ASSIGNMENT 1 Report:

By NIKHIL JAIYAM (McGill ID: 260852707)

General workflow of the algorithm:

1. Generate particles and initialize them.
2. Extract image that each particle observes.
3. Compare the robot image with particle image.
4. Assign weights based on similarity of images.
5. Calculate the average over the best 30% of our particles based on their weights.
6. Resample the rest of the particles and keep Best particles.
7. Back to step 2.

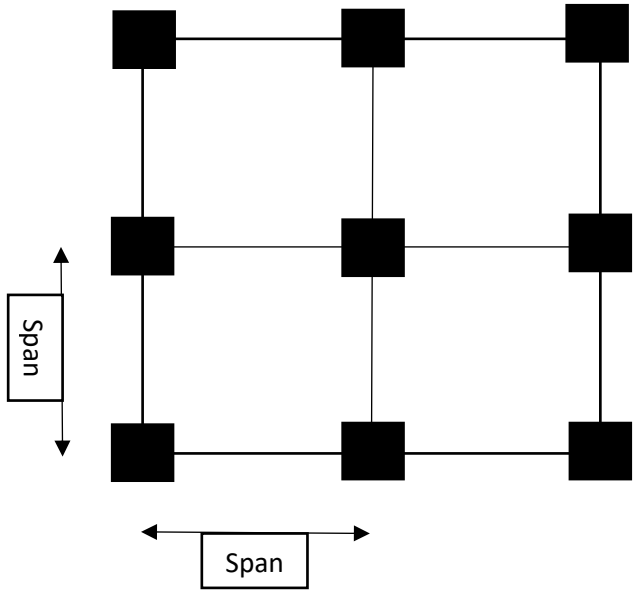
CODE DESCRIPTION:

Sampling approach:

Initially, since we know our robot's position at the beginning of the simulation, the algorithm initializes all particles with their X- coordinate, Y-coordinate and yaw (Abbreviated as Coordinates further in the report.) using the current coordinates of the bot by sampling randomly from a gaussian distribution with mean at the robot's coordinates and an experimentally optimized variance.

After each iteration, the final estimate is calculated by averaging over the best 30% of particles of the total set of particles generated. In successive iterations, all particles are resampled from a gaussian distribution drawn with the final particle's coordinates as mean and an experimentally optimized variance. Since the final particle changes at each iteration, a new distribution is built, and particles are resampled correspondingly.

IMAGE COMPARISON:



In the diagram given above, each black square represents a pixel. In the image comparison model, each image is probed over a 3X3 grid of 9 pixels with are separated by a varying span. The RGB- value at each pixel is compared with the RGB-value of its counterpart in the other image and their norm (an std function.) is saved. Once we go over 9 pixels in each image and we add each pixel's "difference score" (which is just the norm) to form a total picture difference score. Which we use to quantify difference between images.

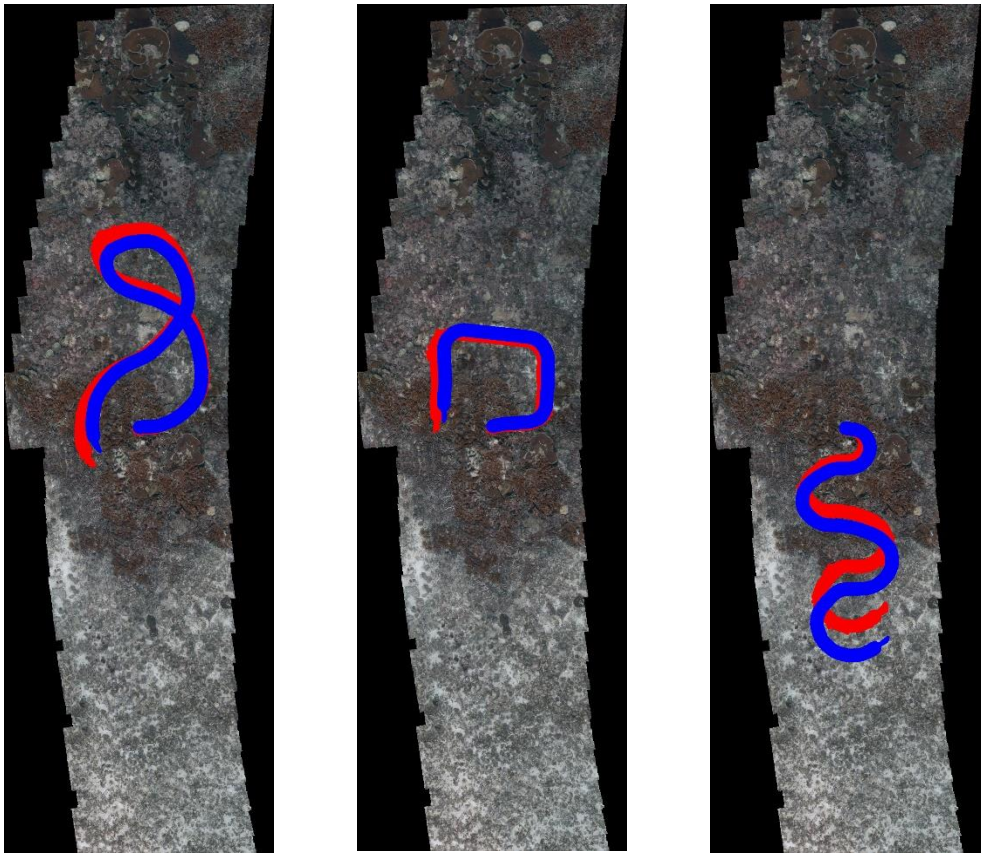
WEIGHING PARTICLES:

There are no extrinsic weights added to particles, just their image difference is directly assigned as their weight. Thereby, A particle with lower image difference would be given a lower weight and therefore would end up on top when all the particles are stored in an array of particles and the said array is sorted with respect to their weights.

Resampling Approach:

Once, all are particles are weighed and our array of particles are sorted, the average over the coordinates of the best 30% of our particles and form a final estimate. The rest of the particles are resampled from a gaussian distribution with our final estimate as the mean and an experimental variance. This way, even if the particles that we didn't resample (that were in the best 30%) turn bad in successive time steps, in the next iteration, they would be weighed again and would end up towards the bottom of the sorted vector and eventually resampled, while a better particle would take their place in the best 30%.

Estimated Results over various trajectories:



The above given results are obtained by generating 10 particles in total and selecting the best 3 particles to form the final estimate. The span in the above test run is 1, which means each probe pixels are 1 pixel apart from each other.

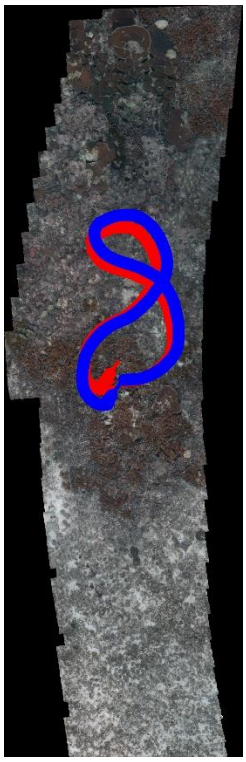
Observations over various parameters:

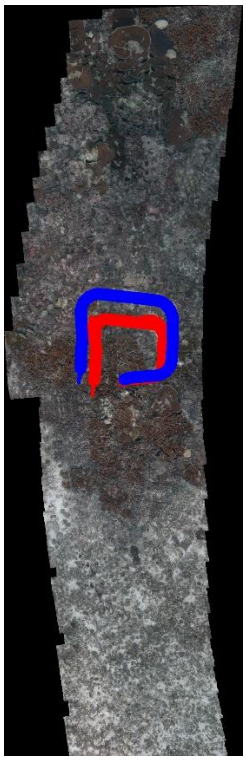
Key parameters varied:

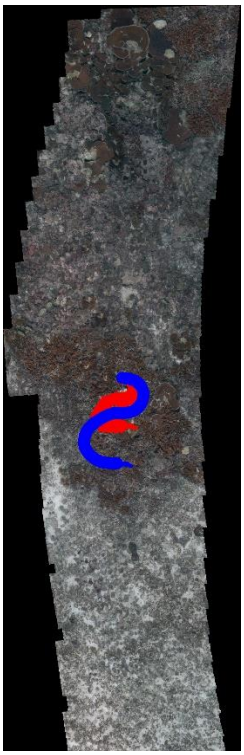
1. Number of particles
2. Span between probe pixels.
3. Different trajectories (emphasis on change in yaw).
4. Variance of the distributions used for sampling.

In the trial runs conducted while varying these parameters, it was observed that the ideal number of particles to get accurate predictions is around 10 particles. In the case where we consider less 10 particles, the model loses accuracy because it does not probe enough pixels (because there's not enough

particles to initiate pixel comparison.) in its vicinity and the probability of landing on a “perfect” particle when they are all sampled from a gaussian distribution is less. When we increase the number of particles to 30, the model again starts performing badly. This is because (due to the best 30% rule for final particle) we end up averaging over 6 particles which introduce considerable noise since they are all sampled from the same distribution and this average also ends up adding 6 random noise samples giving rise to inaccuracy. The model does not gain much from considering an average over 6 particles because the top 3 particles are closest to the mean and capture its approximate position quite accurately, further adding particles to the average does not contribute much to the accuracy of our estimate because the variance considered is small which can be seen from the width of the red line in the above diagram. Results from trial runs with 30 particles are given below.







In the trials conducted, change in span between seems to contribute negligibly to the accuracy of the model. But a loss in accuracy is observed when the span is above 3 pixels.

It can be observed from the results given above that, our estimator performs badly whenever there is a drastic change in yaw. This is because the yaw is being inferred from the X and Y component of the particles and not explicitly being modelled in our algorithm and such behaviour is expected from the particle filter given the guidelines mention on the GitHub page.

The variance of our model is relatively less, which is like a hack given that we already know our initial position. An increase in variance would give us a broader range of estimates which would perform much better in the kidnapped case and which should be the model that we use in a real-world application. An increase in variance would also demand an increase in the number of particles to maintain accuracy of our model because we would end up sampling a wider range of particles and we're more likely to end up with good particles if we increase their number appropriately. The variance in X and Y coordinate in our model depicted above is 0.3 which was selected experimentally.