# Particle Filters

10 April 2018

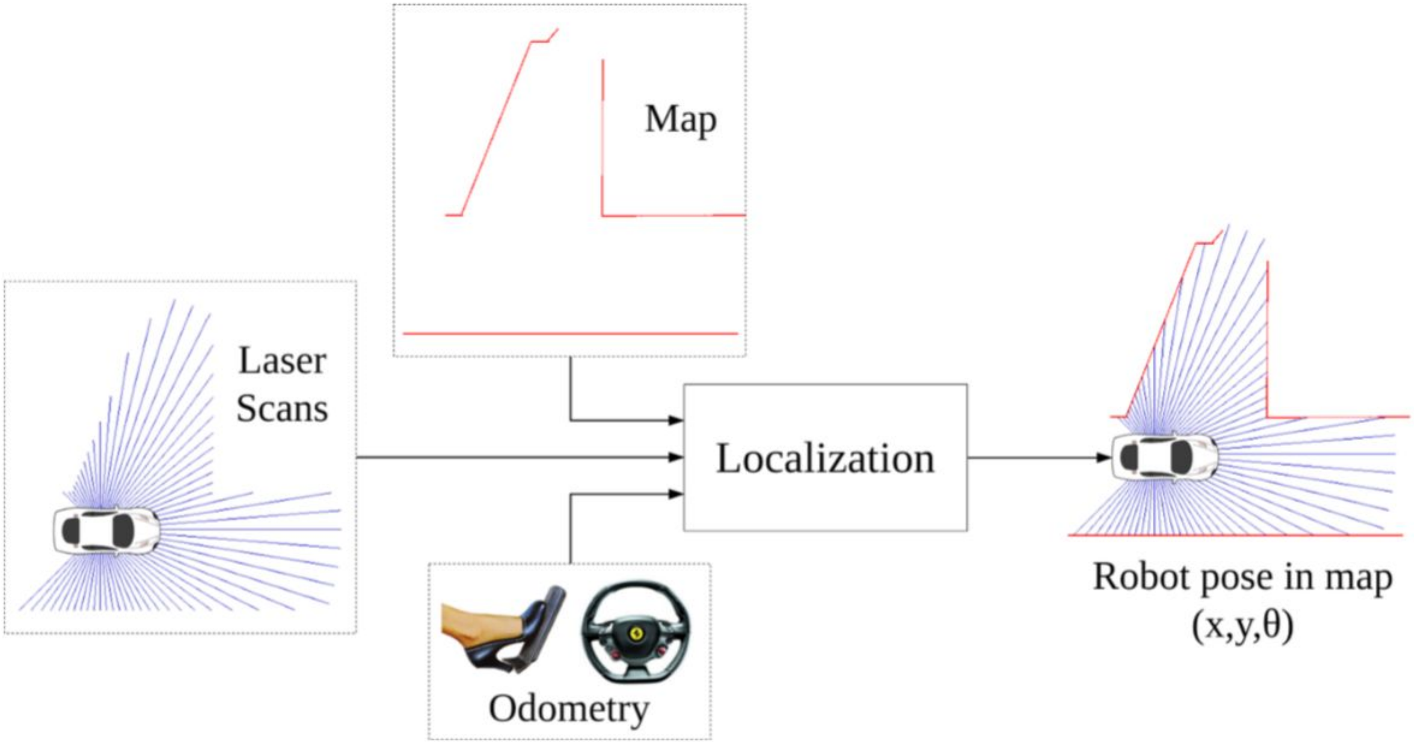
# Outline

- What is it?
- Real world examples
- Key Concepts
- The PF Algorithm
- PF Algorithm Review Videos
- Resampling Algorithms
- Compare PF with other Probabilistic Filters

# Outline

- **What is it?**
- Real world examples
- Key Concepts
- The PF Algorithm
- PF Algorithm Review
- Resampling Algorithms
- Compare PF with other Probabilistic Filters

# What is it?



Map
Laser
Scans
Localization
Odometry
Robot pose in map
(x,y,θ)

# What is it?

A probabilistic algorithm that uses

- **"particles"**
- **"importance/weights"**, *and*
- **"resampling"**

to estimate the current state/location of a moving object.

# Outline

- What is it?
- **Real world examples**
- Key Concepts
- The PF Algorithm
- PF Algorithm Review
- Resampling Algorithms
- Compare PF with other Probabilistic Filters

# Real World Examples

•

**Figure 2**: Particle filters have been used successfully for on-board localization of soccer-playing Aibo robots with as few as 50 particles [26].

# Real World Examples

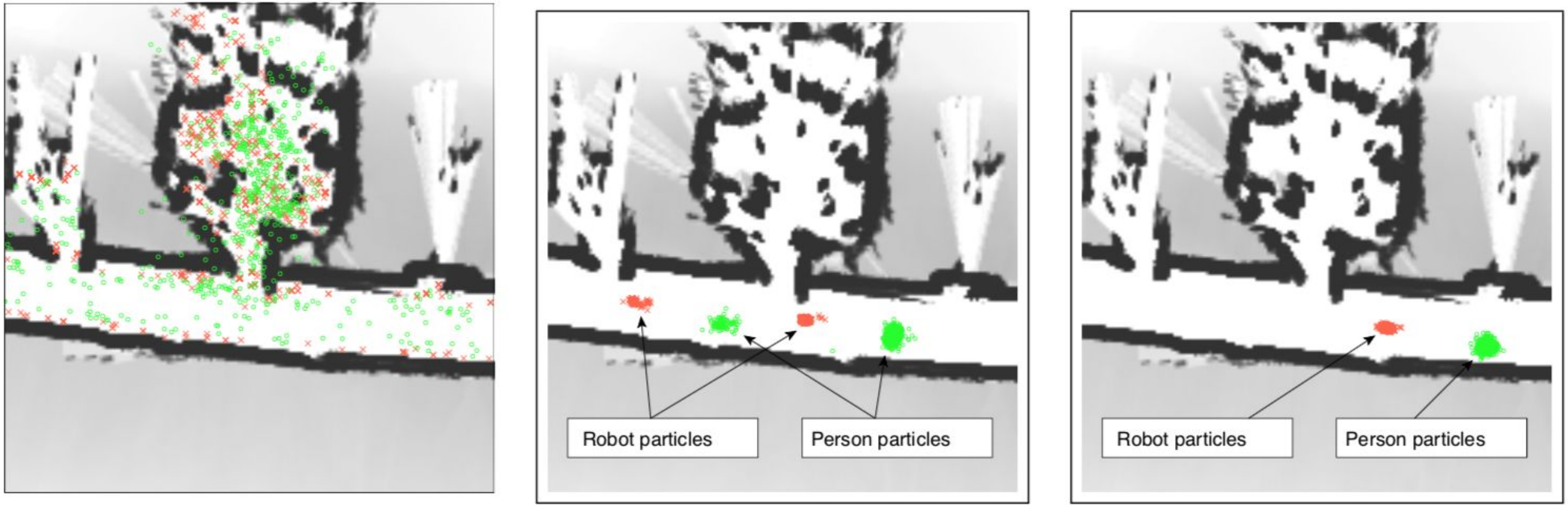



**Figure 7**: Particle filter-based people tracker: This algorithm uses maps to simultaneously localize a moving robot and an unknown number of nearby people. This sequence shows the evolution of the conditional particle filter from global uncertainty to successful localization and tracking of the robot.

# Real World Examples

•

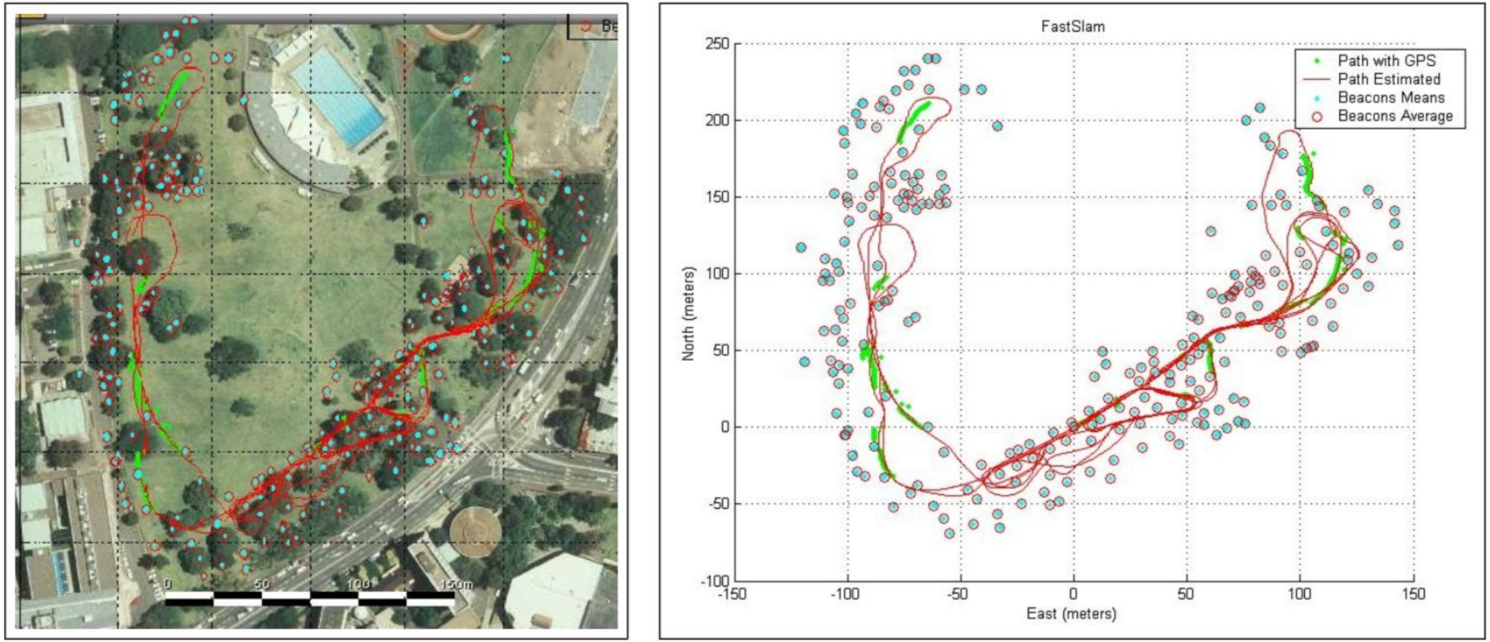



**Figure 5**: FastSLAM with real-world data: Shown here is a map of an outdoor environment (Victoria Park in Sydney), along with GPS information displayed here only for evaluating the accuracy of the resulting map. The resulting map error is extremely small, comparable in magnitude to the EKF solution. These results were obtained Juan Nieto, Eduardo Nebot, and Jose Guivant from the Australian Center of Field Robotics in Sydney, and are reprinted here with permission.

# Real World Examples

Autonomous RC-Car MIT Video
Autonomous RC-Car UPenn Video

# Outline

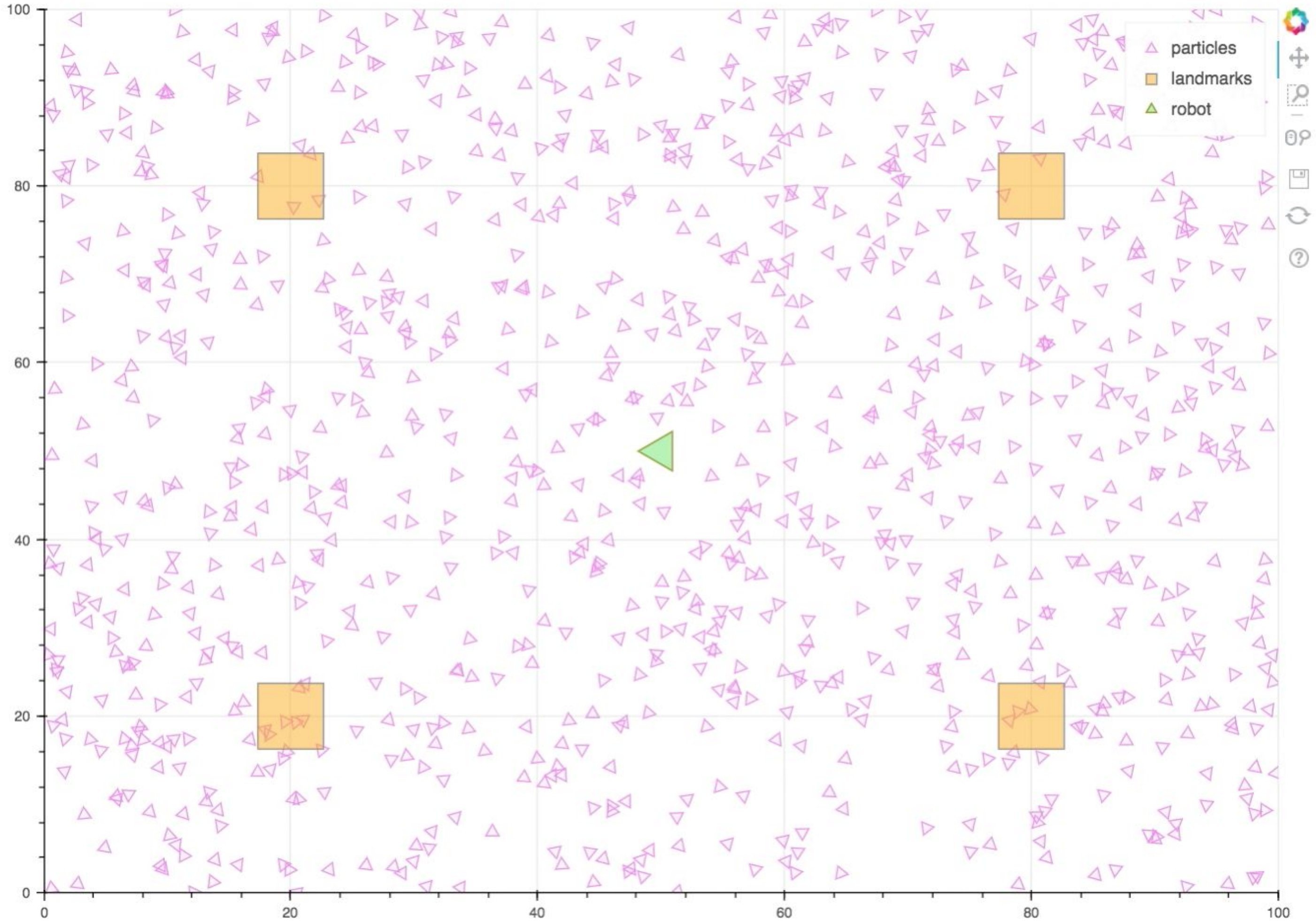
- What is it?
- Real world examples
- **Key Concepts**
- The PF Algorithm
- PF Algorithm Review
- Resampling Algorithms
- Compare PF with other Probabilistic Filters

# Key Concepts (1 / 4)

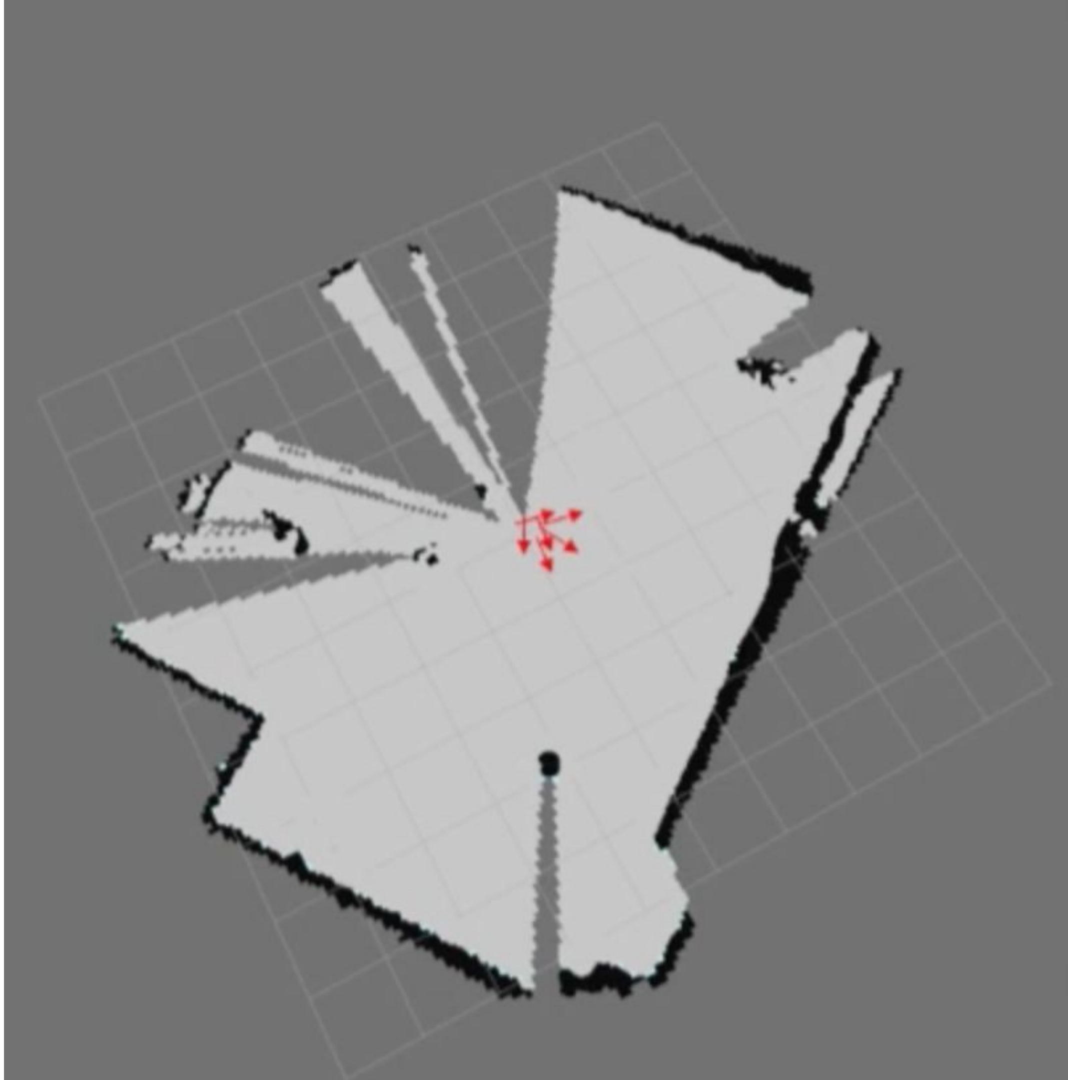
- Each particle is a possible state of a system

w 0

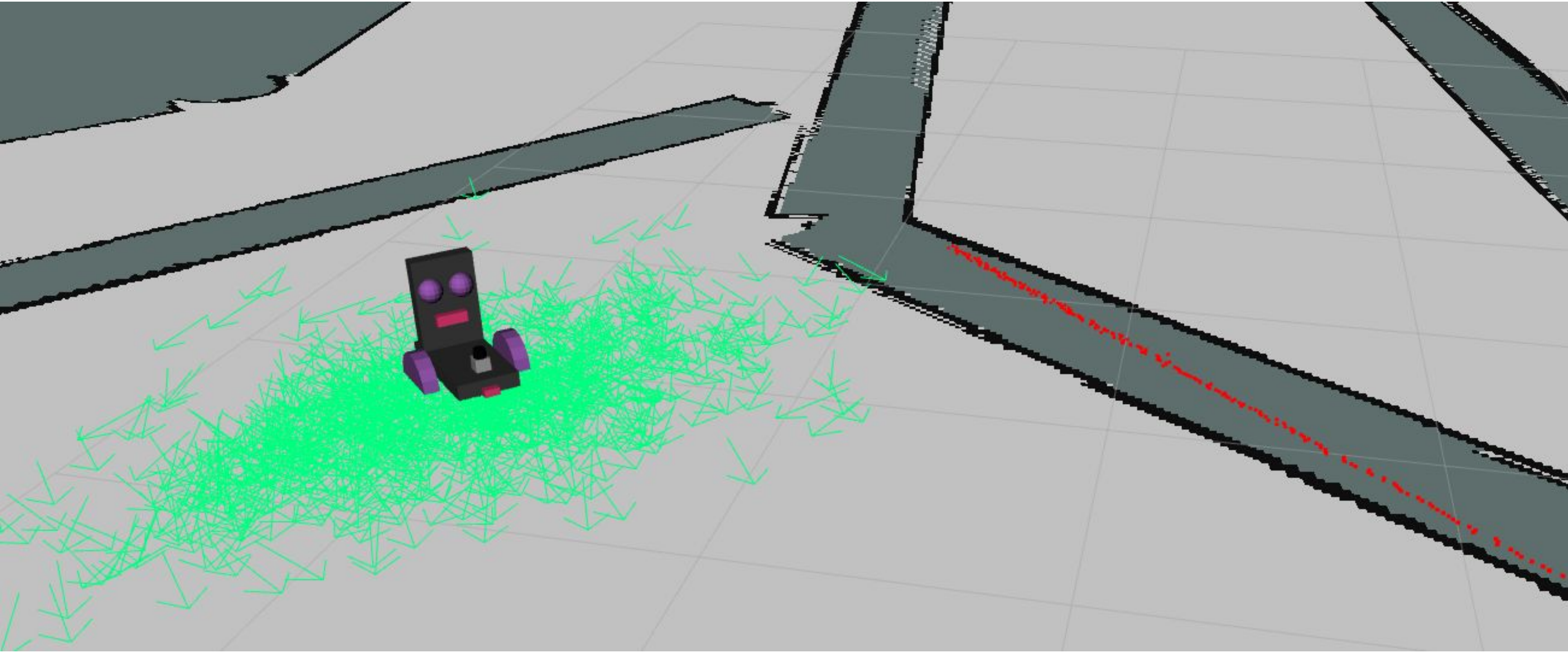
(initial)



# Key Concepts (1 / 4)



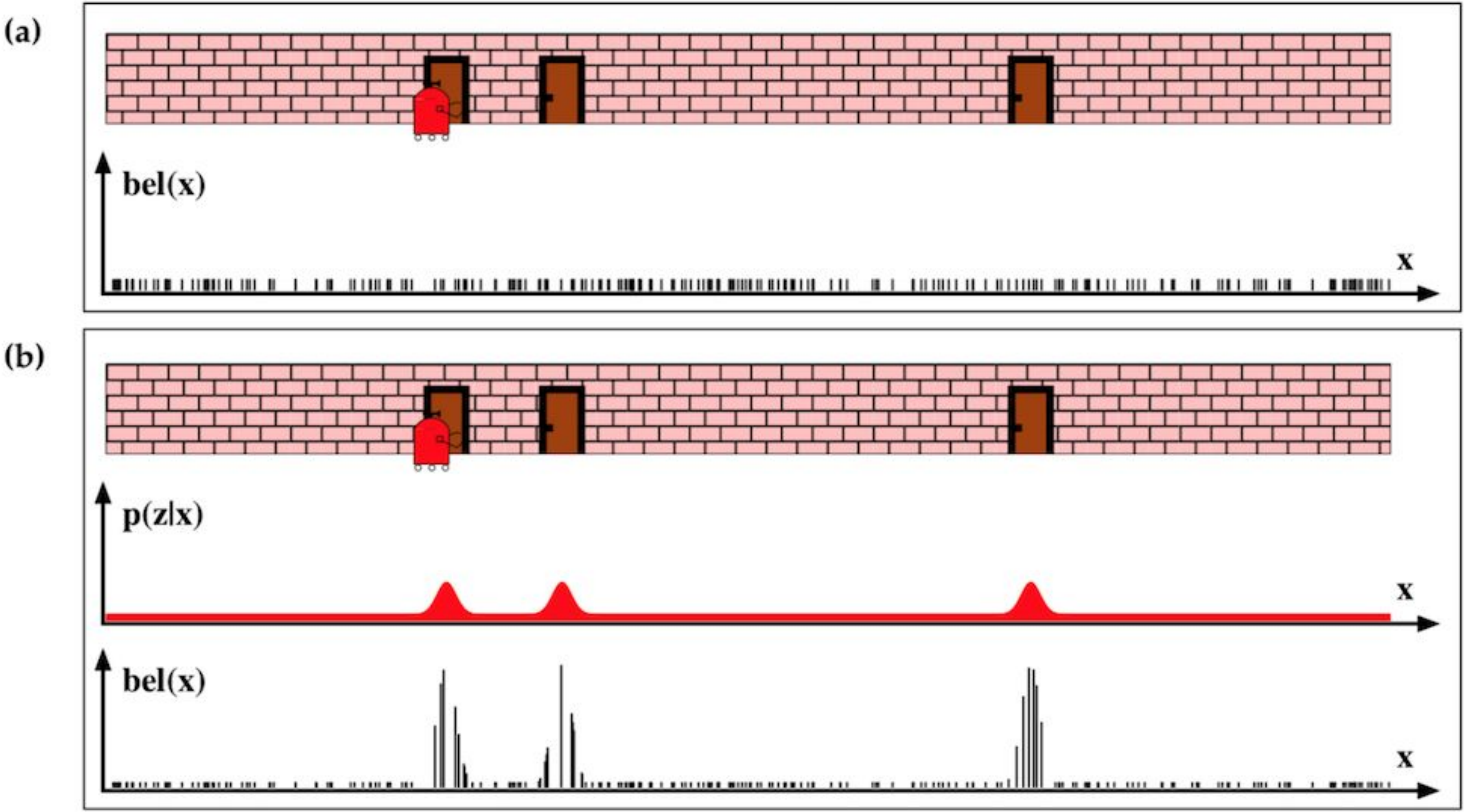
# Key Concepts (1 / 4)



# Key Concepts (2/4)

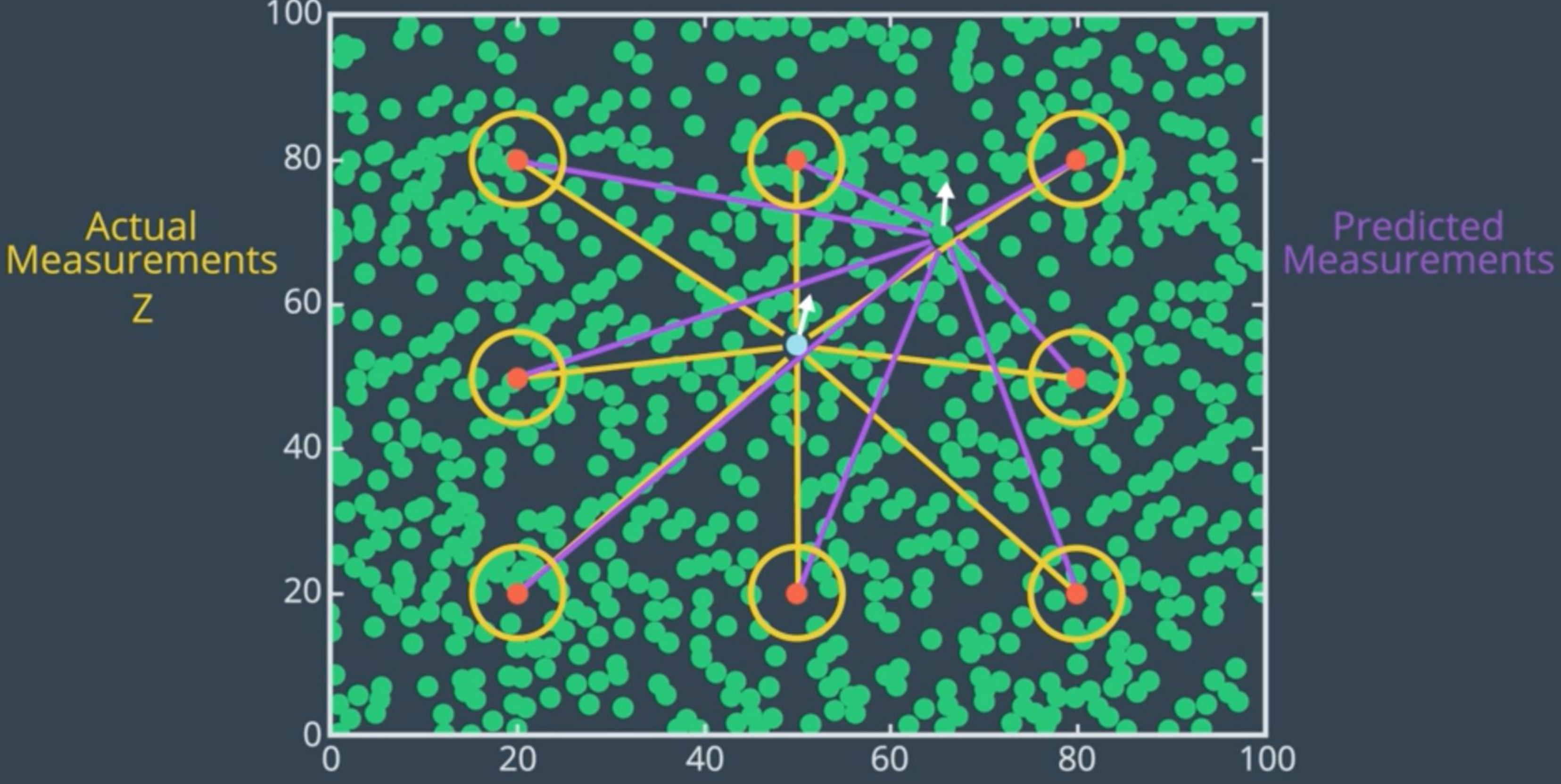
- A particle becomes "more important" if its predicted observation matches the actual observation
- “Data Association”

$x_{Particle}$
$y_{Particle}$
$\Theta_{Particle}$
Weight



(a)
bel(x)
x
(b)
p(z|x)
x
bel(x)
x

# Importance Weight

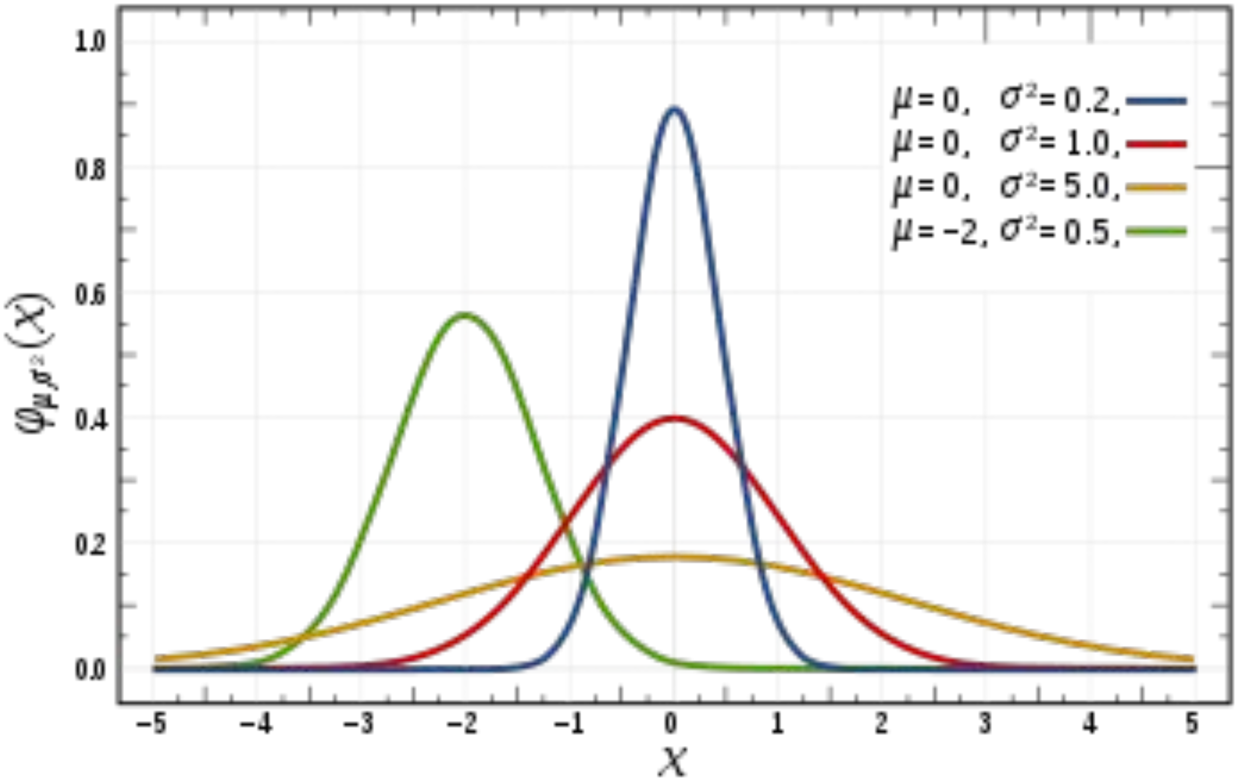


# Key Concepts (2/4)

- **Many ways to compute importance!**
- Univariate
- Bivariate
- Correlation

# Key Concepts (2/4)

- Univariate



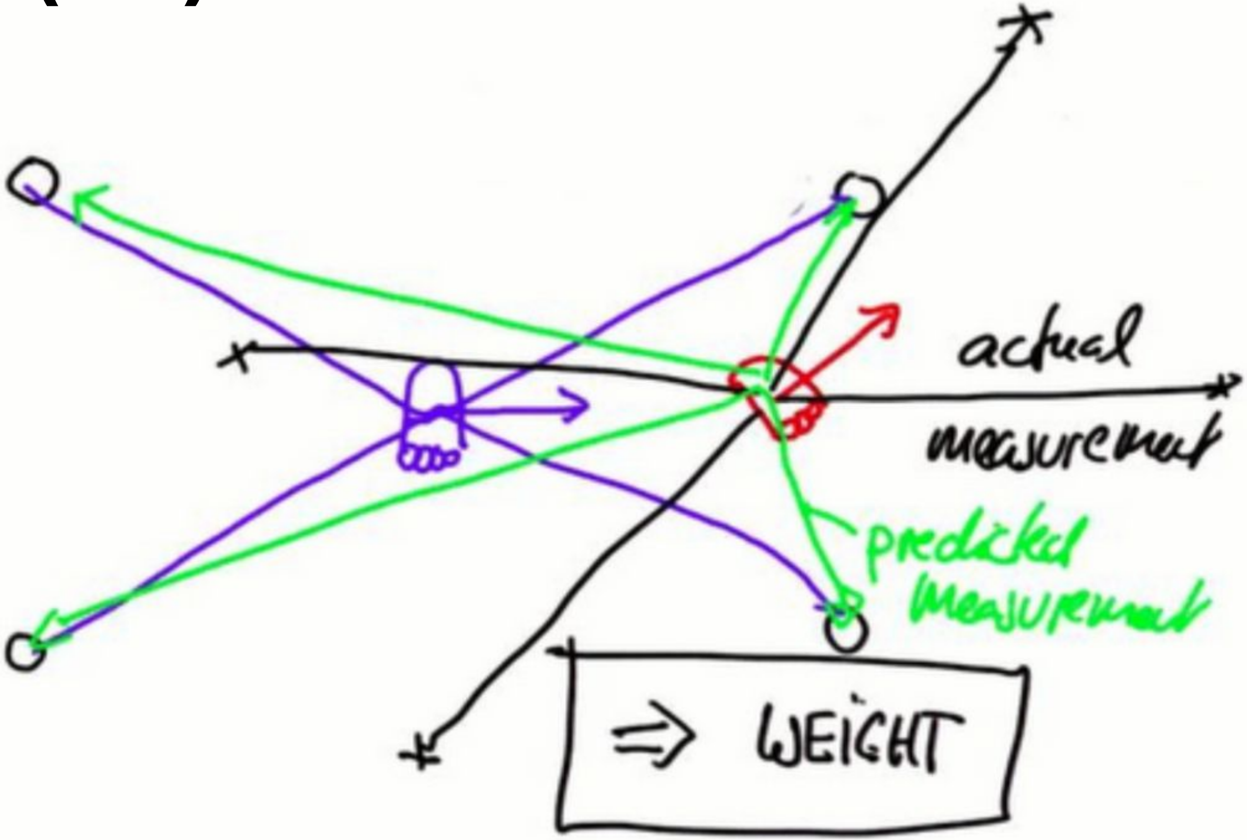
# Key Concepts (2/4)

- Univariate

```python
def gaussian_prob(mu, sigma, x):
    # calculate the probability of x for 1-dim Gaussian with mean mu and var sigma
    return exp( -((mu - x) ** 2) / (sigma ** 2) / 2.) / sqrt( 2. * pi * (sigma ** 2))

def get_weight(my_measurements, ground_measurements, noise):

  w = 1.
  for my_distance, ground_distance in zip(my_measurements, ground_measurements):
    w *= gaussian_prob(mu = my_distance, sigma = noise, x = ground_distance)

  return w + 1.e-300 # avoid round-off to zero
```

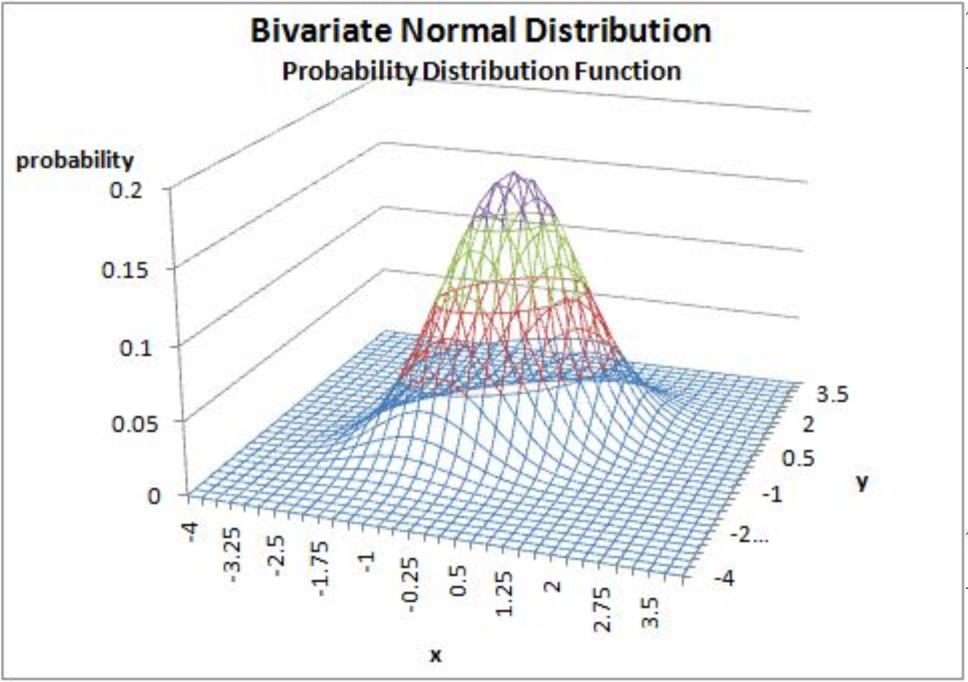
# Key Concepts (2/4)

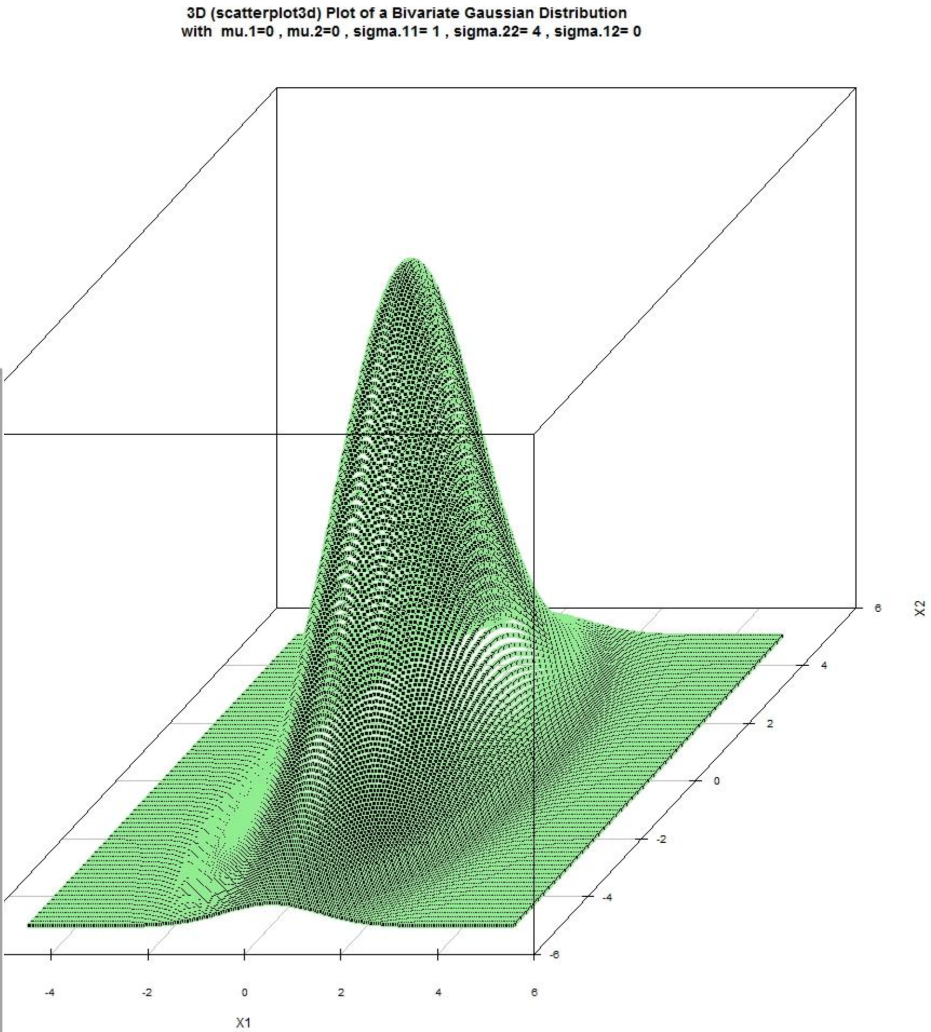
- Univariate



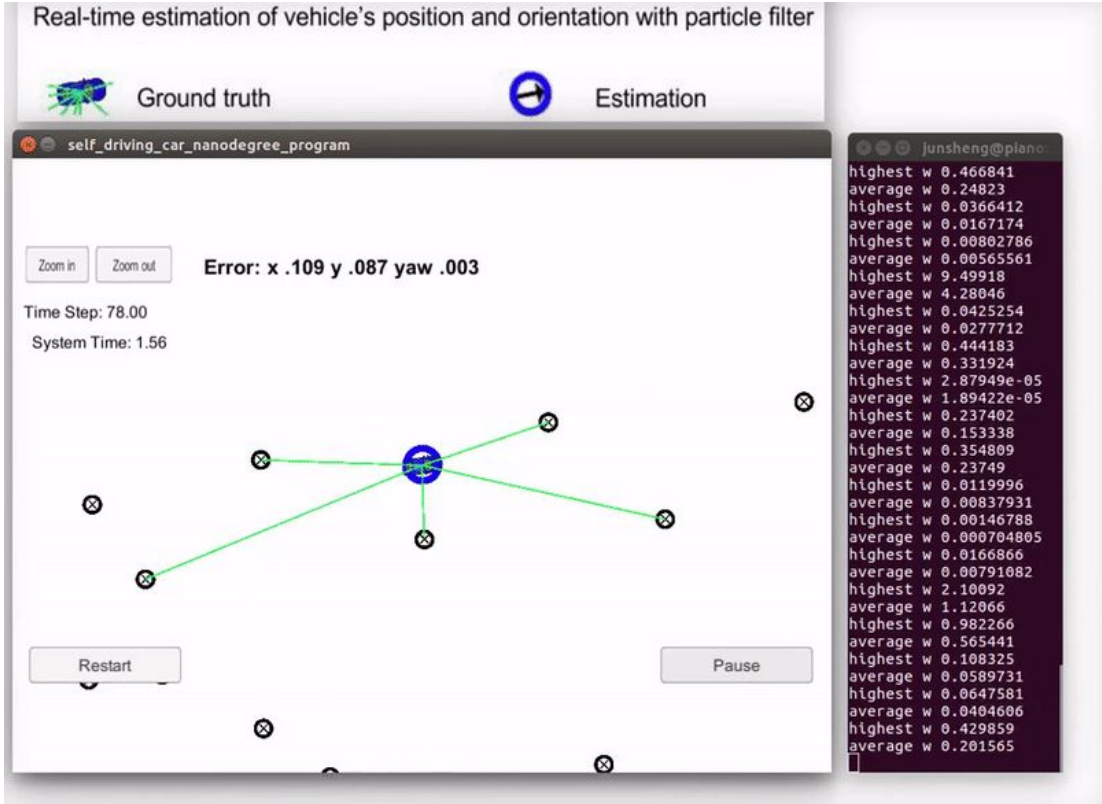
# Key Concepts (2/4)

- Bivariate



# Key Concepts (2/4)



# Key Concepts (2/4)

- Bivariate

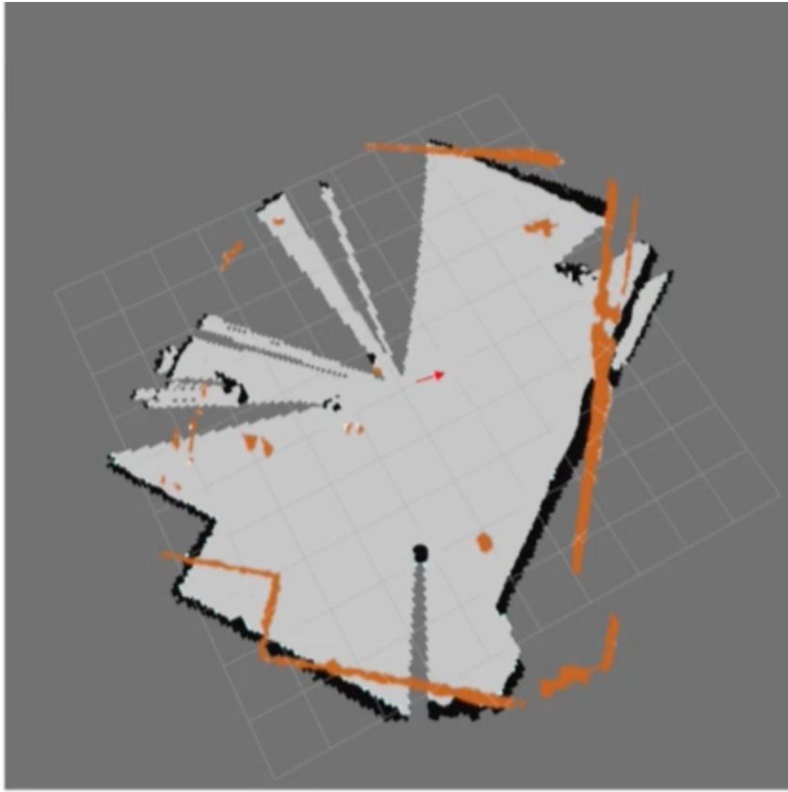
```
const double na = 0.5 / (stdx * stdx);
const double nb = 0.5 / (stdy * stdy);
const double d = sqrt( 2.0 * M_PI * stdx * stdy);
```

```
        const double dx = ox - predicted_x;
        const double dy = oy - predicted_y;

        const double a = na * dx * dx;
        const double b = nb * dy * dy;
        const double r = exp(-(a + b)) / d;
        w *= r;
```

# Key Concepts (2/4)

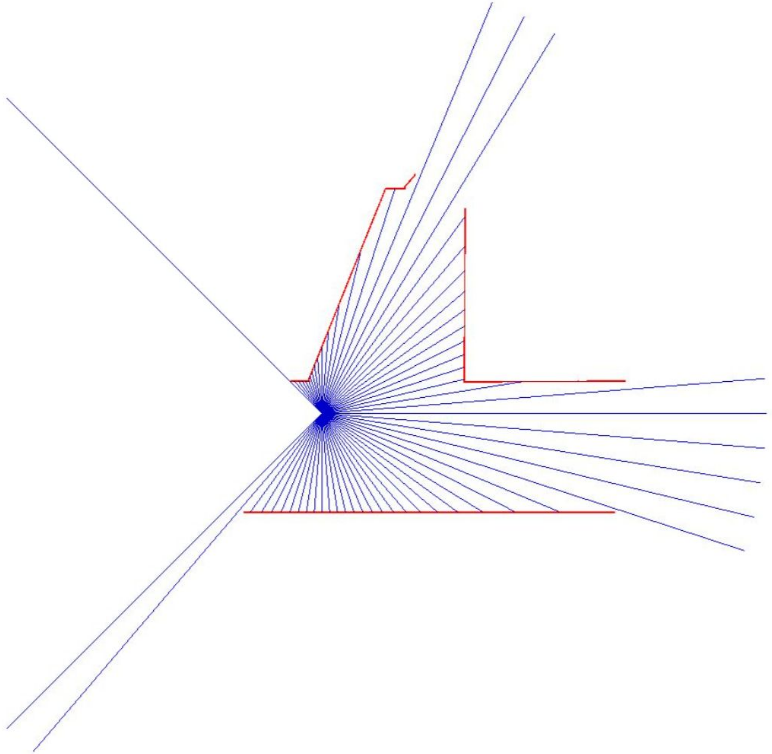
## Scan Correlation

$$S = \frac{\sum_m \sum_n (Amn - \overline{A})(Bmn - \overline{B})}{\sqrt{\left(\sum_m \sum_n (A_{mn} - \overline{A})^2\right)\left(\sum_m \sum_n (B_{mn} - \overline{B})^2\right)}}$$



# Key Concepts (2/4)

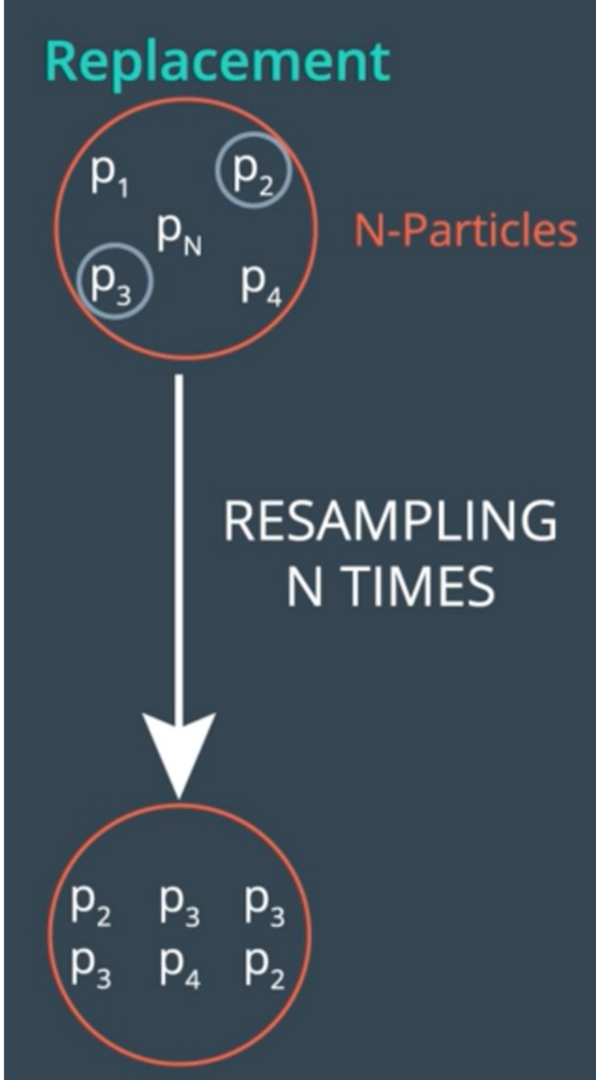
- Scan Correlation



# Key Concepts (3/4)

- Particles that are less important are more likely to die off
- Particles that are more important are more likely to clone itself

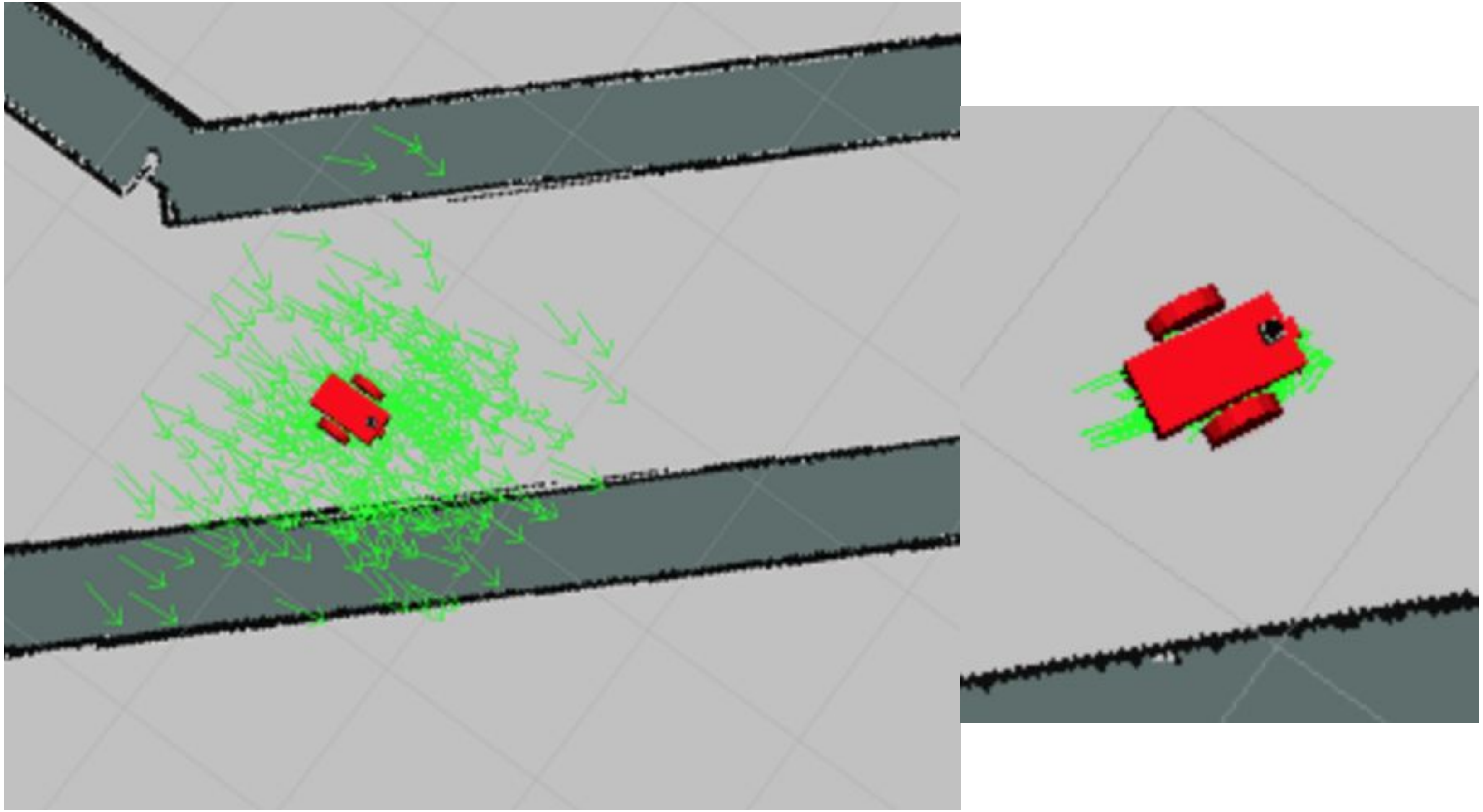
# Key Concepts (3/4)




bel(x)
x

# Key Concepts (4/4)

- If our filter is working properly, as time goes by, ideally we become more and more certain



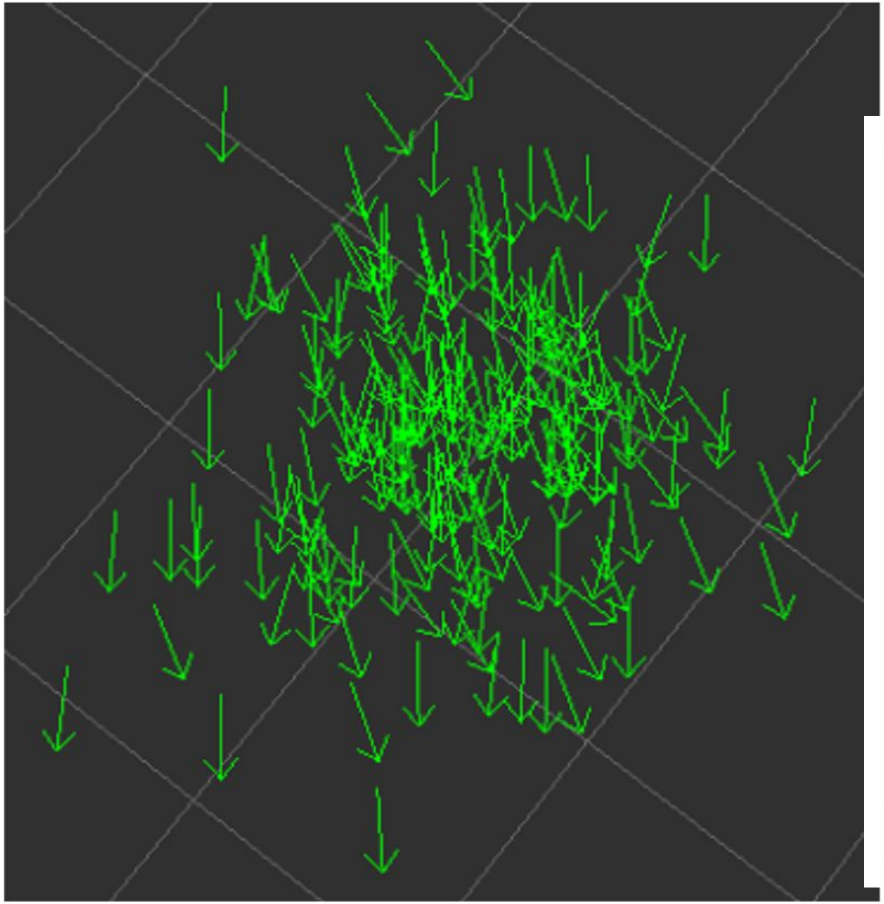


Fig. 3. Initial particle cloud generated based on a parametrized number of particles in the range between 25 and 200.

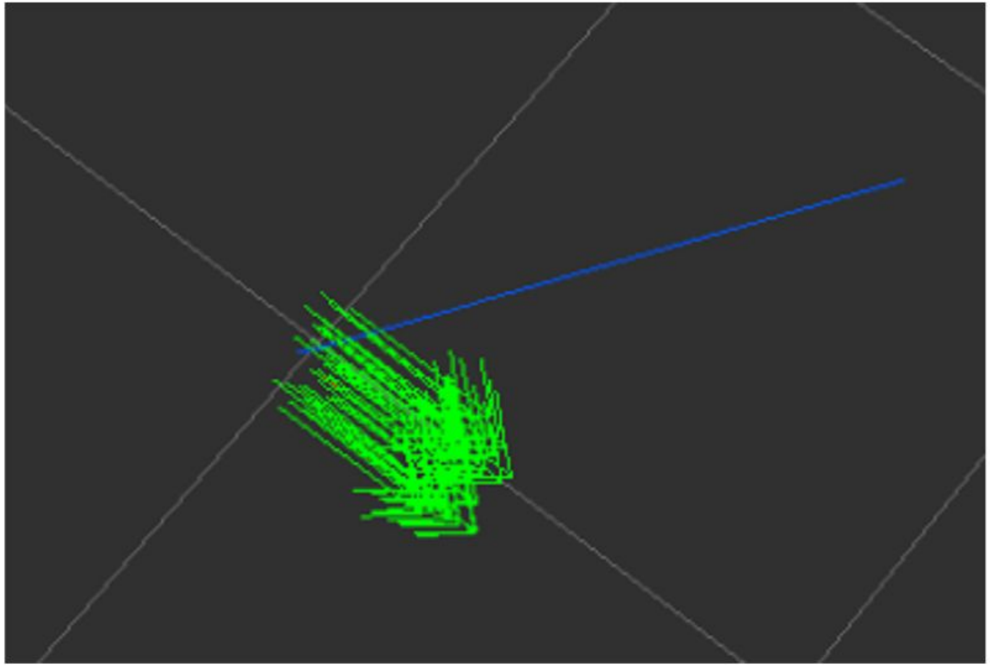
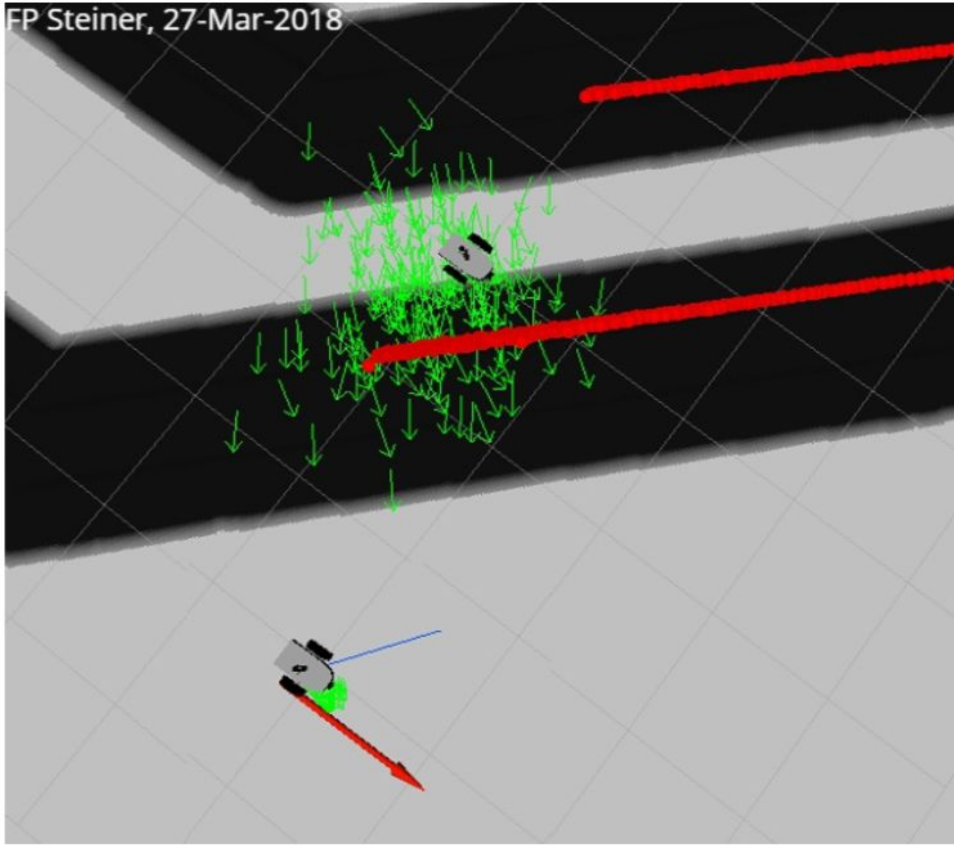


Fig. 4. Final particle cloud after the robot has reached its goal pose.



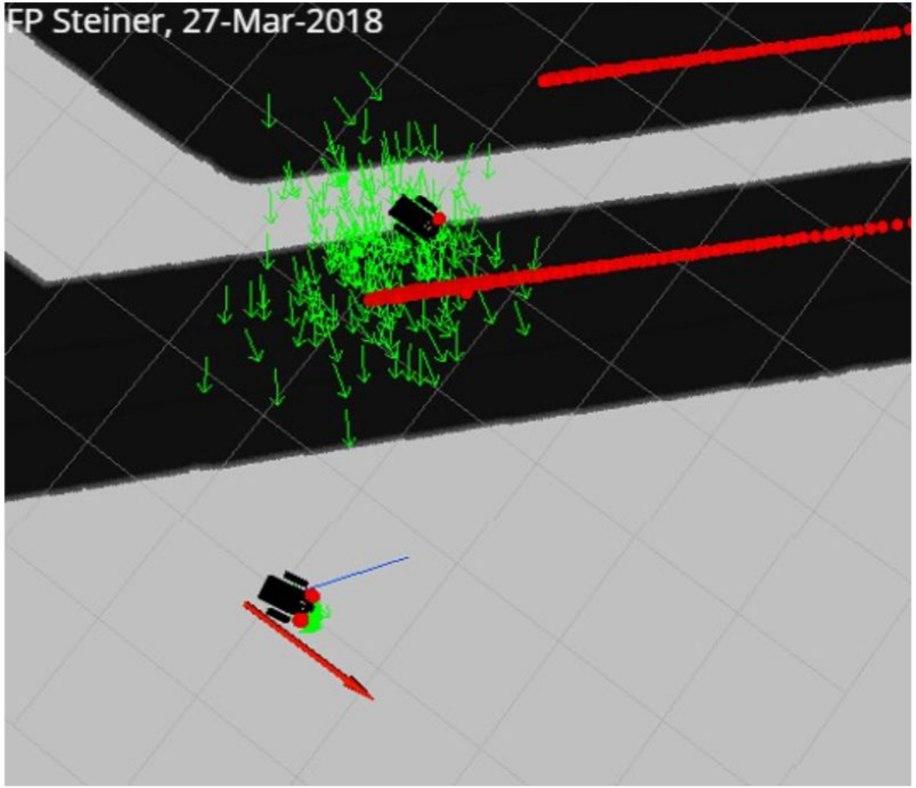




Fig. 9. Superposition of a screenshot of the initial and final pose and particle cloud of the classroom robot.

Fig. 10. Superposition of a screenshot of the initial and final pose and particle cloud of the redesigned robot.

# Outline

- What is it?
- Real world examples
- Key Concepts
- **The PF Algorithm**
- The PF Algorithm Review videos
- Resampling Algorithms
- Compare PF with other Probabilistic Filters

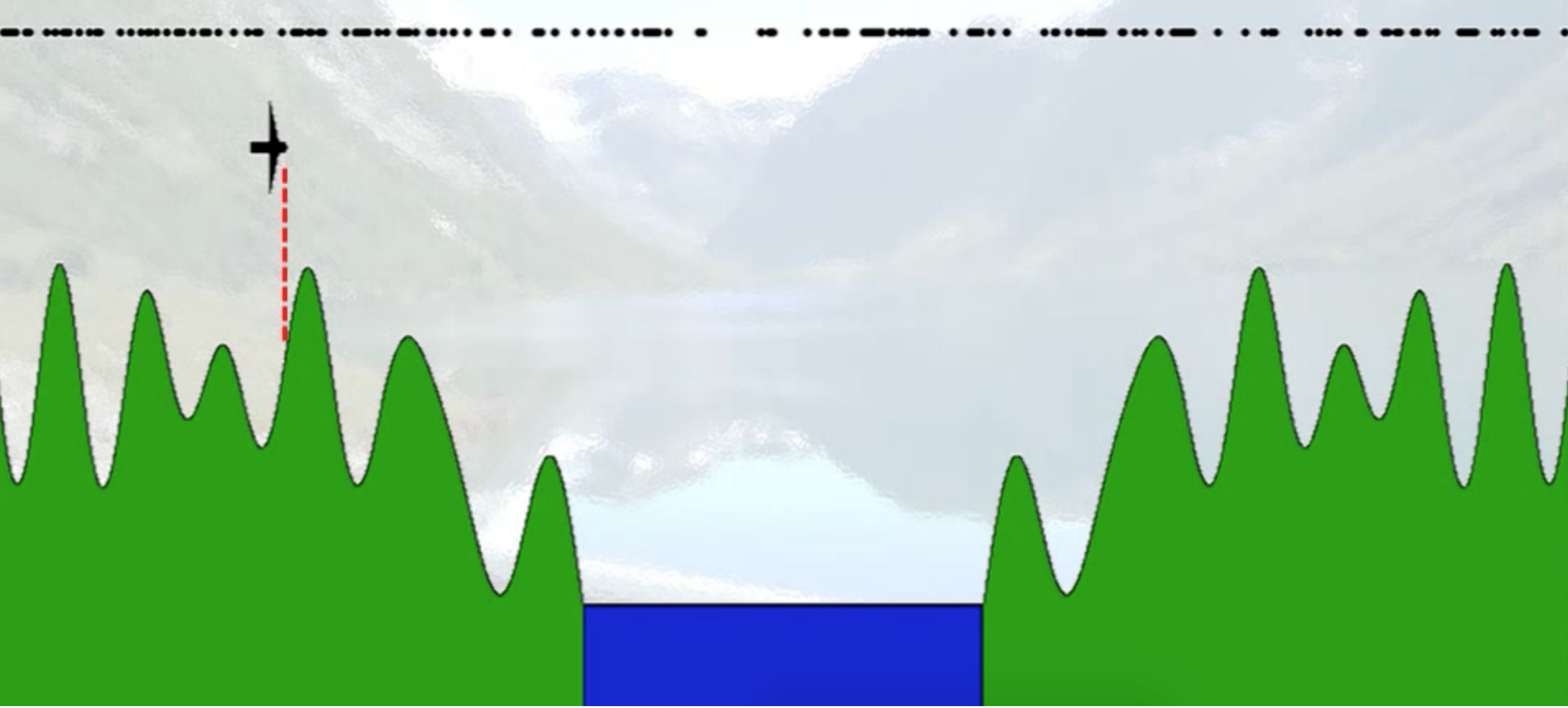
# Particle Filter Algorithm

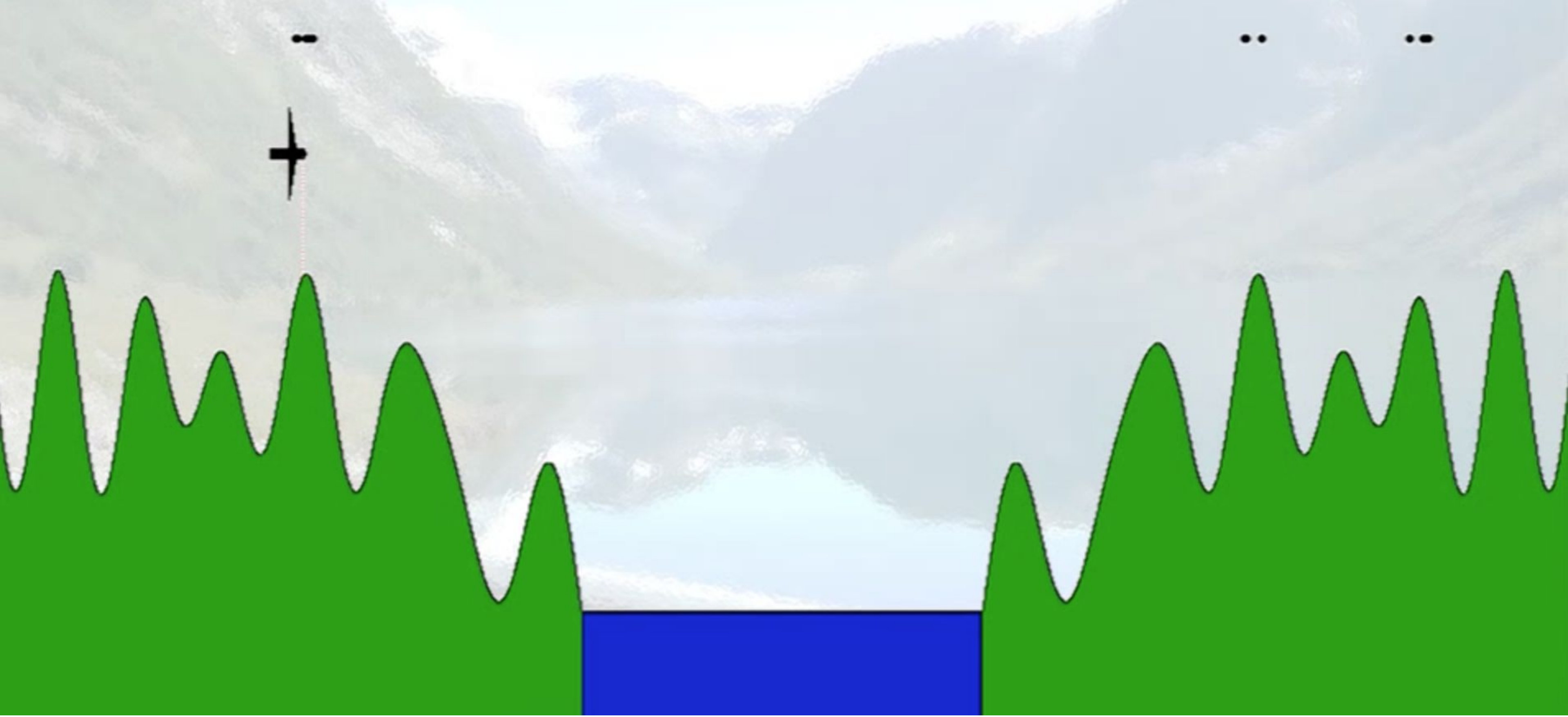
| STEPS | CONCEPTS |
| --- | --- |
| First Step | Previous Belief |
| Second Step | Motion Update |
| Third Step | Measurement Update |
| Fourth Step | Resampling |
| Fifth Step | New Belief |

# Particle Filter Algorithm

## Previous Belief

1. Start with the current set of N particles (equal importance)

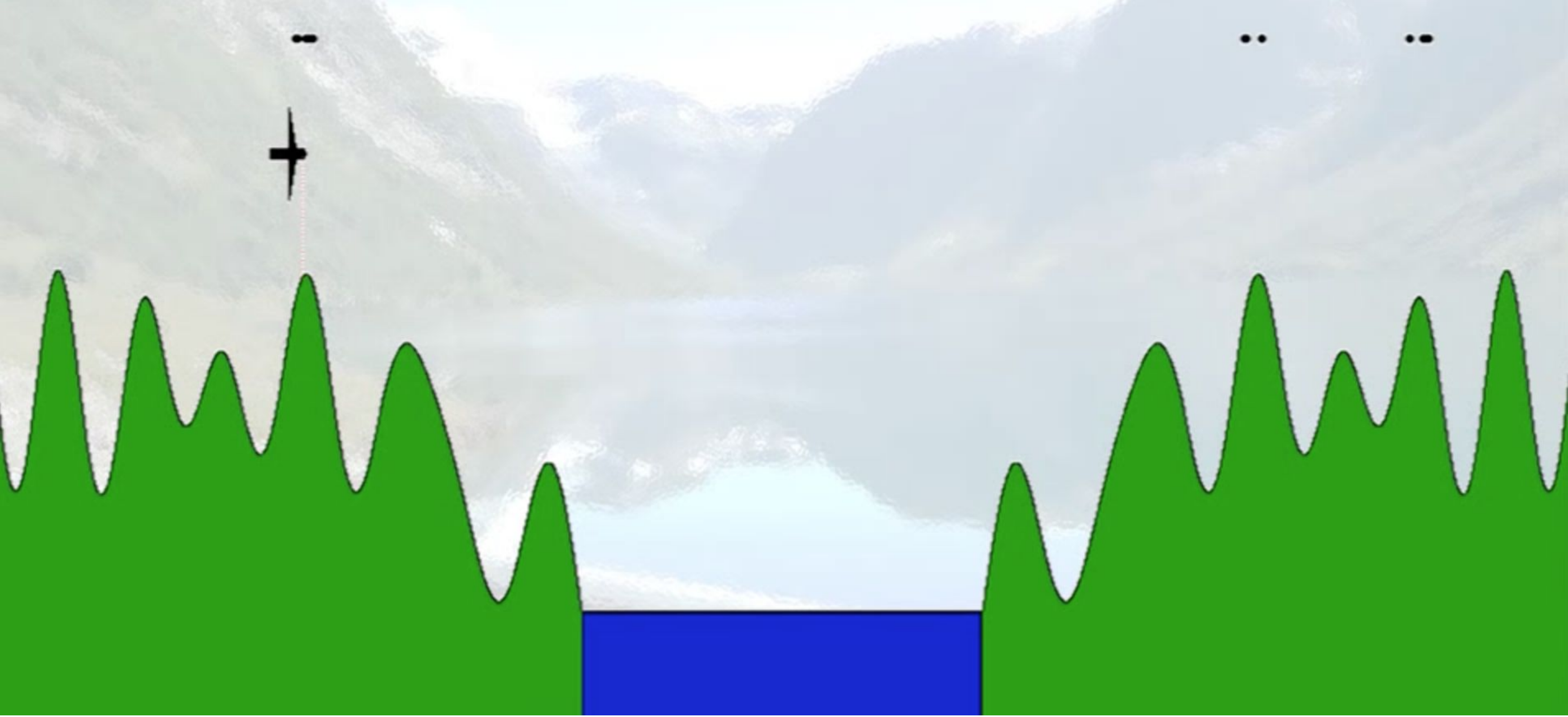


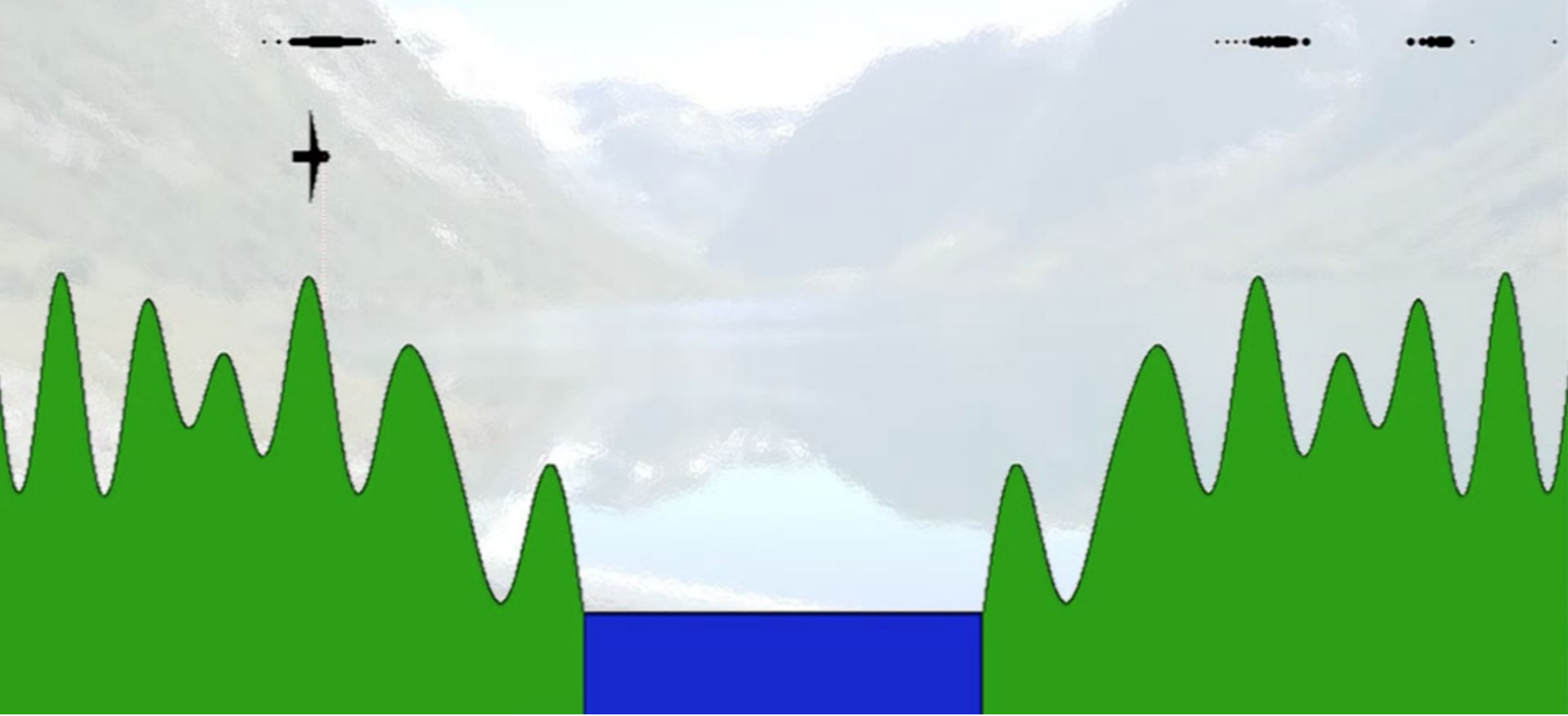


# Particle Filter Algorithm

## Motion Update

2. Move each particles as your robot moved (add noise)



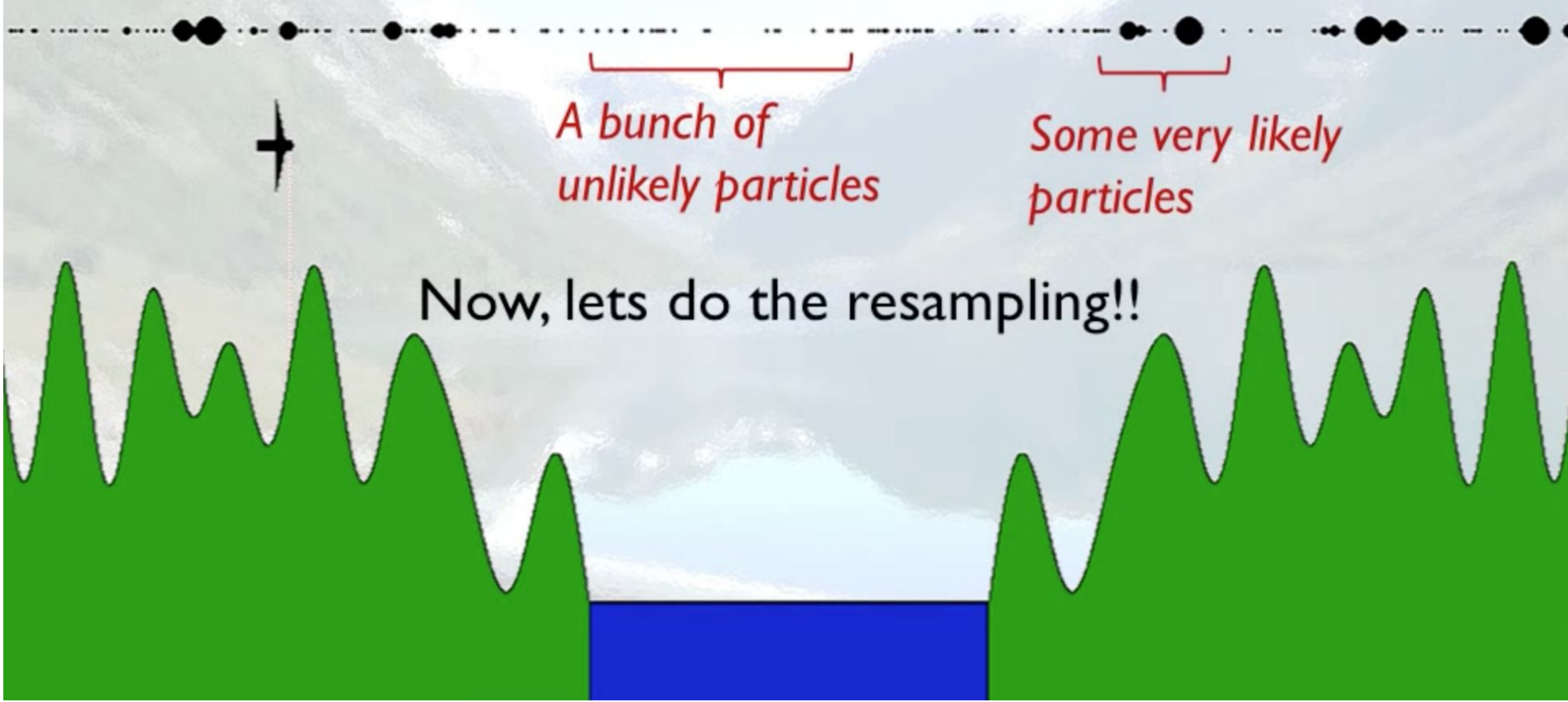


# Particle Filter Algorithm

## Measurement Update

3. "Data Association"

- Get actual observation from "noisy" sensors
- For each particle,
- - Compare the actual observation, and the particle's predicted observation **(include some noise)**
- - If the predicted observation of your particle matches your actual observation then that particle is more important
- - If not, then consequently, less important.

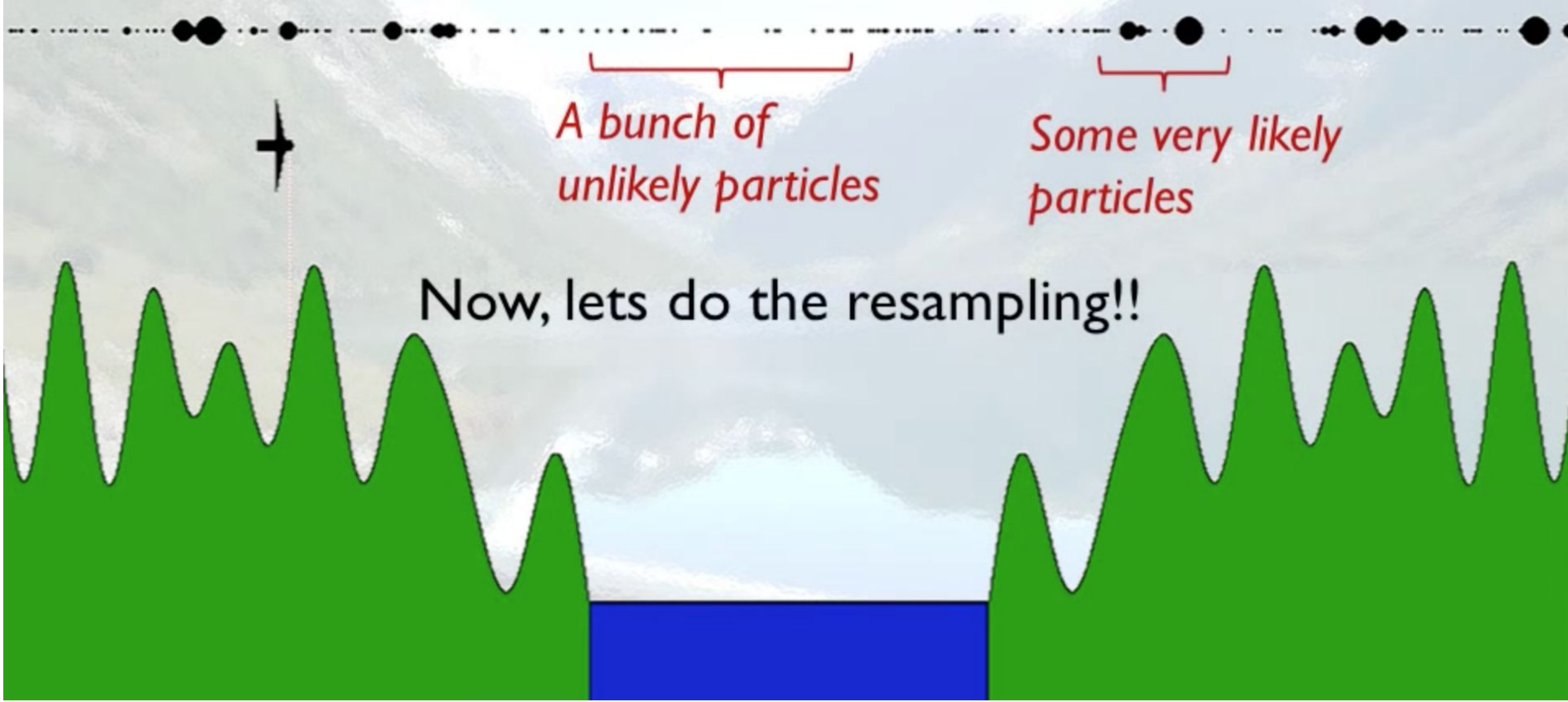

A bunch of unlikely particles
Some very likely particles
Now, lets do the resampling!!

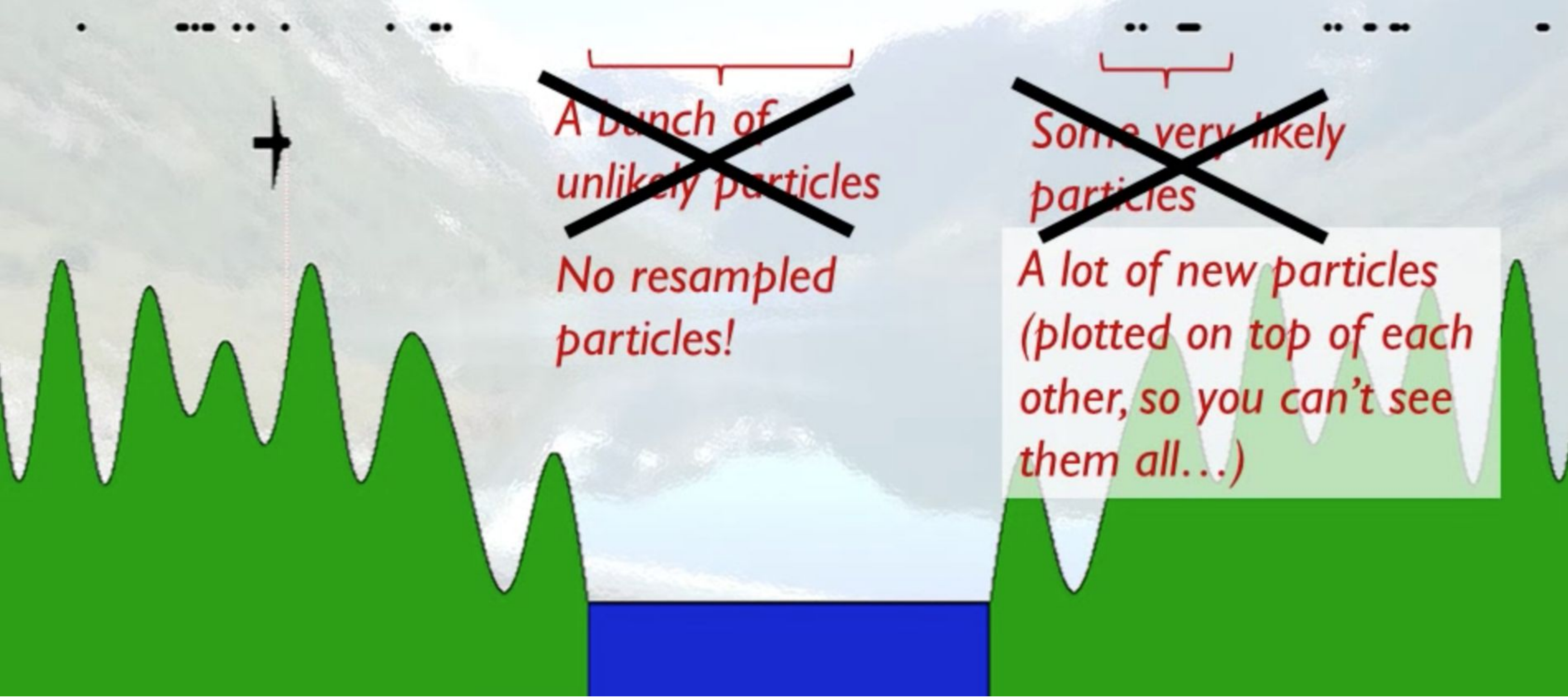
# Particle Filter Algorithm

## Resampling

4. Resample (You can think of it this way)

- Imagine you put all your particles[1…N] in a bag
- The ones that are more important are more likely to get picked
- Each time you pick a particle, you "clone it".
- Put the clone on your "new set" of particles
- Put the particle back in the bag, and pick one again
- Do this N times until you get a new set of particles

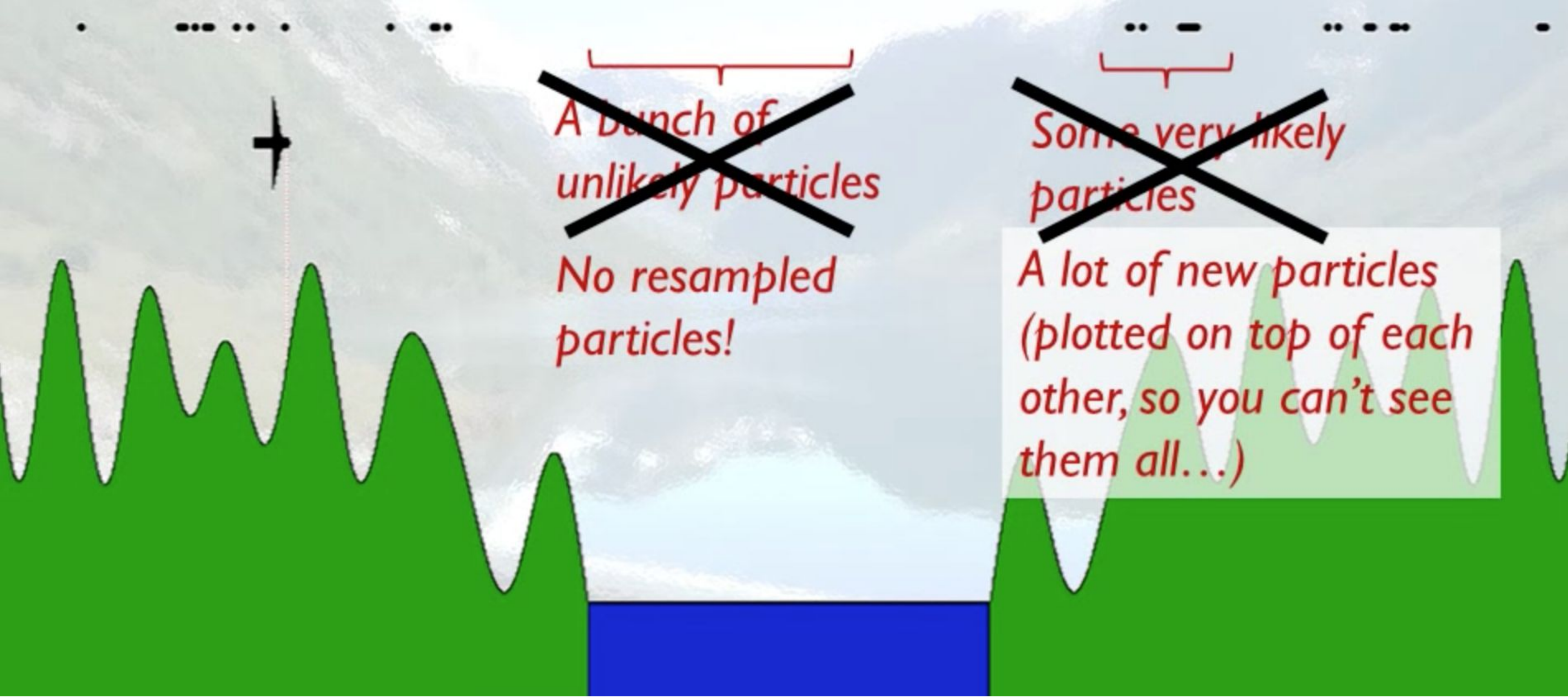

A bunch of unlikely particles
Some very likely particles
Now, lets do the resampling!!


~~A bunch of unlikely particles~~
No resampled particles!
~~Some very likely particles~~
A lot of new particles (plotted on top of each other, so you can't see them all…)

# Particle Filter Algorithm

## New Belief

5. Discard your old set of particles, this new set of particles will be your current set.


~~A bunch of unlikely particles~~
No resampled particles!
~~Some very likely particles~~
A lot of new particles (plotted on top of each other, so you can't see them all…)

# Outline

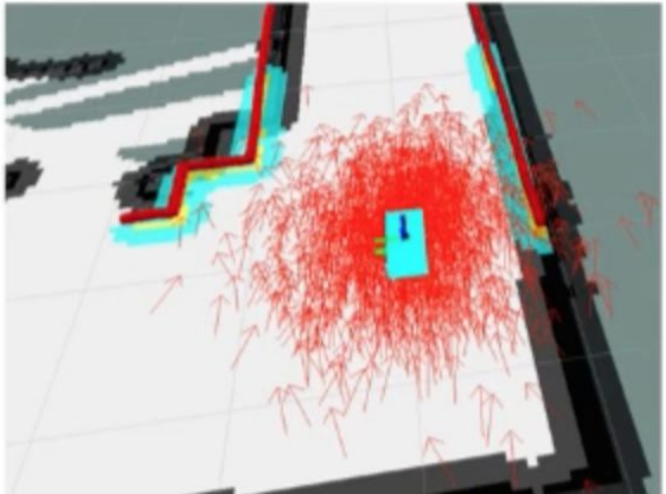
- What is it?
- Real world examples
- Key Concepts
- The PF Algorithm
- **PF Algorithm Review**
- Resampling Algorithms
- Compare PF with other Probabilistic Filters

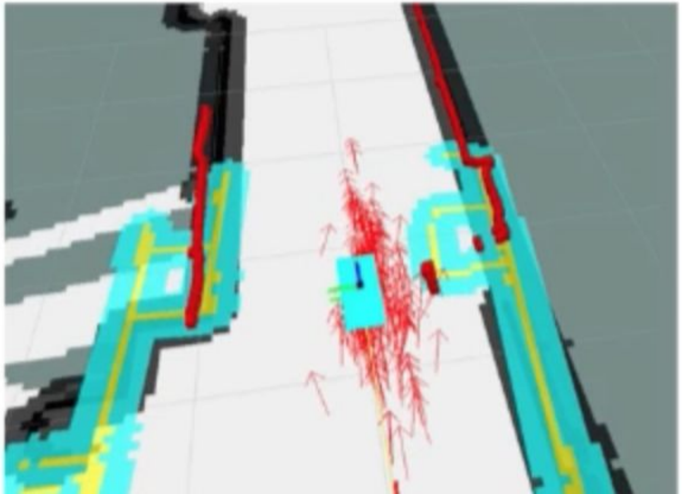
# Particle Filter Algorithm Review

- Video from Udacity
- My Video: Four Landmark Example (Noisy Sensor, Noisy Motion)

# Kullback–Leibler divergence (KLD Sampling)

- Variable Particle size
- Sample size is proportional to error between odometry position and sample based approximation
- i.e smaller sample size when particles have converged





# Outline

- What is it?
- Real world examples
- Key Concepts
- The PF Algorithm
- PF Algorithm Review
- **Resampling Algorithms**
- Compare PF with other Probabilistic Filters

# Resampling Algorithms

- Affects the performance of the filter.

- Research on the topic continues

# Resampling Algorithms

**Properties.**

- preferentially select particles that have a higher probability

- select a representative population of the higher probability particles

- include enough lower probability particles to give the filter a chance of detecting strongly nonlinear behavior.

# Resampling Algorithms

*In practical applications of sequential Monte Carlo methods, residual, stratified, and systematic resampling are generally found to provide comparable results.* ****Despite the lack of complete theoretical analysis of its behavior*, systematic resampling is often preferred because it is the simplest method to implement.***

*Comparison of Resampling Schemes for Particle Filtering 0507025* by: Randal Douc, Oliver Cappe, Eric Moulines.

https://arxiv.org/pdf/cs/0507025.pdf

# Resampling Algorithm: Multinomial

```
print('cumulative sume is', np.cumsum([.1, .2, .1, .6]))
plot_cumsum([.1, .2, .1, .6])
```

```
cumulative sume is [ 0.1  0.3  0.4  1.0]
```



This is an $O(n\log(n))$ algorithm. That is not terrible, but there are $O(n)$ resampling algorithms with better properties v

# Resampling Algorithm: Multinomial

- The performance of the multinomial resampling could be bad.

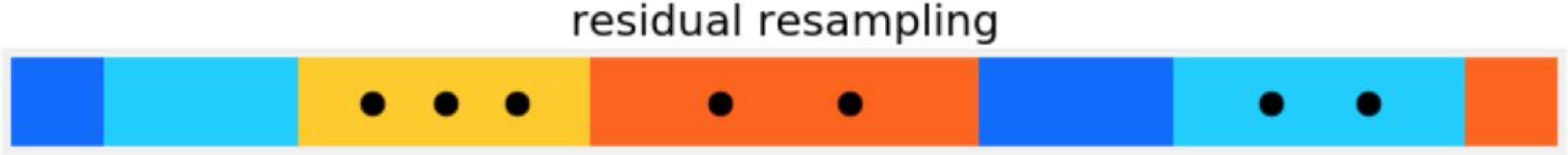
- Very large weight can be not sampled at all

- Smallest weight can be sampled twice.

- Rarely used in the literature or for real problems

# Resampling Algorithm: Residual

- Normalized weights are multiplied by N
- Integer value = how many samples of that particle will be taken.
- w = 0.0012, N =3000, N * w = 3.6
- 3 samples will be taken of that particle.
- Residual = 0.6 <-- use this with other sampling methods



# Resampling Algorithm: Residual

- Ensures all the largest weights are resampled multiple times

- Doesn't necessarily evenly distribute the samples across the particles

# Resampling Algorithm: Systematic

- The space is divided into N divisions
- Choose a random offset to use for all of the divisions
- Each sample is exactly 1/N apart.

# Resampling Algorithm: Systematic

- Ensures we sample from all parts of the particle space

# Resampling Algorithm: Stratified

- Make selections relatively uniformly across the particles
- Divide the cumulative sum into N equal sections
- Select one particle randomly from each section. T
- This guarantees each sample 0 and 2/N apart.

# Resampling Algorithm: Stratified

- a bit better than systematic sampling at ensuring the higher weights get resampled more.

# Resampling Algorithms

## std::discrete_distribution

Defined in header <random>

```
template< class IntType = int >
class discrete_distribution;
```

(since C++11)

`std::discrete_distribution` produces random integers on the interval `[0, n)`, where the probability of each individual integer `i` is defined as $w_i/S$, that is the *weight* of the ith integer divided by the sum of all n weights.

`std::discrete_distribution` satisfies all requirements of `RandomNumberDistribution`

# Resampling Algorithms

```cpp
int main()
{
    std::random_device rd;
    std::mt19937 gen(rd());
    std::discrete_distribution<> d({40, 10, 10, 40});
    std::map<int, int> m;
    for(int n=0; n<10000; ++n) {
        ++m[d(gen)];
    }
    for(auto p : m) {
        std::cout << p.first << " generated " << p.second << " times\n";
    }
}
```

Output:

```
0 generated 4028 times
1 generated 978 times
2 generated 1012 times
3 generated 3982 times
```

# Outline

- What is it?
- Real world examples
- Key Concepts
- The PF Algorithm
- PF Algorithm Review
- Resampling Algorithms
- **Compare PF with other Probabilistic Filters**

| QUIZ | state space | belief | efficiency | in robotics |
| --- | --- | --- | --- | --- |
| Class 1<br>Histogram Filters | x Discrete<br>o Continuous | o unimodal<br>x multimodal | o quadratic<br>x exponential | o exact<br>x approximate |
| Class 2<br>Kalman Filters | o Discreto<br>x Continuous | x unimodal<br>o multimodal | x quadratic<br>o exponential | o exact<br>x approximate |
| Class 3<br>Particle Filters | Continuous | multimodal | ? | approximate |

| | MCL | EKF |
|---|---|---|
| Measurements | Raw Measurements | Landmarks |
| Measurement Noise | Any | Gaussian |
| Posterior | Particles | Gaussian |
| Efficiency(memory) | ✔ | ✔✔ |
| Efficiency(time) | ✔ | ✔✔ |
| Ease of Implementation | ✔✔ | ✔ |
| Resolution | ✔ | ✔✔ |
| Robustness | ✔✔ | x |
| Memory & Resolution Control | Yes | No |
| Global Localization | Yes | No |
| State Space | Multimodel Discrete | Unimodal Continuous |

QUIZ QUESTION

How is MCL different from other localization algorithms? Select all that apply:

- [ ] MCL uses grids to localize the robot pose
- [x] MCL uses particles to localize the robot pose
- [ ] MCL uses histograms to localize the robot pose
- [ ] MCL is restricted by a Gaussian state space distribution
- [x] MCL can approximate almost any state space distribution

- EKFs cannot incorporate negative information

- They cannot use the fact that a robot failed to see a feature even when expected.

- **Sebastian Thrun**

# Outline

- What is it?
- Real world examples
- Key Concepts
- The PF Algorithm
- KLD Sampling (Preview)
- PF Algorithm Review
- Resampling Algorithms

# Sources 1

- UPenn - http://f1tenth.org/session3#l1-3
- MIT RACE CAR https://www.youtube.com/watch?v=-c_0hSjgLYw
- RLABBE RESAMPLING https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python/blob/master/12-Particle-Filters.ipynb
- Particle Filter Explained without Equations https://www.youtube.com/watch?v=aUkBa1zMKv4

# Sources 2


- Comparison of Resampling Schemes for Particle Filtering 0507025* by: Randal Douc, Oliver Cappe, Eric Moulines. https://arxiv.org/pdf/cs/0507025.pdf

- Particle Filters in Robotics In Proceedings of Uncertainty in AI (UAI) 2002 Sebastian Thrun

- Udacity Robotics Software Engineer Nanodegree

- Udacity Intro to AI for Robotics Course

- https://github.com/mithi/particle-filter-prototype