

# Kidnapped Vehicle

## 1. Purpose :

The purpose of project is to build a Particle filter and combine it with real map to localize the vehicle.

## 2. Project Inputs :

- a. A Github repo with starter code(Workspace can also be used)
- b. A Simulator which provides the script for the noisy position data, vehicle controls, and noisy observations. The simulator can also display the best particle's sensed positions, along with the corresponding map ID associations.
- c. Files in the Source (src) folder includes –
  - i. main.cpp
  - ii. particle\_filter.cpp
  - iii. particle\_filter.h
  - iv. json.hpp
  - v. map.h
  - vi. helper\_functions.h

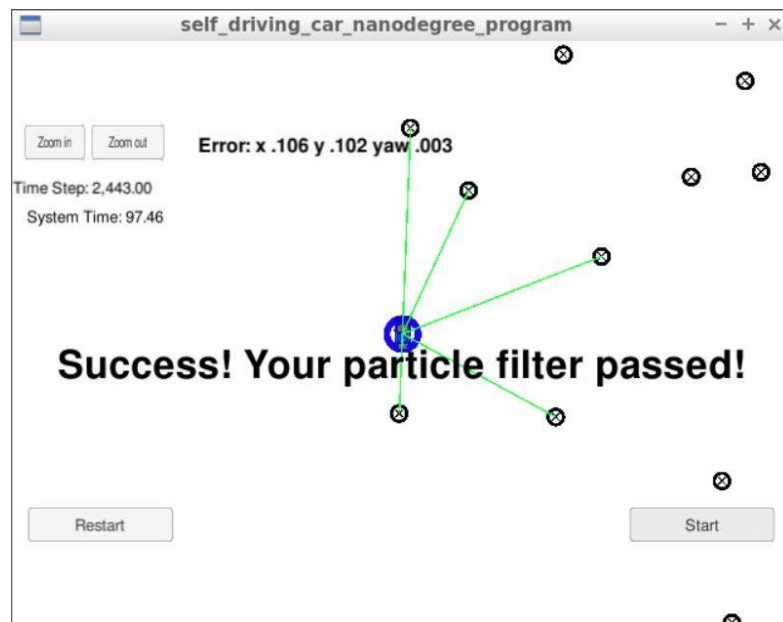


Figure 1: Simulator – Vehicle status information

### **3. Project Outputs :**

- a. To satisfy project rubric following artifacts are expected –
  - i. build Folder
  - ii. particle\_filter run file
- b. To satisfy project rubric following specifications to be satisfied –
  - i. Particle filter shall localize the vehicle within the desired accuracy.
  - ii. Particle run successfully within the specified time of 100 seconds.
  - iii. Code shall use a particle filter to localize the vehicle.

#### 4. **Introduction to Particle Filters :**

Particle filters or Sequential Monte Carlo (SMC) methods are a set of Monte Carlo algorithms used to solve filtering problems arising in signal processing and Bayesian statistical inference. The filtering problem consists of estimating the internal states in dynamical systems when partial observations are made, and random perturbations are present in the sensors as well as in the dynamical system. The objective is to compute the posterior distributions of the states of some Markov process, given some noisy and partial observations. The term "particle filters" was first coined in 1996 by Del Moral in reference to mean field interacting particle methods used in fluid mechanics since the beginning of the 1960s. The term "Sequential Monte Carlo" was coined by Liu and Chen in 1998.

Particle filtering uses a set of particles (also called samples) to represent the posterior distribution of some stochastic process given noisy and/or partial observations. The state-space model can be nonlinear and the initial state and noise distributions can take any form required. Particle filter techniques provide a well-established methodology for generating samples from the required distribution without requiring assumptions about the state-space model or the state distributions. However, these methods do not perform well when applied to very high-dimensional systems.

Particle filters update their prediction in an approximate (statistical) manner. The samples from the distribution are represented by a set of particles; each particle has a likelihood weight assigned to it that represents the probability of that particle being sampled from the probability density function. Weight disparity leading to weight collapse is a common issue encountered in these filtering algorithms; however it can be mitigated by including a resampling step before the weights become too uneven. Several adaptive resampling criteria can be used, including the variance of the weights and the relative entropy with respect to the uniform distribution. In the resampling step, the particles with negligible weights are replaced by new particles in the proximity of the particles with higher weights.

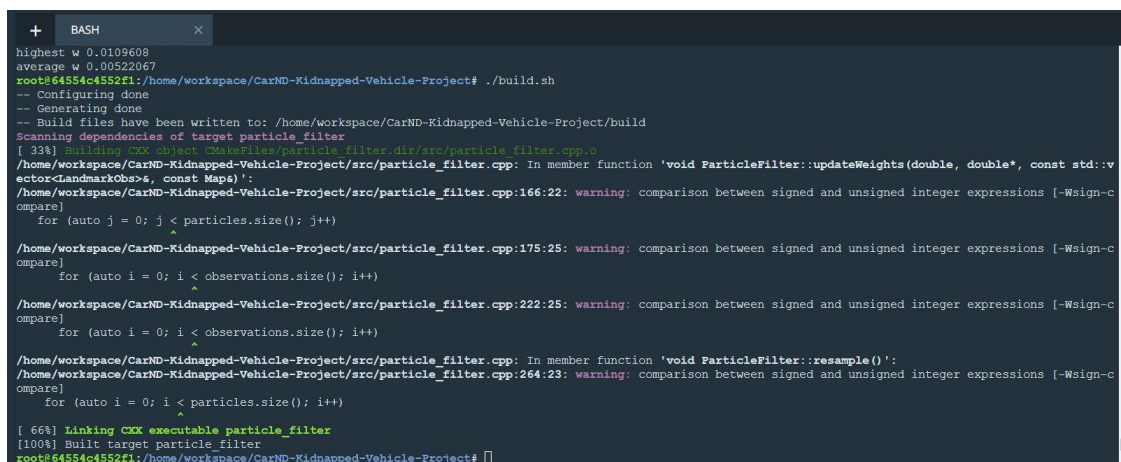
In general, a particle filter includes 4 steps:

1. Generating a set of particles
2. Measure the probability of each particle being the actual car(robot)
3. Resample based on the probability weight
4. Repeatedly move to approach orientation

## 5. Project Execution :

### a. Code Build :

Following are the snapshots from code compilation – Execution of *build* command:



```
+ BASH x
highest w 0.0109608
average w 0.00522067
root@64554c4552f1:/home/workspace/CarND-Kidnapped-Vehicle-Project# ./build.sh
-- Configuring done
-- Generating done
-- Build files have been written to: /home/workspace/CarND-Kidnapped-Vehicle-Project/build
Scanning dependencies of target particle_filter
[ 33%] Building CXX object CMakeFiles/particle_filter.dir/src/particle_filter.cpp.o
/home/workspace/CarND-Kidnapped-Vehicle-Project/src/particle_filter.cpp: In member function 'void ParticleFilter::updateWeights(double, double*, const std::vector<LandmarkObs>*, const Maps)':
/home/workspace/CarND-Kidnapped-Vehicle-Project/src/particle_filter.cpp:166:22: warning: comparison between signed and unsigned integer expressions [-Wsign-compare]
    for (auto j = 0; j < particles.size(); j++)
                        ^
/home/workspace/CarND-Kidnapped-Vehicle-Project/src/particle_filter.cpp:175:25: warning: comparison between signed and unsigned integer expressions [-Wsign-compare]
    for (auto i = 0; i < observations.size(); i++)
                        ^
/home/workspace/CarND-Kidnapped-Vehicle-Project/src/particle_filter.cpp:222:25: warning: comparison between signed and unsigned integer expressions [-Wsign-compare]
    for (auto i = 0; i < observations.size(); i++)
                        ^
/home/workspace/CarND-Kidnapped-Vehicle-Project/src/particle_filter.cpp: In member function 'void ParticleFilter::resample()':
/home/workspace/CarND-Kidnapped-Vehicle-Project/src/particle_filter.cpp:264:23: warning: comparison between signed and unsigned integer expressions [-Wsign-compare]
    for (auto i = 0; i < particles.size(); i++)
                        ^
[ 66%] Linking CXX executable particle_filter
[100%] Built target particle_filter
root@64554c4552f1:/home/workspace/CarND-Kidnapped-Vehicle-Project#
```

Figure 2: Build command execution - Successful

The build command executed without error but with few warnings.

Following is another snapshot with errors -

```
/home/workspace/CarND-Kidnapped-Vehicle-Project/src/particle_filter.cpp:258:32: error: expected primary-expression before '>' token
root8879cc81bc7d1:/home/workspace/CarND-Kidnapped-Vehicle-Project# ./build.sh
-- Configuring done
-- Generating done
-- Build files have been written to: /home/workspace/CarND-Kidnapped-Vehicle-Project/build
Scanning dependencies of target particle_filter
[ 33%] Building CXX object CMakeFiles/particle_filter.dir/src/particle_filter.cpp.o
/home/workspace/CarND-Kidnapped-Vehicle-Project/src/particle_filter.cpp: In member function 'void ParticleFilter::init(double, double, double, double*)':
/home/workspace/CarND-Kidnapped-Vehicle-Project/src/particle_filter.cpp:37:3: error: 'normal_distribution' was not declared in this scope
    normal_distribution<double> dist_x(x, std(0));
    ^
A
/home/workspace/CarND-Kidnapped-Vehicle-Project/src/particle_filter.cpp:37:3: note: suggested alternative:
In file included from /usr/include/c++/5/random:49:0,
                 from /usr/include/c++/5/bits/stl_algo.h:66,
                 from /usr/include/c++/5/algorithm:62,
                 from /home/workspace/CarND-Kidnapped-Vehicle-Project/src/particle_filter.cpp:11:
/usr/include/c++/5/bits/random.h:1920:11: note: 'std::normal_distribution'
```

Figure 3: Build command execution - build fail

Final Folder Structure –

```
/ > home > workspace >
CarND-Kidnapped-Vehicle-Project
├── build
├── data
├── src
├── .gitignore
├── CMakeLists.txt
├── LICENSE
├── README.md
├── build.sh
├── clean.sh
├── cmakepatch.txt
├── install-mac.sh
├── install-ubuntu.sh
├── run.sh
└── set_git.sh
```

Figure 4: Final Folder Structure

b. Output:

Following are the values which I observed after the code execution with different number of element configurations -

Particles	Error X	Error Y	Error Yaw	System Time	Output
4	7.921	22.624	0.103	13.54	Fail – x error is larger than max
8	1.036	5.057	0.017	21.226	Fail – Y error is larger than max
16	0.134	0.122	0.005	48.86	Success
32	0.124	0.116	0.004	48.88	Success
110	0.113	0.106	0.004	48.88	Success
128	0.113	0.107	0.004	48.88	Success
512	0.092	0.086	0.003	48.90	Success
1024	0.094	0.089	0.003	51.12	Success
2048	0.106	0.102	0.003	97.46	Success >> Time taken is too much
4096	0.107	0.101	0.003	103.92	Fail – Time ran out

All output snapshots are placed in output folder.

Following are few snapshots for Failures and Success –

a) Output with number of particles set to 4 -

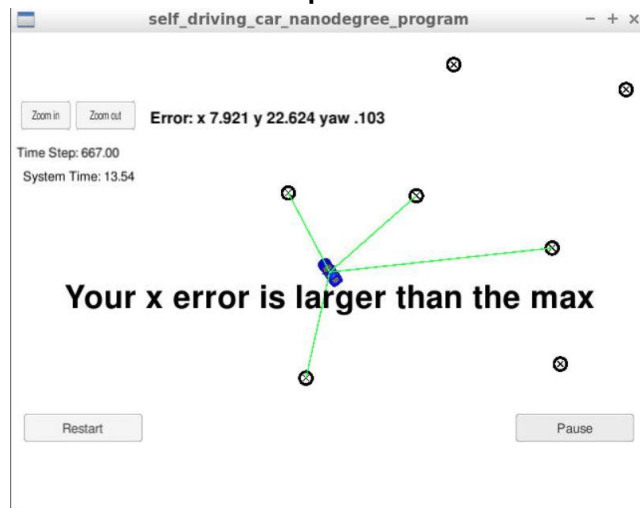


Figure 5: Output with number of Particles set to 4

b) Output with number of particles set to 512 –

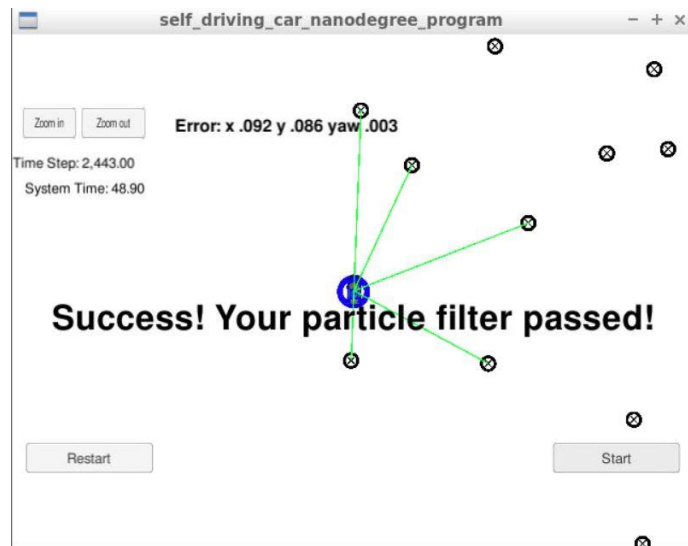


Figure 6: Output with number of Particles set to 512

c) Output with number of particles set to 4096 –

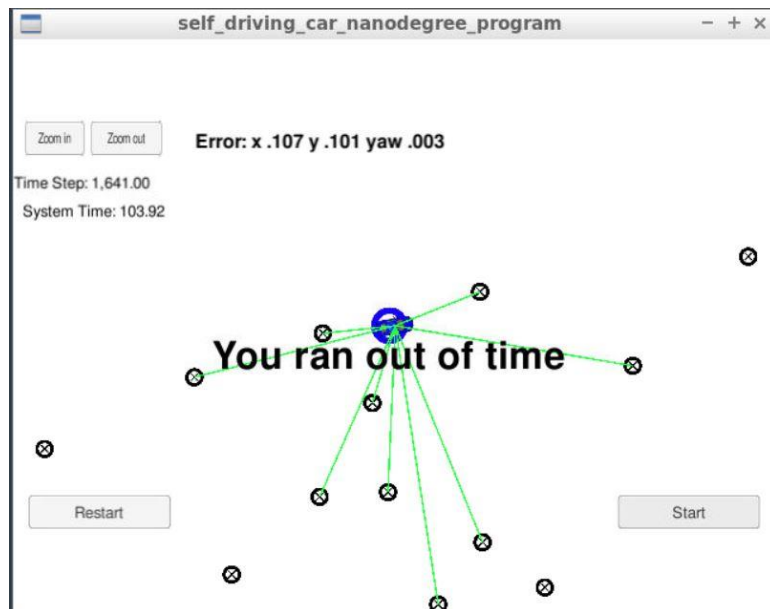


Figure 7: Output with number of Particles set to 4096

c. Conclusion:

*After analyzing the Particle filter behavior with multiple values from  $4(2^2)$  to  $4096(2^{12})$  it is observed that for the less number of particles lead to larger error of X, Y and Yaw. Whereas if we go for large number of particles then it results in lesser error but needs a large execution time. After evaluation all values I found that in current code the particle count of 512 is more suitable considering the error values and execution time.*

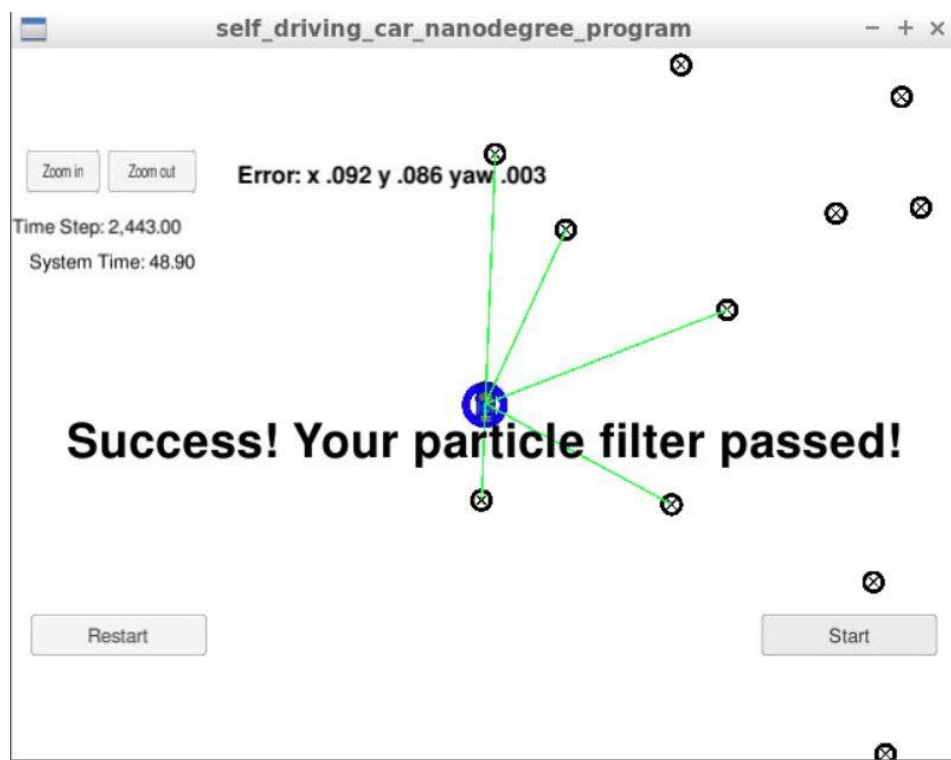


Figure 8: Optimized output with particles = 512