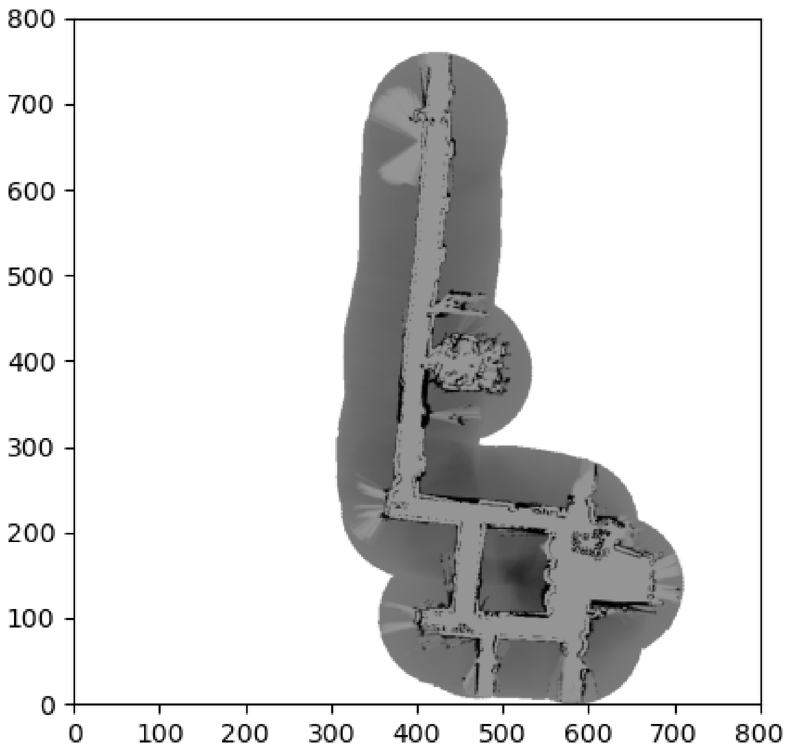
Introduction to Robotics: Homework 4

The goal of homework 4 is to practice and implement what you have learned in the recent courses. In the following exercises, you will implement your own **Monte Carlo Localization** algorithm step by step and test on a real map visualized as following.



The overall algorithm is shown as below.

Algorithm 1 Particle Filter for Robot Localization

```
$\bar{\mathcal{X}}_t = \mathcal{X}_t = \phi$
for $m = 1$ to $M$ do
    sample $x_t^{[m]} \sim p(x_t \mid u_t, x_{t-1}^{[m]})$        (motion model)
    $w_t^{[m]} = p(z_t \mid x_t^{[m]})$                           (sensor model)
    $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \left\langle x_t^{[m]}, w_t^{[m]} \right\rangle$
end for
for $m = 1$ to $M$ do
    draw $i$ with probability $\propto w_t^{[i]}$                 (resampling)
    add $x_t^{[i]}$ to $\mathcal{X}_t$
end for
return $\mathcal{X}_t$
```

Exercise 0

Let's take a look at what we have now. In the scripts folder, you will get:

1. `main.py` : this file is the main program for the Monte Carlo Localization algorithm. By calling other functions you write in this homework, you will be able to help a robot localize itself. You should run this file after you finished implementing all the other modules. **DO NOT MODIFY!!!**
2. `MapReader.py`: this file reads the map for you. **DO NOT MODIFY!!!**

3. `MotionModel.py`: this file is the motion model for the robot. You need to fill in the `MotionModel.update` function, which will be described in detail below.
4. `Resampling.py`: this file is for resampling particles. You need to fill in the `Resampling.low_variance_sampler` function, which will be described in detail below.
5. `SensorModel.py`: this file is the sensor model for the robot. You need to fill in the `SensorModel.beam_range_finder_model` function, which will be described in detail below.

You can add any other functions in the files for your convenience.

Then you need to install the dependencies. Python version 3.7 or 3.8 is recommended. It is recommended to use *conda* and virtual environment to configure the programming environment.

Please run

```
pip install -r requirements.txt
```

to install the dependencies for this homework.

Exercise 1: Motion Model 【30% score】

You need to implement the `MotionModel.update` function. Please follow the notes below when implementing your algorithm.

```
def update(
   self,
   u_t0: np.array, # 1x3
   u_t1: np.array, # 1x3
   x_t0: np.array  # 1x3
) -> np.array:     # 1x3
   # TODO
   return x_t1
```

NOTE1: This exercise asks you to implement the **sample odometry motion model**, which can be found in the slides.

NOTE2: **DO NOT CHANGE THE SEQUENCE OF THE `np.random` TERM!!!**. The sequence of the terms strongly correlates with our autotest program. Changing the sequence may affect the grading of this homework.

Here are the definitions of the variables you are going to be used in your implementation:

1. `u_t0`: input, vector $(\bar{x}, \bar{y}, \bar{\theta})$ in the odometry motion model algorithm.
2. `u_t1`: input, vector $(\bar{x}', \bar{y}', \bar{\theta}')$ in the odometry motion model algorithm.
3. `x_t0`: input, vector (x, y, θ) in the odometry motion model algorithm.
4. `x_t1`: output, vector (x', y', θ') in the odometry motion model algorithm.
5. `del_rot1`: intermediate variable, δ_{rot1} in the odometry motion model algorithm.
6. `del_trans`: intermediate variable, δ_{trans} in the odometry motion model algorithm.
7. `del_rot2`: intermediate variable, δ_{rot2} in the odometry motion model algorithm.
8. `del_rot1_h`: intermediate variable, $\hat{\delta}_{rot1}$ in the odometry motion model algorithm.
9. `del_trans_h`: intermediate variable, $\hat{\delta}_{trans}$ in the odometry motion model algorithm.
10. `del_rot2_h`: intermediate variable, $\hat{\delta}_{rot2}$ in the odometry motion model algorithm.
11. `self.alpha_1`, `self.alpha_2`, `self.alpha_3`, `self.alpha_4`: hyper-parameters correlating with α_1, α_2, α_3, α_4 in the odometry motion model algorithm.

Hint: Make sure that the output lies in $(-\pi, \pi]$.

Exercise 2: Resampling 【30% score】

You need to implement the `Resampling.low_variance_sampler` function. Please follow the notes below when implementing your algorithm.

```
def low_variance_sampler(
   self,
   X_bar: np.array        # Mx4
) -> np.array:
   # TODO
   return X_bar_resampled # Mx4
```

NOTE1: This exercise asks you to implement the **low variance sampling** algorithm, which can be found in the slides.

NOTE2: For sampling $r \sim U(0, \frac{1}{M})$, use the `np.random.uniform` function instead of other random sampling libraries to pass the autotest. Not using the suggested function may affect the grading of your homework.

NOTE3: **Strictly follow the low variance sampling algorithm** in the slides or you might not pass the autotest program and might affect the grading.

Here are the definitions of the variables you are going to be used in your implementation:

1. `X_bar`: input, the set $\{(x_t^{[i]}, y_t^{[i]}, \theta_t^i, w_t^{[i]}), i = 1, 2, \ldots, M\}$ in the low variance sampling algorithm. In the implementation, `X_bar` is an $M \times 4$ shaped array.
2. `X_bar_resampled`: output, the set $\{(\bar{x}_t^{[i]}, \bar{y}_t^{[i]}, \bar{\theta}_t^i, \bar{w}_t^{[i]}), i = 1, 2, \ldots, M\}$ in the low variance sampling algorithm. In the implementation, `X_bar_resampled` is an $M \times 4$ shaped array.

Exercise 3: Sensor Model 【40% score】

You need to implement the `SensorModel.beam_range_finder_model` function. Please follow the notes below when implementing your algorithm.

```
def beam_range_finder_model(
   self,
   z_t1_arr: np.array, # array of 180 values
   x_t1: np.array      # 1x3
) -> float:
   # TODO
   return q
```

NOTE1: This exercise asks you to implement the **beam range finder model**. The algorithm can be found in the slides.

NOTE2: The `SensorModel.rayCast` module is given. Please **DO NOT MODIFY** and do not implement your own ray cast algorithm even though the given one might be suboptimal.

NOTE3: The lidar is 25cm to the center of the agent, while the sensor data is centered at the location of the lidar. Therefore, in the sensor model, you need to calculate the coordinates of the laser based on the agent's coordinates and orientations and then calculate the related weights.

Here are the definitions of the variables you are going to be used in your implementation:

1. `self.map`: the map given by the main program. Please refer to `instruct.txt` for more information. **DO NOT MODIFY**.
2. `self.Z_MAX`: the z_{max} parameter for the four distributions.
3. `self.P_HIT_SIGMA`: the σ_{hit} parameter for the Gaussian distribution p_{hit}.
4. `self.P_SHORT_LAMBDA`: the λ_{short} parameter for the exponential distribution p_{short}.
5. `self.Z_PHIT`, `self.Z_PSHORT`, `self.PMAX`, `self.Z_PRAND`: the weights z_{hit}, z_{short}, z_{max}, z_{rand} for the four distributions.
6. `z_t1_arr`: laser range readings at time t, correlates with z_t in the algorithm.
7. `x_t1`: particle state belief at time t, correlates with x_t in the algorithm.
8. `q`: probability of z_t at time t, correlates with q in the algorithm.

Hint1: maybe you can use $e^{\sum_k \log p_k}$ to replace the original $\prod_k p_k$ for implementation convenience.

Hint2: when calculating the $\log$ probabilities, make sure you filter out all the 0s.

Exercise 4: Run the entire program

After finishing all the modules required for the localization task, you need to run the program `main.py` to generate the final result. Here are the steps for finishing the whole homework.

Step 1: make sure that you are in the `.../hw4/code/scripts` directory before you run your entire program.

Step 2: set the `save_flag` to 1 and clear the **CONTENTS** of `.../hw4/code/scripts/results` directory if any exists. **DO NOT DELETE THE DIRECTORY ITSELF!!!**

Step 3: run `python main.py`. The `main.py` program will first read a series of data from `.../hw3/code/data/log/`. Then it will try to localize the robot using the modules you have implemented.

Step 4: after running `main.py`, there will be a number of photos generated into the `.../hw4/code/scripts/results` folder. Run `python make_video.py` to generate video according to the photos generated. You can check the generated video with the given one to see if everything went right.

Tip1: there is a visualization of the sensor data in `.../hw4/code/data/robotmovie1.gif`, it might help you verify whether your implementation goes right.

Tip2: **DO NOT MODIFY ANY SEED OR PARAMETER**.

How to submit your homework

Compress the entire `hw4` directory to a **ZIP** file named `20xxxxxxxx(your student ID).zip` and submit it to Web Learning (网络学堂).

If you have any problems/bugs on the homework, please contact us via *WeChat group* or email: `chen-xy21@mails.tsinghua.edu.cn` or `zy-jiang21@mails.tsinghua.edu.cn` or leave a message on Web Learning (网络学堂).