UDACITY

Return to Classroom

DISCUSS ON STUDENT HUB

Kidnapped Vehicle

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Nice work in implementing a 2 dimensional particle filter in C++, and applying it to locate a kidnapped vehicle, the vehicle's state is successfully recovered from uncertain control and measurement environment, given known map data.

Histogram Filter, Kalman Filter and Particle Filter are all Bayes filters and can be used to solve localization problem, they differ in the fact that different mathematical representation of system state distribution is assumed in each filter. Please read here for more information.

If you wish to dig deeper into localization, the Robust and Precise Vehicle Localization based on Multi-sensor Fusion in Diverse City Scenes paper published by Baidu Appolo team could be an interesting read. It fuses information from complementary sensors such as GNSS, Lidar, and IMU to achieve centimeter level accuracy in various challenging scenes including urban downtown, highways, and tunnels.

Keep up the great work and Happy learning!

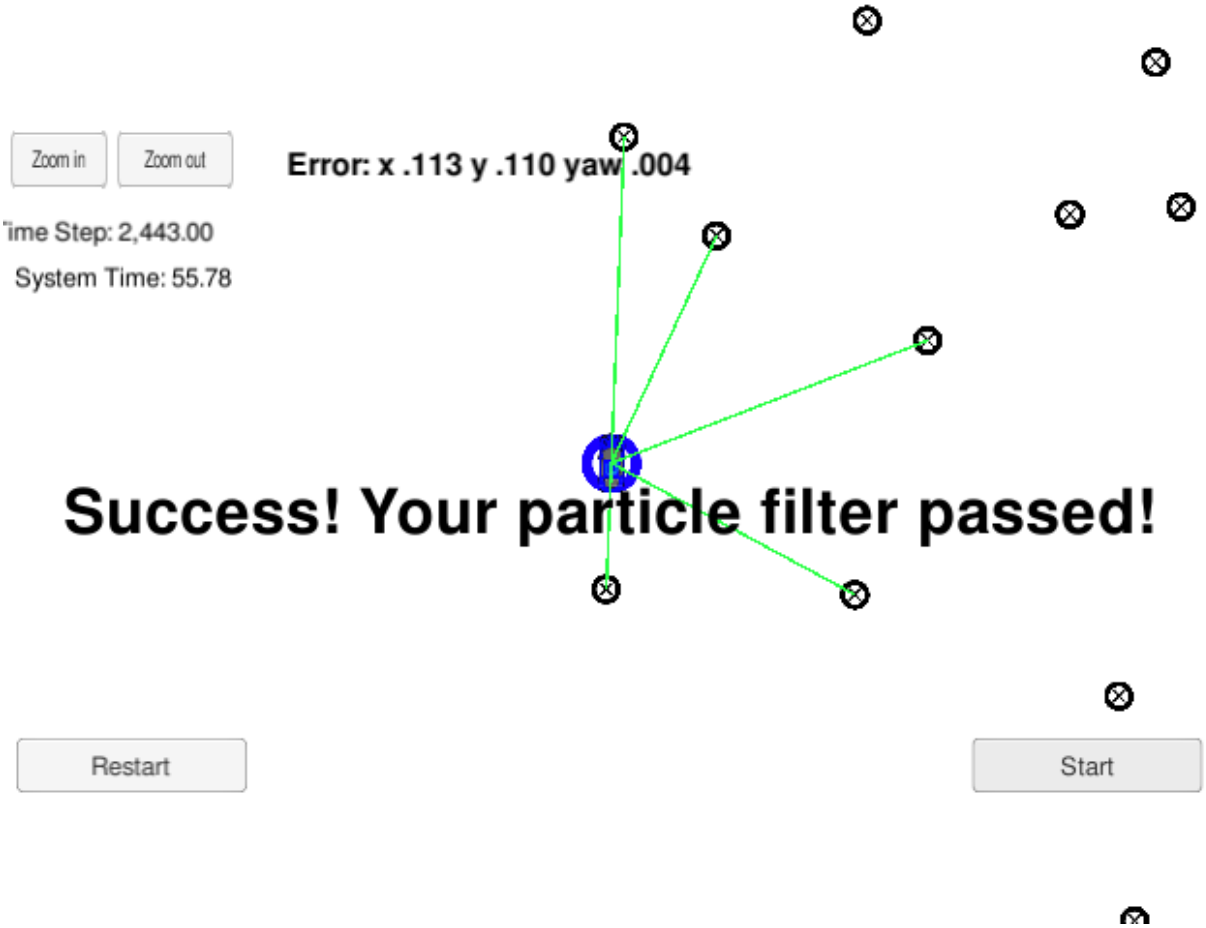
Accuracy

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.
After running ./run.sh in the terminal, the output says "Success! Your particle filter passed!". Well done!

Performance

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.

After running ./run.sh in the terminal, the output says "Success! Your particle filter passed!". Your implementation met performance criteria when running on my laptop.



General

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 is properly implemented. Good job!

Regarding the resampling step, an alternative approach is to use std::discrete_distribution, http://en.cppreference.com/w/cpp/numeric/random/discrete_distribution

```
std::random_device seed;
std::mt19937 random_generator(seed());
// sample particles based on their weight
```

```
std::discrete_distribution<> sample(weights.begin(), weights.end());

std::vector<Particle> new_particles(num_particles);
for(auto & p : new_particles)

p = particles[sample(random_generator)];
particles = std::move(new_particles);
```

DOWNLOAD PROJECT

RETURN TO PATH