Particle Filer – Resampling for localization: implementation and Analysis

The following report summarizes the method of implementation of the home-work 4 problem statement. The report analyses the performance of the particle filter algorithm based on number of particles used, the spatial – angular variances used during creation of a new particle, and the similarity scoring schemes used to quantify the match between the expected/generated laser scan to the observed laser scan.

Contents:

Laser.py

ray_tracing():

It is the function which finds the nearest obstacle to the robot given robot position and direction (angle) in which the obstacle is to be searched. It is easy prove that the maximum distance at which a obstacle could be found is the diagonal length of the map i.e. the worst-case scenario would be when the robot is positioned at a corner of a free map, where the furthest obstacle it can find is the diagonally opposite map boundary(corner).

Hence trace a ray from current position of the robot to a point at distance = diagonal of map in direction given by the variable angle.

```
def ray_tracing(x0, y0, angle, the_map):
  diag = hypot(the_map.info.width,the_map.info.height);
  end_x = x0 + diag * cos(angle)
  end_y = y0 + diag * sin(angle)
  ray = line_seg(x0,y0,end_x,end_y)
  p_obs = None;
  for p in ray:
    if (the_map.data[to_index(p[0],p[1],the_map.info.width)] == 100):
      p_obs = p
      break
  return p_obs
```

expected_scan():

This function generates expected laser scan message given a robot position(x,y) and pose (theta); laser sensor properties like minimum angle, angular increment between each range vector, total number rays; and actual map. The expected scan function uses the ray_tracing() function to get the obstacle reading at each angle.

```
def expected_scan(x, y, theta, min_angle, increment, n_readings, max_range, the_map):
  # print ('hello'+str(theta))
  ranges = []
  for i in range(n_readings):
    theta_ = min_angle + i * increment + theta
    #print('theta_ = '+str(theta_))
    p_obs_map = ray_tracing(x,y,theta_,the_map)
    if p_obs_map is not None:
      p_rob_world = to_world(x, y, the_map.info.origin.position.x, the_map.info.origin.position.y, the_map.info.width,
the_map.info.height, the_map.info.resolution)
      p_obs_world = to_world(p_obs_map[0],p_obs_map[1], the_map.info.origin.position.x, the_map.info.origin.position.y,
the_map.info.width, the_map.info.height, the_map.info.resolution)
      dx = p_rob_world[0] - p_obs_world[0]
      dy = p_rob_world[1] - p_obs_world[1]
      ranges.append(min([hypot(dx,dy),max_range]))
    else:
      ranges.append(max_range)
    # print("theta_" + str(rad2deg(theta_)) + " range = " + str(ranges[len(ranges)-1])+' max_range ' + str(max_range))
  return ranges
```

scan_similarity(ranges0, ranges1, max_range):

This function uses a cost-function to major similarity between the ranges0 I.e. Expected laser scan and ranges1 i.e. actual laser scan seen by the robot.

The various cost functions are used to compare their performance in estimating the similarity between the two scans. The table below shows the comparison of the cost functions at various points in CSE550 map. The map is shown in fig.1 - 4

Fig. 1

Fig. 2

Fig. 3

Fig. 4

Fig. 5 query: Experiments to query result - comparing scores from different functions

Fig.	L1 Norm	L2 Norm	Cosine	Correlation
1.	0.76811618688812	0.646777315739	0.72066146534	0.706185603467
2.	0.86256115557812	0.817732951056	0.6977678890305	0.845070804467
3.	0.93686388762112	0.82019007675	0.899369727403	0.871721658456
4.	0.97225825797712	0.918728734585	0.955725594189	0.95974216692

Although l1 norm is able to differentiate between two different norms, it fails in certain places. L2 norm performs better than l1 norm and but cosine similarity is relatively better to quantify the difference between two scans. The correlation works best in defining the similarity between 2 scans and is able to also adequately quantify the difference between expected and true scans.

However, correlation heavily penalizes the offset between the two vectors and thus, experiments were conducted with score either as correlation-score or as weighted average of correlation and l2 Norm. There are instances at which the mean works better than correlation – especially cse550-5 example, where slight off between the true and estimated position heavily changes the expected scan. Availability of more correlating positions add more to the weakness of the scoring mechanism, making the algorithm converge at false positions.

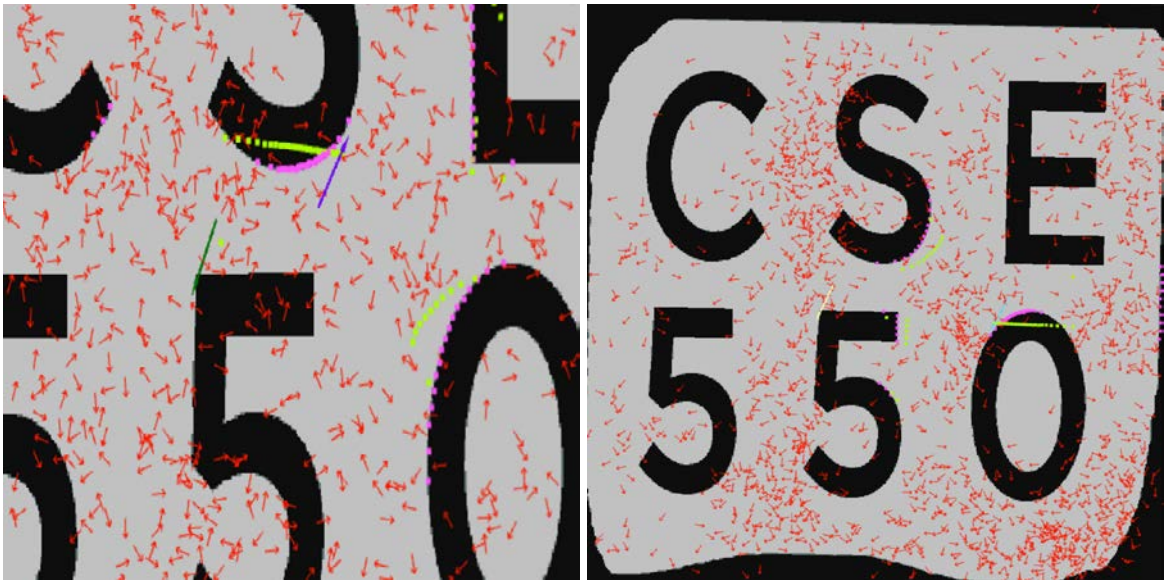


Fig. 6 cse-550-7 false positive: the correlations score good, but the range distances are off

Fig. 7 cse-550-7 1500 / cse550-7 1000: mean of correlation and square cost functions being used

Since, all experiments depend on the initial conditions and randomness, it is however, a hypothesis to conduct further tests. The above results shown in Fig. 7 are of different number of particles, 1500 and 1000 respectively.

Brute Force Output:

Following is the output when we run :

rosrun localization find_pose.py EasyMap.bag easy-1.bag

```
chirag@thunderbolt:~/catkin_ws/src/localization$ rosrun localization find_pose.py EasyMap.bag easy-1.bag
True Position: (0.0, 0.0, 0.0)
Current best: 0.137454925654 (-4.5, -4.5, 0.0)
Current best: 0.572567890825 (-3.5, -0.5, 0.0)
Current best: 0.588031411133 (-3.5, -0.5, 0.7853981633974483)
Current best: 0.690389292473 (-3.5, 0.5, 0.0)
Current best: 0.701232875557 (-3.5, 1.5, 0.0)
Current best: 0.75027162264 (-2.5, -0.5, 0.0)
Current best: 0.759145051959 (-2.5, -0.5, 0.7853981633974483)
Current best: 0.827353904033 (-2.5, 0.5, 0.0)
Current best: 0.842691750528 (-1.5, -0.5, 0.0)
Current best: 0.893887106554 (-1.5, 0.5, 0.0)
Current best: 0.911992892933 (0.5, -0.5, 0.0)

Final Estimate: (0.5, -0.5, 0.0) 0.911992892933
True Position: (0.0, 0.0, 0.0)
```

We get Final Estimate as 0.5, -0.5, 0.0 with correlation based scoring method. With square loss, we get final estimate to be -0.5,-0.5,0.0

Following is the output when we run :

rosrun localization find_pose.py CSE550Map.bag cse550-1.bag -resolution 8

```
^Cchirag@thunderbolt:~/catkin_ws/src/localization$ rosrun localization find_pose.py CSE550Map.
solution 8
rue Position: (-6.389476460186453, -4.036843938683462, 0.0)
urrent best: 0.367581786776 (-9.980000000447035, -9.980000000447035, 0.0)
urrent best: 0.511289682065 (-9.020000021904707, -5.820000093430281, 0.0)
urrent best: 0.533217861254 (-9.020000021904707, -5.820000093430281, 0.7853981633974483)
urrent best: 0.568824876195 (-9.020000021904707, -5.820000093430281, 2.356194490192345)
urrent best: 0.577116096574 (-9.020000021904707, -5.500000100582838, 2.356194490192345)
urrent best: 0.578535047416 (-9.020000021904707, -5.180000107735395, 5.497787143782138)
urrent best: 0.581027406577 (-9.020000021904707, -4.860000114887953, 5.497787143782138)
urrent best: 0.584489228558 (-9.020000021904707, -4.54000012204051, 5.497787143782138)
urrent best: 0.58479131736 (-9.020000021904707, -4.2200001291930676, 2.356194490192345)
urrent best: 0.58832286 (-9.020000021904707, -4.2200001291930676, 5.497787143782138)
urrent best: 0.601505926005 (-9.020000021904707, -3.900000136345625, 2.356194490192345)
urrent best: 0.612934221894 (-9.020000021904707, -3.5800001434981823, 2.356194490192345)
urrent best: 0.62974037778 (-9.020000021904707, -3.2600001506507397, 2.356194490192345)
urrent best: 0.636722464561 (-9.020000021904707, -2.940000157803297, 2.356194490192345)
urrent best: 0.639070115755 (-9.020000021904707, -2.6200001649558544, 2.356194490192345)
urrent best: 0.63950734736 (-9.020000021904707, 1.5399997420608997, 2.356194490192345)
urrent best: 0.648790770361 (-9.020000021904707, 1.8599997349083424, 2.356194490192345)
urrent best: 0.652693744168 (-9.020000021904707, 2.179999727755785, 2.356194490192345)
urrent best: 0.656245272489 (-9.020000021904707, 2.4999997206032276, 2.356194490192345)
urrent best: 0.65643996935 (-9.020000021904707, 2.8199997134506702, 2.356194490192345)
urrent best: 0.672516567934 (-8.700000029057264, -7.7400000505149364, 3.9269908169872414)
urrent best: 0.676507917122 (-8.700000029057264, -3.5800001434981823, 2.356194490192345)
urrent best: 0.686790028423 (-8.700000029057264, -3.2600001506507397, 2.356194490192345)
urrent best: 0.689932163013 (-8.700000029057264, -2.940000157803297, 2.356194490192345)
urrent best: 0.702344559232 (-8.700000029057264, 1.219999749213457, 2.356194490192345)
urrent best: 0.724769115722 (-8.700000029057264, 1.5399997420608997, 2.356194490192345)
urrent best: 0.732247625493 (-8.700000029057264, 1.8599997349083424, 2.356194490192345)
urrent best: 0.738420198522 (-8.700000029057264, 2.179999727755785, 2.356194490192345)
urrent best: 0.740129835932 (-8.700000029057264, 2.4999997206032276, 2.356194490192345)
urrent best: 0.741671925241 (-8.700000029057264, 3.139999706298113, 2.356194490192345)
urrent best: 0.746390527561 (-8.380000036209822, 2.179999727755785, 2.356194490192345)
urrent best: 0.749498289212 (-8.380000036209822, 2.8199997134506702, 2.356194490192345)
urrent best: 0.768651163723 (-7.7400000505149364, 6.0199996419250965, 4.71238898038469)
urrent best: 0.808793954858 (-6.460000079125166, -4.2200001291930676, 0.0)
urrent best: 0.905815526335 (-6.460000079125166, -3.900000136345625, 0.0)
urrent best: 0.957890698584 (-6.140000086277723, -4.2200001291930676, 0.0)
```

This ouput is as a result of mean of square and correlation scores

Particle.py

random_particle():

generates a random particle which is within the free space of the map

new_particle():

generates a new particle around a given particle passed as parameter by adding noise to its position of spatial variance and adding noise in orientations of angular variance – both passed as parameters.

resampling():

This function re-samples the particles based on the weights. Weight of a particle is measure of likelihood of the robot receiving a current laser scan message. The particles with less probability are discarded and new particles are generated around the particles with higher weights. We used cumulative density function to model a roulette wheel – which uses a random number from a normal distribution to choose which particle is to be used as reference to create a new particle – by adding noise. As mentioned, the arc of the roulette wheel is thus proportional to the weight of the particle and hence, higher the weight of the particle, higher is the chance of it being used as a reference for the new particle.

The hyper-parameters for re-sampling are in the generation of the new particle. The spatial variance and angular variance place great role in making the algorithm converge to a most-likely point.

Also if there is symmetry in the Map, it is possible to have multiple positions in the map which would give same similarity-score. Hence it becomes extremely difficult once, if the particle filter gets biased towards a wrong position. Therefore, we modify the resampling algorithm to have some percentage k of new samples, in addition of roulette wheel – just to provide a chance for the algorithm to look at different place. In addition to have better convergence, we always keep the particle with the best estimate, unaltered.

Consider system with 10 particles, spatial variance of 5% of maximum area and angular variance to be 10% of pi:
We see that system converges to the actual position. It can be seen, that normally distributed random particles are dropped on the map. After few iterations, it can be noticed that the particles are grouped into 2 or 3 clusters, which eventually decrease to 1 cluster with all 10 particles. Since the spatial covariance is large, the 10 particles is relatively widely spread as compared to the experiment in which the spatial variance is low. This increases the probability of the particle filter to generate a particle which is closer to the actual estimated value, and hence the convergence rate is fast.

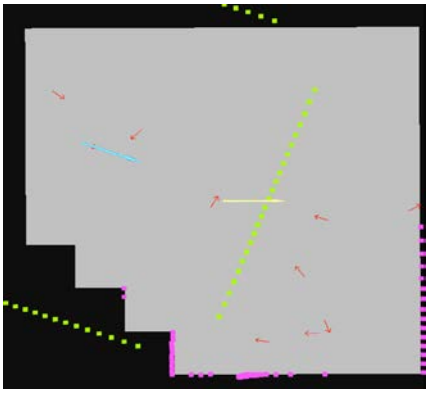


Fig. 8 Spatial Variance 5 iteration 1 Best Score 0.74

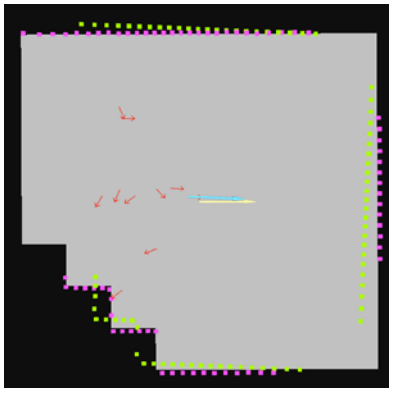


Fig. 9 Spatial Variance: 5, iteration 67, : Best Score 0.87

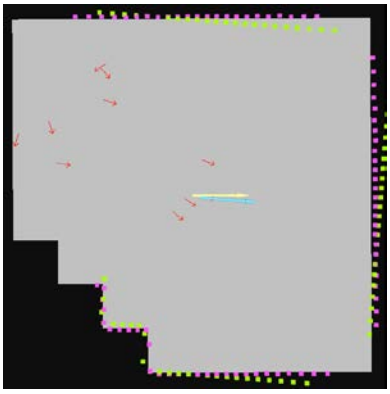


Fig. 10 Spatial Variance 5: iteration 116: Best Score 0.976

It is to be noticed that, although the particle filter convergence faster to the actual pose, the precision of the position is less compared to the lower spatial variance. This is because, if a particle reaches to a close vicinity of the actual position, due to large spatial variance, the probability of the next particle to fall exactly on the actual position is less when compared with an experiment of smaller spatial variance.

This can be seen with experiments when we run with spatial variance 3 and 1 and compare it to that of 5.

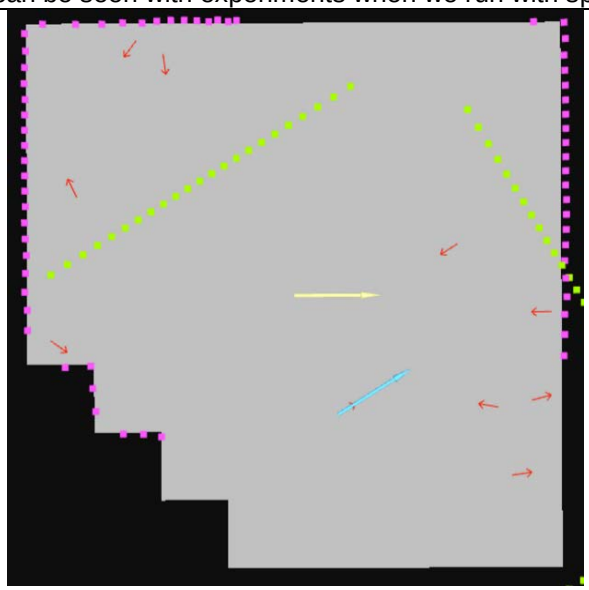


Fig. 11Change this fig

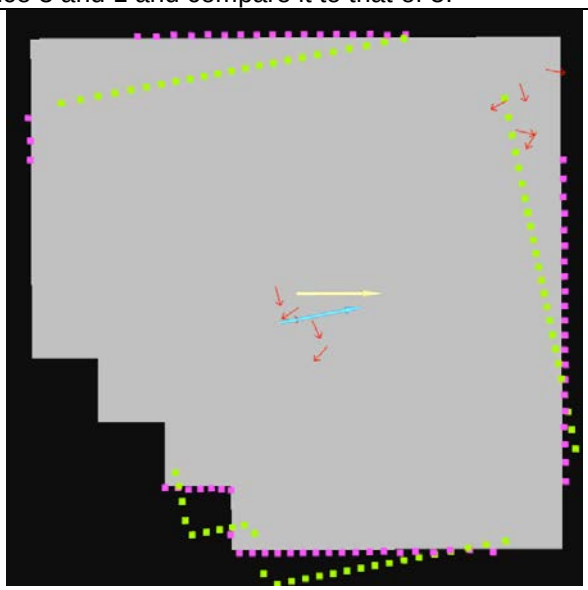


Fig. 12Spatial Variance: 3, iteration 75, : Best Score 0.905

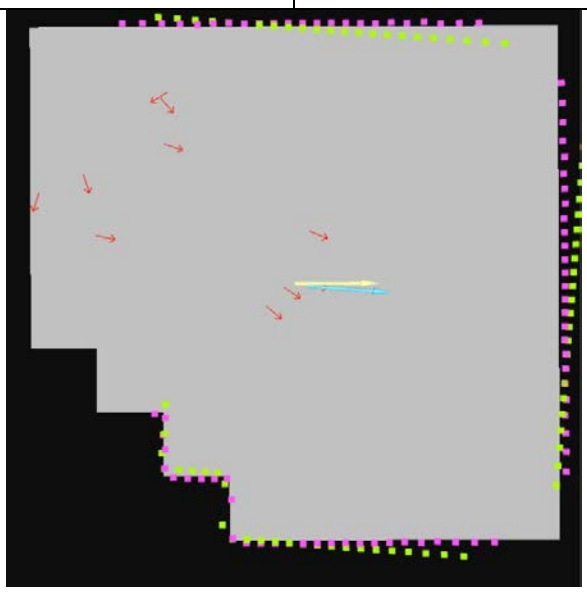


Fig. 13 Spatial Variance: 3, iteration 156, : Best Score 0.97

It can be deduced, initiatively and from the repeated running of the experiments that as the spatial variance is reduced, more number of iterations are required for the algorithm to converge to its best estimate.

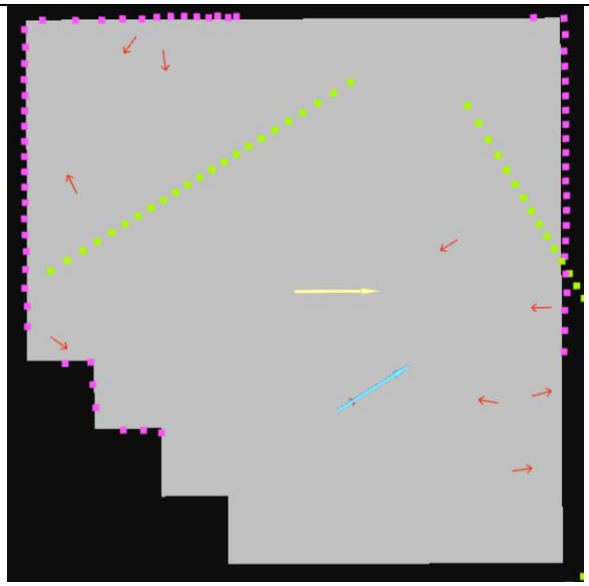


Fig. 14 Spatial Variance: 1 iteration 1, : Best Score 0.77

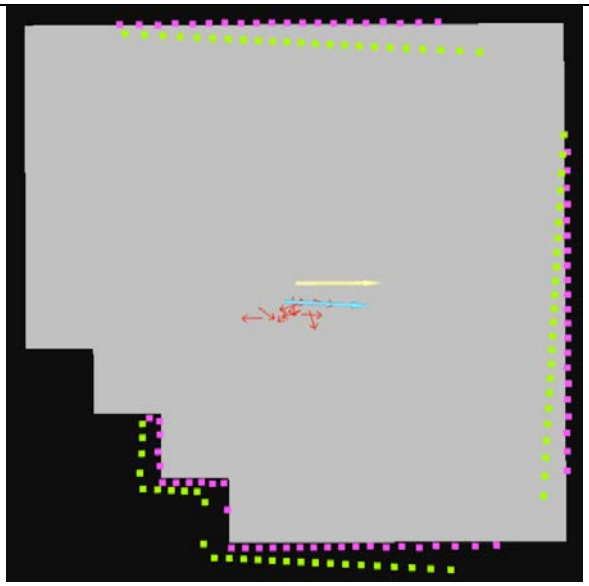


Fig. 15 Spatial Variance:1, 15th iteration, Best Score = 0.915

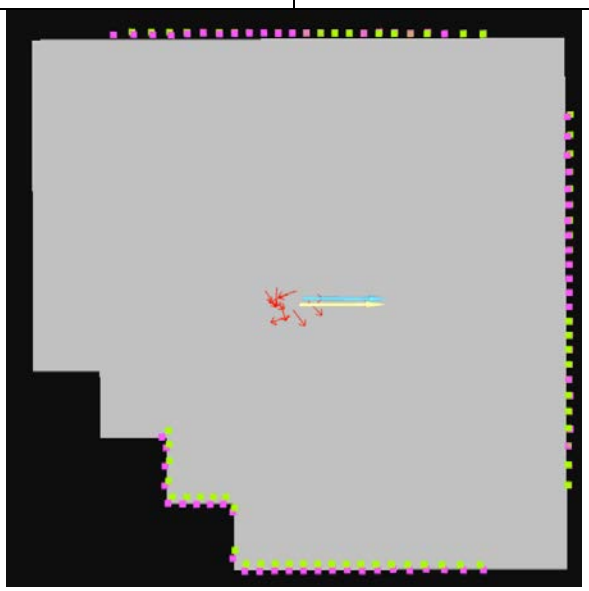


Fig. 16 Spatial Variance:1, 77th iteration, Best Score = 0.998

By chance, due to good the first samples dropped and the following random points generated in the above experiment, the convergence of the algorithm was found to be quicker than with the spatial variance 5. Though, this might look good, the drawback of having a small spatial variance is that the convergence of the algorithm then becomes highly dependent on the first samples dropped. If the nearest best estimate of the position fall far away from the actual positon, all the particles after few iterations may converge near the position of this estimated particle. This may make the probability of the new particle to fall near to the actual position very low, as the new particles are just added by adding noise to existing particles with high weights. Hence, it will take the algorithm a very long time to move from its falsely estimated position to actual position as the spatial variance is very low. A possible solution is to thus add a few normally distributed random particles on map after every few iterations.

It can be seen that a poorly dropped random sample at the start of the experiment makes the algorithm not converge even after 200 iterations (spatial variance 1).

Thus, to have a bench mark, we will run all the experiments with spatial variance 3 and angular variance as 10% of pi – as trade-off between accuracy and convergence.

Running the algorithm on CSE550 Map:

It is noticed that although having small number of particles makes the algorithm converge to a estimated new position on EasyMap, running with small number of particles in a complex map with relatively high resolution will take large amount of iterations and still not guarantee convergence. This is because, once the initial hypothesis of percentage of small number of particles gets confidence, the spatial variance kicks in, instead of normal distribution and hence, testing of positions at distant places becomes less and less unlikely.

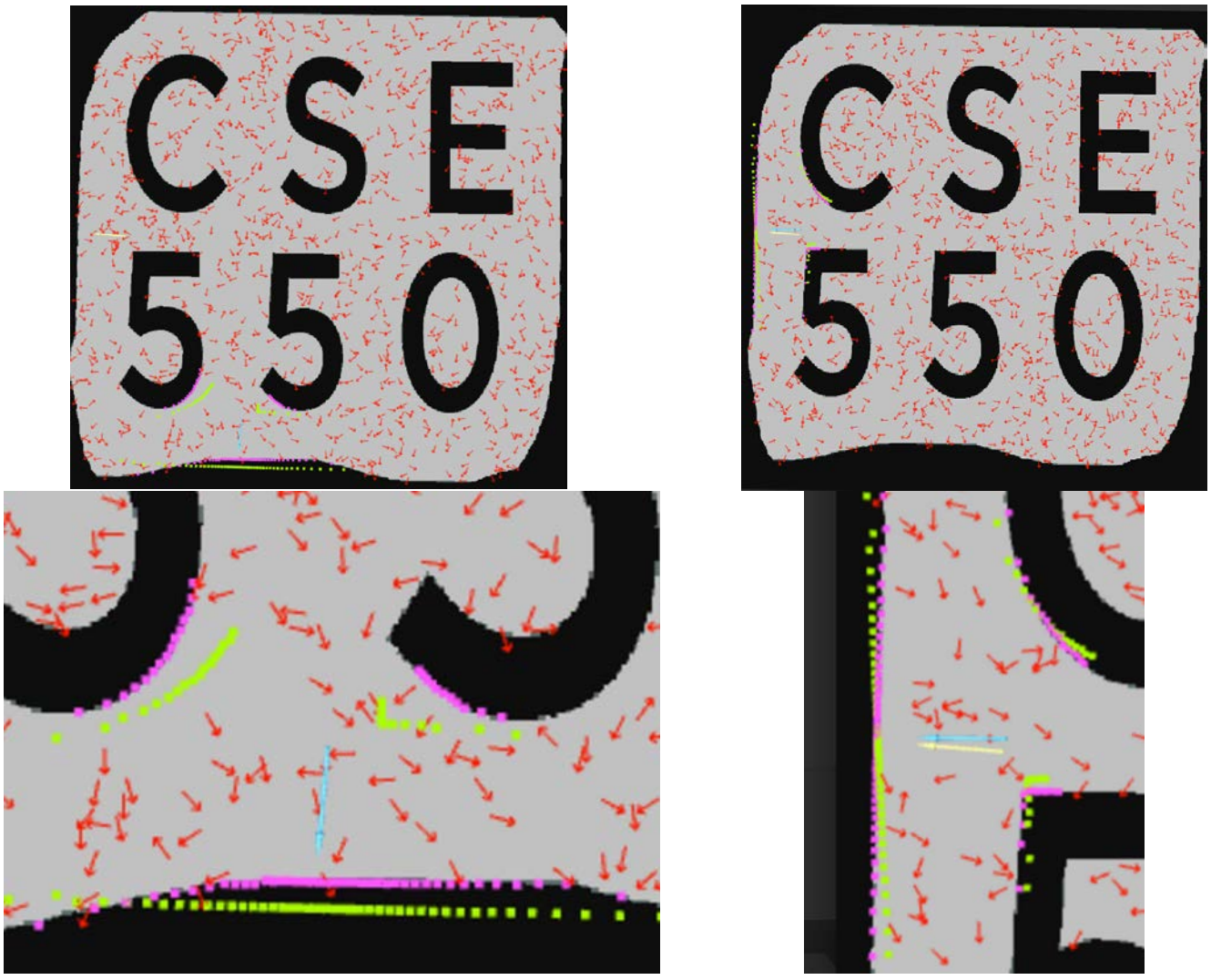
Hence we use large number of particles – say 1000 to 1500, to test the algorithm on CSE550 Map and observe following results.

1. csemap-1 example:

Fig. 17 cse550-1 Iterations

Based on the first normally distributed samples, the filter chooses the position with best score – which is not a true estimate. All the positions with “better-score” are tweaked a little by adding noise and a new score is calculated.
It can be seen that the next best score occurs in the second iteration itself. We have added a constraint of always using the best-sample without tweaking it. Hence, it preserves the last best position in successive iterations. Hence, for rest of the iterations – from 2 to 9, the no other position is found better than the one estimated in 2nd iteration and thus, it remains preserved.

2. csemap-2 example:




```
^Cchirag@thunderbolt:~/catkin_ws/src/localization$ rosrun localization hill_climb.py CSE550Map.bag cse550-2.
True Position: (-7.278950478542229, 0.44473704409224823, 3.0766762671271106)
Best estimate (0): (-2.7800001613795757, -6.500000078231096, -1.6460730981133167) 0.935506602438
Best estimate (1): (-2.7800001613795757, -6.500000078231096, -1.6460730981133167) 0.935506602438
Best estimate (2): (-2.4361376157424024, -6.391546471356257, -1.5807079993754893) 0.960051207851
Best estimate (3): (-7.242876154365307, 0.6330124791020841, -3.136808336528307) 0.97071723913
Best estimate (4): (-7.242876154365307, 0.6330124791020841, -3.136808336528307) 0.97071723913
Best estimate (5): (-7.242876154365307, 0.6330124791020841, -3.136808336528307) 0.97071723913
Best estimate (6): (-7.242876154365307, 0.6330124791020841, -3.136808336528307) 0.97071723913
Best estimate (7): (-7.242876154365307, 0.6330124791020841, -3.136808336528307) 0.97071723913
Best estimate (8): (-7.242876154365307, 0.6330124791020841, -3.136808336528307) 0.97071723913
Best estimate (9): (-7.242876154365307, 0.6330124791020841, -3.136808336528307) 0.97071723913
True Position: (-7.278950478542229, 0.44473704409224823, 3.0766762671271106)
```

Fig. 18 cse555-2: Output of the second experiment - run for 1000 particles - correlation score

It can be seen that the particles start with false estimated starting position, but after 4 iterations the particles start converging to the real position. This shows the importance of the hyperparameter - number of iterations in running the algorithm. It can be safely deduced that, if the number of iterations are small the algorithm may converge to a false position. On other hand, if the number of iterations is sufficiently large, there may be no improvement in the estimate – considering we have a 'good' estimation of variance, and make the program run more computations expensive.

3. cse550-3 example:

```
chirag@thunderbolt:~/catkin_ws/src/localization$ rosrun localization hill_climb.py
CSE550Map.bag cse550-3.bag -particles 1500 -iterations 10 -ros
True Position: (-2.3028718742999215, 6.636562527237244, 2.187984076568699)
Best estimate (0): (-2.220000173896551, 6.779999624937773, 2.1064312697063667)
0.917463631817
Best estimate (1): (-2.220000173896551, 6.779999624937773, 2.1064312697063667)
0.917463631817
Best estimate (2): (-2.220000173896551, 6.779999624937773, 2.1064312697063667)
0.917463631817
Best estimate (3): (-2.220000173896551, 6.779999624937773, 2.1064312697063667)
0.917463631817
Best estimate (4): (-2.220000173896551, 6.779999624937773, 2.1064312697063667)
0.917463631817
Best estimate (5): (-2.220000173896551, 6.779999624937773, 2.1064312697063667)
0.917463631817
Best estimate (6): (-2.220000173896551, 6.779999624937773, 2.1064312697063667)
0.917463631817
Best estimate (7): (-2.220000173896551, 6.779999624937773, 2.1064312697063667)
0.917463631817
True Position: (-2.3028718742999215, 6.636562527237244, 2.187984076568699)
```

Fig. 19 cse550-3: True results with 1500 particles and 10 iterations

Here, as the number of particles dropped in the first scan (normal distribution) is high (1500), the map gets the best estimate quite close to the original position with score of 0.9174.

The figures show the positions of all particles at 1st iteration and last iteration. It is clearly seen that the particles converge towards the position of high-likelihood of the positions on the map where robot can view the received scan.

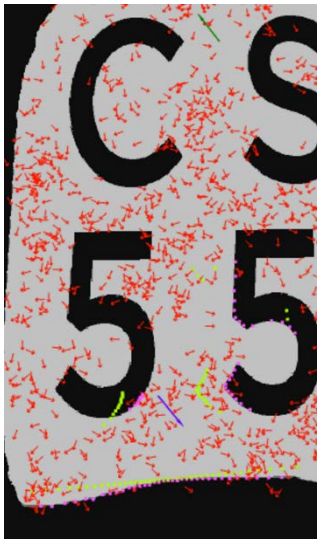
Hence it can be inferred that more the number of particles, more likely the algorithm will converge faster – in terms of number of iterations, but it is important to note that, since – scoring is done for each particle in a iteration, the algorithm will take longer time to converge due to increased computation.

Also, with high number of particles dropped, the precision of obtaining exact position is increase, if a ‘good’ variance is selected for the noise added to individual particles, as there are many particles to choose the best from, and with each iteration, all the particle move towards the best-expected positions. This is depicted in cse550-4

4. cse550-3 false positive.

Since, there are multiple positions in the map where the received scan can match the estimated scan, it is possible for the particle filter to converge at false position with similarity between the scans giving a high score. This is more probable if the number of particles we began with are less, and they start to converge to a false position during successive iterations

It is unlikely that out of the blob of particles around the true positions, none of them give a good score over the one chosen. But if we see the figure closely, we find that there is strong orientation mismatch between the particles in that region and exact pose. This is because, we have high number of dimensions to estimate the pose [x,y,theta] and all three are randomly dropped



After few iterations, of generating points near high likelihood of the estimated scan, the algorithm jumps to another position in the map which shoes better correlation with the true scan compared to the 1st iteration. It's just by chance that the algorithm did not generate enough points near the true position as particles got low weights due to orientation mismatch

Finally it converges to the best estimate it finds amongst the clustered particles

Fig. 20 cse550-3 false positive

An argument is to add a percentage of fresh samples at every iteration to increase the likelihood of them falling near the true position. As seen in the figure above, there is empty space near the green – arrow which is the actual position. However, I feel, with high dimension of vector defining the position i.e. [x,y,thetha] – and with a very few new-samples (percentage of total number of points), it is unlikely that they fall would fall in correct position with correct orientation and give better score than 'false-positive' result we get in some other part of the map.
In addition, if we propagate the current belief when the robot moves and incorporate the action taken by the robot to generate the next expected scan. This, will eventually make the robot correct its belief over time and get better estimates.

5. cse550-4 example:

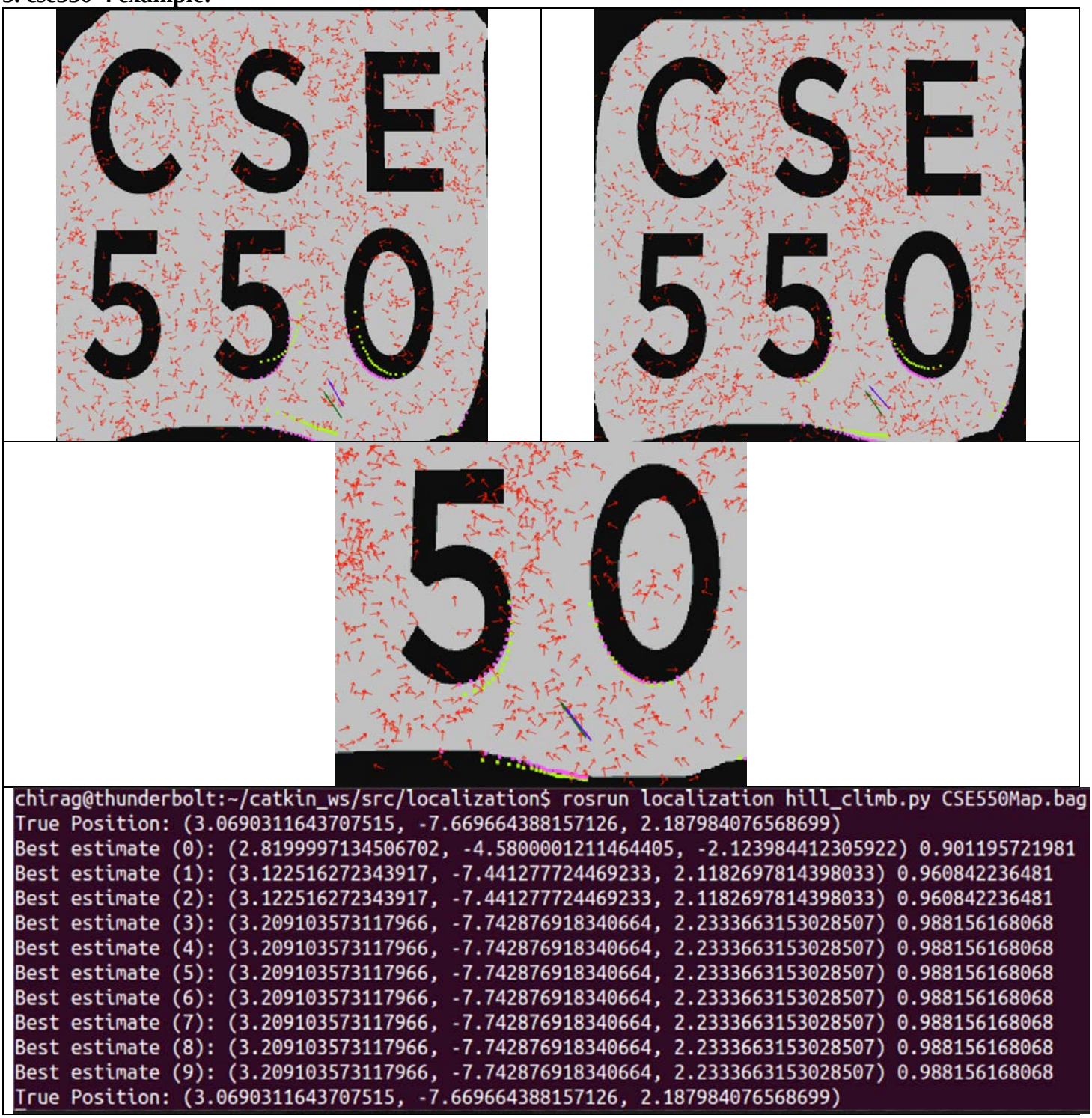


Fig. 21 cse550-4: 3 improvements in estimate of the position

The increased precision is because of increased number of particles which are flogged towards the expected scan. With each iteration than tweaks to these 'best' particles with some noise and expect to get a better similarity score. Here, we start with similarity score of 0.90, few more particles are re-located to the region around this best score position (due to its high weight) by adding noise to this position. We find that in 2nd iteration, we get better similarity match of 0.96. Re-iterating the above logic, we get a even better score of 0.988 in 4th iteration.

6. cse550-5 example:

Importance of iterations is evident in this example:

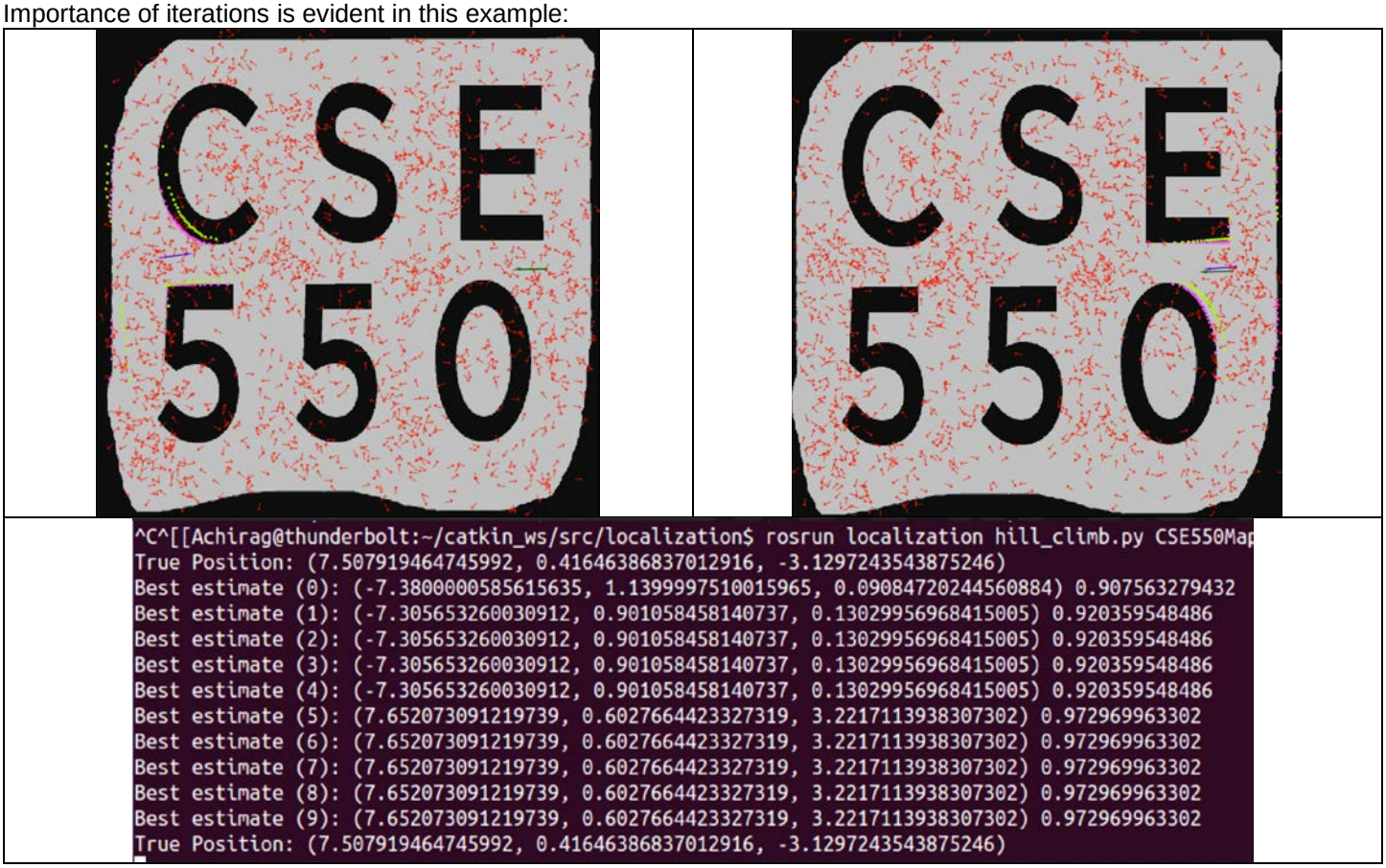


Fig. 22cse50-5

As seen, if the number of iterations where limited to 5, the algorithm would have converged to a false positon. The best-score makes the algorithm jump to exactly opposite position where a similar scan data can be found with better score! Hence, the algorithm finds exactly on 6th iteration, this new point and converges at correct estimate with 0.97 score.

7. cse550-6 example:

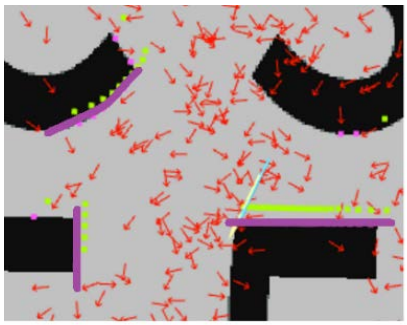
Fig. 23 cse550-6

This is one of the configurations where we need large number of sample for algorithm to converge at correct position. This

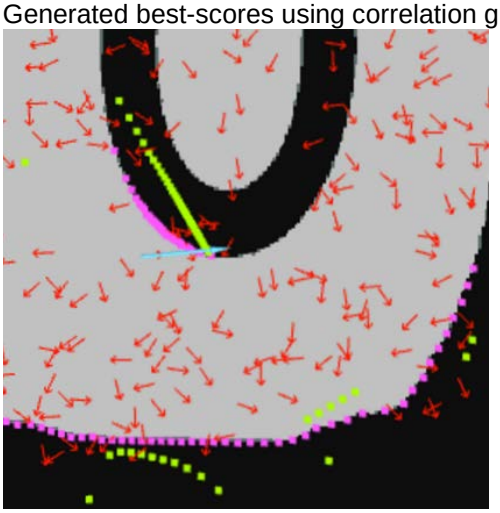
is because, after initial dropping of points, the only way the algorithm can best-estimate is by tweaking its initial best hypothesis with noise. Since, the area is enclosed inside an obstacle, no particles placed outside the closure would be able to make through the obstacle walls, as the scans generated in intermediate steps will be off – and the weights of those particles will be reduced – denying the possibility to further tweak that position with noise. If there was a free space, as seen in EasyMap, less particles will still be able to move towards better estimate due to spatial variance and increase in similarity in expected scan with each iteration. This will not be the case for a particle outside aiming for a position which enclosed in the obstacle.
A way to counter this problem would be to increase the variance of the noise added to the particles, but it would be a problem-specific fix and may worsen the general case.

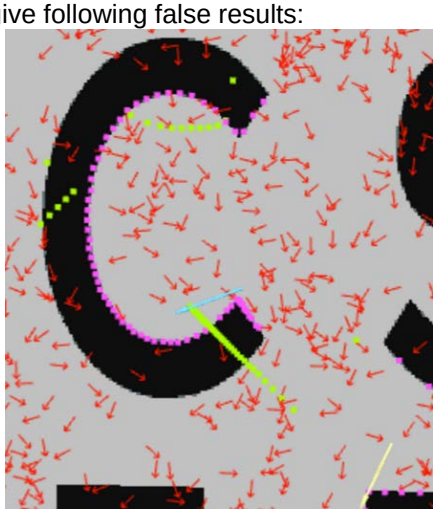
8. cse550-7 false results:

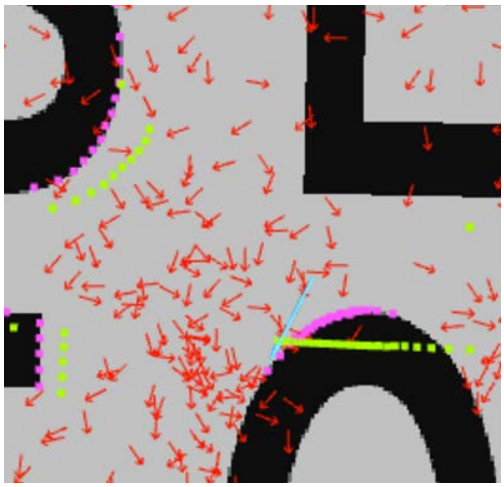
Here, we get to know about importance of robustness of the cost-function. The correlation is sensitive to the sequence of matching the vectors, more so than its magnitude. Correlation is measure that indicates the extent to which two or more variables fluctuate together



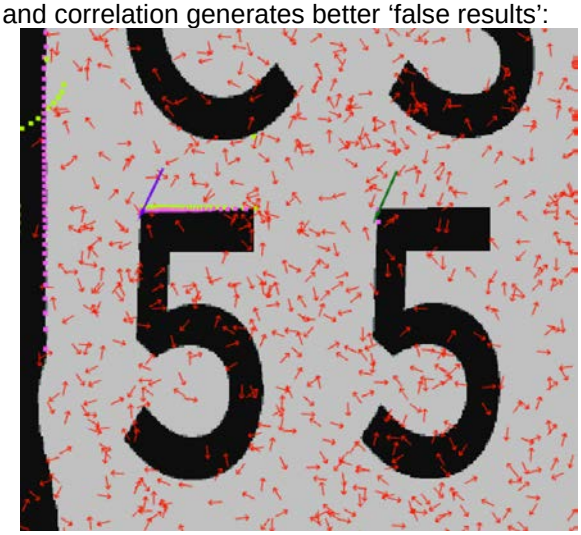
Whereas the least-square measures the difference in the magnitudes of the corresponding vectors.
Here, we can see that the laser-scan is much skewed – most points lie on 2 perpendicular lines as highlighted in the figure below. This causes the correlation function to perform poorly as a slight change in robot position from its expected place will change the sequence of the expected sequence dramatically.
This makes the correlation function score more to particles which generate estimates which match the given fluctuation in true scan rather than depend more on the magnitudes.
Generated best-scores using correlation give following false results:







Whereas, the mean of least-square and correlation generates better ‘false results’:



This is just a hypothesis, we need more experiments to prove this – the results may just be chance. All the figures can be found on drive /cse550 false positives

9. cse550-7 true results:

Fig. 24 cse550-7 1500

It is thus, to be noticed, that with mean of square-correlation loss, the scores are relative lower as compared to the ones before. This is because, although the least-squares gives a ‘good’ score to a relatively matching scans, co-relation pulls it down if there is mismatch in ordering. Again, when the correlation gives a good score – to the cases shown in figures above, the square loss fines them heavily – thus improving the results.