Particle Filters

Sean Holschier z5308039 28/07/2021

A particle filter is a Monte Carlo method which addresses the Hidden Markov Model. Monte Carlo methods utilises randomness to determine a most probable answer. In this piece, I explore the way particle filters operates regarding object tracking and use an example implemented in Java to outline the algorithms used. The project can be found at: https://github.com/Sean-Hol/ParticleFilterProject1

A particle filter, in the context of object tracking, deduces the location of an object within an area by creating a set of possible particles, calculating the probability of each particle matching the location of the object and resampling new particles based on the weights of the current particles. Each 'step' of the object will be incorporated and used to narrow down possible particles.

The Hidden Markov Model

The Hidden Markov Model is the statistical model, which has a goal of identifying states $X_{0,} \cdots, X_{t,}$ when only the observation variables $Y_{0,} \cdots, Y_{t,}$ are known (Jurafsky, 2020). A particle filter attempts to solve this problem, while also accounting from some noise that will be present in any sense data, as technology is not always perfect.

Particle filter Overview

At the beginning of the operation, particles are spread out throughout the possible locations of the object on a known map. This can be done randomly, uniformly spread out. In an article by Dhanoop Karunakaran, where he explores object tracking in a situation where an autonomous vehicle has been 'kidnapped' and a GPS estimate is available, the particles can be normally distributed with the GPS estimate used as the mean (Karunakaran, 2018).

In the Java example that I will reference in this report, the object being tracked will return the distance to the closest 'landmark' from the object. The sensor is assumed to be noisy and not perfect. The object will be placed somewhere randomly on the map and its location to the closest landmark and its movement direction and distance for each step is given.

Particle filter Cycle

Once initialised, the particle filter will begin the process of weighting, resampling, moving, and incorporating noise (Jurić, 2015) until most particles converge on one spot, determining the most

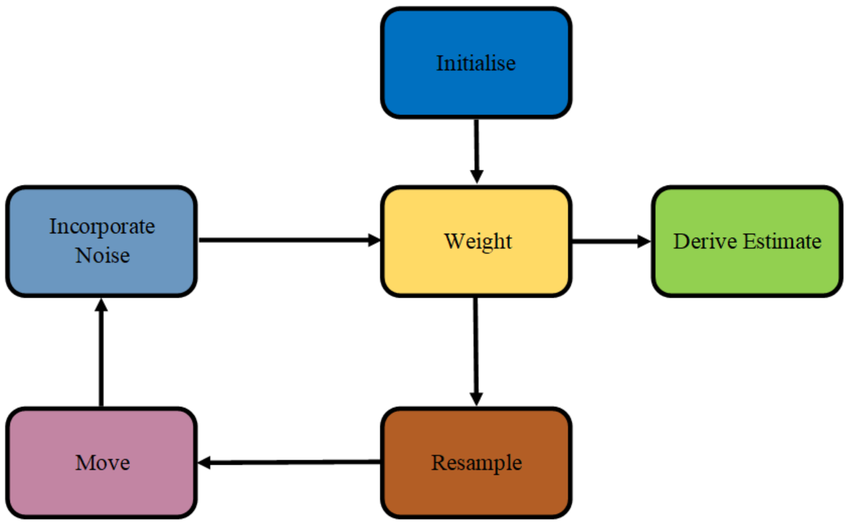
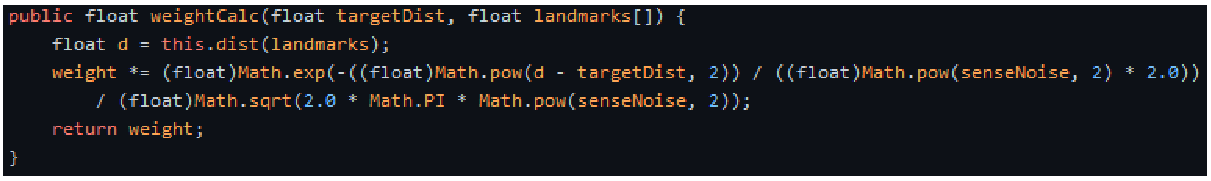


Figure 1. Particle Filter Cycle

probable location of the object. In the project, I used a 95% confidence interval to determine whether or not the filter was successful.

- **Weighting**

All the particles are given a weight, which will be the probability that the particle has a matching location and orientation. These weights will be derived from the observation variable from the actual object at that state in time.



```
public float weightCalc(float targetDist, float landmarks[]) {
    float d = this.dist(landmarks);
    weight *= (float)Math.exp(-((float)Math.pow(d - targetDist, 2)) / ((float)Math.pow(senseNoise, 2) * 2.0))
        / (float)Math.sqrt(2.0 * Math.PI * Math.pow(senseNoise, 2));
    return weight;
}
```

Figure 2. Weighting method for the project

$$g(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2} \frac{(x-\mu)^2}{\sigma^2} \right)$$

Figure 3. Gaussian Probability Density Formula

The Gaussian Probability Density Formula was used, with the actual distance of the object to the closest landmark as (mu) μ, and the sensor noise used as the standard deviation (σ). By substituting x with the distance between the particle and the closest beacon, the probability that the particle matches the target object can be calculated.

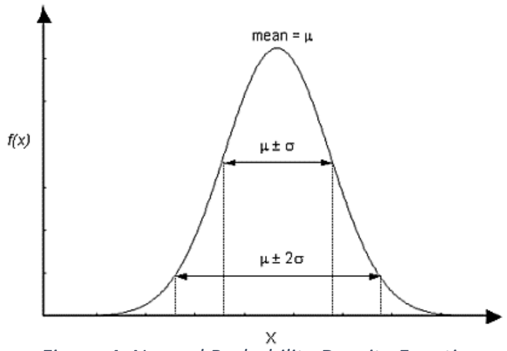


Figure 4. Normal Probability Density Function (rocscience, 2021)

Rather than resetting the weight for each step, I found that the filter was more accurate when the new weighting for a step was multiplied onto previous steps. This would be because the particles do not only weigh based on the current step, but also the previous steps. This saw a decrease in the number of steps needed to narrow down ‘particle clouds’, as the previous steps of the particles becomes a larger factor in weight calculation. However, to confirm these suspicions, further tests may need to be carried out as this may be creating bias as the previous weighting are taken into consideration twice, in both the reweighting and resampling.

$\text{ParticleProbability}_{\text{Step } 0\sim t} = \text{ParticleProbability}_{\text{Step } 0} * \text{ParticleProbability}_{\text{Step } 1} * \ldots * \text{ParticleProbability}_{\text{Step } t}$

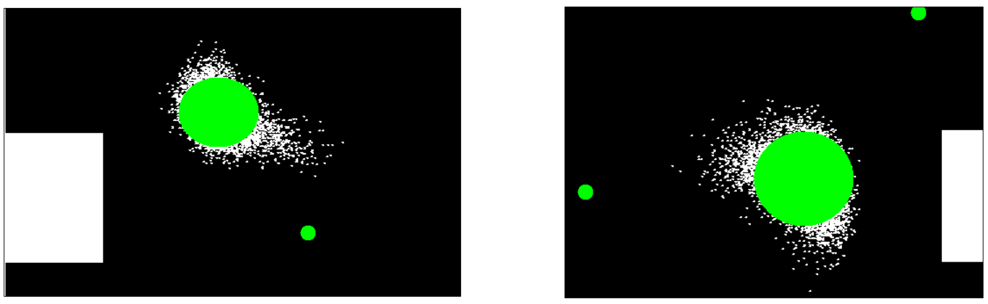


Figure 5. 95% confidence interval for particles with probability inheritance (left) and without inheritance (right)

Another observation seen was how much the std deviation depended on the movement of the final step. Larger movements are more helpful in the beginning when attempting to narrow down the particles to one area or spot, as the change in closest landmark would be a large hint as to which ‘particle cloud’ is around the target object. However, A larger distance would have a bigger effect on the turn noise distribution, spreading the particles out more. This creates a larger standard deviation compared to when the final step does not feature a large movement. In the following graph, the standard deviations and movement sizes for the last 5 steps of 5 different particle filter scenarios have been graphed. For all 5 scenarios, a slight but apparent upwards trend can be seen.

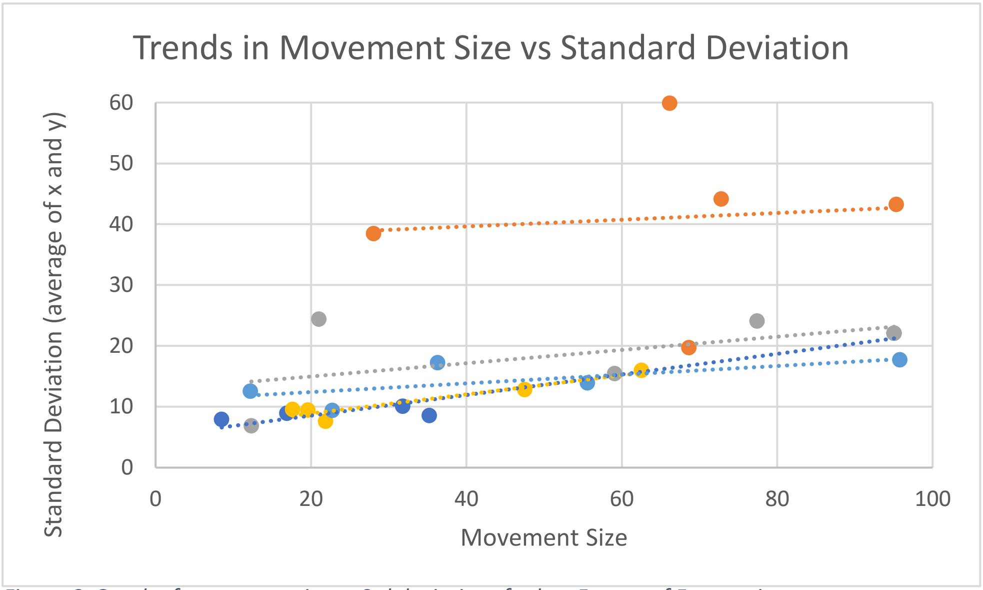
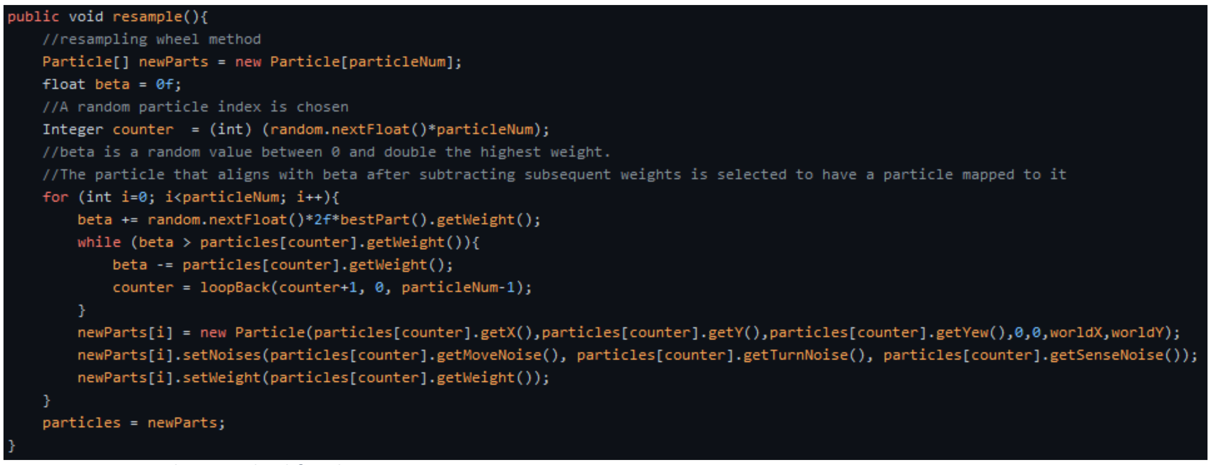


Figure 6. Graph of movement size vs Std deviations for last 5 steps of 5 scenarios

- **Resampling**

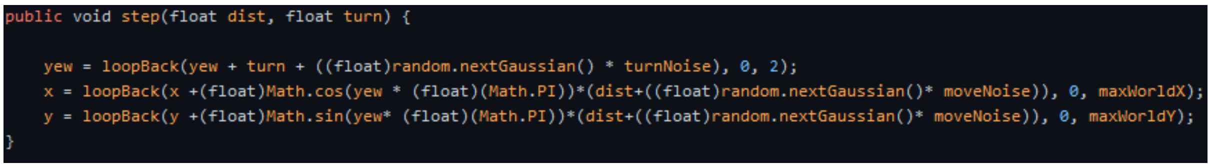
Resampling is the task of selecting a new set of particles X_t based on the weightings of the previous particles X_{t-1}. There are many forms of resampling, but here a method called the ‘resampling wheel method’ (Madrigal, 2012), a type of multinomial resampling, is utilised (Fernández-Madrigal, 2013). A random index value within the particle array is chosen, where the resampling will begin. From there, for each particle a beta value, which will be a random value between 0 and double the maximum weight, is selected, and the method will iterate through the particle array starting at the index value. Weights are subtracted from beta until a weight value becomes larger than beta. The particle that weight belongs to is then chosen to have a particle to success it. The particle and all its attributes are then mapped onto the new set of particles.


```java
public void resample(){
    //resampling wheel method
    Particle[] newParts = new Particle[particleNum];
    float beta = 0f;
    //A random particle index is chosen
    Integer counter  = (int) (random.nextFloat()*particleNum);
    //beta is a random value between 0 and double the highest weight.
    //The particle that aligns with beta after subtracting subsequent weights is selected to have a particle mapped to it
    for (int i=0; i<particleNum; i++){
        beta += random.nextFloat()*2f*bestPart().getWeight();
        while (beta > particles[counter].getWeight()){
            beta -= particles[counter].getWeight();
            counter = loopBack(counter+1, 0, particleNum-1);
        }
        newParts[i] = new Particle(particles[counter].getX(),particles[counter].getY(),particles[counter].getYew(),0,0,worldX,worldY);
        newParts[i].setNoises(particles[counter].getMoveNoise(), particles[counter].getTurnNoise(), particles[counter].getSenseNoise());
        newParts[i].setWeight(particles[counter].getWeight());
    }
    particles = newParts;
}
```
Figure 7. Resampling method for the project

This method allows for a factor of randomisation in the resampling, but also gives particles with larger weightings a larger chance of being chosen.

- **Move**

Once the particles have been resampled, all the particles are moved in the same manner that the object moves. By moving all the particles in a similar manner to the object that is being tracked, the particles can be given new weightings to further narrow down the possible particles



```
public void step(float dist, float turn) {

    yew = loopBack(yew + turn + ((float)random.nextGaussian() * turnNoise), 0, 2);
    x = loopBack(x +(float)Math.cos(yew * (float)(Math.PI))*(dist+((float)random.nextGaussian()* moveNoise)), 0, maxWorldX);
    y = loopBack(y +(float)Math.sin(yew* (float)(Math.PI))*(dist+((float)random.nextGaussian()* moveNoise)), 0, maxWorldY);
}
```

Figure 8. Step method for the project

- **Incorporating Noise**

To compensate for the inaccuracy of the sensor data and the movement variables, noise is added to each particle individually so that each particle will receive a differing weight and determine more accurate. As seen in the move step above, the noise in incorporated in the moving process in the project.

Particle Filter Operation Example

Frame: 0

In the initial frame, the particles are randomly distributed, with the red dot signifying the target object. All particles that are in white zones will automatically be given a weight of 0.0, eliminating any possibility of surviving the resampling.

Frame 1

After 1 step, the particles have created rings around their closest particle, as only data from one step have been taken into consideration

Frame 2

As a second distance is taken into consideration, only a smaller part of the rings survived, as the extra data point narrows down the possible areas.

Frame 3

Since the object has now changed which landmark it is closest to, the particles have greatly narrowed down possible locations.

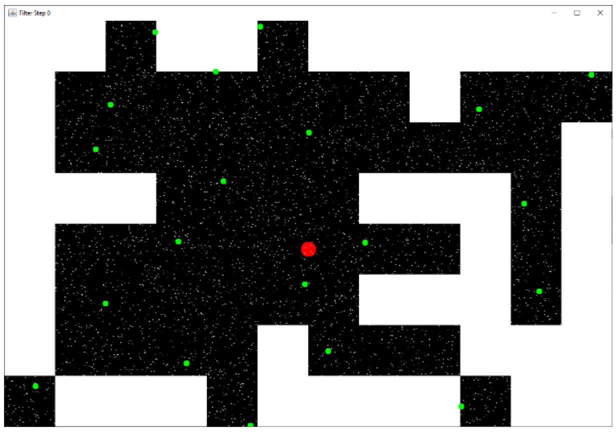


Figure 9. Frame 0

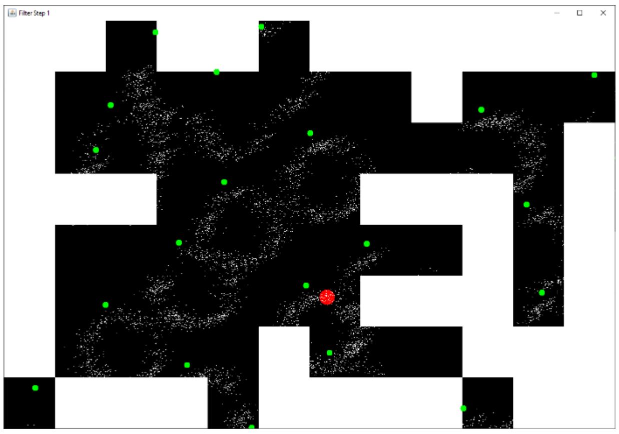


Figure 10. Frame 1

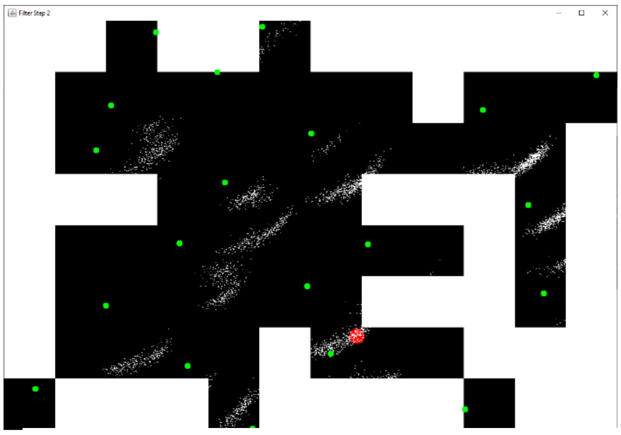


Figure 11. Frame 2

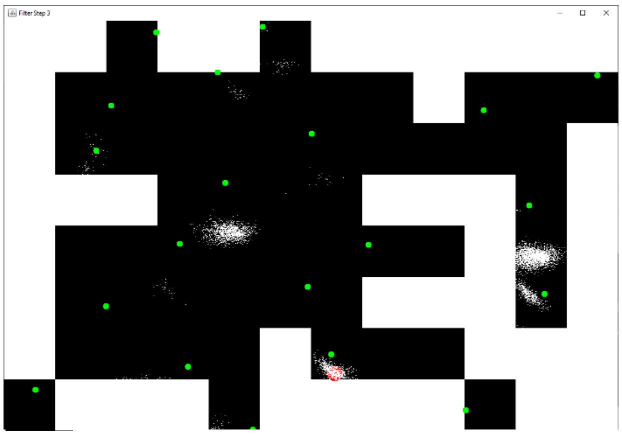


Figure 12. Frame 3

Frame 5

There are only a few locations where the particles cluster, with a majority over the actual object.

Frame 10

By frame 10, nearly all other clusters have been eliminated, with the large cluster spreading out more the larger movements the object takes.

Frame 15

At frame 15 the object location has been identified and it is just a matter of following the object accurately.

Frame 20

After the final step, the mean and standard deviations for x and y for all the particles have been calculated. With the final object coordinates being (644.51263, 592.87396), the means (639.3203, 590.5263) are very accurate. The particles had a standard deviation of (12.671295, 9.80258) for x and y respectively. (green oval signifies the 85% confidence interval – it would be orange if the object was not within it)

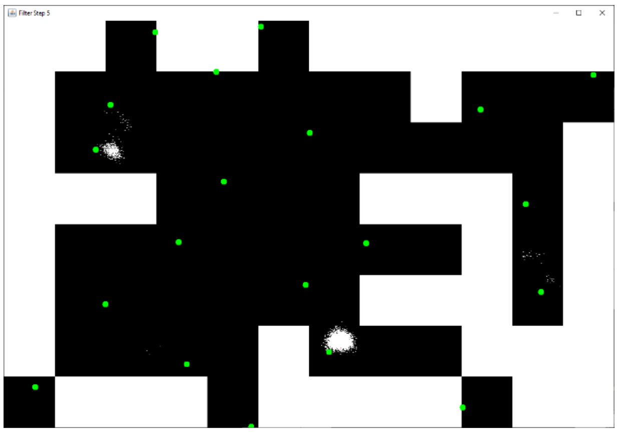


Figure 13. Frame 5

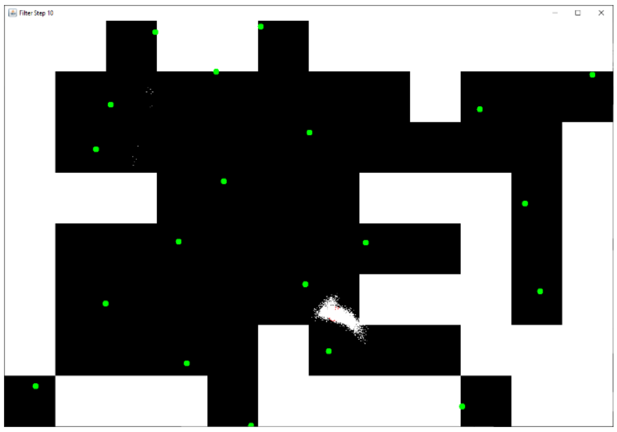


Figure 16. Frame 10

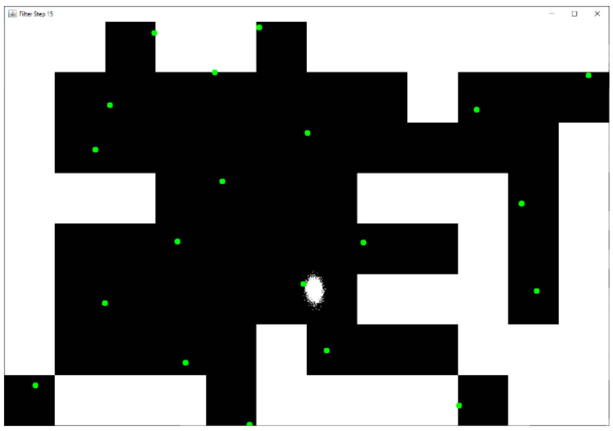


Figure 14. Frame 15

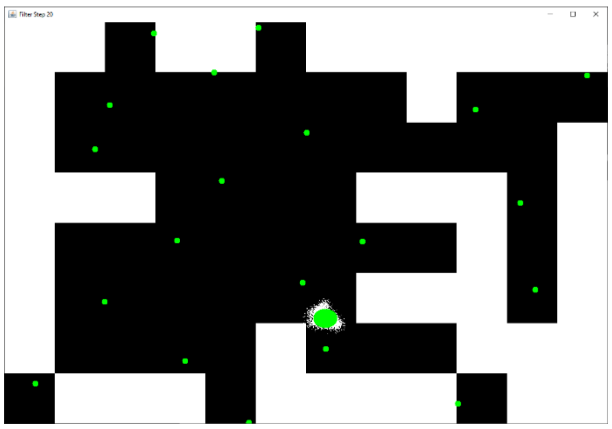


Figure 15. Frame 20

Summary

The particle filter was largely successful and would be viable for tracking objects when only vector and distance to closest landmarks are known. However, the algorithms would need to be refined to increase accuracy, with more research into whether having inherited weights do create bias in the filter.

Bibliography

Buchner, J. (2019, 2). *johnhw/pfilter*. Retrieved from Github: https://github.com/johnhw/pfilter erhs-53-hackers. (2013, 4). *erhs-53-hackers /Particle-Filter*. Retrieved from Github: https://github.com/erhs-53-hackers/Particle-Filter

Fernández-Madrigal, J.-A. (2013). Appendix B: Resampling Algorithms. In J.-A. Fernández-Madrigal, *Simultaneous localization and mapping for mobile robots introduction and methods* (pp. 407-411). Hershey, PA : Information Science Reference.

Jurafsky, D. M. (2020). Hidden Markov Models. In D. M. Jurafsky, *Speech and Language Processing.*

Jurić, D. (2015, 4). *Object Tracking: Particle Filter with Ease*. Retrieved from CodeProject: https://www.codeproject.com/Articles/865934/Object-Tracking-Particle-Filter-with-Ease

Karunakaran, D. (2018, March 14). *Kidnapped vehicle project using Particle Filters-Udacity's Self-driving Car Nanodegree*. Retrieved from Medium: https://medium.com/intro-to-artificial-intelligence/kidnapped-vehicle-project-using-particle-filters-udacitys-self-driving-car-nanodegree-aa1d37c40d49

Kenton, W. (2021, 7). *Monte Carlo Simulation*. Retrieved from Investopedia: https://www.investopedia.com/terms/m/montecarlosimulation.asp

Madrigal, C. (2012, 06). *resampling-wheel-algorithm*. Retrieved from calebmadrigal.com: https://calebmadrigal.com/resampling-wheel-algorithm/

rocscience. (2021, 07). *Normal Distribution in an RS2 Probabilistic Analysis*. Retrieved from RocScience.com: https://www.rocscience.com/help/rs2/phase2_model/normal_distribution_02.htm

Srinivasan, S. (2019, 8). *Particle Filter : A hero in the world of Non-Linearity and Non-Gaussian*. Retrieved from TowardsDataScience: https://towardsdatascience.com/particle-filter-a-hero-in-the-world-of-non-linearity-and-non-gaussian-6d8947f4a3dc