16-833 HW1: Particle Filter

Julius Arolovitch

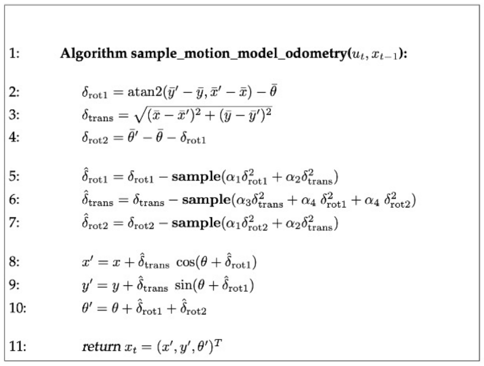
February the 18th, 2025

The following describes the implementation of the particle filter for localizing a robot within a map of Carnegie Mellon's Wean Hall using a LiDAR sensor and odometry. Videos demonstrating localization are attached for log 1 and log 5 and correspond to approximately 30 localization timesteps per second.

Experiments were conducted on a machine with an AMD Ryzen Threadripper PRO 5995WX (128 threads), 512GB RAM, and an NVIDIA RTX 4090.

Motion Model

The motion model is implemented to propagate particles according to information available from the odometry data stream from the robot. The implementation uses the canonical sampling odometry algorithm from the Probabilistic Robotics textbook:



```
1:  Algorithm sample_motion_model_odometry($u_t, x_{t-1}$):

2:      $\delta_{\text{rot1}} = \text{atan2}(\bar{y}' - \bar{y}, \bar{x}' - \bar{x}) - \bar{\theta}$
3:      $\delta_{\text{trans}} = \sqrt{(\bar{x} - \bar{x}')^2 + (\bar{y} - \bar{y}')^2}$
4:      $\delta_{\text{rot2}} = \bar{\theta}' - \bar{\theta} - \delta_{\text{rot1}}$

5:      $\hat{\delta}_{\text{rot1}} = \delta_{\text{rot1}} - \textbf{sample}(\alpha_1 \delta_{\text{rot1}}^2 + \alpha_2 \delta_{\text{trans}}^2)$
6:      $\hat{\delta}_{\text{trans}} = \delta_{\text{trans}} - \textbf{sample}(\alpha_3 \delta_{\text{trans}}^2 + \alpha_4 \delta_{\text{rot1}}^2 + \alpha_4 \delta_{\text{rot2}}^2)$
7:      $\hat{\delta}_{\text{rot2}} = \delta_{\text{rot2}} - \textbf{sample}(\alpha_1 \delta_{\text{rot2}}^2 + \alpha_2 \delta_{\text{trans}}^2)$

8:      $x' = x + \hat{\delta}_{\text{trans}} \cos(\theta + \hat{\delta}_{\text{rot1}})$
9:      $y' = y + \hat{\delta}_{\text{trans}} \sin(\theta + \hat{\delta}_{\text{rot1}})$
10:     $\theta' = \theta + \hat{\delta}_{\text{rot1}} + \hat{\delta}_{\text{rot2}}$

11:     return $x_t = (x', y', \theta')^T$
```

In the program, the `update` function accepts parameters `u_t0`, `u_t1`, and `x_t0`, where the first two arguments correspond to robot's odometry readings from the prior and current time-step and the final argument contains the position of the addressed particle at the prior time-step, respectively. In order to approximate the control input u_t, the implementation computes `u_t1-u_t0`.

The motion model assumes that the motion of the robot can be represented as an initial rotation δ_{rot1}, a linear translation δ_{trans}, and another rotation δ_{rot2}. These transformations are subject to Gaussian noise, which is controlled by four parameters: α_1, α_2, α_3, and α_4; each parameter changes the behavior of the motion model with respect to the translation and rotation behavior of the robot. The inclusion and tuning of these parameters is essential to the performance of the particle filter; too little noise may fail to overcome inherent error in odometry and result in poor convergence, while too much noise may result in unpredictable motion that dominates the contribution of the odometry and may, too, cause poor convergence. We discuss the tuning process and results later in the report.

Sensor Model

The sensor model is implemented according to the sensor model defined in Probabilistic Robotics, and the algorithm is shown below. The model relies on the weighted combination of four probability distributions p_{hit}, p_{short}, p_{max}, and p_{rand}. The four probability distributions correspond to the Normal-distributed likelihood of having an accurate sensor reading (with appropriate variance defined by σ_{hit}), the exponentially-distributed likelihood of having a transient obstruction that will block the true region in the environment the sensor is expected to detect (parametrized by λ_{short}), the Dirac delta-distributed likelihood of a maximum-range reading being accurate, and the uniformly-distributed likelihood of having a random sensor reading, respectively.

The sensor model is implemented using a pre-computed raycast map that is computed offline before beginning localization. This approach follows a prior implementation that attempted to use a cache that would save raycasts performed online during localization in an attempt to use them again in future calls, but this was found to yield insignificant runtime improvements while increasing memory consumption. The raycasting is done by taking 1cm steps radially outwards in order to ensure accurate obstacle detection, which is thresholded by a predetermined probability of interpreting a position as blocked. While all 180 perspectives were used in order to achieve the localization results linked in this report, it was found that as few as 20 uniformly spaced perspectives can be sufficient to yield accurate localization. For localizing in particularly noisy environments, such as in logs 1 and 5, more perspectives can help overcome the clutter of the environment and the resultant noise in the sensor readings. Utilizing too few perspectives in such regions can subject the localization to high variance harmful to performance.

```
1: Algorithm beam_range_finder_model($z_t, x_t, m$):

2:     $q = 1$
3:     for $k = 1$ to $K$ do
4:         compute $z_t^{k*}$ for the measurement $z_t^k$ using ray casting
5:         $p = z_{\text{hit}} \cdot p_{\text{hit}}(z_t^k \mid x_t, m) + z_{\text{short}} \cdot p_{\text{short}}(z_t^k \mid x_t, m)$
6:             $+z_{\max} \cdot p_{\max}(z_t^k \mid x_t, m) + z_{\text{rand}} \cdot p_{\text{rand}}(z_t^k \mid x_t, m)$
7:         $q = q \cdot p$
8:     return $q$
```

Resampling

Resampling is implemented to be adaptive relative to the distribution of particles, both in respect to the quantity of particles sampled and the sampling method used. Low variance sampling is well-suited for a broadly spread distribution of particles and, thus, weights. Low variance sampling excels at resampling particles with lower bias than multinomial sampling by leveraging a random seed to initialize regularly spaced selection points.

Multinomial sampling, however, tends to concentrate toward already high-weight particles, making it more prone to sample impoverishment, where a few particles dominate the new set while lower-weight particles are discarded more aggressively. This can lead to higher variance in the resampled set, reducing particle diversity.

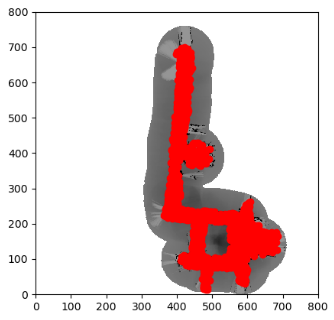
To balance efficiency and accuracy, the implemented adaptive resampling strategy adjusts both the quantity of resampled particles and the choice of resampling method based on the Effective Sample Size (ESS). When ESS is high, indicating well-distributed particles, fewer particles are resampled, and low variance resampling is preferred to maintain diversity. When ESS is low, meaning that particles have collapsed toward a few high-weight regions, multinomial resampling is applied with a reduced number of resampled particles.

This resampling approach is adaptive to the progress of localization and is also aimed to reduce runtime towards the conclusion of the localization query by progressively collapsing the quantity of particles.

Parameters & Tuning

Parameter tuning was the defining contribution to the performance of the particle filter, and involved tuning a variety of parameters.

Particles are initialized in free space, and an example of an initialization is shown below. Particle quantity is important to the quality of localization, as a sufficient quantity of particles must be present near the true location of the robot in order for particle resampling to occur within the region in future time steps of the filter. Any quantities below 500 particles were found to be insufficient to reliably result in accurate localization. 2000 particles were found to be reliably sufficient, above which performance was found to saturate.



For the sensor model, parameters z_{hit}, z_{short}, z_{max}, z_{rand}, σ_{hit}, and λ_{short} were tuned. While the other parameters were also critical to performance, it was found that z_{rand} particularly influenced performance. From an intuitive perspective, z_{rand} quantifies how much random sensor data should be weighted, and thus indirectly quantifies how much we should trust the sensor output. In contrast, other parameters such as z_{short} and z_{max} simply define the influence of elements in the environment, such human presence in proximity to the sensor or simply environments where the true reading extends beyond the manufacturer-guaranteed accurate range of the sensor hardware. Parameters were tweaked one-by-one, noting their impact on performance across several runs before iterating to another value. The tuning was done by initializing all particles in free space and watching the filter's performance on converging to the starting position of the first robot from the first log before it starts moving, eliminating influence of motion model performance from the tuning process. The final parameters used were $z_{hit} = 1.1$, $z_{short} = 0.25$, $z_{max} = 0.1$, $z_{rand} = 400$, $\sigma_{hit} = 90$, and $\lambda_{short} = 0.1$.

For the motion model, parameters α_1, α_2, α_3, and α_4 were tuned. The influence of each parameter on the motion of the particles is somewhat apparent from the parameters that they influence in the motion odometry algorithm. α_3, for one, appears to be a reference to translational noise that occurs during translational motion, and should thus be roughly proportional to the linear translations that are seen from the robot odometry. α_1, on the other hand, appears to reference rotational noise due to rotational motion, while α_4 appears to reference rotational noise due to translational motion. Finally, α_2 correlates with translational noise due to rotational motion. Through manual inspection of odometry, where it is seen that the robot does not appear to move more than several centimeters or rotate fractions of a radian at a time, we can assume approximate values that could be reasonable. The performance of these parameters was also tested in isolation on a simulated robot performing various maneuvers, and the distribution of the particles over time was noted for various parameter permutations. After tuning, the final parameters used were $\alpha_1 = 0.0003$, $\alpha_2 = 0.0001$, $\alpha_3 = 0.0005$, and $\alpha_4 = 0.0001$.

Performance & Future Work

Initially, a query on the 1st log took approximately 20 minutes to run. This was improved to approximately **2.1 seconds** through the two methods described below.

First, a raycast map is precomputed from each free position in the map at each of 360 degree angles; we note that while the heading of the robot and its sensor are not guaranteed to be at an integer-valued heading, we approximate the heading of the robot by rounding its heading to the nearest integer and find that this does not greatly impact the quality of localization performance. The aforementioned raycasting map took approximately 50 minutes to compute sequentially, though this was improved to 2 minutes by implementing parallel computation; the map is saved for future queries, and thus only needs to be computed once assuming a static map.

Second, the implementation of the sensor model, motion model, and resampling of particles is vectorized. NumPy was initially used for vectorized operations, and resulted in a 10X reduction in runtime. However, NumPy does not leverage GPU for vectorized operations, and the implementation was then swapped to leverage PyTorch. PyTorch results in slight initial overhead due to the initialization of the CUDA runtime, though it experimentally resulted in a mean 2.3X performance boost over NumPy in terms of localization time steps processed per second, and the initialization overhead would be overcome for longer localization queries.

Finally, the performance of localization greatly depends on the quality of the data used to perform it. Fine-tuning hyperparameters does define whether localization can yield usable results, and can always be improved to better converge (through formal optimization against a loss function and ground truth queries or trajectories, for example). However, while tuning can result in approximations of environmental disturbances and sensor error, having better models or understanding of sensor failure modes or expected environmental disturbance would enable development of sensor models that could better qualify and discriminate data.