8. `MotionModel` - Abstract class representing $p(s_t|s_{t-1}, u_t)$, a probabilistic motion model over the abstract `RobotPose` and `MotionCommand` types. For the concrete implementation, see *Probabilistic Robotics*, Chapter 5.3, Table 5.3 (pg. 124).
     - Abstract function: Given s_{t-1}, u_t, and Δt, samples s_t.
  9. `MeasurementModel` - Abstract class representing $p(z_t^i|s_t, \theta_{n_t^i})$, a measurement likelihood model over the `Observation`, `LandmarkLocation`, and `RobotPose` types. For details, see *Probabilistic Robotics*, Chapter 6.6, Table 6.4 (pg. 179).
     - Abstract function: Given z_t^i, s_t, and $\theta_{n_t^i}$, compute $p \in [0, 1]$.
  10. `EKF` - An Extended Kalman Filter over the abstract `LandmarkGaussian` type. For details, see *Probabilistic Robotics*, Chapter 13.3, Table 13.1, lines 12-17 (pg. 450).
     - Almost certainly calls functions from the `MeasurementModel` class.
- `FastSLAM_2D` - A concrete implementation of `FastSLAM` class for the 2D case.

References

[1] Michael Montemerlo et al. "FastSLAM: a factored solution to the simultaneous localization and mapping problem". In: *Eighteenth national conference on Artificial intelligence*. USA: American Association for Artificial Intelligence, July 2002, pp. 593–598. ISBN: 978-0-262-51129-2. URL: https://dl.acm.org/doi/10.5555/777092.777184.

[2] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. 1st ed. The MIT Press, Aug. 2005. ISBN: 978-0-262-20162-9. URL: https://dl.acm.org/doi/book/10.5555/1121596.