note 44 views

Guideline of Lab 5

Hi all,

Please **start working on Lab 5 early** since it is not straightforward to debug particle filter from simulation behaviors. We TAs will move our office hours to Thursday class. So preparing yourself with lab 5 related questions before the class can keep yourselves from late struggling.

The following resources can provide some additional information on Particle Filters for Lab 5:

- Section 6.5 of the Corke book on Robotics, Vision and Control
- Chapter 5 of the Autonomous Mobile Robots book (less detailed)
- Chapter 4, 5, and 6 of Probabilistic Robotics

At a high level, you are responsible for filling in the two core steps of monte-carlo localization: **motion update** and **measurement update**.

Motion update

Move each particle in accordance with the odometry estimate received, i.e. for each particle, update it's position and add some noise.
Notes:

- if the robot doesn't move, the particles shouldn't move either
- remember we use motion_model_odometry. You need to extract motion δ_{rot1}, δ_{trans}, δ_{rot2} from odometry u_t.
- Please refer to course slide to sample from motion_model_odometry.
- Note for each particle, it should get its own sample update. So over time, if the robot does not see any landmarks, the particles should spread out.

Measurement update

Update the weight of each particle according to how well it matches the current sensor readings. There are two steps for this process.

Step 1: Set weights

for each particle:

1. Obtain the list of localization markers that a robot would see if it were really at the pose (position/orientation) specified by this particle. `particle.py` has a helper function to simulate this: `markers_visible_to_particle = particle.read_markers(grid)`
2. Now we want to compare the list of markers obtained by simulating the particle's field of view to the list of particles reported by Cozmo. Note that in our domain all the localization markers look the same, we have no way to distinguish one from another. As a result, we will try to match the markers by distance. Your code should do something like this:

```
for cm in markers_seen_by_cozmo:
  find m, the closest marker out of markers_visible_to_particle

  store the pairing [cm, m] for later calculations

  remove m from list of possible future pairings (i.e. if Cozmo sees two markers, they need to be matched to different markers_visible_to_part
icle)
```

Once you have identified the list of marker pairings, we can calculate the weight of the particle itself based on how well the markers seen by cozmo match up with the markers "seen" by the particle.

```
prob= 1.0;
for each landmark
  d = Euclidean distance to landmark
  prob *= Gaussian probability of obtaining a reading at distance d for this landmark from this particle

return prob
```

The Gaussian probability density function is defined as:

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

In our case μ would be a reading from Cozmo and x would be a simulated reading from a particle, thus we would obtain the probability of them matching. However, in the case of Cozmo we want to account for angle in addition to distance. Also, for simplification, we can ignore the first part of the equation because it remains the same for every particle. Therefore our probability update becomes:

$$p = e^{-\left(\frac{(distBetweenMarkers)^2}{2\sigma_t^2} + \frac{(angleBetweenMarkers)^2}{2\sigma_r^2}\right)}$$

where $distBetweenMarkers$ can be obtained using the grid_distance helper function in the provided code, and $angleBetweenMarkers$ can be obtained using the diff_heading_deg helper function. σ_t denotes standard deviation of the gaussian model for translation; in our code this is MARKER_TRANS_SIGMA. σ_r is the standard deviation for rotation measurements, MARKER_ROT_SIGMA.

Additional notes:

- particles within an obstacle or outside the map should have a weight of 0
- if there are no new sensor readings, assign equal weight to all particles (there is no new sensor information, so there should be no bias in the resampling process)

Step 2: Resampling (a.k.a. importance sampling)

The final step of the pipeline is resampling, in which we will generate a new set of n particles.

First, now that we have the particle weights, remember we have to normalize them. Sum up all the weights and divide the weight of each particle by the sum.

Next, we want to do probabilistic sampling with replacement to generate a new particle distribution. In other words, each particle should be resampled with a probability equal to the particle's normalized probability calculated in the previous step. There are a number of ways to do this in python, `numpy.random.choice` is one possible way.

There are a number of additional implementation options available to you here. Examples include:

- eliminating any particles with very low probabilities and replacing them with random samples
- always maintaining some small percentage of random samples
- throwing away all particles and starting again with a uniform distribution if all the particles are very unlikely

Feel free to experiment with what works best.

lab5

Updated 1 day ago by Yang Tian

followup discussions *for lingering questions and comments*

[● Resolved] Unresolved

Anonymous 19 hours ago
When we are comparing the measurement to the markers visible from a single particle, if the number of markers does not match up can we just assign weight 0 to that particle? For example if the robot sees two markers but at a particular particle only one is visible, should we even consider that particle in the sampling?

Yang Tian 19 hours ago In that case, you can safely assign 0 to that particle.

Resolved [● Unresolved]

Anonymous 10 hours ago
For motion update, we just need to calculate the new position according to odom right? No probability is involved.

Yang Tian 9 hours ago You do need to sample from a certain distribution.