Part I

Implementation

1 Filter Parameters

1.1 Constants

Name	Type	Default value	Description
Basic params			
INIT_ESTIMATED_POSE_UNCERTANTY	Python tuple of size 3	(10, 10, 2*math.pi)	Uncertanties of the initial robot pose estimation, used to decide how spread out the particalcloud will be. Postition [0] and [1] coorisponding to the Translation and Drift uncertanties respectively, postiton [2] represent the Ratational uncetanty measure in radian.
Normal resampaling params			
POSE_STD_THRESHOLD_TO_RESAMPLE	Python tuple of size 3	(0.2, 0.2, 0.05*math.pi)	Thresholds for the standar diviation of particles' poses in the particlecloud, if any of the threasholds are exceed, the filter will resample the particles in particlecloud. Postition [0] and [1] coorisponding to the Translation and Drift thresholds respectively, postiton [2] represent the Ratational threshold measure in radian.
RESAMPLE_NOISE	Python tuple of size 3	(0.3, 0.3, 0.1*math.pi)	Noise apply to the particles during resampling.
MIN_NEW_PARTICLE_WEIGHT	Int	4	The minimun weight of the particals during resampaling
MAX_GENERATE_ATTEMPT	Int	5	The maximum attempt to gengrate a partical with weight above MIN_NEW_PARTICLE_WEIGHT
Noisy resampaling params			
POSE_STD_THRESHOLD_TO_NOISY_RESAMPLE	Python tuple of size 3	(4, 4, 0.5*math.pi)	Thresholds for the std of the particles' poses in particalcloud to do a resampaling with more noise
MEAN_PARTICLE_WEIGHT_THRESHOLD_TO_NOISY_RESAMPLE		10	Thresholds for the mean weight of the particles' poses in particalcloud to do a resampaling with more noise
NOISY_RESAMPLE_NOISE	Python tuple of size 3	(10, 10, 0.5*math.pi)	Noise apply to the particles during noisy resampling.
Other			
ESTIMATION_CLUSER_POSITION_RANGE	Python tuple of size 2	(2,2)	The radius of the cluster use for pose estimation

2 Initialise Particle Cloud

The global variable *particlecloud* and *particle_importance* are initialised in this function. *particlecloud* is initialised by calling the helper function *get_random_pose()*, with *initialpose* as the mean and the global constant *INIT_ESTIMATED_POSE_UNCERTANTY* as the noise_range. This function is repeatly called in a for loop, coltrolled by the global constant *NUMBER_PREDICTED_READINGS*, so a cloud of particles are generated around the *initialpose*. *particle_importance* are initialised to an array of 1s with the size equals to *NUMBER_PREDICTED_READINGS*.

3 Update Particle Cloud

In this part, the filter will update the global variable partical_importance according to the newest scan, and optionally chooes to do a *normal resampaling* of a *noisy resampaling* according to the, standard diviation of the poses in particlecloud and the preset constant *POSE_STD_THRESHOLD_TO_RESAMPLE* and *POSE_STD_THRESHOLD_TO_NOISY_RESAMPLE*.

3.1 Update particle importance

When this function is called, it will construct an array by iterate throught the poses in *particlecloud* and get its weight by calling the given *get_weight()* function in sensor model. The elements in the array then averaged with the global *particle_importance* array's elements. The reason behind this is to previent the accumulated weight gets too big and also deal with the situation where the robot is static for a long time but also reciving laser data.

3.2 Resampling

During this part of the filter, systematic resampaling was used to update the poses in the particlecloud, and noise will be reintroduce to the particles accoring to the preset constances *RESAMPLE_NOISE* or *NOISY_RESAMPLE_NOISE*. With one modification to the basic systematic resampaling process, I have decided to apply the constrain that the newly generated particles cannot below a threshold declare by *MIN_NEW_PARTICLE_WEIGHT*, this constrain will prevent the particle being generated outside of the map, and will reduced the time takes for the particles to converge.

4 Estimate Pose

In this part, the filter will first identify the particle with the highest *partical_importance* value, and then return the average pose of the particles around this particle according to the preset constant *ESTIMATION_CLUSER_POSITION_RANGE*

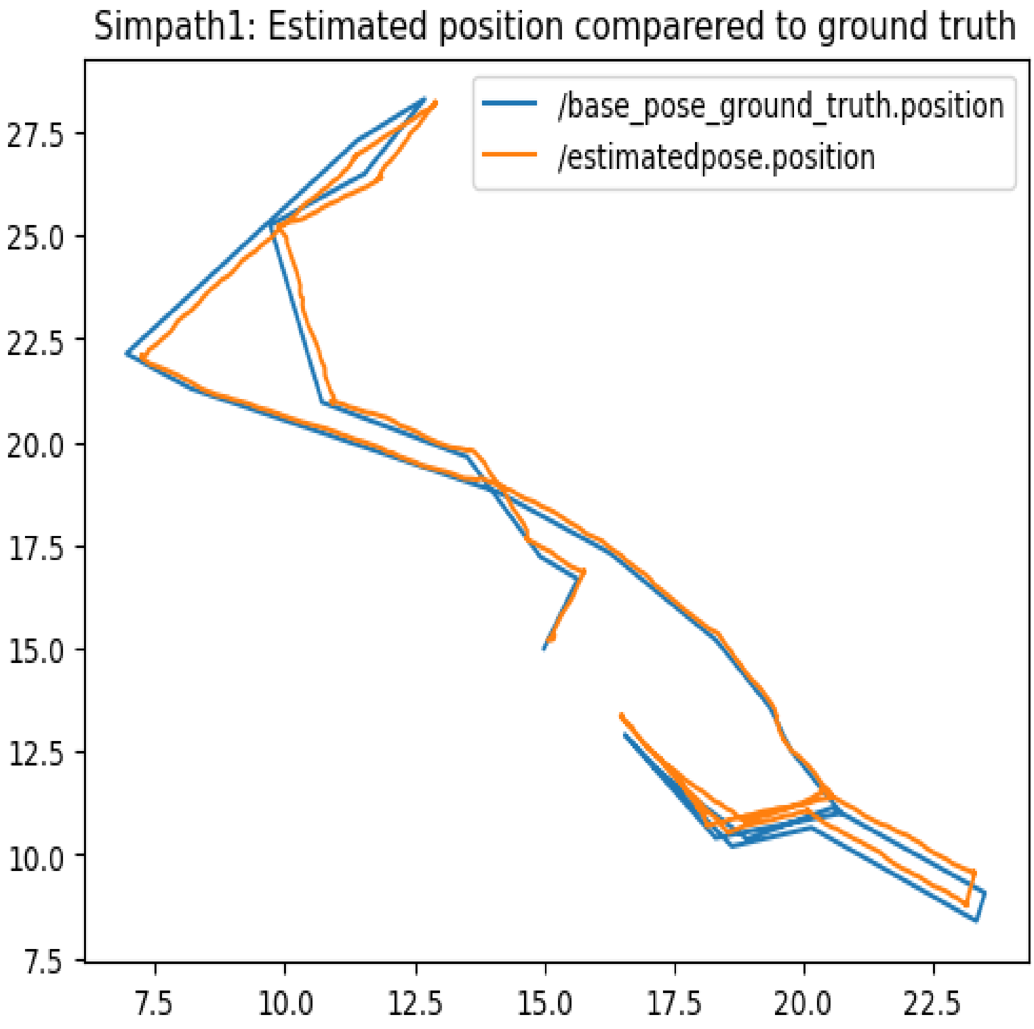
Part II

Benchmark

4.1 The kidnapped robot problem

The filter was able to deal with some of the kidnapped robot problem, also this is the reason why I decided to have 2 different noise for resampaling. It now could work in situation which the movement is parallel to the heading of the robot, however have a poor performance, if the movement is perpendicular, which i assume is because of the motion modle is not expecting such movement and is not spreading the particlecloud perpendicularly.

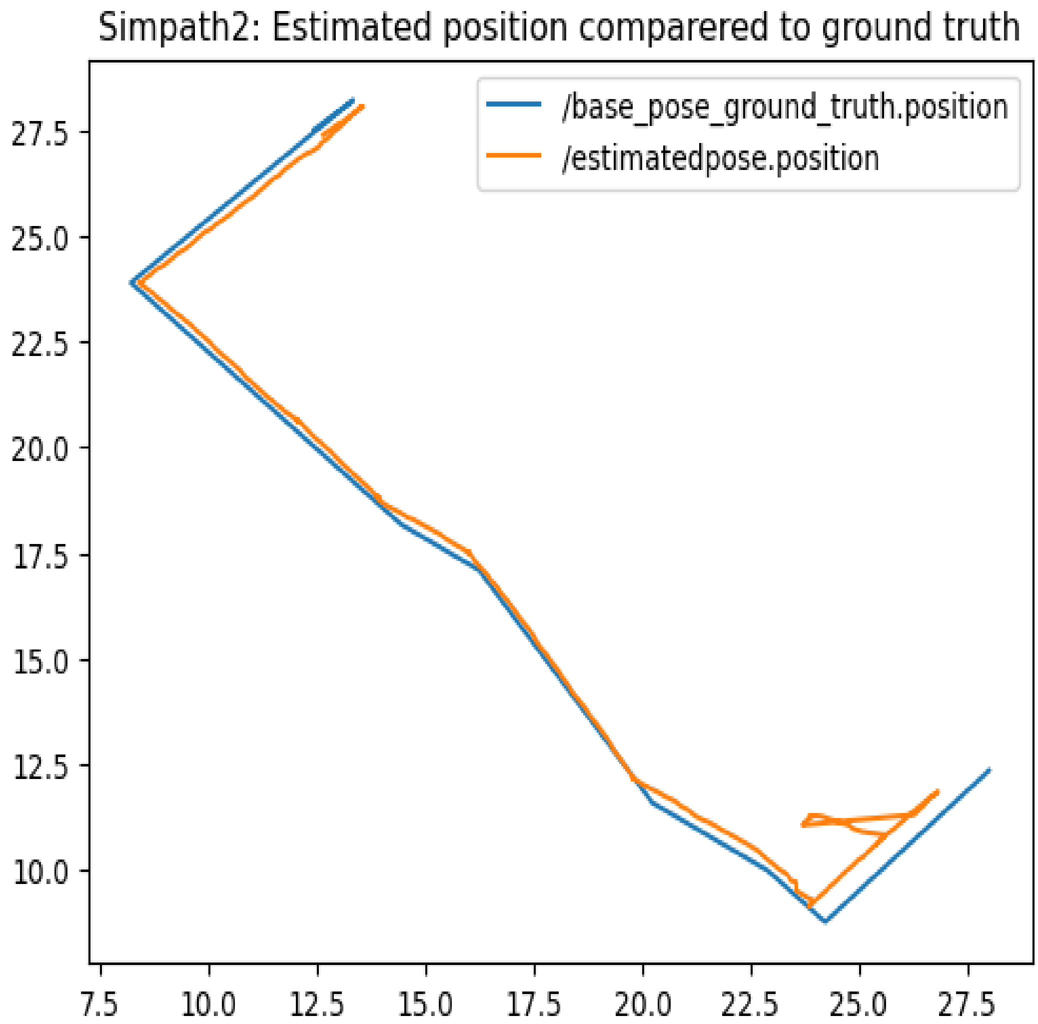
4.2 Simpath1: Estimated position comparered to ground truth



TODO: compare heading TODO: calculate RMSE

The performance of the Particle filter is decent when running the given *Simpath1* bag file, the path produced have a roughtly correct shape, most of the diviation happes while the robot is tuning, which I tried could be reduced by reducing the preset *POSE_STD_THRESHOLD_TO_RESAMPLE* constant. The bag file used to produce this graph can be found here.

4.3 Simpath2: Estimated position comparered to ground truth



TODO: compare heading TODO: calculate RMSE

The performance of the Particle filter is decent when running the given *Simpath2* bag file, However when comparing to the performance on *Simpath1*, it struggled to localise at the start, due to the movement of the robot starts sooner also the distance of the movement are further, which left a small window for the particles to form a cluster. The bag file used to produce this graph can be found here.

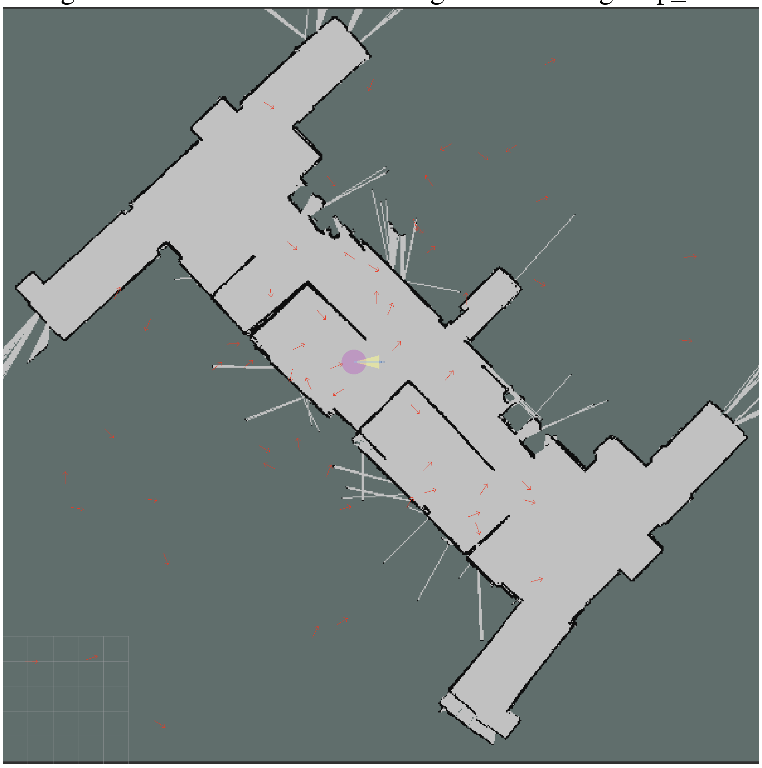
Part III
Appendix

5 Initialise Particle Cloud (Past iterations)

5.1 Iteration#1 - Random particles according to map size

At the start, I had this thinking that if the particles of the filter are more spread out, the particles will have a better chance of getting to the correct pose. So I decided to use the dimensions of the map as the variance in an gaussian distribution to set the initial noise, so the particles will be very spread out on the map1. I realised later in the delopment process that althought the particles are exploring a lots of places, but the low particle density have led to a very slow convergence. Also sometime the actual map might only take up a small space in some of the map files. So I decided using the Map_info to introduce noise to the particles is not practical.

Figure 1: Particlecloud of size 60 generated using map_info



5.2 Iteration#2 - Random particles according to constants

In this iteration, i tried to generate the initial *particlecloud* with constants, which allowed me to predict the particle density way better. However hard-coding the constants into the code results in reduced generalisation while runing the filter in different maps. So in my final implementation, I introduced a global constant *INIT_ESTIMATED_POSE_UNCERTANTY*, which allow user to discribe how inaccurate the initial pose estimation is, and hence control how spread out the particles will be.

6 Update Particle Cloud (Past iterations)

6.1 Iteration#1 - Resample everytime when laser data is recived

This is the iteration where I simply call the *get_weight()* function in sensor model everytime a base_scan is recived. The problem of doing this is althought the particlecloud will converge into clusters, it will result in an very wobbly pose estimation even when the robot is not moving, this is due to the constant updating particlecloud.

6.2 Iteration#2 - Resample according to difference in accumulated particle_importance

After decided I do not want to resample the particlecloud everytime this function is called, I came up with the idea to accumulate the weight of the particles before resampaling, thinking that this will favour the more weighted particles because the there will be a bigger difference in importance between the more accurate particles and the less accurate ones. The result of this has reduced the frequency of resampaling, however it also increased the time taken for the initial convergence of the particlecloud.

7 Estimate Pose (Past iterations)

7.1 Iteration#1 - Averaging everything

7.2 Iteration#2 - Taking the particle with hightest weight