ROBIT
ROBOT SPORT GAME TEAM

Team. ROBIT | Localization_v2

2022.11.29

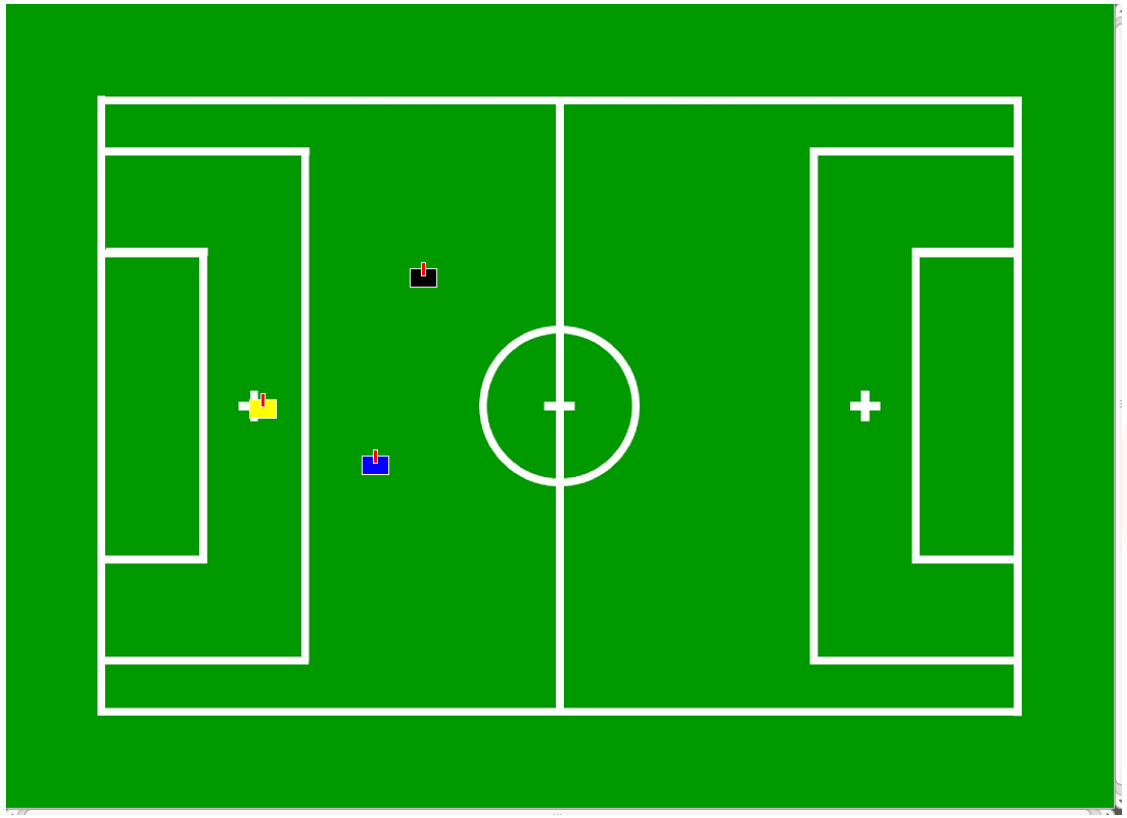
최초 노드 제작자 : 13th 김동현

작성자 : 14th 한도형, 15th 안인균

INDEX

개요

- 최근의 Robocup humanoid league에서는 로봇이 자신의 위치를 파악하는 Localization 능력을 요구한다.
- 따라서 로빛에서도 위 대회를 준비함에 있어 Vision 기반의 Localization Node를 별도로 개발하였다.
- 이 자료는 그러한 기능을 가진 localization_v2 노드에 대한 설명을 다룰 것이다.

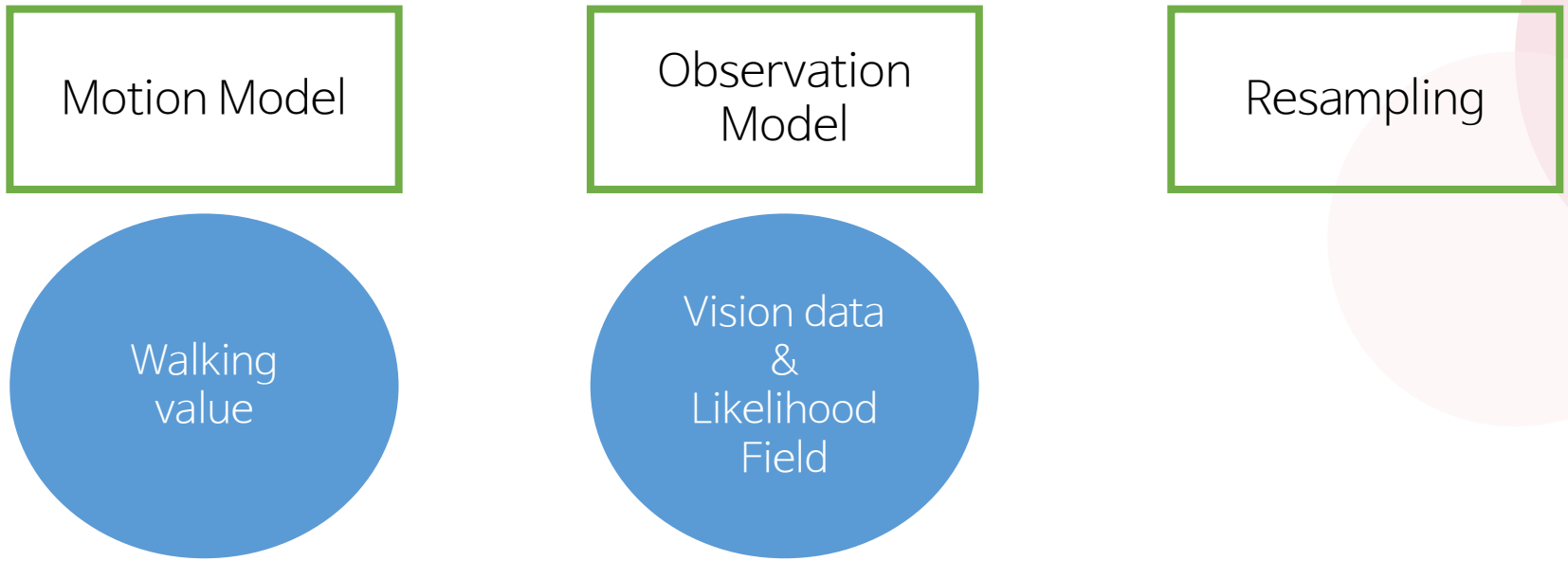


Monte-Carlo Localization(MCL)

ROBIT ROBOT SPORT GAME TEAM

- Introduction

MCL은 Motion Model, Observation Model, Resampling 으로 이뤄진 세 가지 과정을 각각의 particle에 순차적으로 적용한다. Motion Model은 초기 로봇의 위치를 알기 때문에 보행 값을 기반으로 각 particle들에게 Gaussian Noise를 적용 하였고, Observation Model은 Vision Module로부터 얻은 데이터인 경기장 라인을 나타내는 점들을 Likelihood Field를 이용하여 각 particle들에 weight를 주었다. 이후 Resampling과정에서 Observation Model을 통해 구한 weight를 이용하여 particle들을 선별한다.



Monte-Carlo Localization(MCL)

ROBIT ROBOT SPORT GAME TEAM

- Motion model

로봇의 초기 위치는 지정해 주며, IMU 센서를 통해 로봇의 현재 바라보는 방향을 얻을 수 있고, 로봇에게 주는 보행 값을 통해 로봇의 이동 거리를 알 수 있으므로 로봇이 움직였을 때 초기 위치에서부터 어디에 있는지 대략적으로 유추할 수 있다. 하지만 이러한 방식은 시간이 지날수록 정확도가 떨어지기 때문에 MCL을 통해 위치를 보정해 주는 과정을 거친다.

MCL의 Motion Model은 앞서 소개한 보행 값을 기반으로 한 Odometry-based model을 사용하였다. 그러나 로봇에게 보행 값을 주었을 때 이론적 이동 거리와 실제 이동 거리의 오차가 존재한다. 이러한 이유로 로봇을 움직일 때마다 Gaussian noise를 주어 particle들이 맵 전역으로 더욱 퍼지게 하였다.

아래 그림은 ocalization UI이다. UI에서 작은 점들은 particle이며 로봇이 있을 수 있는 곳이고, 파란색 네모는 뿌려진 particle 중 로봇이 있을 확률이 제일 높은 곳이다. 시간이 지날수록 particle들은 Gaussian noise에 의하여 맵 전역에 뿌려지는 모습을 확인할 수 있다.

Monte-Carlo Localization(MCL)

ROBIT ROBOT SPORT GAME TEAM

- Observation Model

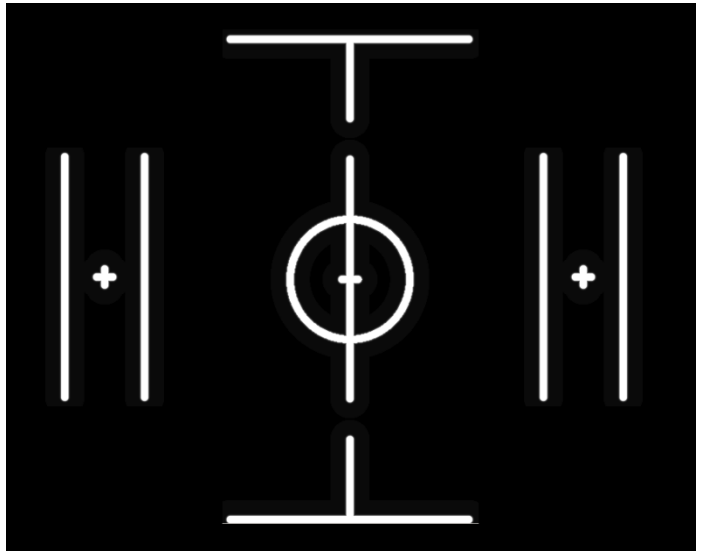
Likelihood Field는 각각의 위치에 따른 weight 값을 저장하고 있는 모델이다. 현재 모델에서는 경기장 내 흰색 라인의 중심에서 멀어질수록 낮은 weight를 갖고 있다. 그림 3.2에서 밝을수록 높은 weight가 저장되어 있고 어두울수록 낮은 weight가 저장되어 있다.

먼저 HSV 색공간을 이용하여 경기장 위에 있는 흰색 라인의 외곽을 검출하고 외곽의 선들 중 몇 개의 점들을 뽑게 된다.

흰색 라인의 외곽선 중 몇 개의 점을 뽑고 난 후에는, Tilt의 각도와 카메라 Calibration을 통해 얻은 정보를 이용하여 각 점들의 거리 정보를 얻을 수 있다. 이렇게 얻은 거리 정보를 가지고 현재 로봇이 있는 위치로부터 각 점들의 위치를 역으로 유추할 수 있게 된다.

이렇게 현재 위치에서 라인을 나타내는 점들이 어디에 있는지 알 수 있다면 Motion Model에서 뿌려 놓은 각 particle들의 위치에서도 각 점들의 위치를 유추할 수 있게 된다. 이후 Likelihood Field에서 각 점들의 위치에 따른 픽셀 값들의 곱의 결과가 한 particle의 weight가 된다.

Observation Model 에서는 모든 particle에 대하여 이 과정을 반복하여 각각의 weight를 부여하게 된다.



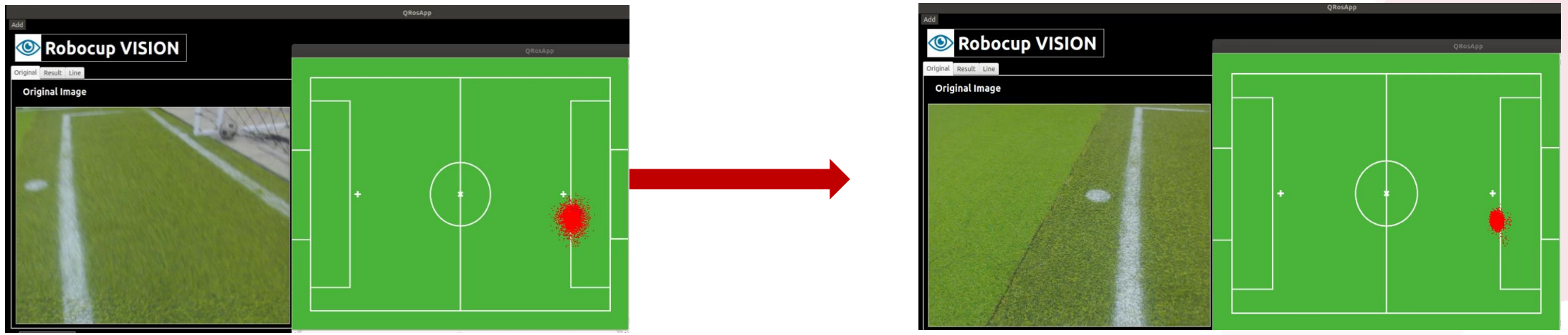
현재 사용 중인 Likelihood

Monte-Carlo Localization(MCL)

- Resampling

Motion model를 통해 particle을 뿌려 놓고, Observation model를 통해 weight를 주고 난 다음 Resampling과정을 통하여 weight가 높은 particle들이 더 많이 뽑히게 한 사진이다.

보이는 사진처럼 Motion model 만을 사용하여 로봇의 위치를 추정할 경우 시간이 갈수록 로봇의 위치가 맞지 않을 수 있으므로 위에서 말한 과정과 Resampling 과정을 통하여 로봇의 위치를 보정해 준다.



Monte-Carlo Localization(MCL)

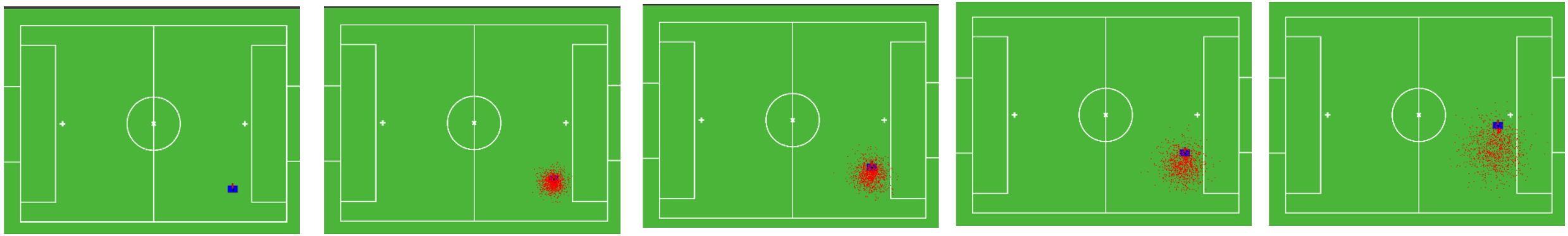
- Algorithm

1: **Algorithm MCL**($\mathcal{X}_{t-1}, u_t, z_t, m$)**:**
2: $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
3: for $m = 1$ to M do
4: $x_t^{[m]} =$ **sample_motion_model**$(u_t, x_{t-1}^{[m]})$
5: $w_t^{[m]} =$ **measurement_model**$(z_t, x_t^{[m]}, m)$
6: $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
7: endfor
8: for $m = 1$ to M do
9: draw i with probability $\propto w_t^{[i]}$ } Resampling
10: add $x_t^{[i]}$ to $\mathcal{X}_t$
11: endfor
12: return $\mathcal{X}_t$

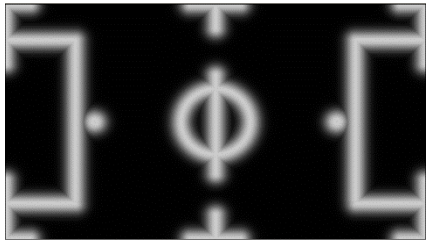
그림 3.5(Pseudocode for MCL (Thrun, Burgard, and Fox 2005)) [3]

그림 3.5는 현재 사용하고 있는 MCL 알고리즘에 대한 Pseudocode이다.
4번째 줄의 sample_motion_model은 앞서 설명했던 Motion Model 부분을 적용하며, 5번째 measurement_model은 weight를 구하는 부분으로 Observation Model에서 나온 Likelihood Field를 통해 weight를 구하게 된다.
이러한 방법은 축구 경기장과 같이 특징점(Land Mark)이 없는 환경에서도 위치 추정이 가능하다.

Localization-구현된 기능(~2021)

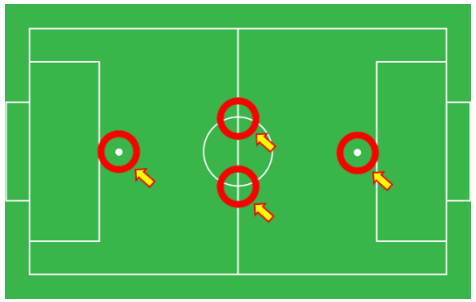


Walk_Odometry



과거 Likelihood Field

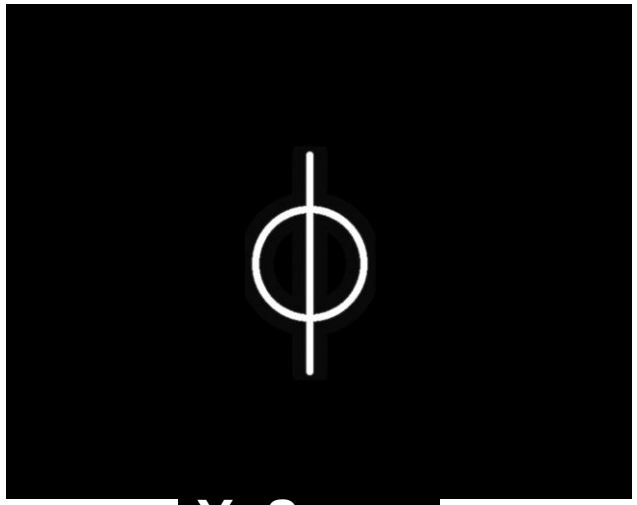
현재 Likelihood Field

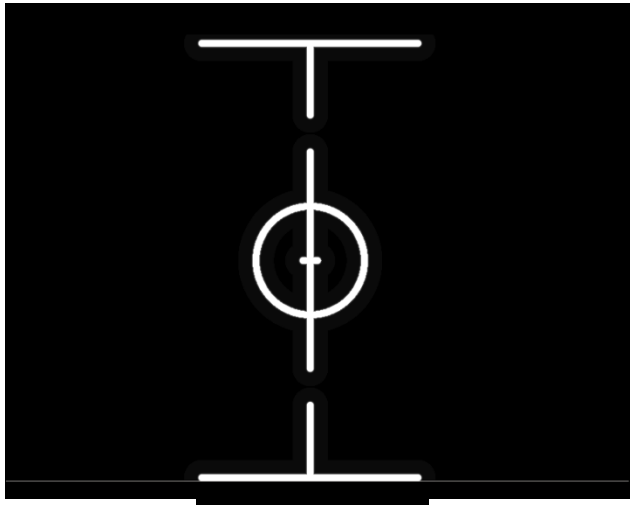


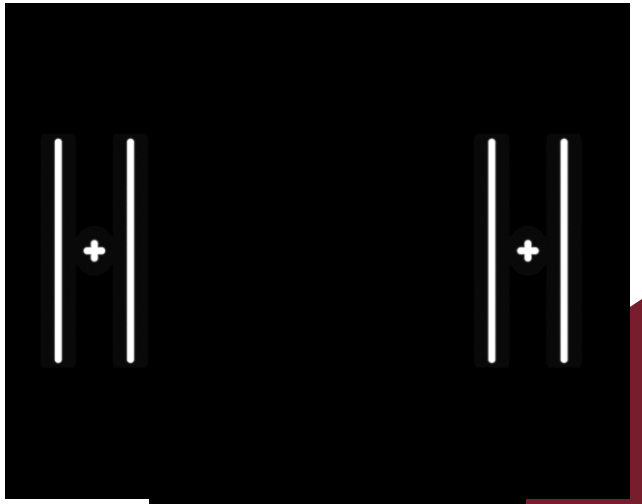
Likelihood Field 생성 및 비전 베이스 위치 보정.

Localization-새로 구현된 기능

전반적인 코드 수정
-꼬여 있던 MCL 쪽 구현 코드를 전면적으로 수정, 이론에 가깝게 변경.
보정 타이밍 변경
-기존 Walk Callback 기준에서 Vision Callback 신호 기준으로 변경.
Penalty Mark 보정 기준을 수식적 위치 추정에서 MCL 기반으로 변경
-Penalty Mark가 원에서 십자가로 변해 수식적으로 위치를 추정하는 것보다 MCL 기반 보정 방식이 더 정확하다고 판단.
Likelihood Field의 변경
-기존 Likelihood의 경우 중앙 X_Cross 영역 및 Penalty Mark 영역으로만 보정을 했지만 신규 Likelihood의 경우 중앙 T Line을 포함한 2-Lines 인식 영역을 추가했다. (비전에서 2개 이상의 라인이 인식되면 보정 진행.)


X_Cross


2-Lines


Penalty Mark

Localization-새로 구현된 기능

새로 구현된 기능(2 lines,Penalty Mark,X_cross 기반 보정) 동작 모습

Localization - Test

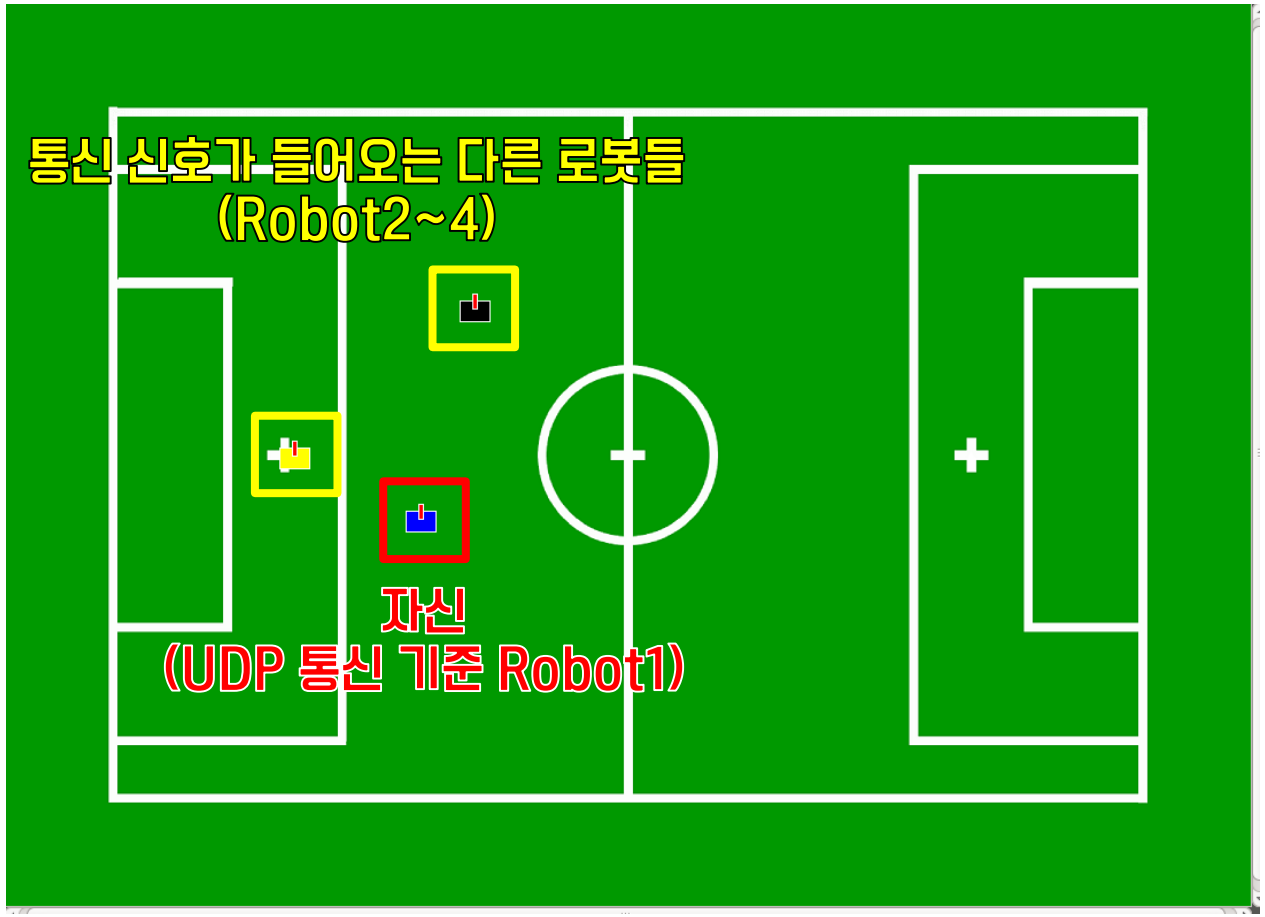
전체 테스트 진행 영상.

T자 인식.

페널티 마크 인식

5분간 중요 인식점을 포함한 경로로 공을 따라 로봇을 이동시키는 실험 수행.
최종적으로 약 5회 연속, 약 10cm~20cm 정도의 평균 오차.
(좌표 기준 약 10~20 pixel)

Localization- Coordinate Sharing



기존에는 로봇의 좌표 공유를 마스터에서만 부분 적용. (Collision)
-〉공유 되는 위치를 정확히 판단하기 어려워 디버깅이 불편
-〉로봇 간 좌표 공유 상황을 Local node UI에 표시해 디버깅을 편하게 함.
-〉공 좌표 표시 및 보정은 해봤는데 불안정해서 폐기.

Likelihood Field 수정 및 생성 방법

- 원하는 likelihood field를 얻기 위해서는 **main_window.cpp** 에서 drawField()를 수정하면 된다.

```
void MainWindow::drawField()
{

#if ROBIT_TEAM
    m_width = 850;//700;
    m_height = 500;
    nRadius =70; // center circle radius
    int Goal_area_length = 142;//92;
    int Goal_area_width = 342;//362//412;
#else
    m_width = 900;
    m_height = 600;
    int nRadius =75; // center circle radius
    int Goal_area_length = 100;
    int Goal_area_width = 500;
#endif

    m_field = new Mat(m_height, m_width, CV_8UC1);
    m_penaltyCircle = new Mat(m_height, m_width, CV_8UC1);
    m_Lcross = new Mat(m_height, m_width, CV_8UC1);
    int i = 0;
    image = imread("/home/robit/catkin_ws/src/localization_v2/img/2021korea_open_robocup_humanoid_field_robit_made_hdh.png");
    cv::resize( image, image, cv::Size( (m_width+100), (m_height+100)), 0, 0, CV_INTER_NN );
```

주의! 맵 전체 크기는 정확히 설정할 것

Likelihood Field 수정 및 생성 방법

ROBIT ROBOT SPORT GAME TEAM

- 각 line을 주석으로 구별해 두었기 때문에 주석을 확인하며 원하는 길이로 설정하면 된다.

```
vector<cv::Point> point;
/************************* START 0824 modified by dohyeong **********************/
/************************* This part is a likelihood field of full field **********************/
for(i = 0; i < 51; i++) // Top Horizontal_left
{
    m_field->data[m_field->cols*0 + i] = 255;
    point.push_back(cv::Point(i,0));
}

for(i = 373; i < 476; i++) // Top Horizontal_middle
{
    m_field->data[m_field->cols*0 + i] = 255;
    point.push_back(cv::Point(i,0));
}

for(i = 798; i < m_width; i++) // Top Horizontal_right
{
    m_field->data[m_field->cols*0 + i] = 255;
    point.push_back(cv::Point(i,0));
}

for(i = 0; i < 51; i++) // Bottom Horizontal_left
{
    m_field->data[m_field->cols*(m_field->rows-1) + i] = 255;
    point.push_back(cv::Point(i,m_field->rows-1));
}

for(i = 373; i < 476; i++) // Bottom Horizontal_middle
{
    m_field->data[m_field->cols*(m_field->rows-1) + i] = 255;
    point.push_back(cv::Point(i,m_field->rows-1));
}

for(i = 798; i < m_width; i++) // Bottom Horizontal_right
{
    m_field->data[m_field->cols*(m_field->rows-1) + i] = 255;
    point.push_back(cv::Point(i,m_field->rows-1));
}

for(i = 1; i < 52; i++)//center vertical_top
{
    m_field->data[m_field->cols*i + m_field->cols/2] = 255;
    point.push_back(cv::Point( m_field->cols/2,     i));
}
/************************* END 0824 modified by dohyeong **********************/

for(i = 150; i < 362; i++)//center vertical_middle
{
    m_field->data[m_field->cols*i + m_field->cols/2] = 255;
    point.push_back(cv::Point( m_field->cols/2,     i));
}

/************************* START 0824 modified by dohyeong **********************/
/************************* This part is a likelihood field of full field **********************/
    for(i = 448; i < m_height-1; i++)//center vertical_bottom
    {
        m_field->data[m_field->cols*i + m_field->cols/2] = 255;
        point.push_back(cv::Point( m_field->cols/2,     i));
    }
```

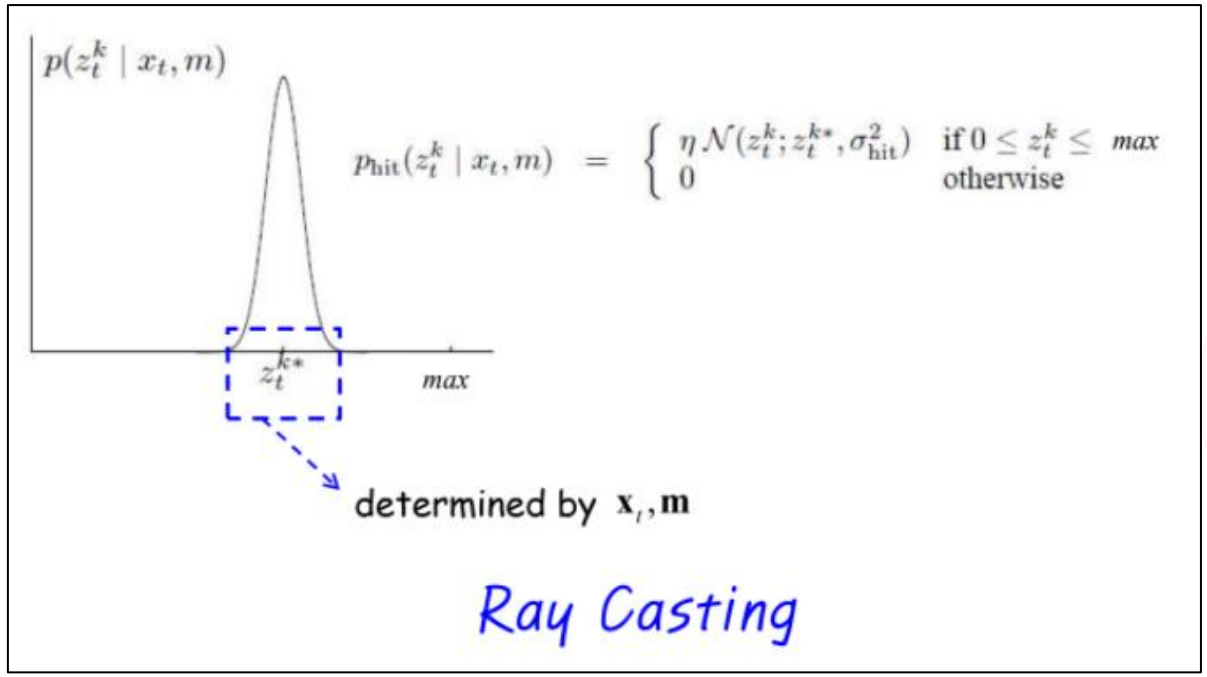
Likelihood Field 수정 및 생성 방법

- likelihood field생성은 우선 패키지를 실행시킨 후, UI의 우측 아래를 본다.

Likelihood Field 수정 및 생성 방법

- p_{hit}

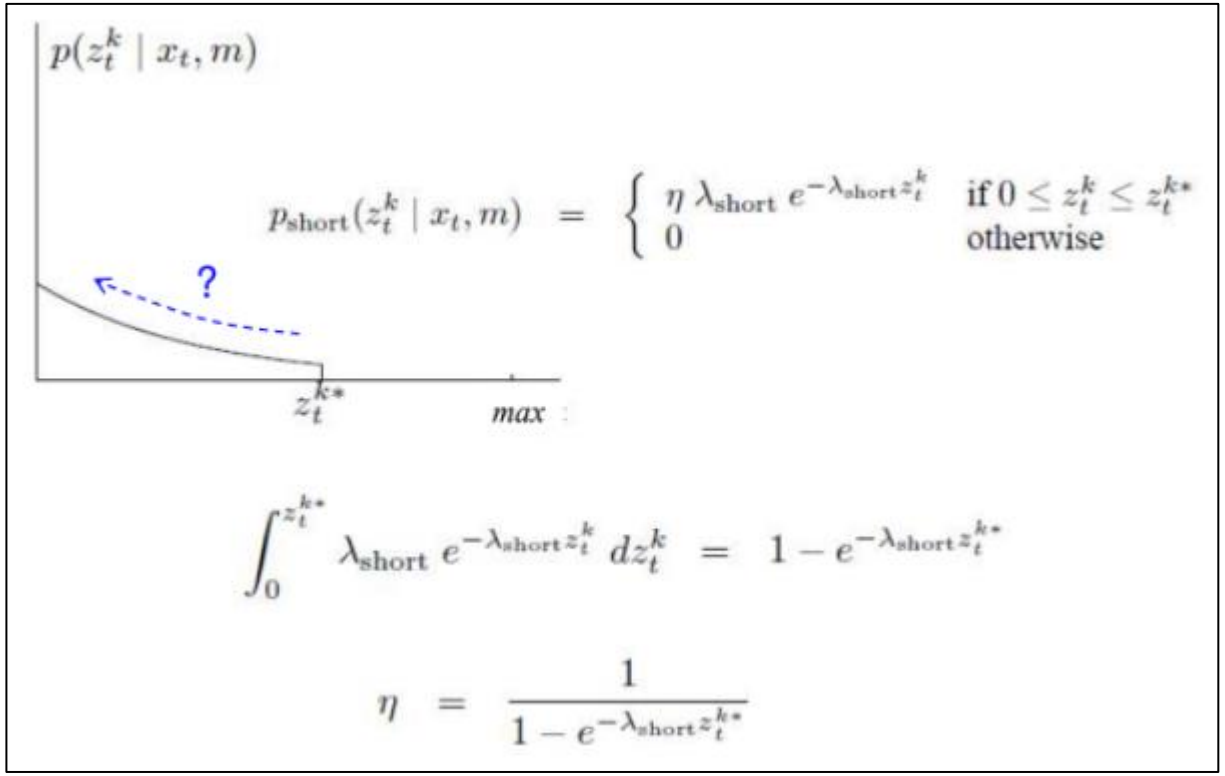
p_{hit} 은, 현재의 로봇의 위치(x_t) 와 맵의 정보(m)을 알고 있기 때문에
현재 바라보는 각도에서 대략적인 거리 즉, Ray casting(z_t^{k*})을 알 수 있다. z_t^{k*}을 평균으로 두고, ${\sigma^2}_{hit}$(뒤에서 설명)을 분산으로 설정하여 가우시안 분산을 구하여 p_{hit}을 구한다.



Likelihood Field 수정 및 생성 방법

- p_{short}

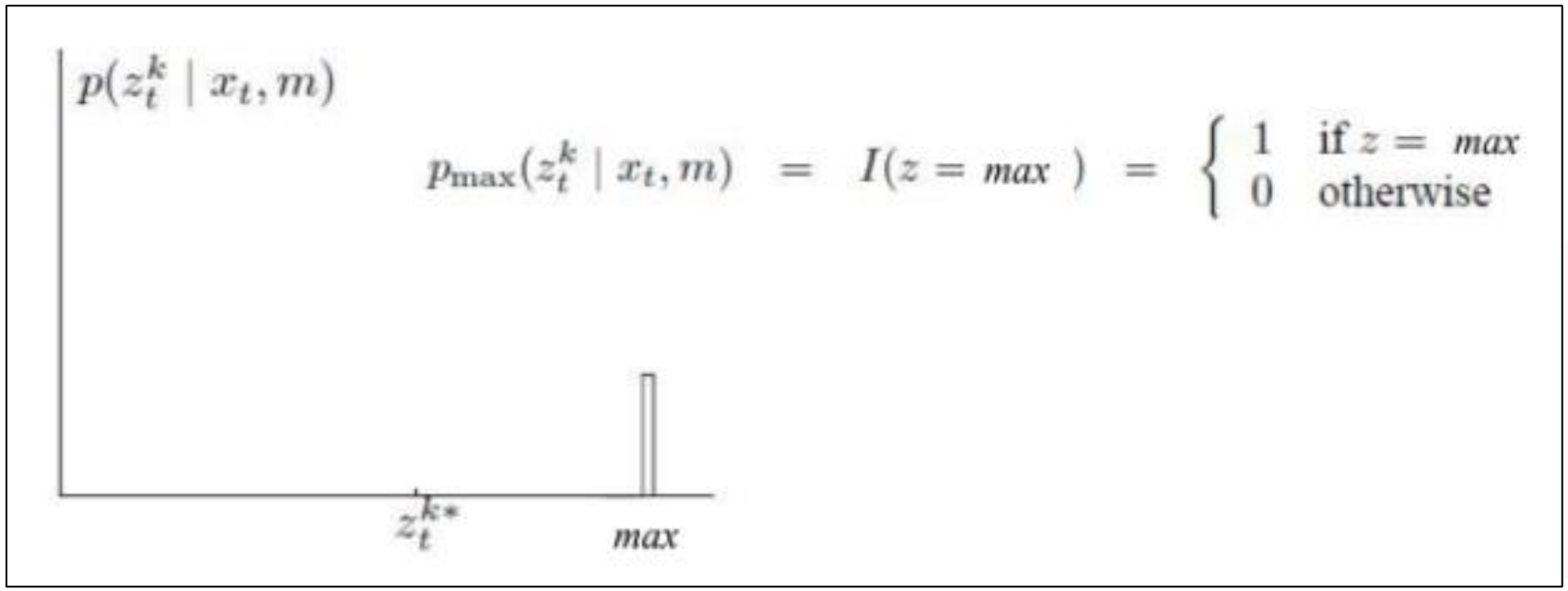
p_{short} 은 예측되는 거리 보다 더 짧은 거리가 인식되는 경우, 즉 중간에 예상하지 못한 장애물(ex〉사람)이 존재하는 경우의 확률은 구하는 것이다. 실제 예측거리 보다 더 짧으면 예상하지 못한 장애물일 확률이 높기 때문에 거리가 짧을 수록 확률이 높고 z_t^{k*} 일 수록 확률이 낮아진다.



Likelihood Field 수정 및 생성 방법

- p_{max}

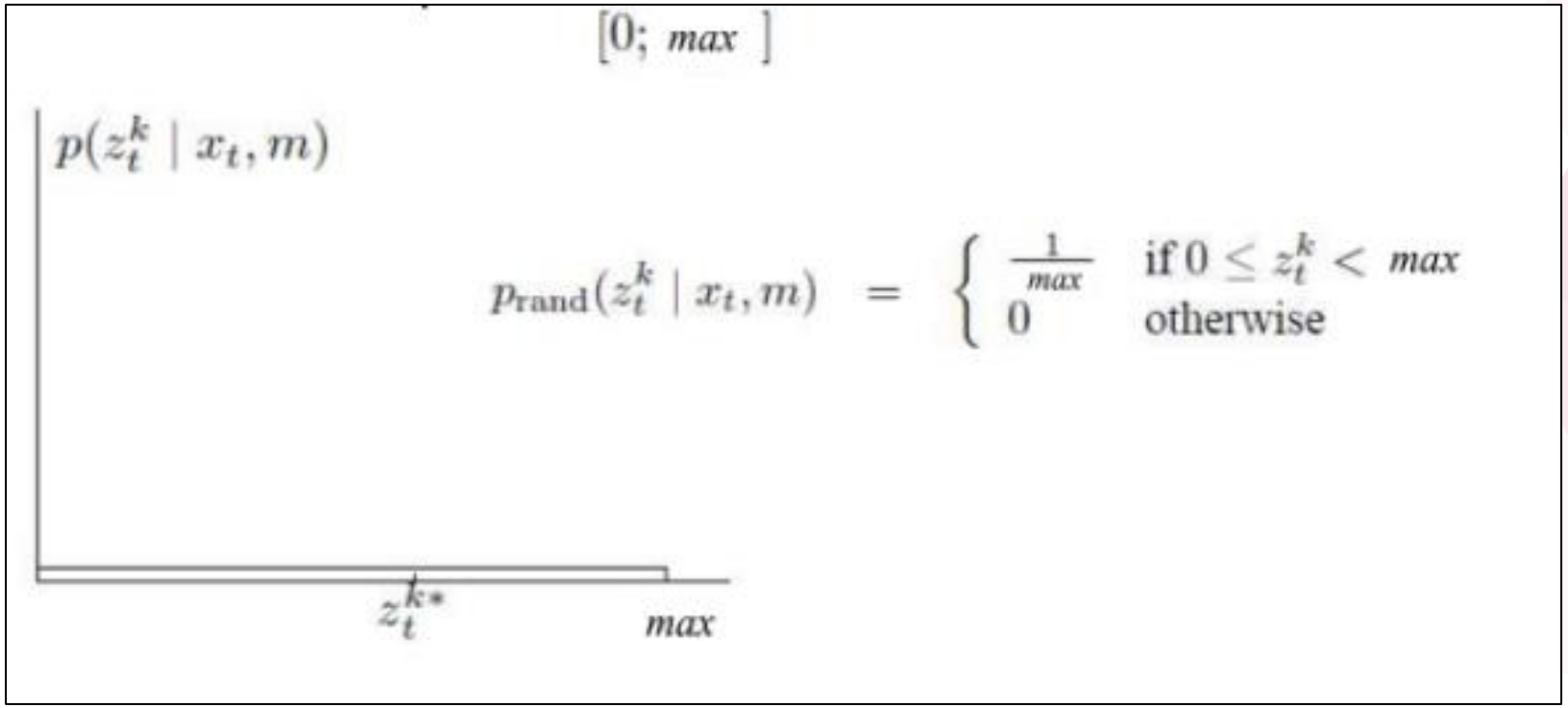
  p_{max} 은 센서가 측정가능한 범위를 벗어난 경우를 나타낸다.



Likelihood Field 수정 및 생성 방법

- p_{rand}

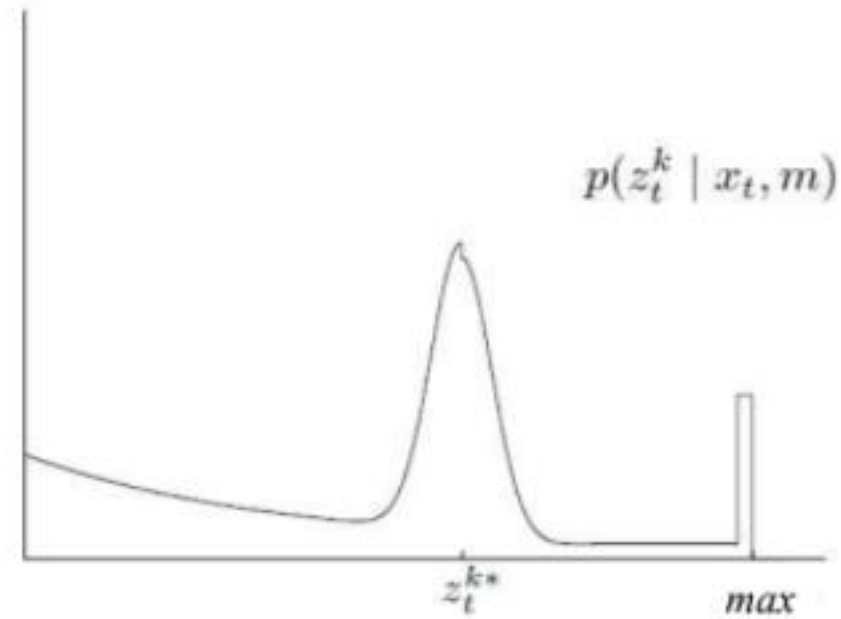
  p_{rand} 은 센서의 오류로 인해 랜덤 값이 발생하는 경우(Noise)



Likelihood Field 수정 및 생성 방법

- $z_{hit}, z_{short}, z_{max}, z_{rand}$ (weighted mean으로 표시) (확률 분포의 비중을 어떻게 둘지 정하는 부분)

  $p(z_t^k|x_t, m)$ 확률 분포는 다음의 식을 만족한다.



$$p(z_t^k \mid x_t, m) = \begin{pmatrix} z_{\text{hit}} \\ z_{\text{short}} \\ z_{\text{max}} \\ z_{\text{rand}} \end{pmatrix}^T \cdot \begin{pmatrix} p_{\text{hit}}(z_t^k \mid x_t, m) \\ p_{\text{short}}(z_t^k \mid x_t, m) \\ p_{\text{max}}(z_t^k \mid x_t, m) \\ p_{\text{rand}}(z_t^k \mid x_t, m) \end{pmatrix}$$

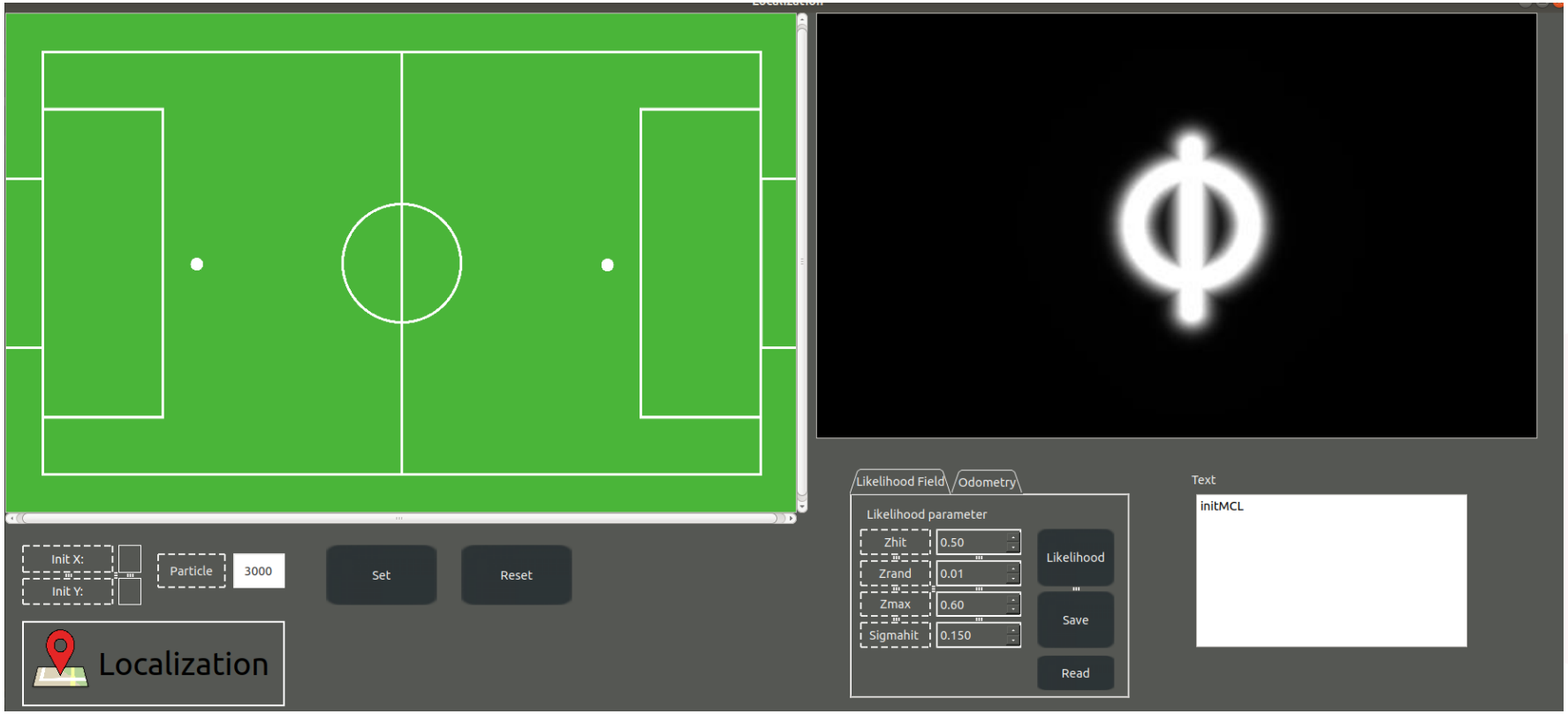
$$z_{\text{hit}} + z_{\text{short}} + z_{\text{max}} + z_{\text{rand}} = 1$$

Likelihood Field 수정 및 생성 방법

- $z_{hit}, z_{short}, z_{max}, z_{rand}$을 설정한 후 Likelihood 버튼을 누른다. (2~3분 소요될 수 있다.)

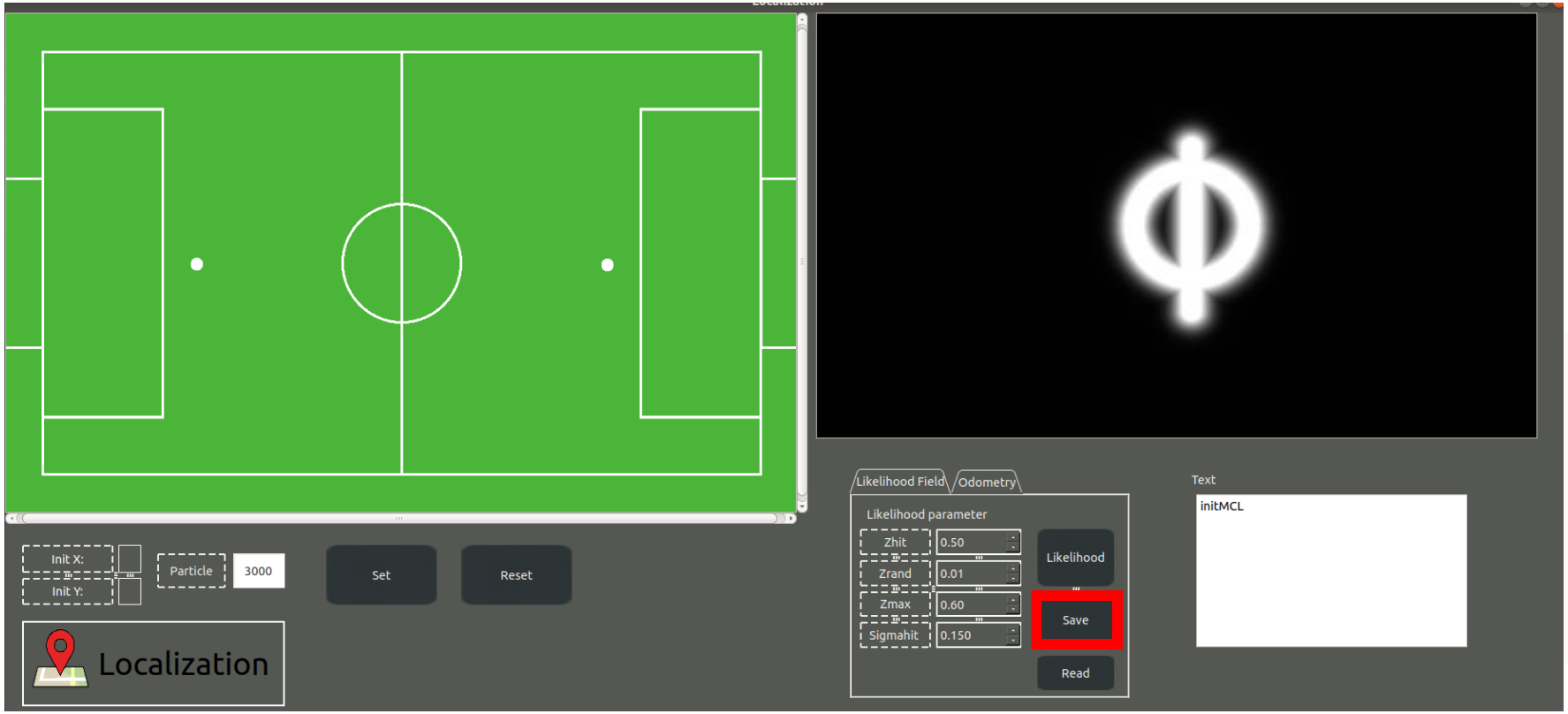
Likelihood Field 수정 및 생성 방법

- Likelihood Field가 생성된 것을 확인 할 수 있다.
- 단, 너무 밝게 생성하는 것은 weight가 그 만큼 커서 로봇이 한번에 다른 위치로 크게 이동하는 문제가 발생할 수 있다



Likelihood Field 수정 및 생성 방법

- Likelihood Field가 생성이 되면, save 버튼을 눌러 파일을 저장한다.
  - 파일을 저장하면 1)Penlaty Circle 2)X_Cross 3)나머지 부분의 3가지 Likelihood로 나뉘어 저장됨.



Likelihood Field 수정 및 생성 방법

- 저장된 Likelihood Field 파일을 불러올려면, Read 버튼을 눌러 원하는 파일을 선택하여 사용하면 된다.

Z_MAX설정

- main_window.cpp에서 ZMAX를 설정한다.
- Z_max는 vision노드에서 보낸 point들의 거리가 Z_max이상이면 오차값으로 간주하여 사용하지 않는다.

```
    *****************************************************

#include <QtGui>
#include <QMessageBox>
#include <iostream>
#include "../include/localization_v2/main_window.hpp"

/*****************************************************
    ** Namespaces
    *****************************************************
#define ROBIT 21
#define ROBIT_TEAM 1
#define ADD 1
#define deg(r)  (180.0/M_PI) * r // radian to degree
#define rad(d)  (M_PI/180.0) * d // degree to radian
//#define ROBIT                 37      //37

#define ROBOT_WIDTH     80
#define ROBOT_HEIGHT    80

#define ZMAX            120 // cm
#define TEMPSIZE        10
```

```
                theta.push_back(th);
            }
        }
        for(int i = 40; i < theta.size(); i+=30)
        {


            for(int i = 0; i < dist.size(); i++)
            {
                if(dist[i]>ZMAX) continue;   // cm
                point.x = robot_MCL->x - (dist[i])*sin(rad(theta[i]));
                point.y = robot_MCL->y - (dist[i])*cos(rad(theta[i]));
                Points.push_back(point);
            }

            //              if(qnode.localMsg.penaltyCircleDist.size()!=0) only_one
            //              if(qnode.localMsg.xcrossDist[0]  < 1100 && qnode.localMs
            //              if(only_one == false){
            MCL->measurementMCL(dist, theta, *likelihood);//pointdist를 적용하는 부분
            //              re  }
            for(int i =0;i<10;i++) MCL->resamplingMCL();
            RobotPose robotMCL = MCL->getRobotPose();
            robot_MCL->x        = robotMCL.x;
            robot_MCL->y        = robotMCL.y;
            robot_MCL->theta    = robotMCL.theta;
            for(int i = 0; i < dist.size(); i++)
            {
                if(dist[i]>ZMAX) continue;   // cm
                Point2d visualPoint;
                visualPoint.x = robot_MCL->x - (dist[i])*sin(rad(theta[i]));
                visualPoint.y = robot_MCL->y - (dist[i])*cos(rad(theta[i]));
                visualPoints.push_back(visualPoint);
            }
            break;

        }
    }
```

qnode_odometry

- 로봇의 보행 변경, 로봇 변경 시 **필수적**으로 바꿔야하는 부분이다
- 로봇을 필드상에서 일정거리를 일직선으로 걷게 한 후, localization_v2에서의 로봇 값이 일치하도록 값을 수정해준다.
- Ymove도 일정거리를 일직선으로 옆 걸음하게 한 후, localization_v2에서 로봇이 일치하도록 값을 수정한다.

```
void QNode::coordinateCallback(const msg_generate::ikcoordinate_msg::ConstPtr &msg)
{

    Xmoved = (double)msg->X*0.15;
    Ymoved = (double)msg->Y*0.14;
    Q_EMIT step();
}
```

Particle filter

- Particle filter 개념에 대한 설명을 하기 전에 우선, Bayes filter에 대해 소개하겠다.
- Bayes filter는 data들을 기반으로 현재의 위치를 확률적으로 표현한 것을 말한다.

$$Bel(x_t) = p(x_t \mid d_{0\ldots t})$$

Belief or posterior

Robot state

At time t

The data from time 0 to t.

Particle filter

- Bayes filter는 총 2가지의 데이터로 구분할 수 있다.

1) Perceptual data : lidar에서 얻은 데이터 등을 의미한다.

2) Odometry data : 로봇이 이동한 데이터를 의미한다.

$$Bel(x_t) = p(x_t \mid o_t, a_{t-1}, o_{t-1}, \ldots)$$

Belief or posterior

Robot state

Perceptual Observation

Odometry

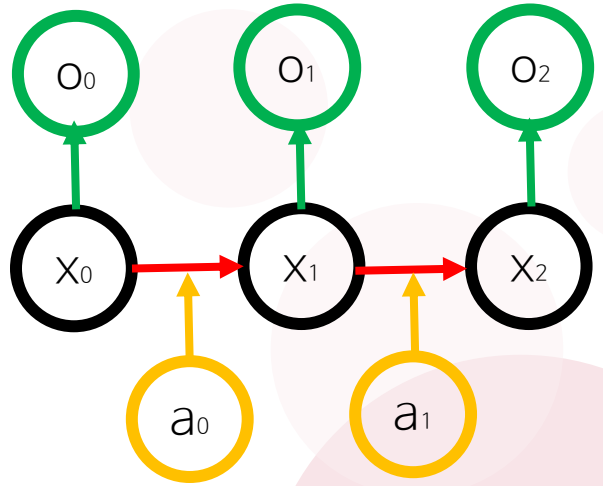
History...

Particle filter

- Bayes filter는 총 2가지의 데이터로 구분할 수 있다.

1) Perceptual data : lidar에서 얻은 데이터 등을 의미한다.

2) Odometry data : 로봇이 이동한 데이터를 의미한다.



$$Bel(x_t) = p(x_t \mid o_t, a_{t-1}, o_{t-1}, \ldots)$$

Belief or posterior

Robot state

Perceptual Observation

Odometry

History...

Particle filter

- Bayes filter식을 정리하면 다음과 같다.

$$p(A \mid B) = \frac{p(B \mid A)p(A)}{p(B)} = \frac{P(A,B)}{P(B)} = P(A \cap B)$$

$$Bel(x_t) = p(x_t \mid o_t, a_{t-1}, o_{t-1}, \dots)$$

$$\frac{P(x_t, o_t, a_{t-1}, o_{t-1}, \cdots) / P(a_{t-1}, o_{t-1}, \cdots, a_0, o_0)}{P(o_t, a_{t-1}, o_{t-1} \cdots) / P(a_{t-1}, o_{t-1}, \cdots a_0 o_0)} \rightarrow \frac{P(x_t, o_t, a_{t-1}, o_{t-1}, \cdots)}{P(a_{t-1}, o_{t-1}, \cdots, a_0, o_0)} \cdot \frac{P(x_t, a_{t-1}, o_{t-1}, \cdots, a_0, o_0)}{P(x_t, a_{t-1}, o_{t-1}, \cdots, a_0, o_0)}$$

$$Bel(x_t) = \frac{p(o_t \mid x_t, a_{t-1}, \dots, o_0)p(x_t \mid a_{t-1}, \dots, o_0)}{p(o_t \mid a_{t-1}, \dots, o_0)}$$

$$Bel(x_t) = \eta p(o_t \mid x_t, a_{t-1}, \dots, o_0)p(x_t \mid a_{t-1}, \dots, o_0)$$

Normal constance

Particle filter

- 여기서 Markov Assumption을 적용한다.
  - Markov Assumption은 바로 이전 데이터만이 현재에 영향을 준다는 내용이다.

$$p(o_t \mid x_t, a_{t-1}, \ldots, o_0) = p(o_t \mid x_t)$$

$$Bel(x_t) = \eta p(o_t \mid x_t) p(x_t \mid a_{t-1}, \ldots, o_0)$$

Particle filter

- 다음 식을 이용하여 식을 정리한다.

$$p(x) = \int p(x,y)dy = \int p(x|y)p(y)dy$$

$$p(x|y,z) = \int p(x|y,z)\,p(y|z)\,dy$$

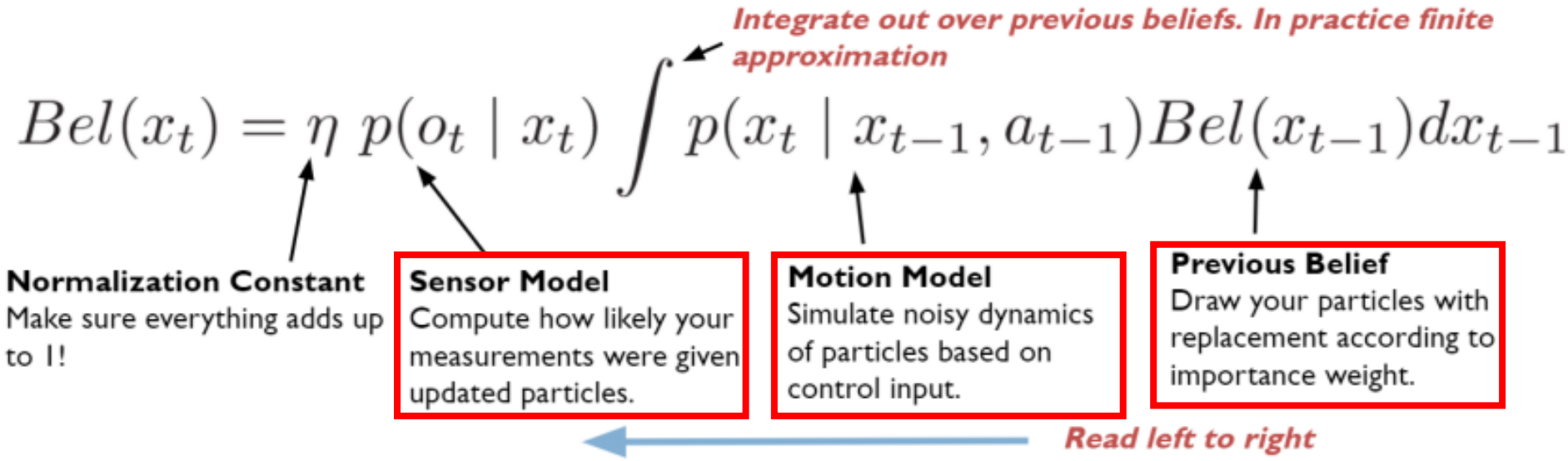
$$Bel(x_t) = p(x_t \mid x_{t-1}, a_{t-1}, \ldots, o_0)$$

$$Bel(x_t) = \int p(x_t \mid x_{t-1}, a_{t-1}, \ldots, o_0)p(x_{t-1} \mid a_{t-1}, \ldots, o_0)dx_{t-1}$$

$$Bel(x_t) = \int p(x_t \mid x_{t-1}, a_{t-1})Bel(x_{t-1})dx_{t-1}$$

Particle filter

- 최종적으로 다음의 식이 유도된다.

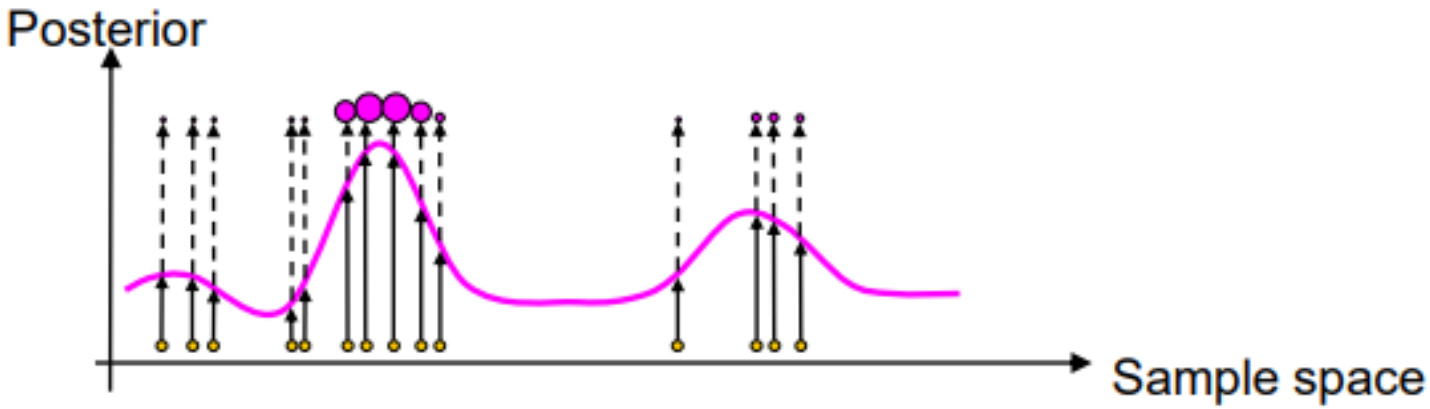


- Particle filter를 이용하여 sensor model과 motion model에 관해 계산한다.

  $Bel(x_t)$ 를 particle로 표현한다.

Particle filter

- Non-Gaussian형태로 추정한다.
- Monte Carlo filter, Particle filter라고 불른다.

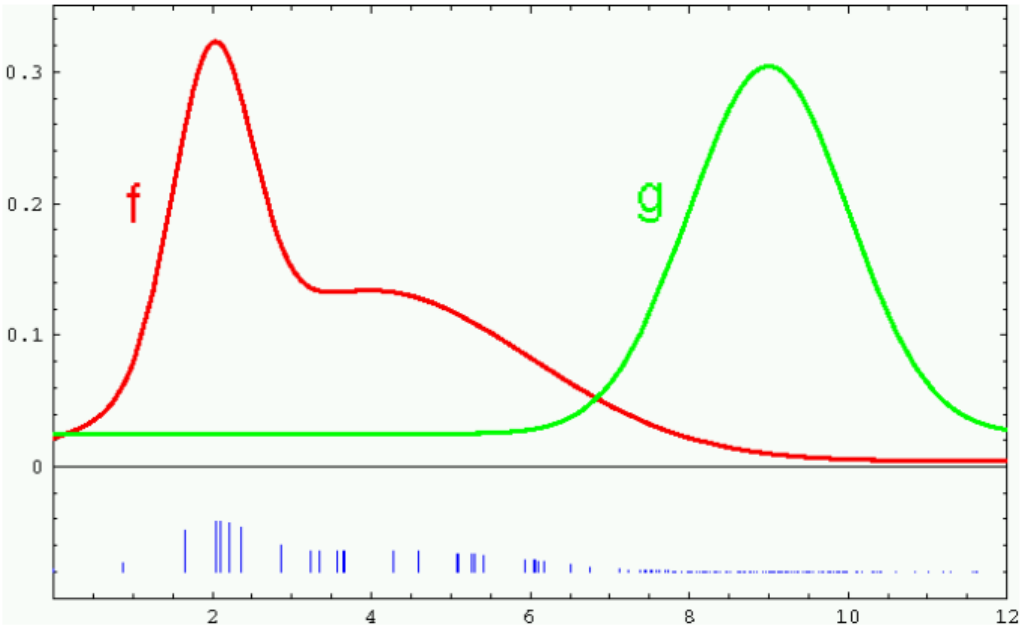


Particle filter

- 실제 f가 주어지고 현재는 f를 모른다고 가정하여 g로 설정하고 비교하면
- 값이 실제는 큰데 g는 작으면 weight sameples 값이 크게 나오고 값이 실제로는 작은데 g는 크면 값이 작게 나오게 된다.

이 과정을 계속 진행하면 결국 g가 f에 수렴하여 w가 1로 수렴하는 것을 확인 할 수 있다.

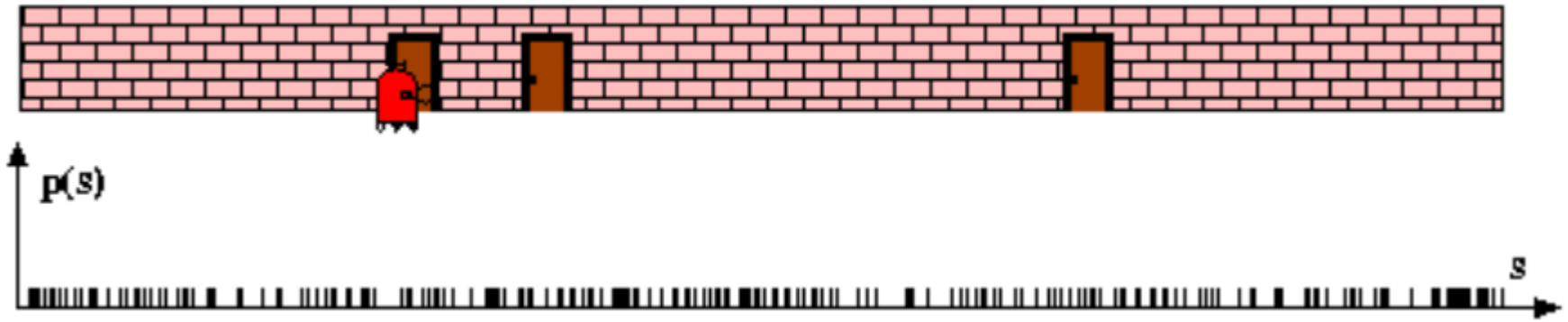
- Importance sampling



Weight samples: $w = f / g$

Particle filter

- 로봇의 위치가 현재는 불분명하기 때문에 weight가 골고루 퍼져있는 것을 확인할 수 있다.

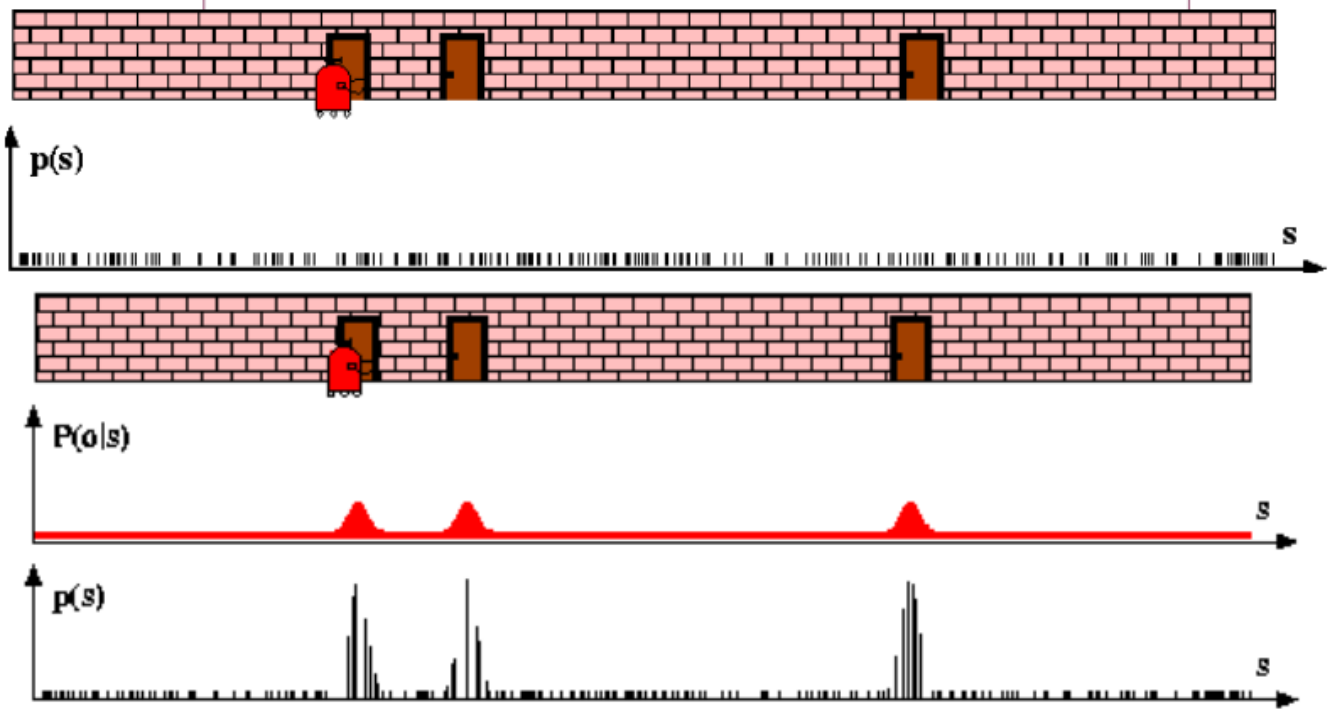


Particle filter

- 로봇 앞에 문이 있다는 것을 확인하면 문 주위의 weight가 커진다.

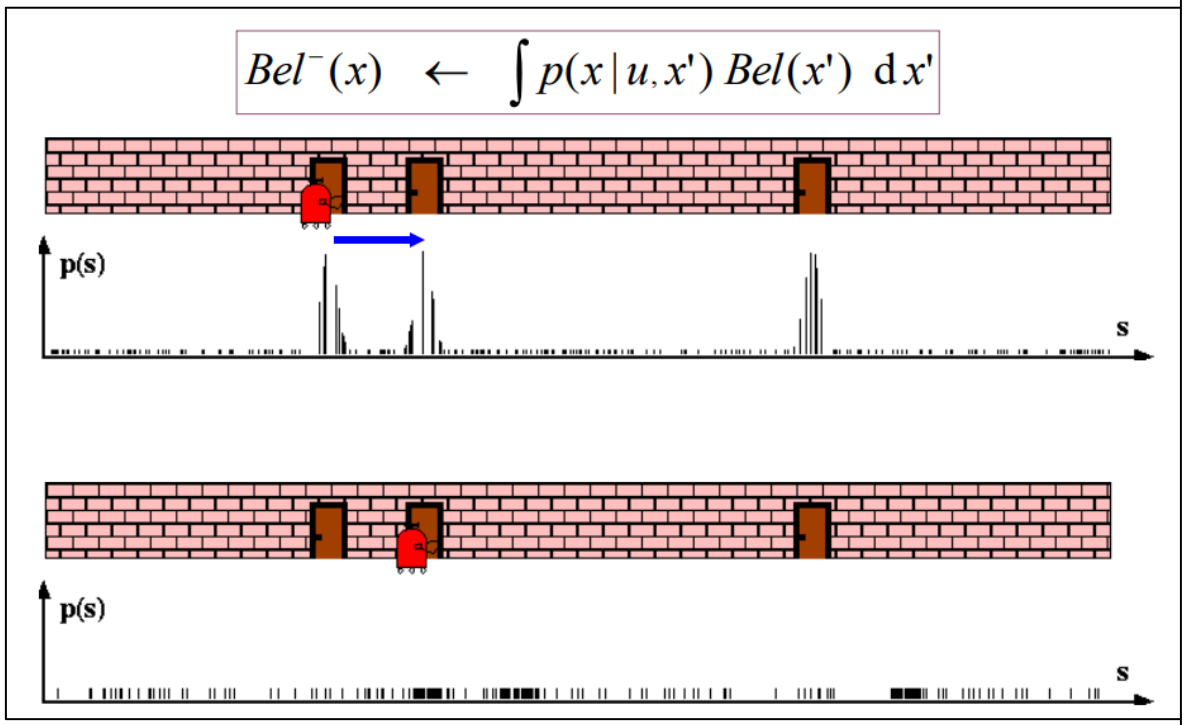
$$Bel(x) \leftarrow \alpha\ p(z \mid x)\ Bel^{-}(x)$$

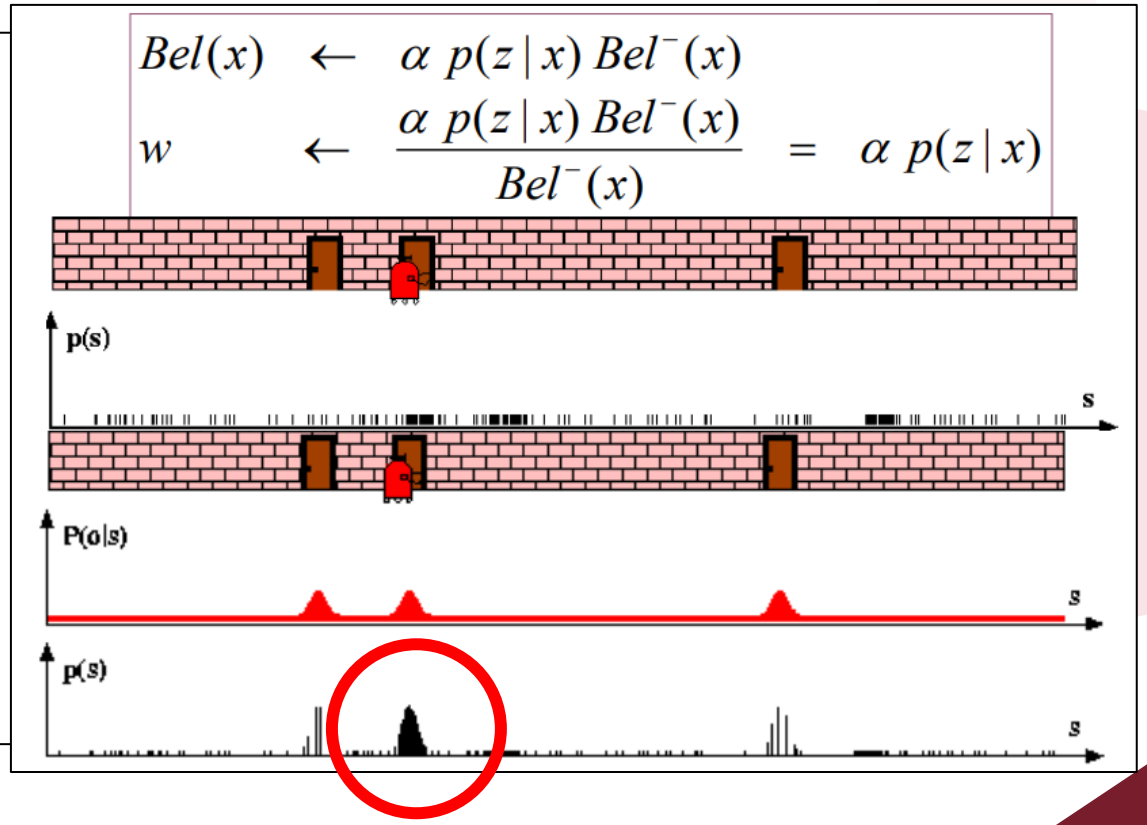
$$w \leftarrow \frac{\alpha\ p(z \mid x)\ Bel^{-}(x)}{Bel^{-}(x)} = \alpha\ p(z \mid x)$$



Particle filter

- 로봇이 옆으로 이동했을 때도 문이 존재하기 하기때문에 2번째 문의 weight가 커진다.
- 관찰횟수가 증가할수록 정확도도 증가한다.





Particle filter

- target distribution을 몰라도 importance factor를 통해 추정할 수 있기 때문에 확률 분포를 가정을 하지 않아도 된다는 장점이 있다.

$$Bel(x_t) = \eta\, p(z_t \mid x_t) \int p(x_t \mid x_{t-1}, u_{t-1})\, Bel(x_{t-1})\, dx_{t-1}$$

draw x^i_{t-1} from $Bel(x_{t-1})$

draw x^i_t from $p(x_t \mid x^i_{t-1}, u_{t-1})$

Importance factor for x^i_t:

$$w^i_t = \frac{\text{target distribution}}{\text{proposal distribution}} \quad \begin{matrix}(f)\\(g)\end{matrix}$$

$$= \frac{\eta\; p(z_t \mid x_t)\; p(x_t \mid x_{t-1}, u_{t-1})\; Bel\;(x_{t-1})}{p(x_t \mid x_{t-1}, u_{t-1})\; Bel\;(x_{t-1})}$$

$$\propto p(z_t \mid x_t)$$

앞으로 할 것.

Localization 개선 방안
-Localization Node가 경기 중에 심각한 문제를 보여주었는가?
-No, 하지만 알고리즘의 Local 의존도가 기존보다 높아짐에 따라 기존보다 더 높은 정확도를 요구.

경기 중 발생한 몇가지 문제점

-1.보행 보폭이 상향됨에 따라 보행의 이동 간격이 불안정해짐.
(특히, 전진 시 Side 보행이 비례해서 들어가 IMU는 앞을 보는 상태로 옆으로 가는 식의 보행이 발생, 오도메트리 오차가 크게 발생.)
-〉IMU 기반 위치 추정으로 변경(SLAM에서 쓰는 것과 비슷한 느낌.)-〉칼만 필터를 이용해 현재 이동량 추정.

-2.보정
실전에서는 상대 로봇 및 공의 위치에 따라 보정점을 전혀 찾지 못하는 변수가 자주 발생, 보정할 기회가 생각보다 많지 않다. 따라서 쌓인 오차 누적을 해결할 마땅한 방법이 존재하지 않는다.
-〉그렇다고 다른 보정 방법이나 보정 포인트를 늘리기에도 마땅치 않다. (오보정 확률이 올라감.)
-〉결국 새로운 보정 방법론을 시도하기 보다 애초에 정확하고 빠른 보행이 가능하게 하여 보정 포인트를 최대한 많이 지나다니게 하는게 제일 확실한 방법 인 것 같다. (그리고 오도메트리 오차를 최대한 줄이는 방향성으로 가는 것이 맞는 방향인 것 같다.)

-3.그 외 UDP 통신과의 연동 및 디버깅을 위한 기능을 좀 더 깔끔하게 수정 및 보완해야 함.

Thank you