Particle filter – report

Dhasharath Shrivathsa & Katie Butler

March 1, 2017

Abstract

Particle filters are a useful way to describe a set of discrete hypotheses $\{\beta_0, \beta_1 ... \beta_n\}$ to approximate (α') some truth about the world α. This process occurs by iteratively updating each β with data D as it comes in. The Bayesian approach is applicable here, as we can frame each D as a Bayesian update on the set of β's, $P(\beta|D)$

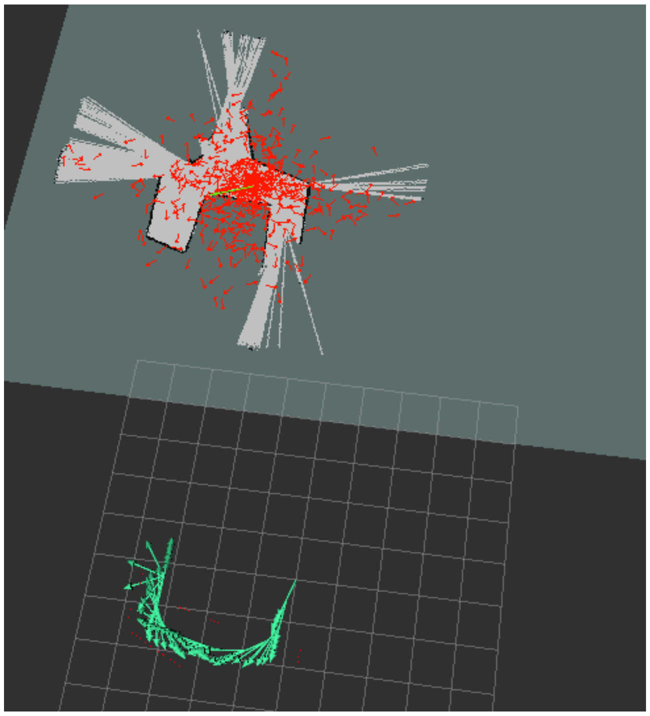

Figure 1: Particle filter localization demo

1 Goal

The goal for this project was to have a working basic particle filter with as few assumptions made about the problem as possible. I decided to go for a 2-day from-the-ground rewrite of a particle filter to identify inefficiencies and work out how one really works, including the hairy coordinate frame stuff that Paul somewhat boxed away.

This was a success

2 Problem solving description

We can decompose the particle filter into several simple steps:

2.1 Initialization

Initialize map M The map is recieved from the ROS service `map_server`, and is wrapped in an `OccupancyField` so we can have python bindings to be able to query the map nicely.

This is worth noting because my implementation differs from Paul's greatly (see `scripts/my_pf.py`). One important distinction is how the single method that `OccupancyField` implements, `get_distance_to_closest_obstacle`, has a different time complexity. Paul's implementation uses a $O(1)$ time, as he wraps it in a `NearestNeighbors(n_neighbors=1)` classifier. My implementation is $O(n)$ because I didn't need the speed. It can be converted to $O(1)$ trivially.

Initialize particles $\{\beta\}$ To spread out the initial set of hypotheses, we sample uniformly across the convex hull of the points in the occupancy field. The convex hull is computed with the initialization of the `OccupancyField` class, and we build a bounding box and sample across that, taking only what lies within the hull up to `n_particles`. This gives us $\{\beta\}$ with equal confidence weights

2.2 Callbacks

Odometry As the robot moves, we'd like to update all our hypotheses. The callback `update_particles_with_odom` handles relative distance and angle transforms every time the robot moves greater than some amount. This math requires a small amount of angle thinking, as moving in the x-direction when $\theta = 0$ is not the same as when $\theta = 90$.

Laser scans L As the robot recieves new laser scans, we project each scan onto each β, measuring its error, and performing a confidence update on each. In the existing model, we square the error and divide each β (higher weights are better particles).

This lets us encode the whole set of sensor readings into each β, as $P(\beta, L_{t-1}...)$ is the weight of β

2.3 Utility and ROS interface

Visualization Each β is a `Pose` in a `PoseArray`. `alpha_pose/alpha_prime`, our best guess, is a pose from $\{\beta\}$, the one with the most weight. I do not calculate the origin-map-frame-to-odom-frame transform, as that's a debugging detail in the problem description

3 Design decisions

To maintain a rugged filter and to enable post-localization unceartainty, the code only samples across a fat-tailed cauchy distribution.

It considers all Laser scans

It considers points only valid if they're within the convex hull of the occupied points on the map

4 Challenges

The timeframe was the main challenge, as I did the project twice over, first with a partner, and then as a time-constrained implementation exercise. Of the two, the time-constrained one was much harder and I learned more technical and implementation details from that. The partner-based project was also interesting, as I was forced to refine my definitions, at the end taking up no more than a single side of a4 written in sharpie.

5 Improvements

I'd want to find ways to enforce multi-modality and parameterize more things to find out how to tune the filter.

Improvements from more sensors would be nice.

I really want to create a cohesive ROS class structure that makes it easier to write and debug code (see `warmup_project/scripts/classes.py`). I would like to rewrite this in C as python/rospy flaws became apparent.