LAB 3: PARTICLE FILTER PART I

Due: Tuesday, October 5th 11:59pm EST

The 5th Annual CS 3630 Skating Competition

The objective of Labs 3 and 4 are to implement a Particle Filter (a.k.a. Monte Carlo Localization). In this lab, you will work entirely in simulation to validate your algorithm. In Lab 4, we will adapt the algorithm to work on Cozmo and Vector.

Task: *Welcome to the 5th annual CS 3630 Skating competition!* You are a world-renowned coach who has successfully trained thousands of skater robots, and you have recently entered your top competitor, Skater-Bot, in the CS 3630 Skating Competition! You'd like to monitor the progress of Skater-Bot as he competes. To do this, you must implement a particle filter which takes robot *odometry measurement* as motion control input and *marker measurement* as sensor input. You must complete the implementation of two functions: `motion_update()` and `measurement_update()`. Since in a particle filter, the belief of a robot's pose is maintained by a set of particles, both `motion_update()` and `measurement_update()` have the same input and output - a set of particles, as the belief representations.

Before giving more details about the functions you need to implement, we first present some definition of the world: the skating rink. In this lab, we will use the grid shown in Fig 1., with the addition of localization markers on the wall of the rink. The rink has the origin (0,0) at the bottom

left, X axis points right and Y axis points up. The robot has a local frame in which the X axis points to the front of the robot, and Y axis points to the left of the robot.

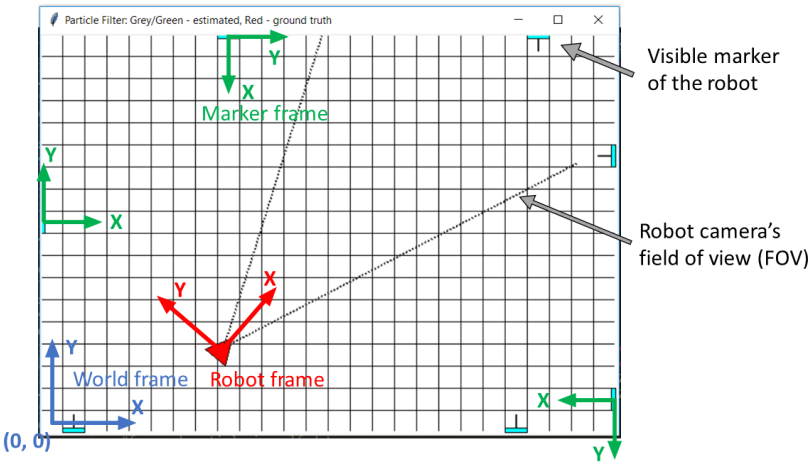

Fig 1. Coordinate frame definitions

The localization markers will appear only on the wall of the rink and will be replaced by real markers which were used in the earlier labs. The direction the marker is facing is defined as the positive X direction of the marker's local coordinate frame, and Y points left of the marker.

The simulated robot skater is equipped with a front-facing camera with a 45 degrees field of view (FOV), like Cozmo. The camera can see markers in its FOV. The simulation will output the position and orientation of each visible marker, measured relative to the Skater-Bot's position. As he competes, Skater-Bot will also report his odometry estimates to you, his world-renown coach.

Motion update: `particles = motion_update(particles, odom, grid):`

The input of the motion update function includes particles representing the belief $p(x_{t-1}|u_{t-1})$ before motion update, and Skater-Bot's new odometry measurement. The odometry measurement is the relative transformation of Skater-Bot's current pose relative to his last pose in the local frame. It has the format odom = (dX, dY, dH), as shown in Fig 2. To simulate noise in a real-world environment, the odometry measurement includes Gaussian noise, and noise level is defined in setting.py. The output of the motion update function should be a set of particles representing the belief $\tilde{p}(x_t|u_t)$ after the motion update.

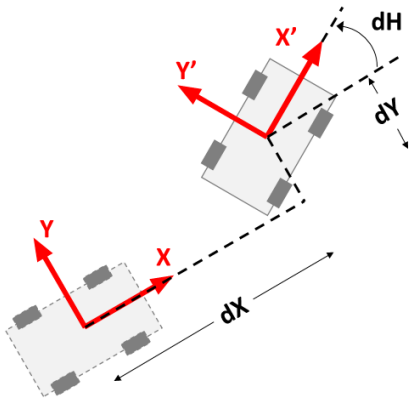

Fig 2. Odometry measurement definition

Measurement update: `particles = measurement_update(particles, mlist, grid)`:

The input of the measurement update function includes particles representing the belief $\tilde{p}(x_t|u_t)$ after motion update, and the list of localization marker observation measurements. The marker measurement is calculated as a relative transformation from Skater-Bot to the marker, in Skater-Bot's local frame, as shown in Fig 3. Same as odometry measurement, marker measurement is also mixed with Gaussian noise, noise level is defined in setting.py. The output of measurement update function should be a set of particles representing the belief $p(x_t|u_t)$ after the measurement update.

In addition to the noise in the odometry measurement, the marker detection may also be noisy. This is to simulate poor lighting conditions, camera issues etc. The marker detection is noisy in two ways. Either the robot could detect "spurious" markers or "drop" the correct markers. This corresponds to false positives and false negatives in your marker detector. We control the number of "spurious" or "dropped" markers using `SPURIOUS_DETECTION_RATE` and `DETECTION_FAILURE_RATE` variables defined in settings.py. These variables are the probabilities of the robot detecting "spurious" markers or "dropping" correct markers respectively.

Note: The measurement update must include resampling to work correctly. We will be setting `DETECTION_FAILURE_RATE` and `SPURIOUS_DETECTION_RATE` to 0 for most test cases and we will be cranking this up for a few test cases (See Grading section below).

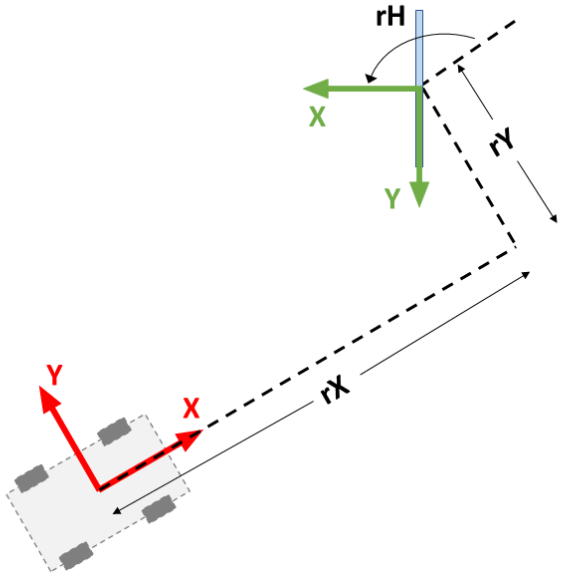

Fig 3. Marker measurement definition

In this lab you are provided the following files:

`particle_filter.py` – Particle filter you should implemented

`autograder.py` – Grade your particle filter implementation

`pf_gui.py` – Script to run/debug your particle filter implementation with GUI

`particle.py` – Particle and Robot classes and some helper functions.

`grid.py` – Grid map class, which contains map information and helper functions.

`utils.py` – Some math helper functions. Feel free to use any.

`setting.py` – Some world/map/robot global settings.

`gui.py` – GUI helper functions to show map/robot/particles. You don't need to look into details of this file

You need to implement `motion_update()` and `measurement_update()` functions in `particle_filter.py`. Please refer to the particle class definition in `particle.py`.

You will need numpy to implement your filter and tkinter to run the GUI. To install these libraries, please run the command: `pip3 install numpy tk`

To run your particle filter in simulation:

```
> py pf_gui.py       # in Windows
```

or

```
> python3 pf_gui.py # in Mac or Linux (or Windows in cases)
```

You will see a GUI window that shows the rink and Skater-Bot after the competition starts. Skater-Bot's ground truth pose will be shown as red (with a dashed line to show FOV). Particles will be shown as red dot with a short line segment indicate the heading angle, and Skater-Bot's estimated pose (average over all particles) will be shown in grey (if not all particles meet in a single cluster) or in green (all particles meet in a single cluster, means estimation converged).

There are 2 rounds of the competition displayed in `pf_gui.py`: (1). The warm up round, where the skater drives forward, and if it hits an obstacle, it bounces in a random direction. (2). The competition round: the skater performs its figure skating routine. You have spent all season training Skater-Bot to perform his specialty routine: driving in circles (this is the motion the autograder uses). Feel free to change the setup in `pf_gui.py` for your debugging.

Grading: Although Skater-Bot is the one competing, you are also being evaluated on your coaching abilities! Your particle will be evaluated by CS 3630's esteemed coaching judges: the `autograder.py` file. Grading will separately evaluate three capabilities: (1). Your particle filter's estimation can converge to correct value within a reasonable number of iterations. (2). Your filter can accurately track Skater-Bot's motion. (3) The filter is robust enough to track Skater-Bot's motion even when some of the markers are dropped or spuriously detected.

If the judges take a long time to discuss, do not be discouraged! The `autograder.py` file can take from 2 to 6 minutes (or potentially longer!) to run.

We use a total of 5000 particles, and we treat the average of all 5000 particles as the filter's estimation. The particles will be initialized randomly in the space. We define the estimation as *correct* if the translational error between ground truth and filter's estimation is smaller than 1.0 unit (grid unit), *and* the rotational error between ground truth and filter's estimation is smaller than 10 degrees. The grading is split into three stages, the total score is the sum of three stages (each rounded up to the nearest whole integer):

1. [45 points] We let Skater-Bot run 95 time steps to let the filter find global optimal estimation. If the filter gives correct estimation in 50 steps, you get the full credit of 45 points. If you spend more than 50 steps to get correct estimation, a point is deducted for each additional step required. Thus, an implementation that takes 66 steps to converge will earn 29 points; one that does not converge within 95 steps will earn 0 points. For reference, our implementation converges within approximately 10 steps.
2. [45 points] We let Skater-Bot run another 100 time steps to test stability of the filter. The score will be calculated as $45*p$, where p is the percentage of "correct" pose estimations

based on the position and angle variance listed above.

3. [10 points] We let Skater-Bot run with `DETECTION_FAILURE_RATE = 0.1` and `SPURIOUS_DETECTION_RATE = 0.1` (Both these constants are defined in `settings.py`) and we will repeat the test cases in stages 1 and 2.

Notes and Suggestions:

1. In the first two stages of grading there will **not** be any spurious markers or dropped markers. We will enable this by setting the `DETECTION_FAILURE_RATE` and the `SPURIOUS_DETECTION_RATE` in `settings.py` to 0.
2. Your final grade will be an integer value. For example [98.5, 99.5) -> 99
3. In `autograder.py` Skater-Bot will follow a circular trajectory (see the `autograder.py` file in details). We provide several example circles in `autograder.py` for your testing, but in the final grading we will use **another 5 different circles**, and the score will be the average between these five tests. So, make sure you test several different cases to ensure reliability!
4. Particle filtering is a randomized algorithm, each time you run you will get slightly different behavior. Make sure to test your code thoroughly.
5. If you need to sample a Gaussian distribution in python, use `random.gauss(mean, sigma)`.

Submission: Submit only your `particle_filter.py` file in a zip folder with you and your partner's first and last name in the format `Last1First1_Last2First2.zip`. Make sure the `particle_filter.py` file remains compatible with the autograder. Only one partner should upload the file to Gradescope. If you relied significantly on any external resources to complete the lab, please reference these in the submission comments.