

Kidnapped Vehicle

[审阅](#)[代码审阅](#)[HISTORY](#)

Meets Specifications

Dear student,

This is a very great attempt as I can see you put a lot of effort to accomplish this project. I enjoyed reviewing your project as your code is very well commented which makes the project easy to follow. I am sure that you also enjoyed working on the project.

- If you are interested to learn more about the topic, you might want to check [Sebastian's article](#) about the particle filter in robotics.

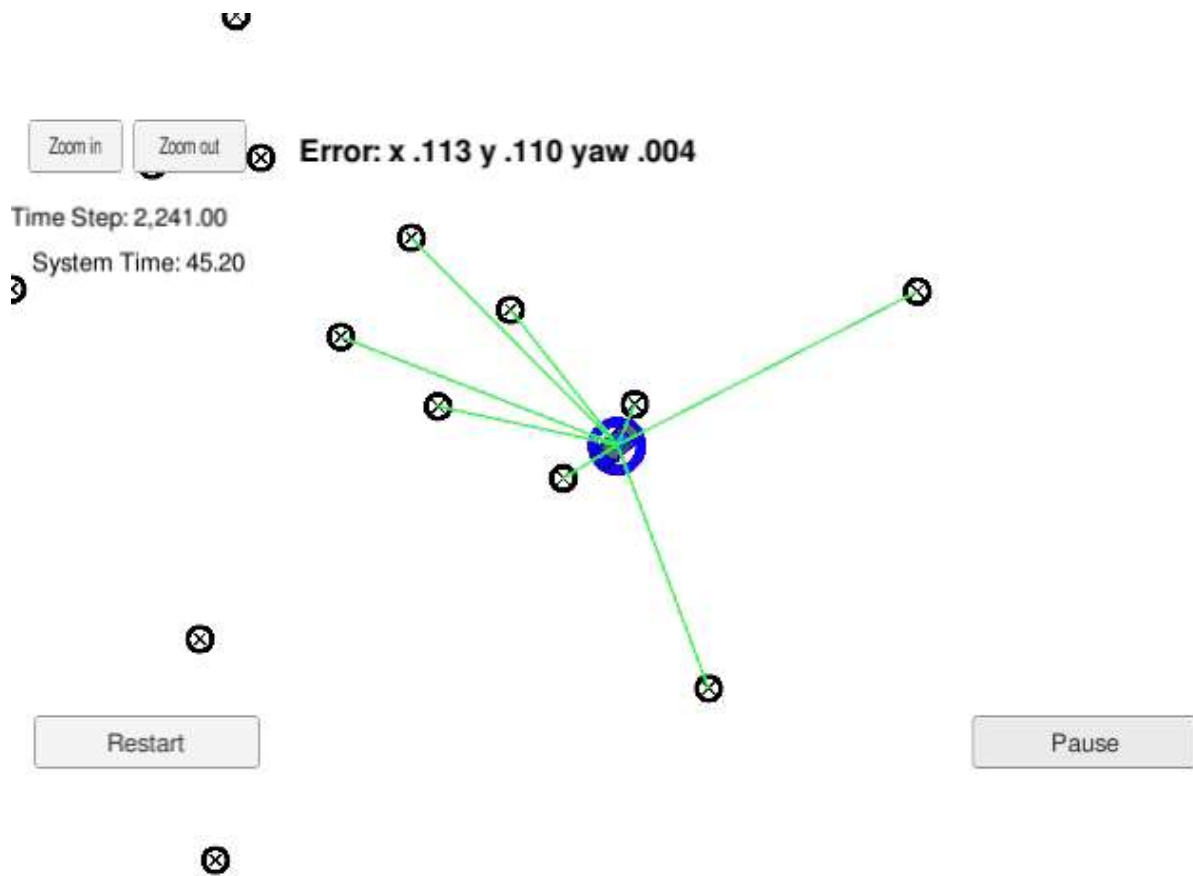
Congratulations 🎉 and keep up the good work for the next project!



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.

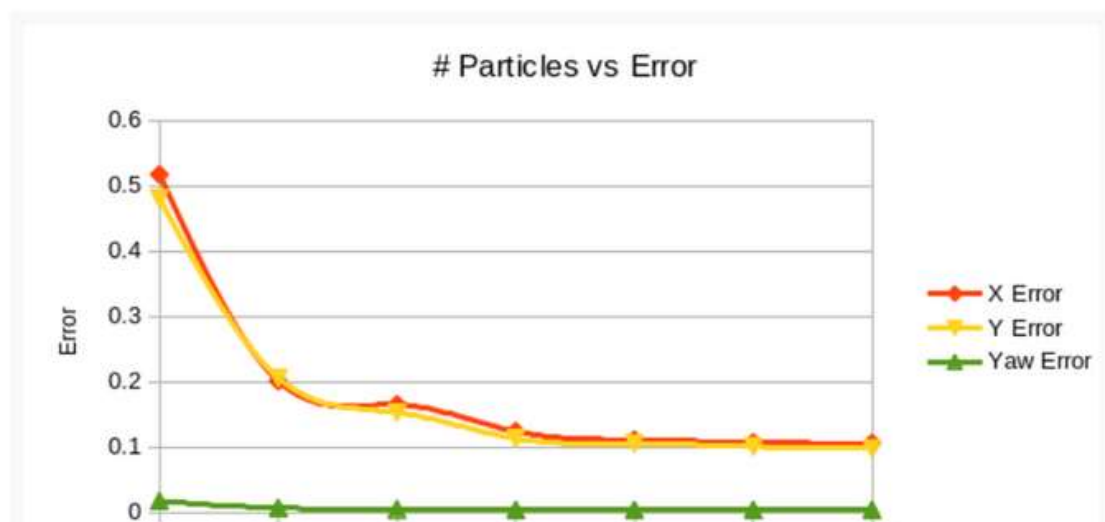
The particle filter implementation passes the grading code in the simulator. Awesome.



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.

- `num_particles = 100;`



2 5 10 50 100 500 1000

Particles

Great choice. It is neither too small which would be prone to error nor too high as that would be computationally expensive.

- ```
double tx = x + cos(theta) * observations[j].x - sin(theta) * observations[j].y;
double ty = y + sin(theta) * observations[j].x + cos(theta) * observations[j].y;
```

Nice work transforming the observations from car coordinates to map coordinates.

- ```
if(fabs(yaw_rate)<0.00001)
```

Good job using the linear model when the yaw rate is close to zero to avoid division by zero.

```
if (yaw_rate is close to zero)
    Use linear Model
else
    use the General Model
```

$$\dot{\theta} = 0$$

Yaw rate

$$x_f = x_0 + v(dt)(\cos(\theta_0))$$

Final x position Initial x position Velocity Time elapsed X-component of velocity

$$y_f = y_0 + v(dt)(\sin(\theta_0))$$

Final y position Initial y position Velocity Time elapsed Y-component of velocity

$$\theta_f = \theta_0$$

Final yaw Initial yaw

$$\dot{\theta} \neq 0$$

Yaw rate

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

Final x position Initial x position Velocity Yaw rate Time elapsed Initial yaw

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

Final y position Initial y position Velocity Yaw rate Time elapsed Initial yaw

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

Final yaw Initial yaw Yaw rate Time elapsed

I went through the rest of the code, and all the functions are properly implemented. Kudos!! 🙌

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.

The methods in `particle_filter.cpp` behave as expected. All the steps in particle filter(Initialization, Prediction, Update and Resample) is correctly implemented. The code is clear and logical. Nice work. 👍

Further Reading

- You may also check this [paper](#) which discusses the resampling methods that allow for increased speed of particle filter and require less memory.
- This [research paper](#) discusses object tracking in a video using particle filter which is an interesting read to get an idea of real world particle filter applications.

 [下载项目](#)

[返回 PATH](#)

给这次审阅打分

开始