Kidnapped Vehicle project – Project 3

Overview

This project implements a Particle filter Algorithm in C++ and tested on Simulator provided by Udacity. The communication between the simulator and the EKF is done using WebSocket using the uWebSockets implementation on the EKF side. The Simulator also provides the end result of whether Particle filter algorithm was able to meet the project criteria

This project utilizes the starter code given by Udacity. Following are the outputs available as outcome of this project:

1. Vide0.mp4 – Contains recorded video of the output in the simulator
2. Cmake-output.log- Contains output of Cmake
3. make-output.log - contains output of make

Environment:

This project was done in Ubuntu 18 LTS and following are the components

- cmake >= 3.5
- make >= 4.1
- gcc/g++ >= 5.4

Ouput:

Below are the output of this project:

cmake output:

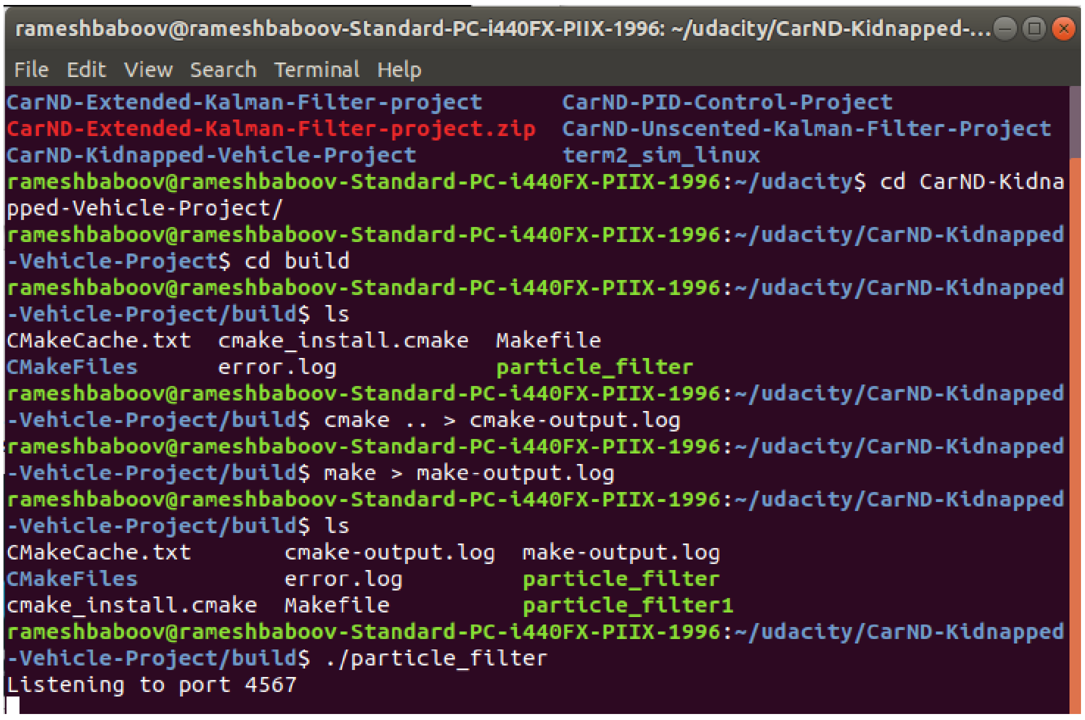
```
-- Configuring done
-- Generating done
-- Build files have been written to: /home/rameshbaboov/udacity/CarND-Kidnapped-Vehicle-Project/build
```

make output:

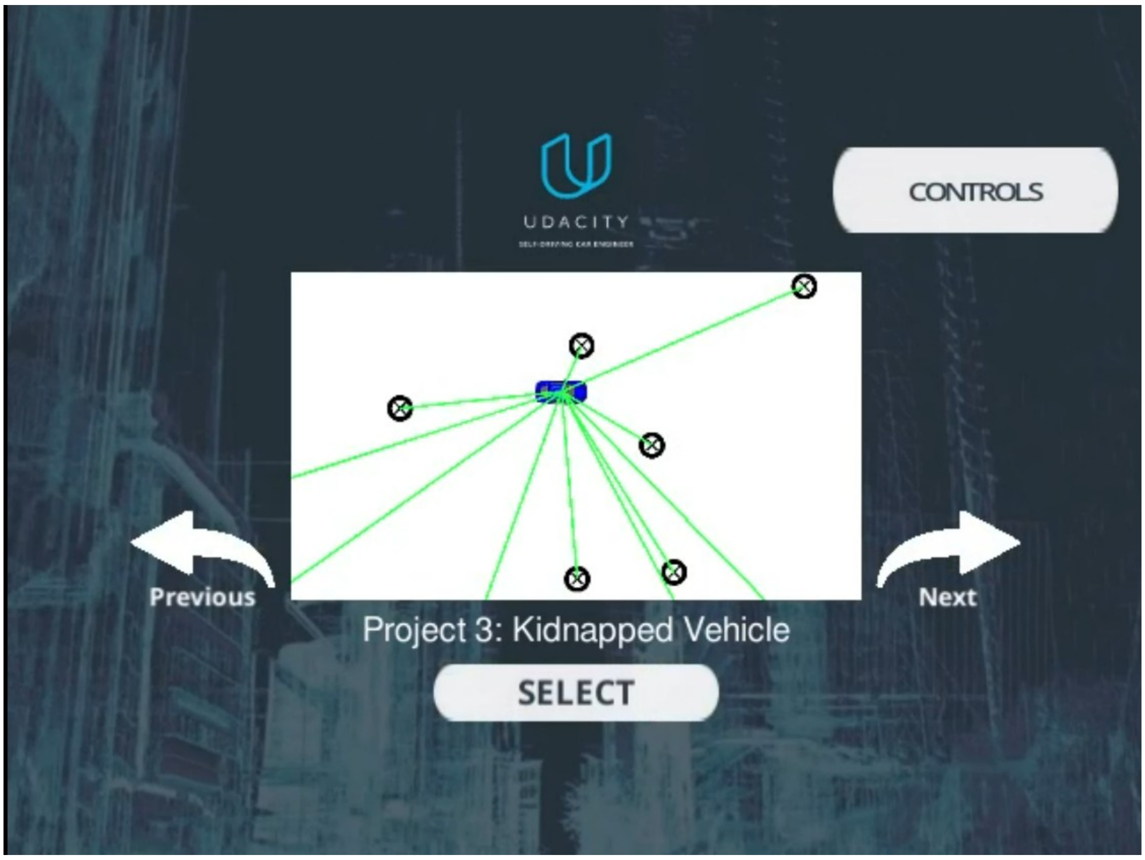
```
[ 33%] Linking CXX executable particle_filter
[100%] Built target particle_filter
```

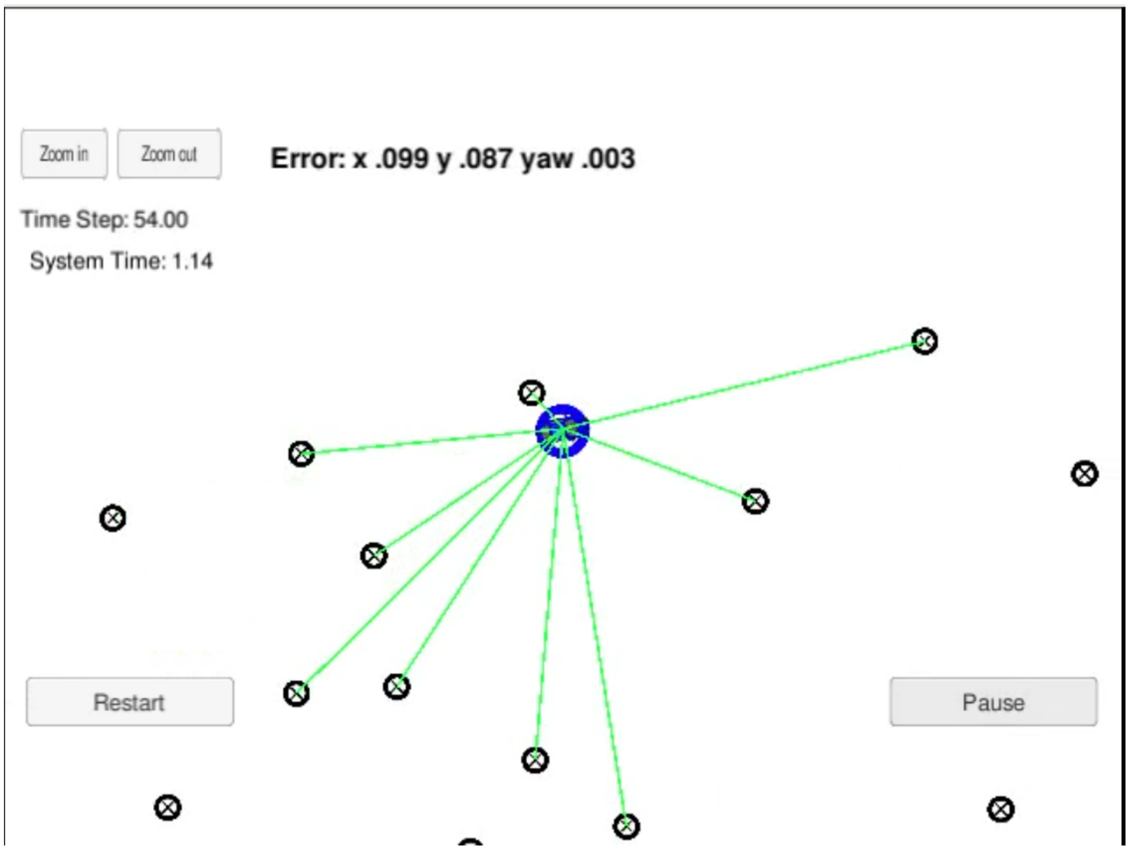
Simulator output:

Once the code is run following message is displayed. The application connects to the Simulator on port # 4567 , once the project-3 is seleted and started on simulator as shown below:


rameshbaboov@rameshbaboov-Standard-PC-i440FX-PIIX-1996: ~/udacity/CarND-Kidnapped-...
File Edit View Search Terminal Help

```
CarND-Extended-Kalman-Filter-project      CarND-PID-Control-Project
CarND-Extended-Kalman-Filter-project.zip  CarND-Unscented-Kalman-Filter-Project
CarND-Kidnapped-Vehicle-Project           term2_sim_linux
rameshbaboov@rameshbaboov-Standard-PC-i440FX-PIIX-1996:~/udacity$ cd CarND-Kidna
pped-Vehicle-Project/
rameshbaboov@rameshbaboov-Standard-PC-i440FX-PIIX-1996:~/udacity/CarND-Kidnapped
-Vehicle-Project$ cd build
rameshbaboov@rameshbaboov-Standard-PC-i440FX-PIIX-1996:~/udacity/CarND-Kidnapped
-Vehicle-Project/build$ ls
CMakeCache.txt  cmake_install.cmake  Makefile
CMakeFiles      error.log            particle_filter
rameshbaboov@rameshbaboov-Standard-PC-i440FX-PIIX-1996:~/udacity/CarND-Kidnapped
-Vehicle-Project/build$ cmake .. > cmake-output.log
rameshbaboov@rameshbaboov-Standard-PC-i440FX-PIIX-1996:~/udacity/CarND-Kidnapped
-Vehicle-Project/build$ make > make-output.log
rameshbaboov@rameshbaboov-Standard-PC-i440FX-PIIX-1996:~/udacity/CarND-Kidnapped
-Vehicle-Project/build$ ls
CMakeCache.txt       cmake-output.log  make-output.log
CMakeFiles           error.log         particle_filter
cmake_install.cmake  Makefile          particle_filter1
rameshbaboov@rameshbaboov-Standard-PC-i440FX-PIIX-1996:~/udacity/CarND-Kidnapped
-Vehicle-Project/build$ ./particle_filter
Listening to port 4567
```

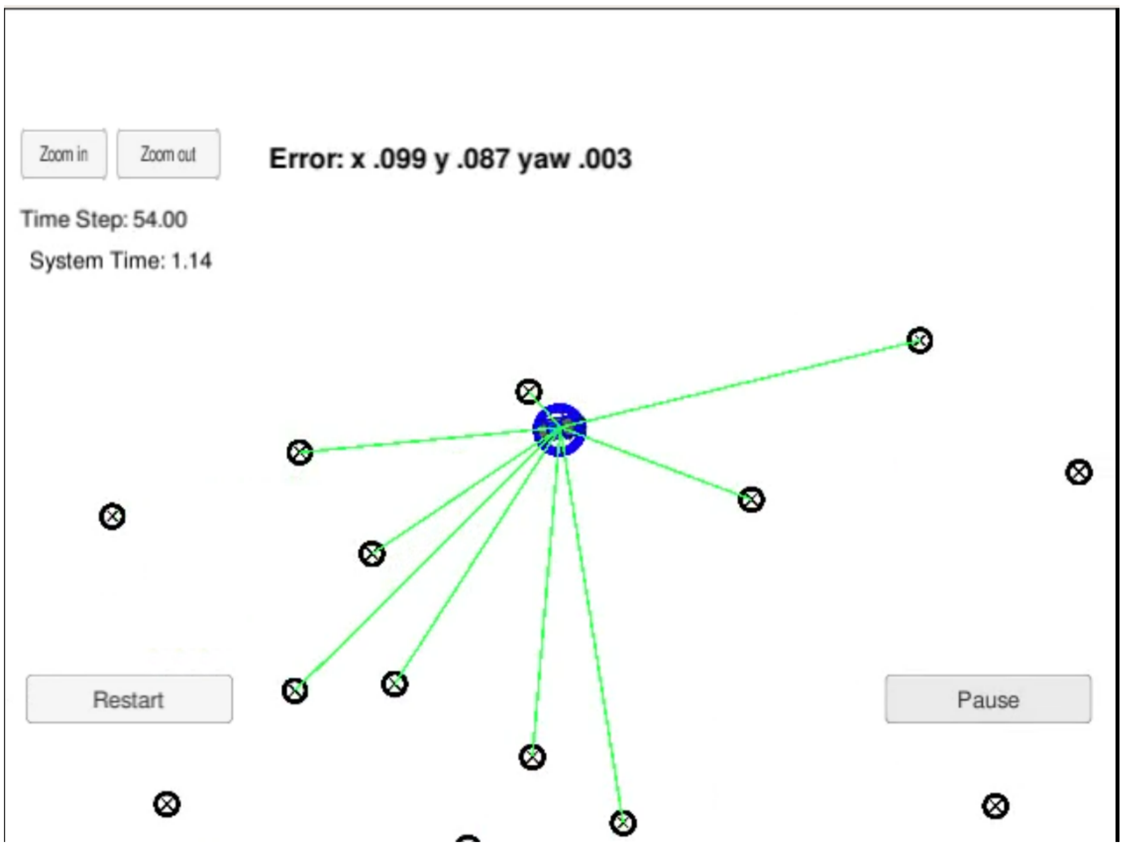




Error: x .099 y .087 yaw .003

Time Step: 54.00

System Time: 1.14


Error: x .099 y .087 yaw .003

Time Step: 54.00

System Time: 1.14

Code:

Particle_filter.cpp – I used the starter code provided by Udacity. Below are the functions of this code:

1. **ParticleFilter::init** - Initiates the no of particles to 100. add random gaussian noise to each of the particle for x, y and theta.
2. **ParticleFilter::prediction:** adds measurements to each particles and the random noise. The following equation is used:

   if yaw rate is zero:

$$x = x + v * \Delta t * \cos(\Theta)$$
$$y = y + v * \Delta t * \sin(\Theta)$$

   if yaw rate is not zero:

$$x_f = x_0 + \frac{v}{\dot{\theta}}[sin(\theta_0 + \dot{\theta}(dt)) - sin(\theta_0)]$$

$$y_f = y_0 + \frac{v}{\dot{\theta}}[cos(\theta_0) - cos(\theta_0 + \dot{\theta}(dt))]$$

$$\theta_f = \theta_0 + \dot{\theta}(dt)$$

3. **ParticleFilter::dataAssociation -** Finds the predicted measurement that is closest to each observed measurement and assign the observed measurement to this particular landmark. Called by ParticleFilter::updateWeights. This uses the dist function to calculate the distance and selects the shortest distance

```
double cur_dist = dist(obs.x, obs.y, pred.x, pred.y);
```

4. **ParticleFilter::updateWeights –** calculates the weights of each particle using a mult-variate Gaussian distribution. Also landmarks that are within the range are only considered because they only would hae been detected ny the partile

$$P(x,y) = \frac{1}{2\pi\sigma_x\sigma_y} e^{-\left(\frac{(x-\mu_x)^2}{2\sigma_x^2} + \frac{(y-\mu_y)^2}{2\sigma_y^2}\right)}$$

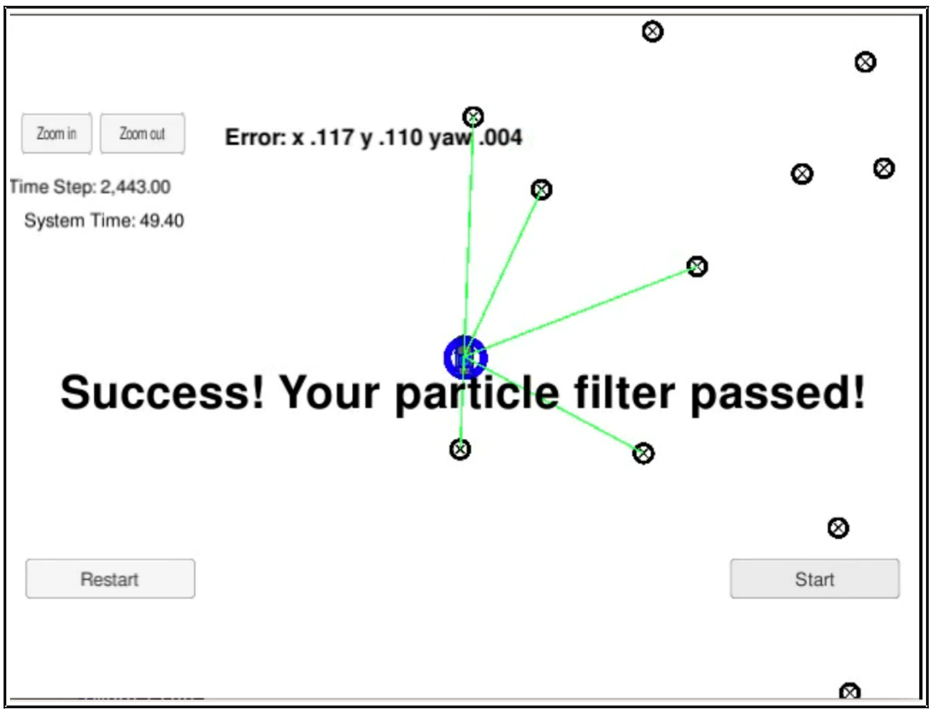
RUBRICS:

This project satisfies all the rubric points

Rubric#1: Accuracy

Criteria	Meets Specifications
Does your particle filter localize the vehicle to within the desired accuracy?	This criteria is checked automatically when you do ./run.sh in the terminal. If the output says "Success! Your particle filter passed!" then it means you've met this criteria.

Yes. The output says - "Success! Your particle filter passed!"



Error: x .117 y .110 yaw .004

Time Step: 2,443.00

System Time: 49.40

Success! Your particle filter passed!

Rubric#2: Performance

Criteria	Meets Specifications
Does your particle run within the specified time of 100 seconds?	This criteria is checked automatically when you do ./run.sh in the terminal. If the output says "Success! Your particle filter passed!" then it means you've met this criteria.

Yes. The output says - "Success! Your particle filter passed!". It runs within 50 seconds.

Rubric#3: General

Criteria	Meets Specifications
Does your code use a particle filter to localize the robot?	There may be ways to “beat” the automatic grader without actually implementing the full particle filter. You will meet this criteria if the methods you write in particle_filter.cpp behave as expected.

Particle filter works as expected