Particle Filter Localization

Monte Carlo method

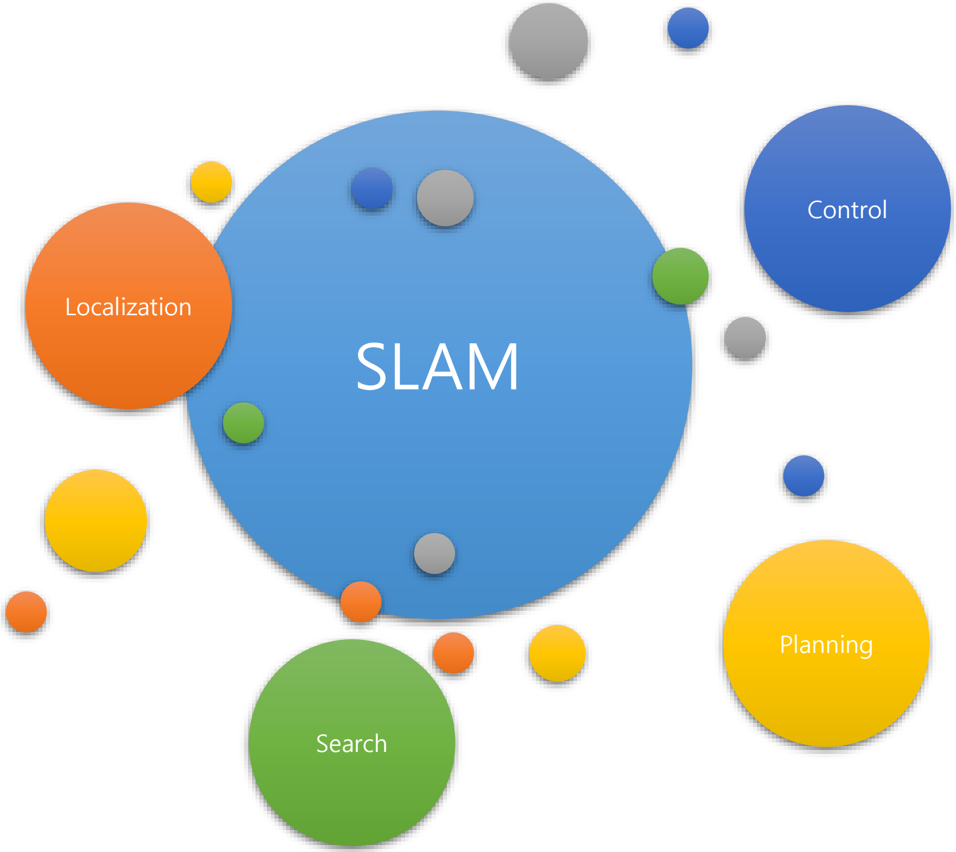
2016.02.04

Hyun Ho Jeon

CONTENTS

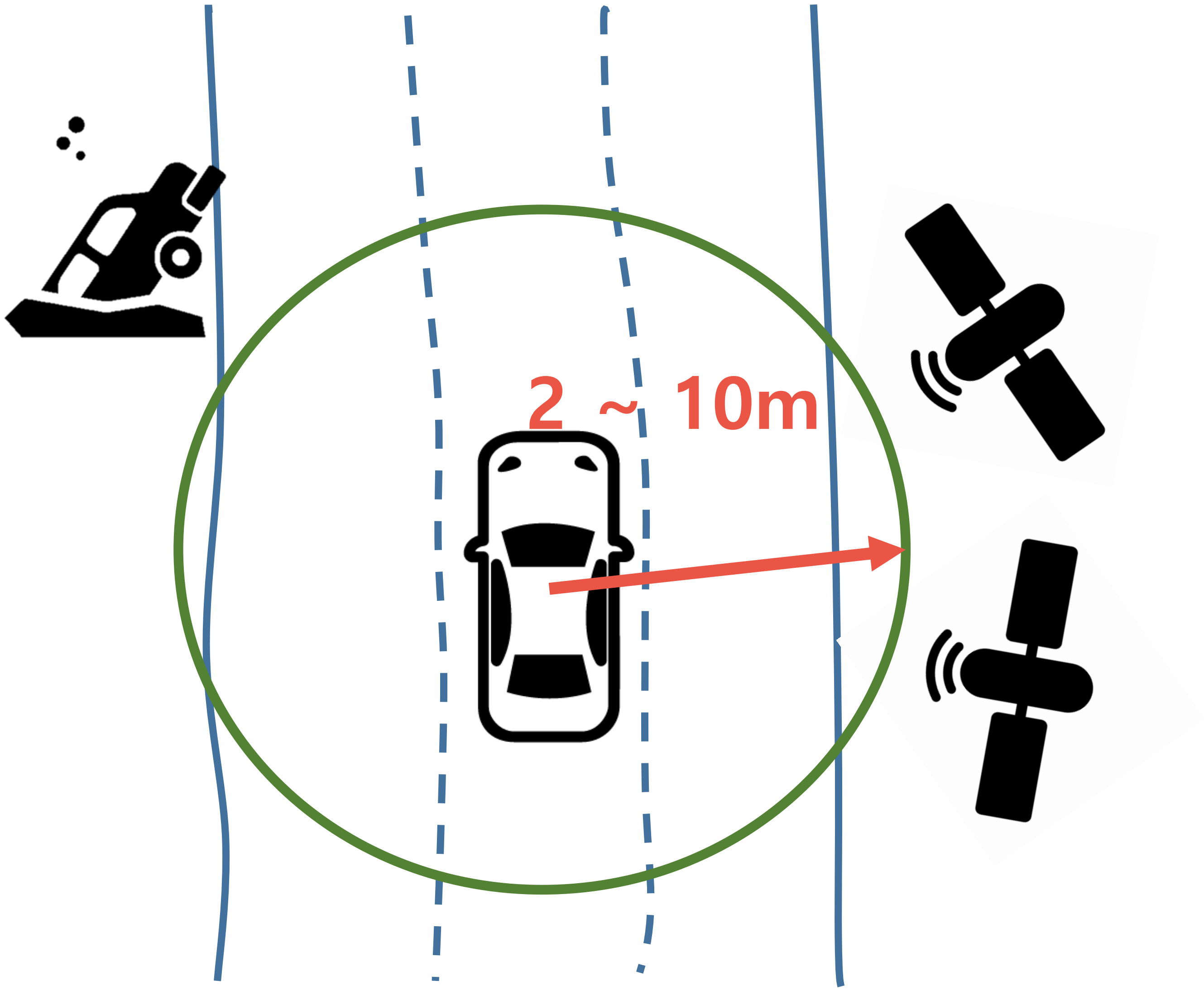
SLAM

- SLAM(Simultaneous localization and mapping) : In robotic mapping, **simultaneous localization and mapping** (**SLAM**) is the computational problem of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it. … … Popular approximate solution methods include the **particle filter** and extended Kalman filter.



2005 DARPA Grand Challenge winner STANLEY performed SLAM as part of its autonomous driving system.

Localization

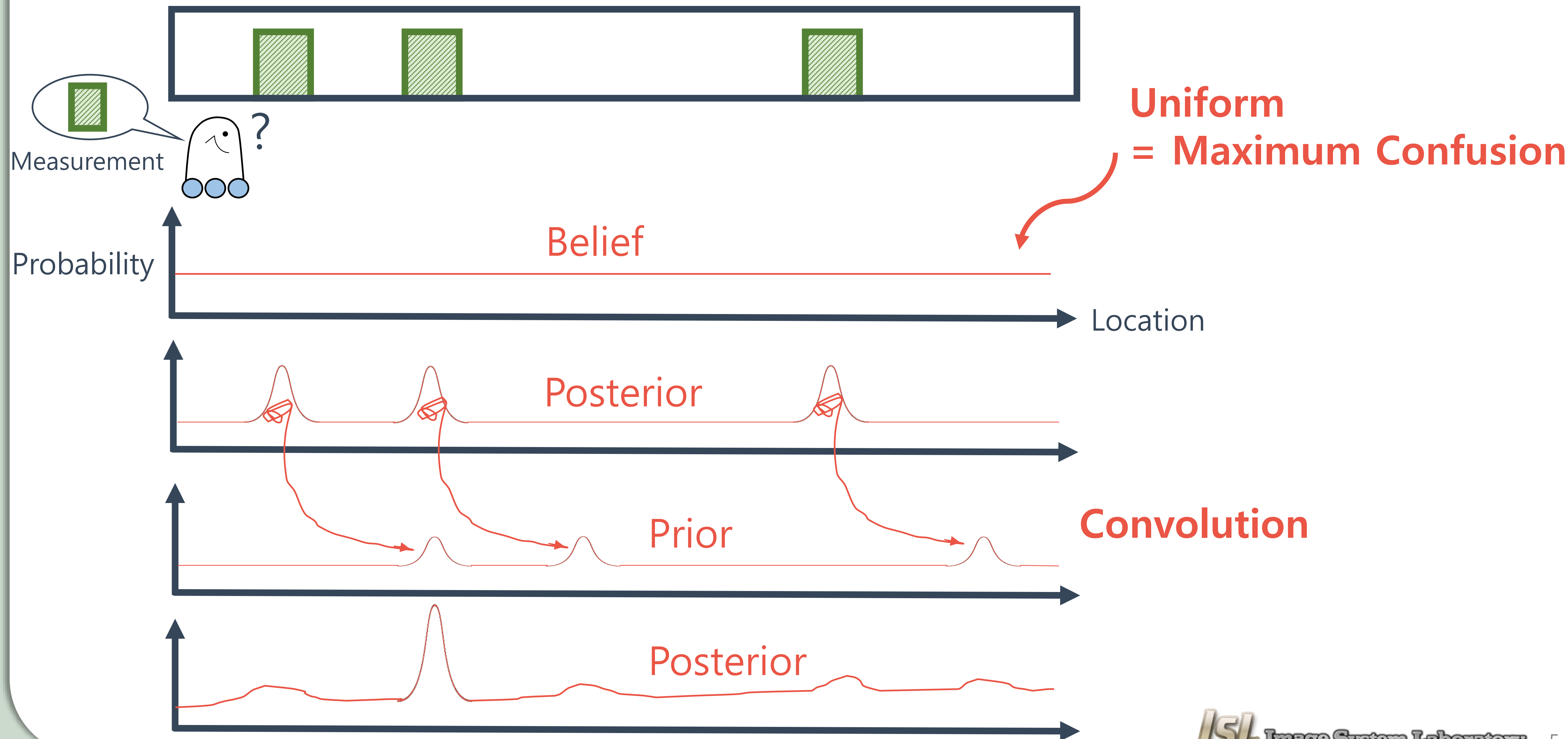


GPS Global Positioning System

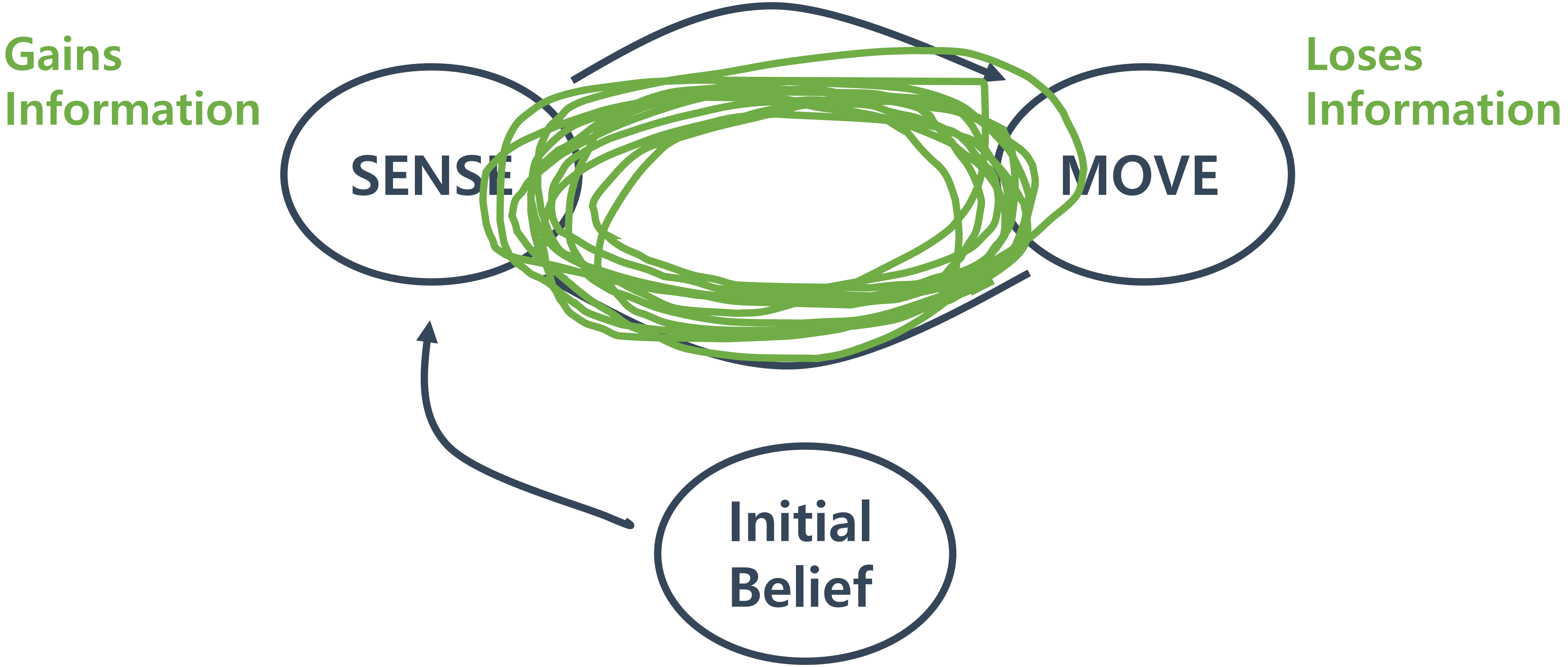
The problem with GPS is its really not very accurate. It's common for a car to believe to be somewhere but **it has error about 2-10 meters.**

If you want to reduce error -> **Localization**

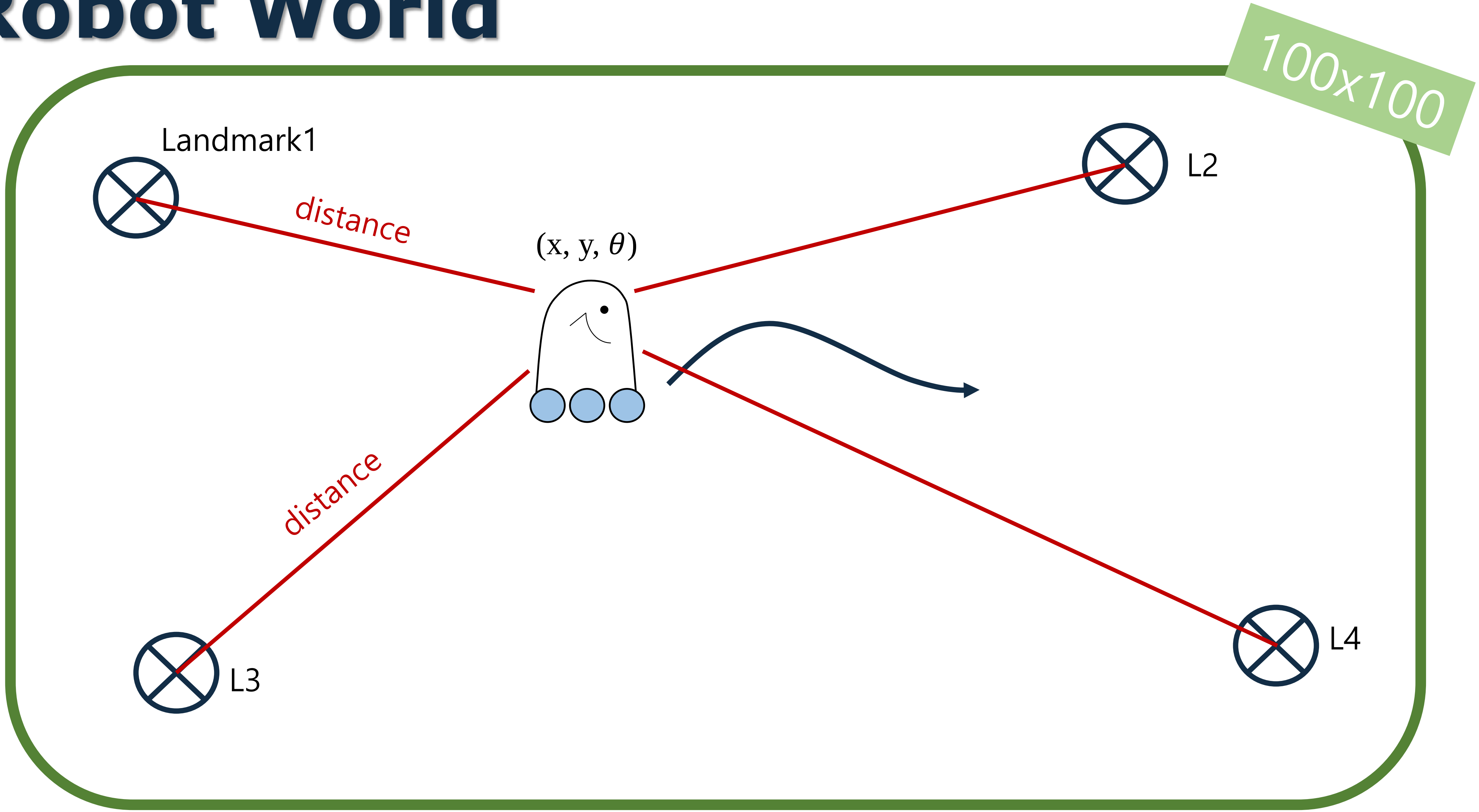
Localization



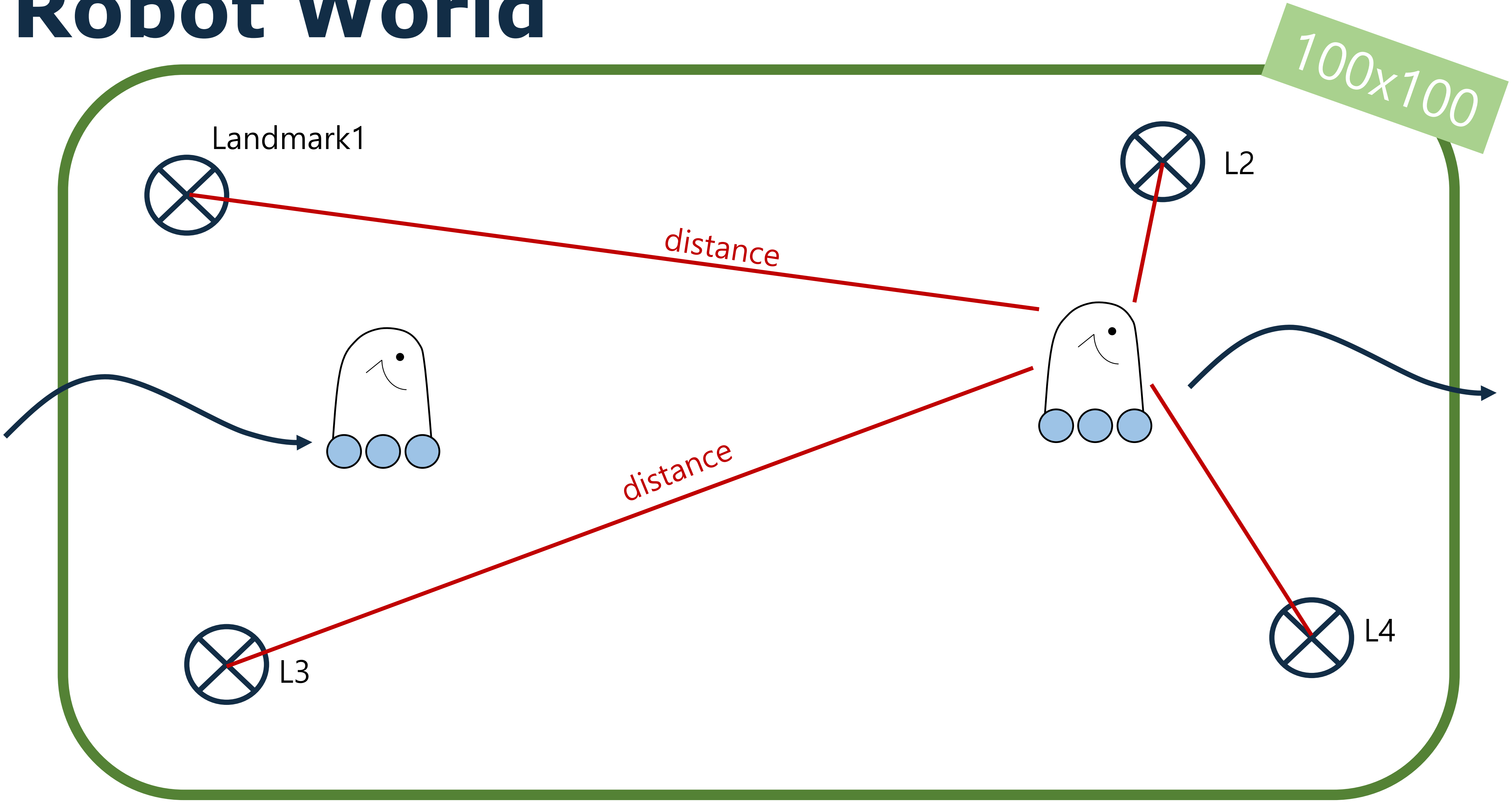
Sense & Move


Gains
Information
SENSE
MOVE
Loses
Information
Initial
Belief

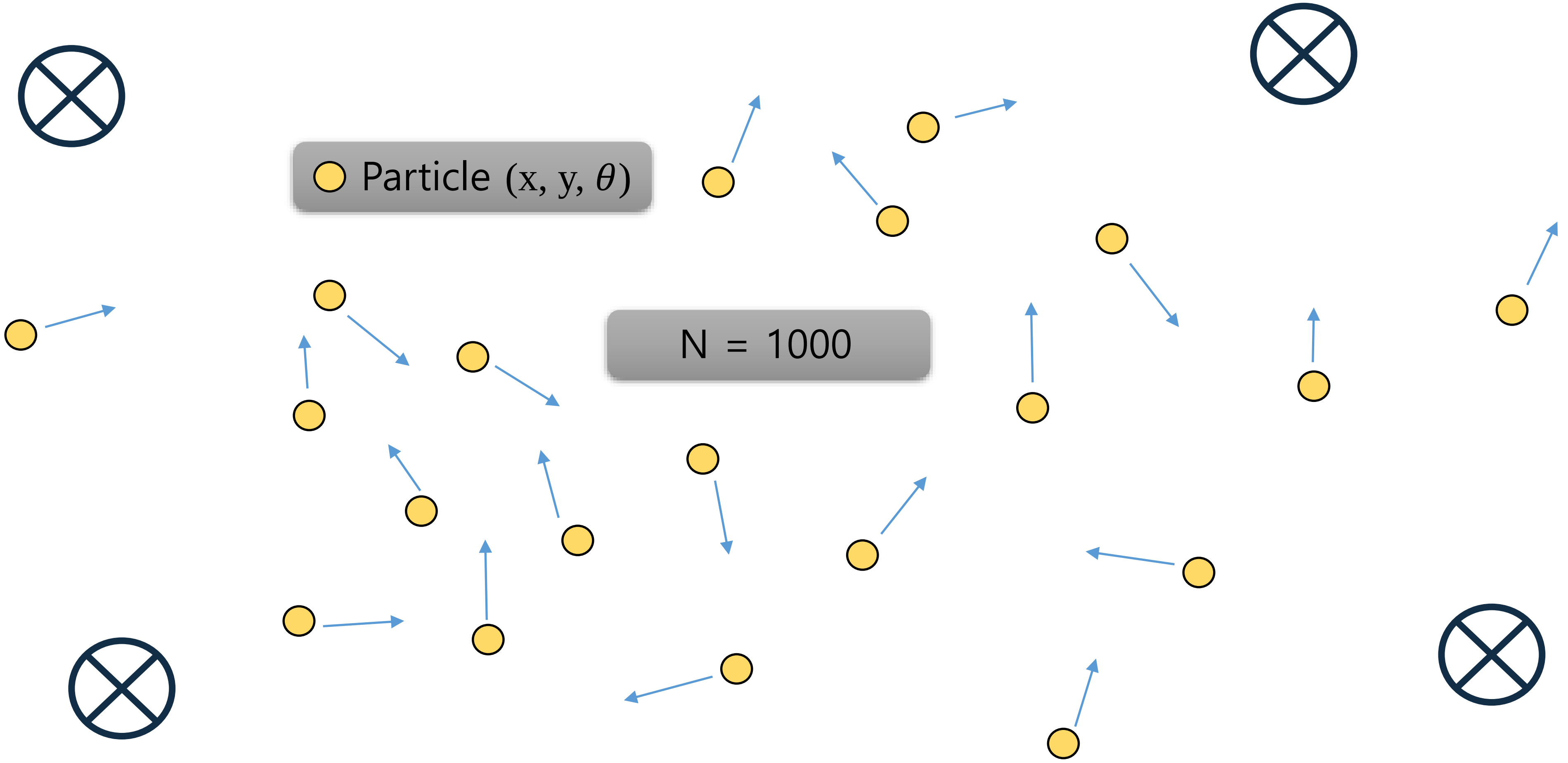
Robot World



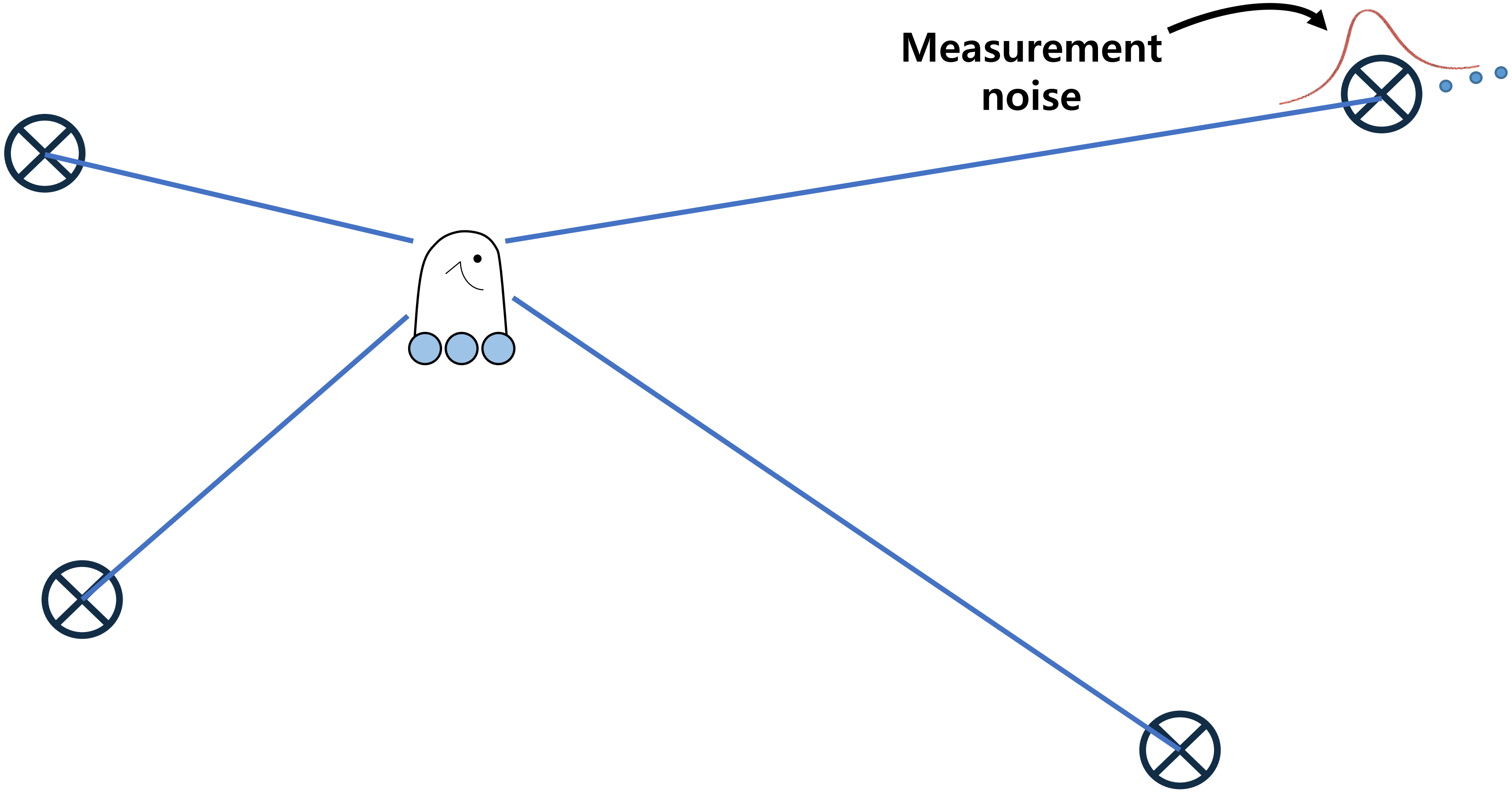
Robot World



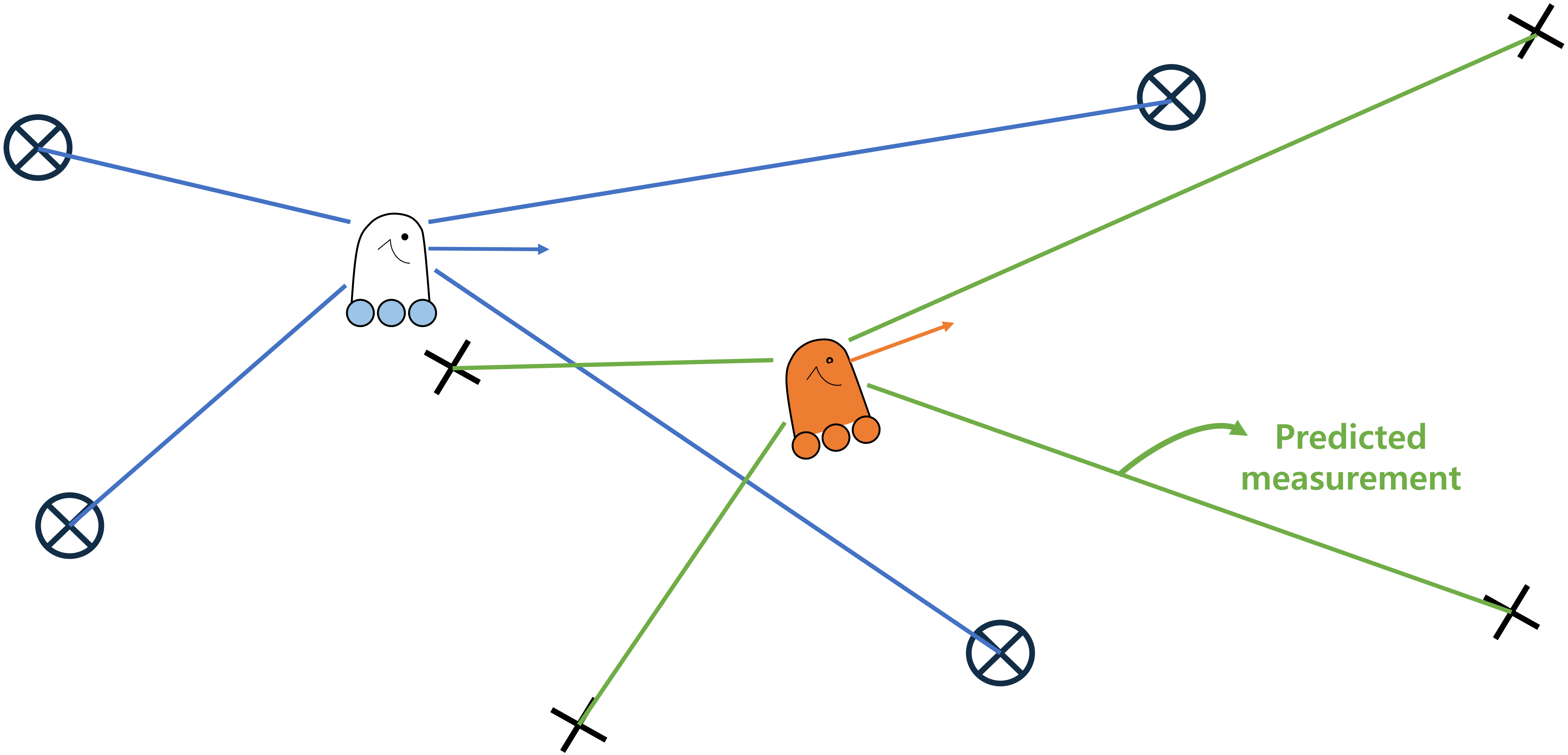
Create Particles



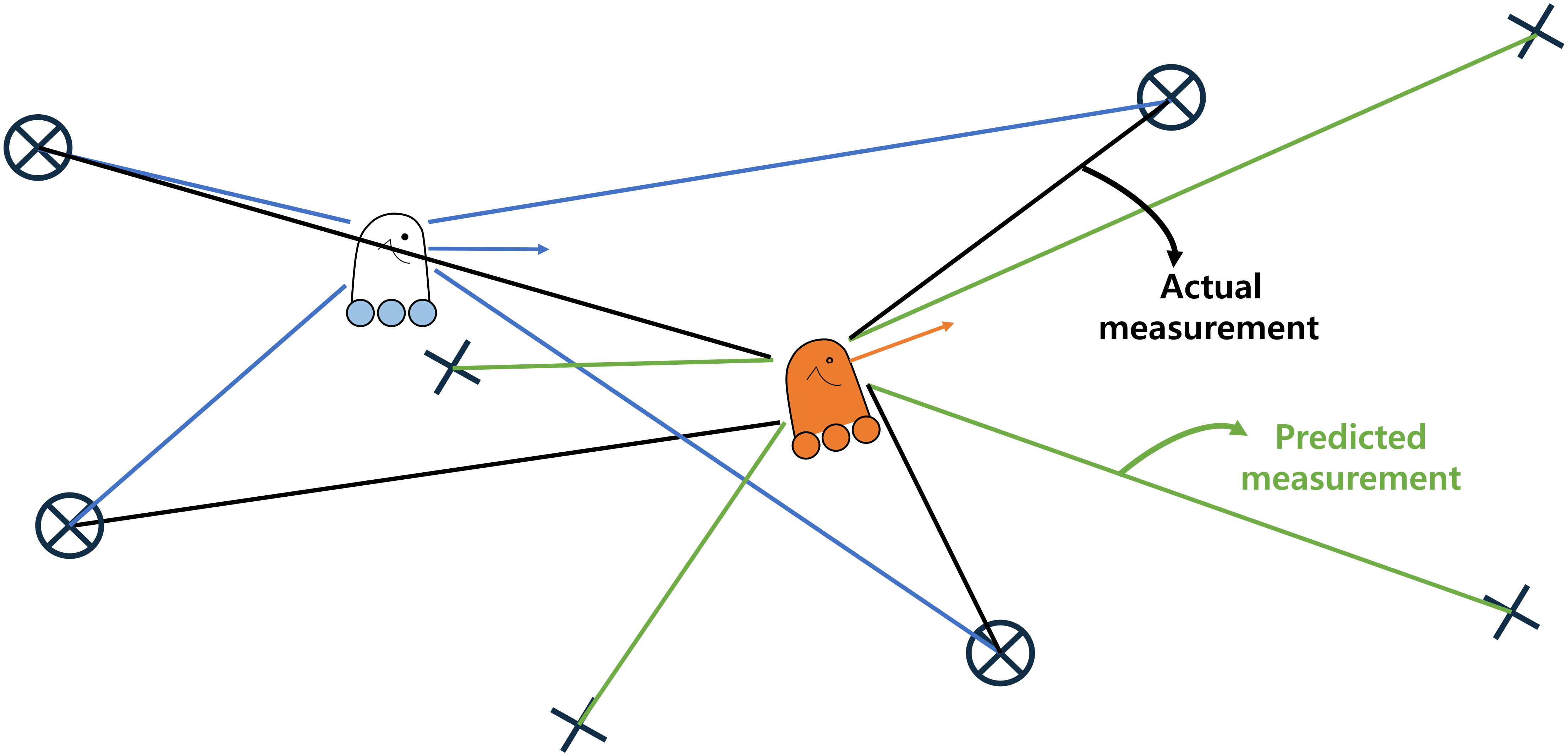
Importance Sampling



Importance Sampling



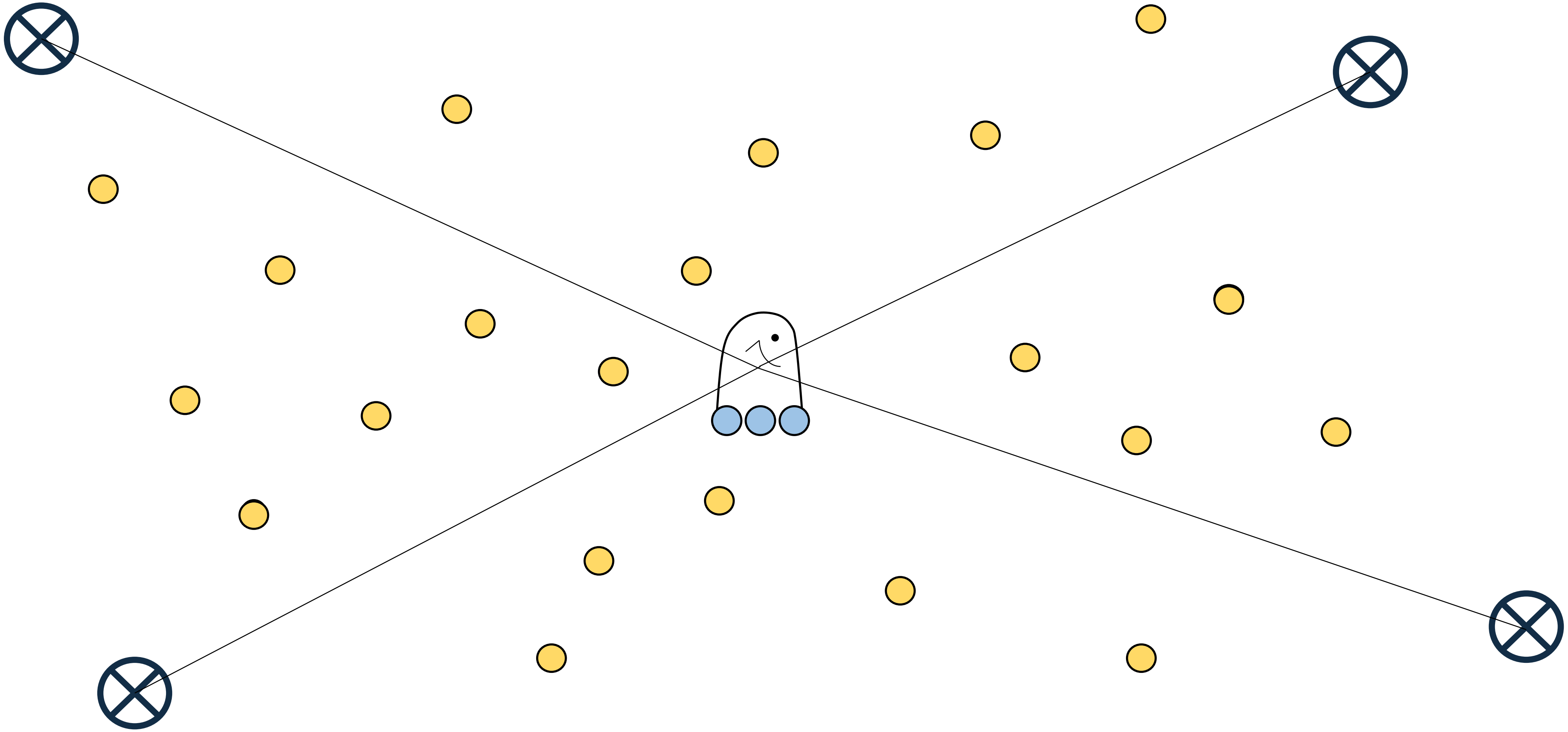
Importance Sampling



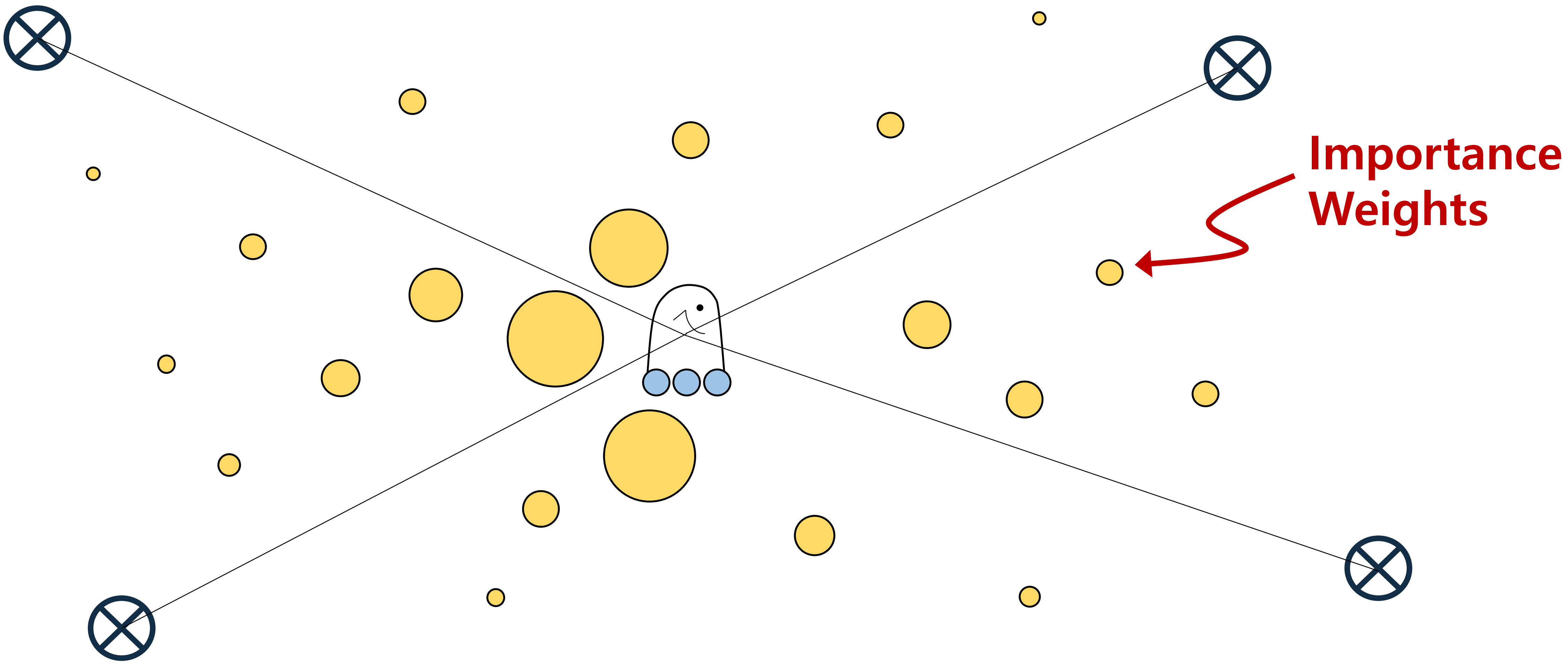
Importance Sampling

Weight

Importance Sampling



Importance Sampling

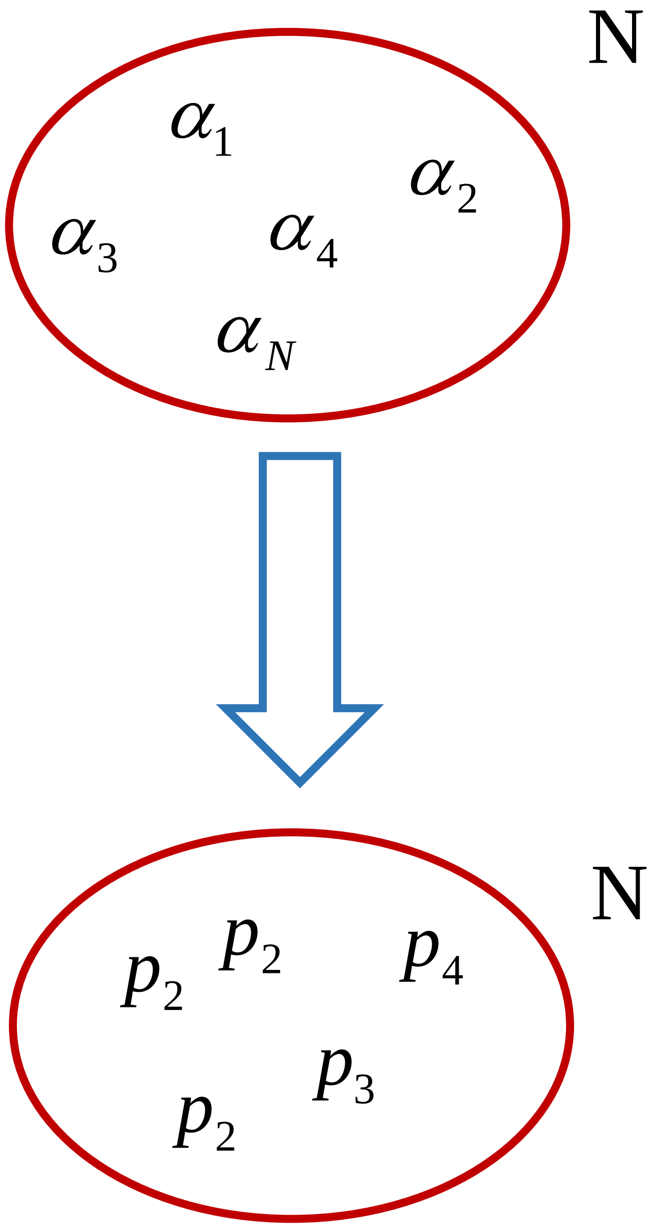


Importance Sampling

Resampling

Resampling

	Particles	Weights	Normalized Weight
N	(x_1, y_1, θ_1)	ω_1	$\alpha_1 = \frac{\omega_1}{W}$
	(x_2, y_2, θ_2)	ω_2	$\alpha_2 = \frac{\omega_2}{W}$
	$\vdots$	$\vdots$	
	(x_N, y_N, θ_N)	ω_N	$\alpha_N = \frac{\omega_N}{W}$
		$W = \sum_i^N \omega_i$	$\sum_i^N \alpha_i = 1$



Quiz 1-1

	Particles	Weights	Normalized Weight
N = 5	p_1	$\omega_1 = 0.6$	$\alpha_1 =$
	p_1	$\omega_2 = 1.2$	$\alpha_1 =$
	p_1	$\omega_2 = 2.4$	$\alpha_1 =$
	p_1	$\omega_2 = 0.6$	$\alpha_1 =$
	p_1	$\omega_2 = 1.2$	$\alpha_1 =$
		$W = 6.0$	

Is it possible that p_1 is <u>**NEVER**</u> sampled?

☑ YES ☐ NO

Quiz 1-2

	Particles	Weights	Normalized Weight
N = 5	p_1	$\omega_1 = 0.6$	$\alpha_1 = 0.1$
	p_1	$\omega_2 = 1.2$	$\alpha_1 = 0.2$
	p_1	$\omega_2 = 2.4$	$\alpha_1 = 0.4$
	p_1	$\omega_2 = 0.6$	$\alpha_1 = 0.1$
	p_1	$\omega_2 = 1.2$	$\alpha_1 = 0.2$
		$W = 6.0$	

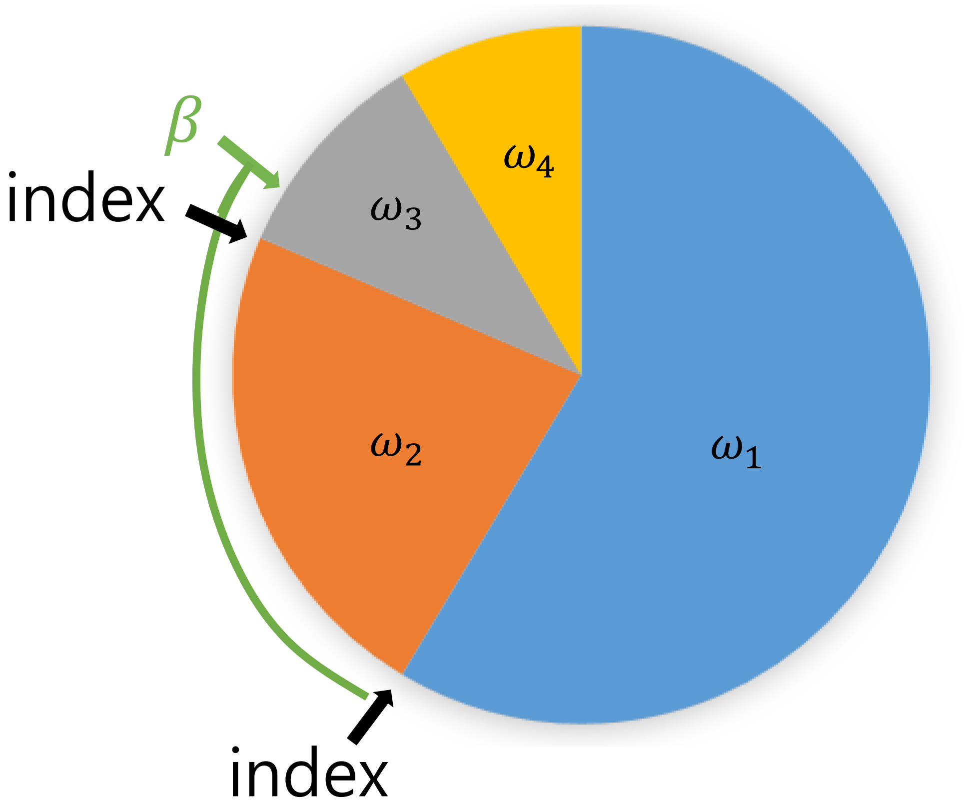
Is it possible that p_3 is **NEVER** sampled?

☑ YES ☐ NO

What is the probability of NEVER sampling p_3?

0.0777

Resampling Wheel



index = U[1 … N]

β = 0

for i = 1…N

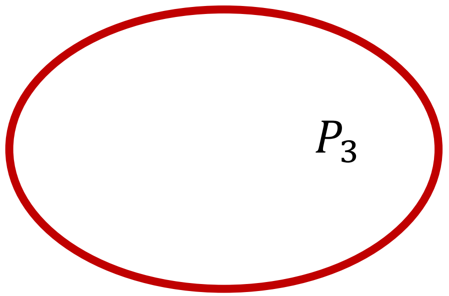
  β = β + U[0 … 2 * ω_{max}]

  while $\omega_{index} < \beta$

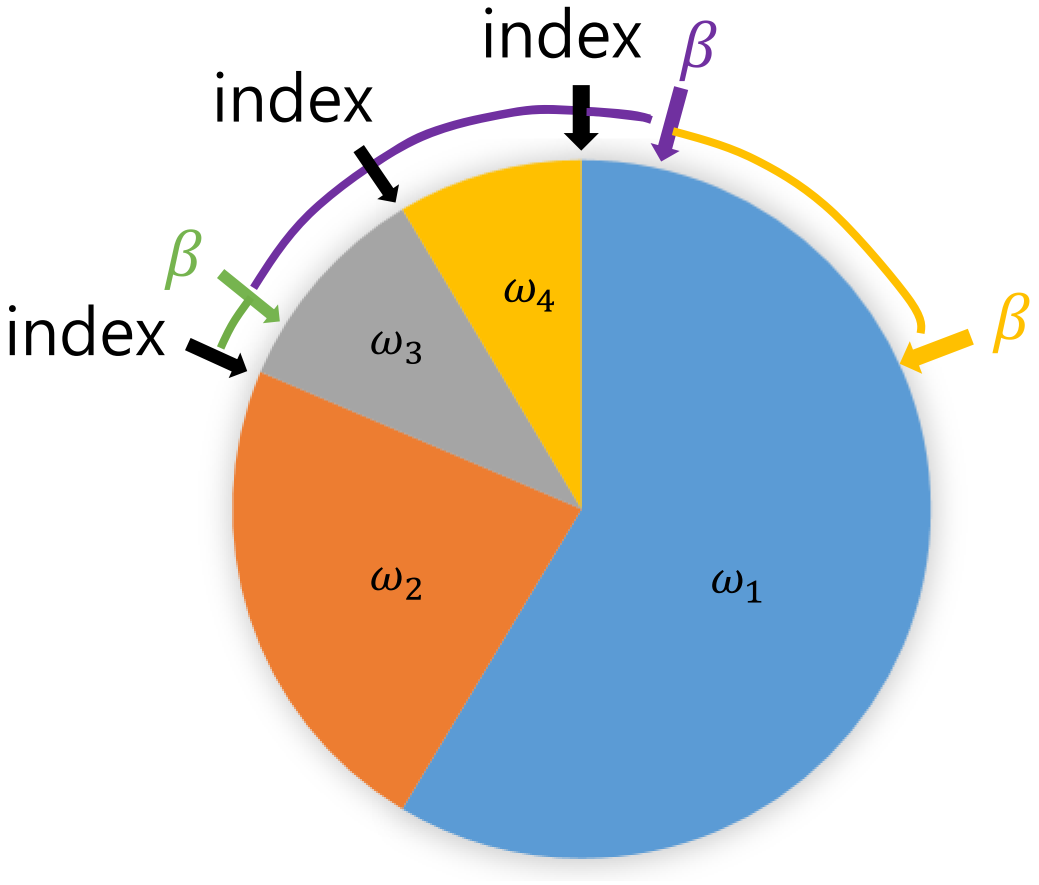
    β = β - ω_{index}

    index = index+1

  Pick P_{index}



Resampling Wheel



index = U[1 … N]

β = 0

for i = 1…N

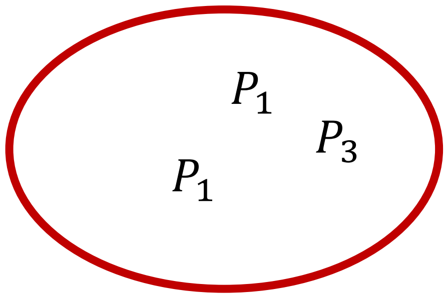
  $\beta = \beta$ + U[0 … 2 * ω_{max}]

  while $\omega_{index} < \beta$

    $\beta = \beta - \omega_{index}$

    index = index+1

  Pick P_{index}



Implementation

```
void main()
{
    srand((unsigned)time(NULL));
    std::cout << "OpenCV Version: " << CV_VERSION << std::endl;
    viz::Viz3d mywindow("test");
    mywindow.showWidget("MyCoordinate", viz::WCoordinateSystem(100.0));


    robot myrobot;
    myrobot = myrobot.move(0.1, 5.0);
    mywindow.spinOnce();
    vector<double> Z = myrobot.sense();
    int N = 1000;
    int T = 1;//10;


    Mat point_particle(1, N, CV_32FC3);
    Mat point_robot(1, 1, CV_32FC3);

    point_robot.at<cv::Vec3f>(0, 0)[0] = myrobot.x;
    point_robot.at<cv::Vec3f>(0, 0)[1] = myrobot.y;
    point_robot.at<cv::Vec3f>(0, 0)[2] = 0.;

    mywindow.showWidget("robot", viz::WCloud(point_robot, viz::Color::red()));
    mywindow.spinOnce();
```

Create Robot

Implementation

```
vector<robot> p;
for (int i = 0; i < N; i++)
{
    robot r;
    r.set_noise(0.05, 0.05, 5.0);
    p.push_back(r);
    point_particle.at<cv::Vec3f>(0, i)[0] = r.x;
    point_particle.at<cv::Vec3f>(0, i)[1] = r.y;
    point_particle.at<cv::Vec3f>(0, i)[2] = 0;
    mywindow.showWidget("particle", viz::WCloud(point_particle, viz::Color(192, 192, 192)));
    mywindow.spinOnce();
//  Sleep(1);
}
```

Create Particle

```
for (int t = 0; t < T; t++)
{
    myrobot = myrobot.move(0.1, 5.0);
    Z = myrobot.sense();

    point_robot.at<cv::Vec3f>(0, 0)[0] = myrobot.x;
    point_robot.at<cv::Vec3f>(0, 0)[1] = myrobot.y;
    point_robot.at<cv::Vec3f>(0, 0)[2] = 0;
    mywindow.showWidget("robot", viz::WCloud(point_robot, viz::Color::red()));

    vector<robot> p2;
    for (int i2 = 0; i2 < N; i2++)
    {
```

Robot Motion & measurement Update

Implementation

```
        p2.push_back(p[i2].move(0.1, 5.0));
        point_particle.at<cv::Vec3f>(0, i2)[0] = p2[i2].x;
        point_particle.at<cv::Vec3f>(0, i2)[1] = p2[i2].y;
        point_particle.at<cv::Vec3f>(0, i2)[2] = 0;
        mywindow.showWidget("particle", viz::WCloud(point_particle, viz::Color(192, 192, 192)));
        mywindow.spinOnce();
    //  Sleep(1);
    }
    p = p2;


    vector<double> w;
    for (int i3 = 0; i3 < N; i3++)
    {
        w.push_back(p[i3].measurement_prob(Z));
    }
```

Particle Motion Update

Importance Weight

Implementation - Weight

```
double robot::Gaussian(double mu, double sigma, double r_x)
{
    // calculates the probability of x for 1-dim Gaussian with mean mu and var. sigma
    return exp(-(pow((mu - r_x), 2.0)) / pow(sigma, 2.0) / 2.0) / sqrt(2.0 * pi * pow(sigma, 2.0));
}

double robot::measurement_prob(vector<double> &measurement)
{
    // calculate the correct measurement

    double prob = 1.0;
    for (int i = 0; i < 4; i++)
    {
        double dist = sqrt(pow((x - landmarks[i][0]), 2.0) + pow((y - landmarks[i][1]), 2.0));
        prob *= Gaussian(dist, sense_noise, measurement[i]);
    }
    return prob;
}
```

Implementation

```
vector<robot> p3;
int index = int((rand() / RAND_MAX) * N);
double beta = 0.0;
double mw = *max_element(w.begin(), w.end());
for (int i4 = 0; i4 < N; i4++)
{
    beta += ((double)rand() / (double)RAND_MAX) * 2.0 * mw;
    while (beta > w[index])
    {
        beta -= w[index];
        index = (index + 1) % N;
    }
    p3.push_back(p[index]);
    point_particle.at<cv::Vec3f>(0, i4)[0] = p3[i4].x;
    point_particle.at<cv::Vec3f>(0, i4)[1] = p3[i4].y;
    point_particle.at<cv::Vec3f>(0, i4)[2] = 0;
```

Resampling

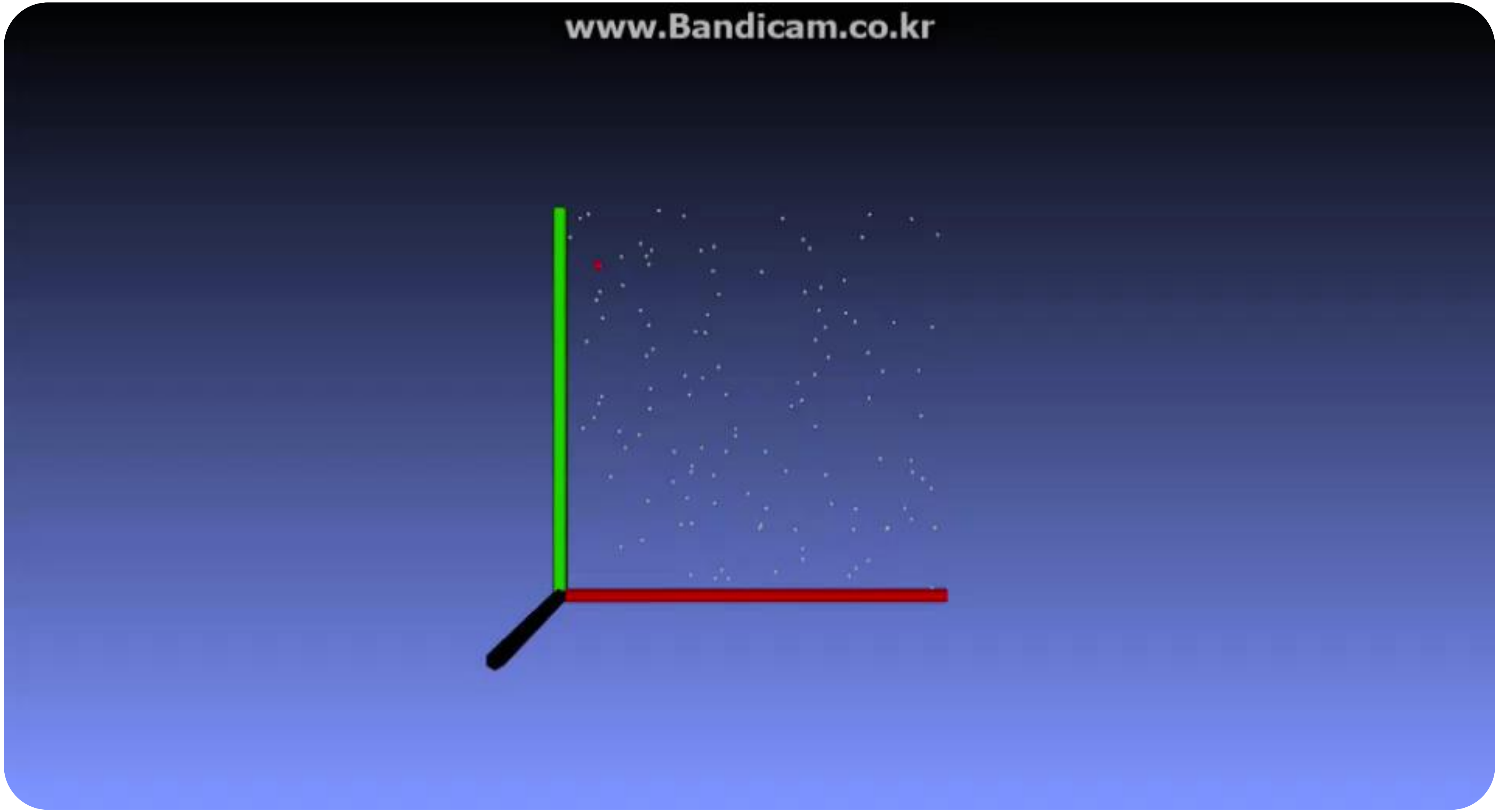
Implementation

```
            mywindow.showWidget("particle", viz::WCloud(point_particle, viz::Color(192, 192, 192)));
            mywindow.spinOnce();
        //  Sleep(1);
            cout << p3[i4].x << " , " << p3[i4].y << endl;
        }
        p = p3;
        cout << "p3.size() =  " <<p3.size() << endl;
        get_position(p);
        cout << t << "." << endl;
        cout << "Ground truth : " << myrobot.x << " " << myrobot.y << " " << myrobot.orientation << endl;
        cout << "Particle filter : " << estimated_position[0] << " " << estimated_position[1] << " " << estimated_position[2] << endl;
        cout << "eval : " << eval(myrobot, p) << endl;
        if (check_output(myrobot, estimated_position))
        {
            cout << "Code check : " << "True" << endl;
        }
        else    cout << "Code check : " << "False" << endl;
    }
    while (!mywindow.wasStopped())
    {
        mywindow.spinOnce();
    }

}
```

Pose Estimation

Result



Q&A

비교

	State space	Belief	Efficiency	In robotics
Histogram Filter	Discrete	Multimodal	Exponential	Approximate
Kalman Filter	Continuous	Unimodal	Quadratic	Approximate
Particle Filter	Continuous	Multimodal	?	Approximate

Mathematical Representation

Measurement Update

$$P(X|Z) \propto P(Z|X)P(X)$$

Motion Update

$$P(X') = \sum P(X'|X)P(X)$$