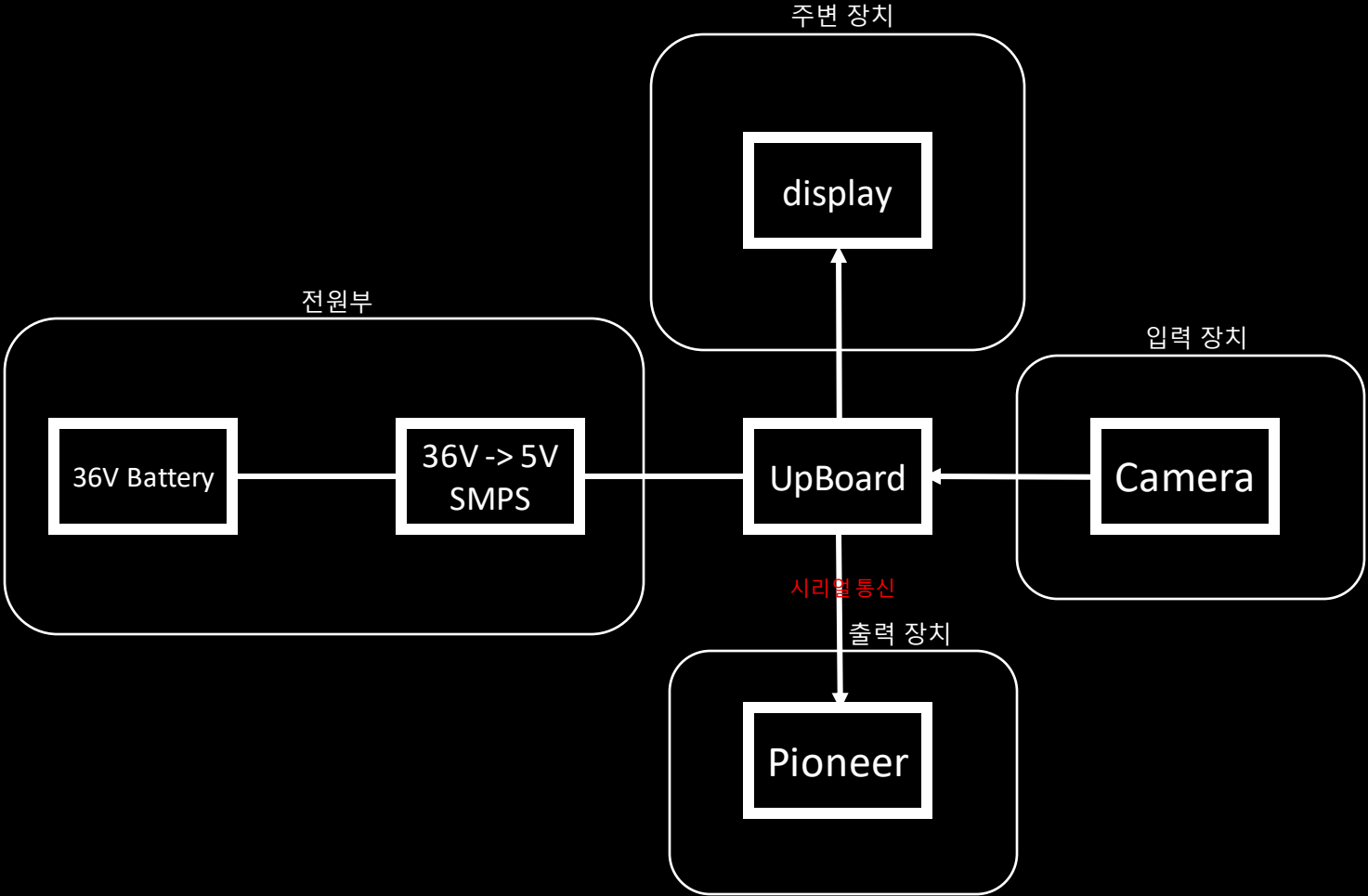


# OJT 5주차 과제 자료

조재현

하드웨어 구성



Pioneer error

10Hz cmd\_vel (linear.x = 0.1 1sec)

9.8 / 10.8 / 10 / 10.8 / 10.8 / 9.8 / 10.8 / 10 / 9.8 / 10.8

20Hz cmd\_vel (linear.x = 0.1 1sec)

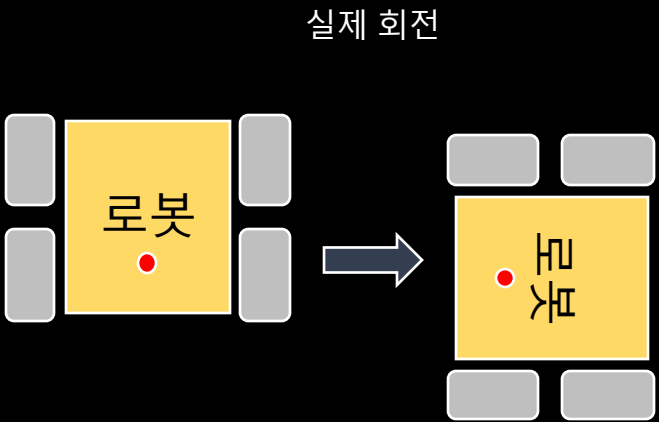
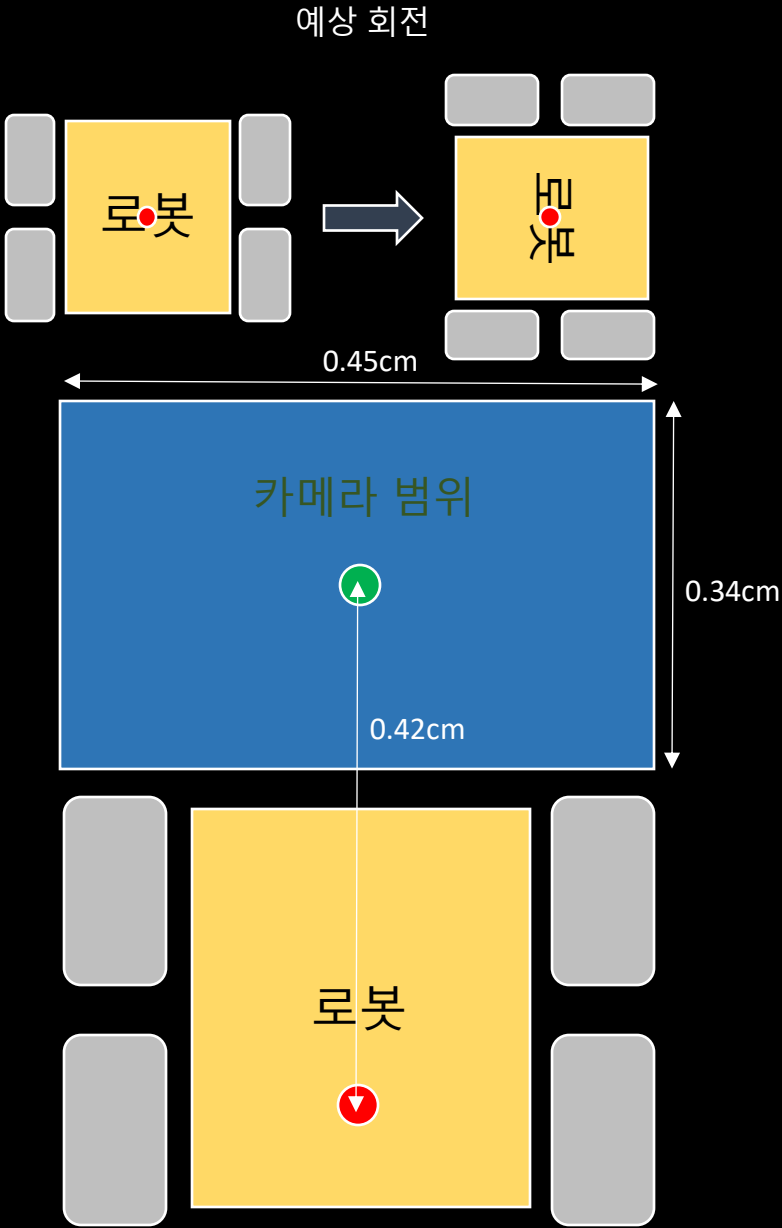
10.8 / 9.8 / 10.8 / 10 / 10.8 / 10 / 10 / 10.8 / 10.8 / 10.8

Hz를 늘려도 큰 차이가 없음을 관측

10Hz cmd\_vel (angular.z = 0.314 5sec)

94 / 95 / 93 / 94 / 95

90도 회전 시 약 4도 정도 오차를 관측



## Camera Calibration(1)

실제 카메라 이미지



캘리브레이션 된 이미지



Camera Calibration : 실세계의 3D 점과 캘리브레이션 된 카메라로 캡처한 이미지의 해당 2D 투영 픽셀 간의 정확한 관계를 파라미터화 하여 구하는 과정

3D->2D로 바꾸는 과정에서  
렌즈의 종류 및 이미지 센서와의 거리와 각도에 의해 왜곡이 제각각 다릅니다.  
이 왜곡을 잡아 주기 위한 파라미터를 구하는 과정입니다.

## Camera Calibration(2)

왼쪽이 실제 카메라 이미지로 모든 800\*600 이미지에서 100\*100 크기의 픽셀 사각형에 가로 5.7 세로 5.7 크기의 L 모양의 테이프가 들어가는 모습을 볼 수 있다.

-> 왜곡이 존재하지 않으며 100pixel당 약 5.7cm이다.





# 메인 문

```
1  #include "robot_control.hpp"
2
3  int main(int argc, char** argv)
4  {
5      ros::init(argc,argv,"Robot_control");
6      Pioneer Pioneer_;
7      Pioneer_.Get_param();
8      Pioneer_.run_robot();
9      return 0;
10 }
```

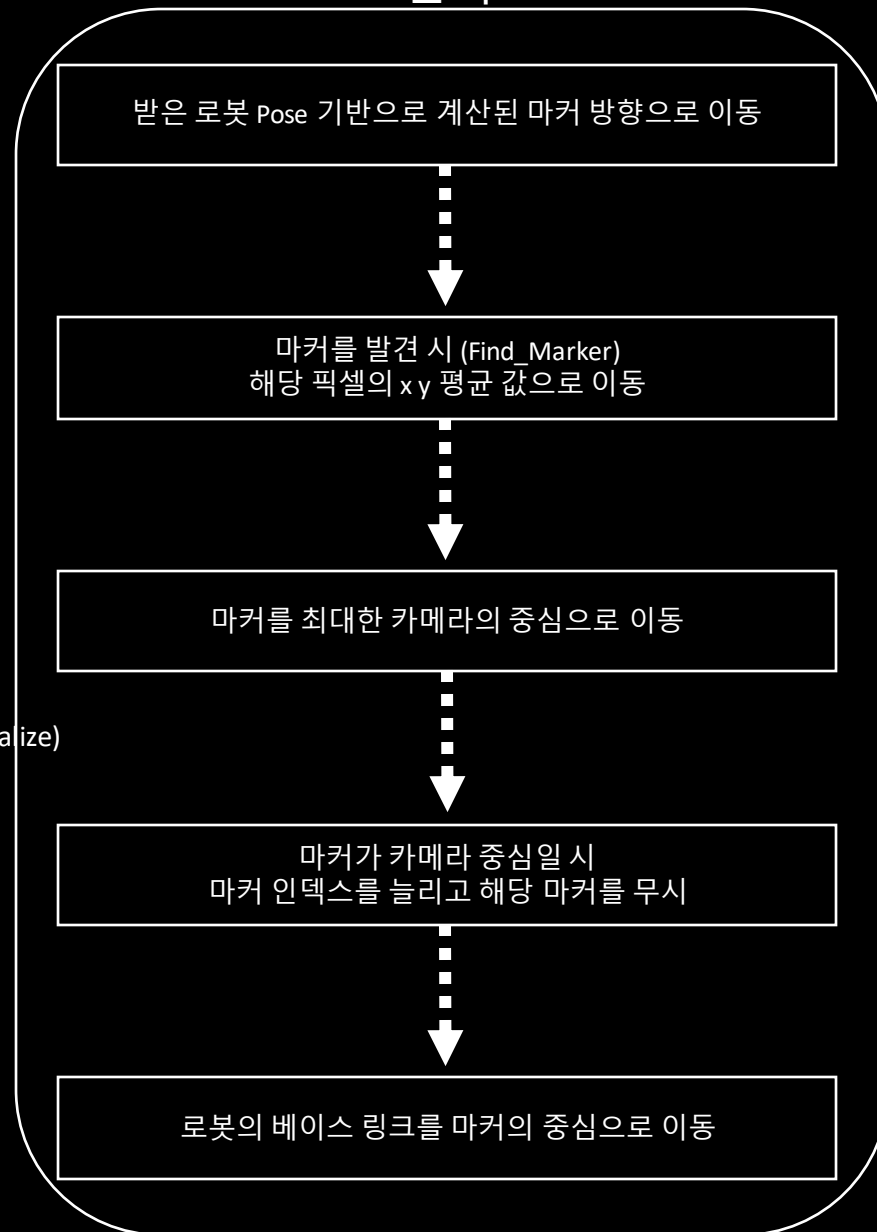
## Get\_param() 의 기능

- Subscribe 및 Publish 토픽과 이어줌
- 로봇의 속도 및 각속도를 가져옴
- 마커의 개수 및 위치 정보를 가져옴

## run\_robot() 의 기능

- 카메라를 이용해 마커가 얼마나 들어왔는지 위치가 어느정도 되는지 계산 (run\_camera)
- 마커가 대략적으로 얼마나 들어왔는지 확인 후 마커 모드를 변경(is\_marker\_on\_sight)
- 로봇의 현재 위치와 마커의 위치를 기반으로 로봇의 움직임을 결정(is\_direction\_match)
- Grid map을 cv Mat을 이용해 만들어 Kalman을 이용한 pose Odom을 이용한 pose와 마커의 위치를 보여준다(visualize)
- Start / Stop / Arrive 상태 시 정지 명령어 실행

## 반복



# Get\_param(1)

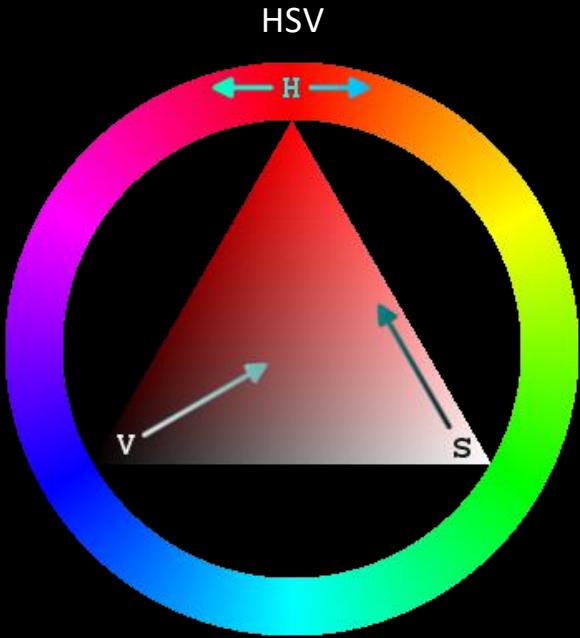
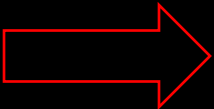
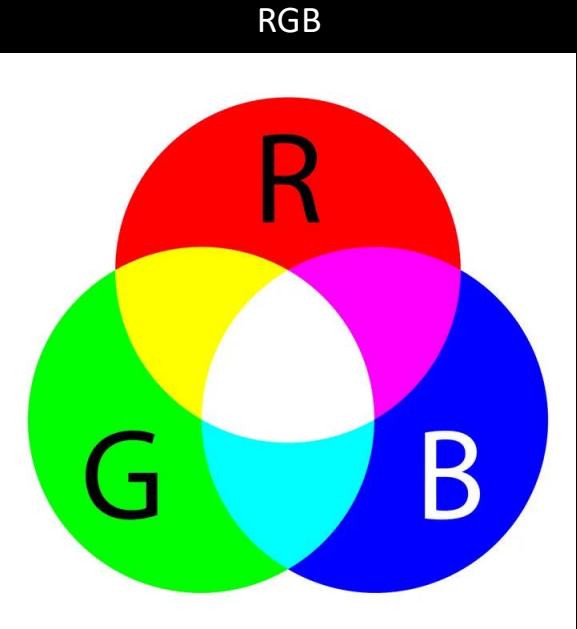
```
ros::NodeHandle nh_private("~");
nh_private.param<double>("speed", speed, 0.2);
nh_private.param<double>("angle_speed", angle_speed, 0.1);
angle_speed*=PI;
nh_private.param<int>("marker_num", marker_num, 10);
nh_private.param<int>("order_num", order_num, 5);

string marker;
string marker_pos;
string order;
string order_pos;

string robot_pos;
nh_private.param<string>("ROBOT_POS", robot_pos,"0,0");
int x=0;
int y=0;
int index=0;
index=robot_pos.find(',');
x=atoi(robot_pos.substr(index+1).c_str());
robot_pos.erase(index,robot_pos.size()-1);
y=atoi(robot_pos.c_str());
cout<<y<<endl;
ROBOT.x=x;
ROBOT.y=y;
ROS_INFO("STARTING ROBOT x: %f ROBOT y: %f ROBOT th: %f",ROBOT.y,ROBOT.x,ROBOT.th);
for(int i=1;i<=marker_num;i++)
{
    x=0;
    y=0;
    marker="Marker";
    marker=marker+to_string(i);
    nh_private.param<string>(marker, marker_pos,"10,10");
    index=marker_pos.find(',');
    y=atoi(marker_pos.substr(index+1).c_str());
    marker_pos.erase(index,marker_pos.size()-1);
    x=atoi(marker_pos.c_str());
    add_path_or_marker(MARKER,Marker_Gap*x,Marker_Gap*y,i);
}
ROS_INFO("%d MARKER SET",marker_num);
```

## <launch> 파일

```
<node name="robot_control_node" pkg="robot_action" type="robot_control" output="screen">
  <param name="speed" type="double" value="0.02"/>
  <param name="angle_speed" type="double" value="0.02"/>
  <param name="ROBOT_POS" type="string" value="0,0"/>
  <!-- *3.14 -->
  <!-- MARKER -->
  <param name="marker_num" type="int" value="11"/>
  <param name="Marker1" type="string" value="1,0"/>
  <param name="Marker2" type="string" value="2,0"/>
  <param name="Marker3" type="string" value="3,0"/>
  <param name="Marker4" type="string" value="4,0"/>
  <param name="Marker5" type="string" value="4,-1"/>
  <param name="Marker6" type="string" value="4,-2"/>
  <param name="Marker7" type="string" value="4,-3"/>
  <param name="Marker8" type="string" value="4,-4"/>
  <param name="Marker9" type="string" value="3,-4"/>
  <param name="Marker10" type="string" value="2,-4"/>
  <param name="Marker11" type="string" value="2,-5"/>
</node>
```



RGB방식으로 표현된 색을 찾을 시 빛에 너무 민감하다.  
HSV로 변환하여 색을 인식하도록 함 (빨강, 파랑, 초록) -> (색조, 포화, 밝기)

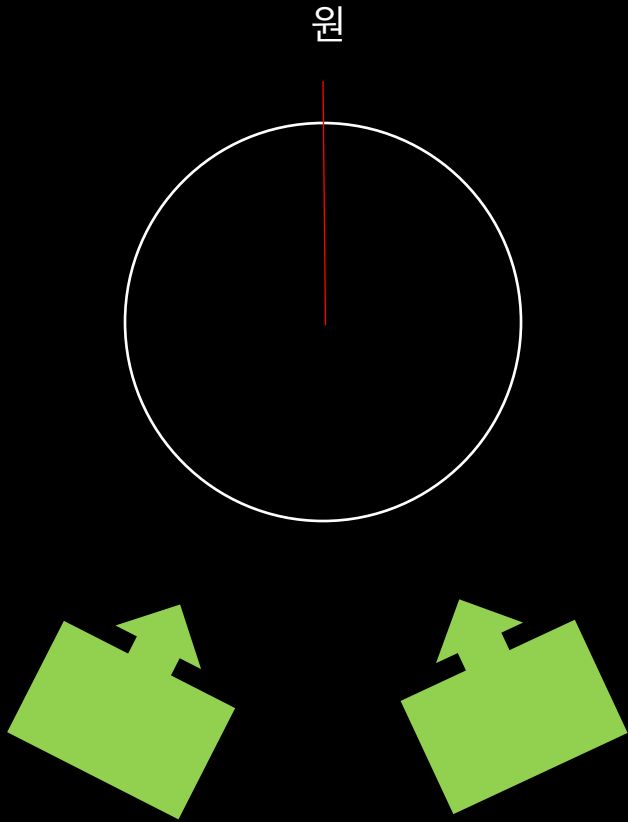
H는 360이 최대값이기에 8비트인 컴퓨터 입장에선 0~179로 나타낸다.  
즉 216-> 118이며 테스트 환경에 없으며 s와 v가 최대한 높게 인식되는 파란색을 선택

Blue (III)	216	100	100	0	102	255	#0066FF
------------	-----	-----	-----	---	-----	-----	---------

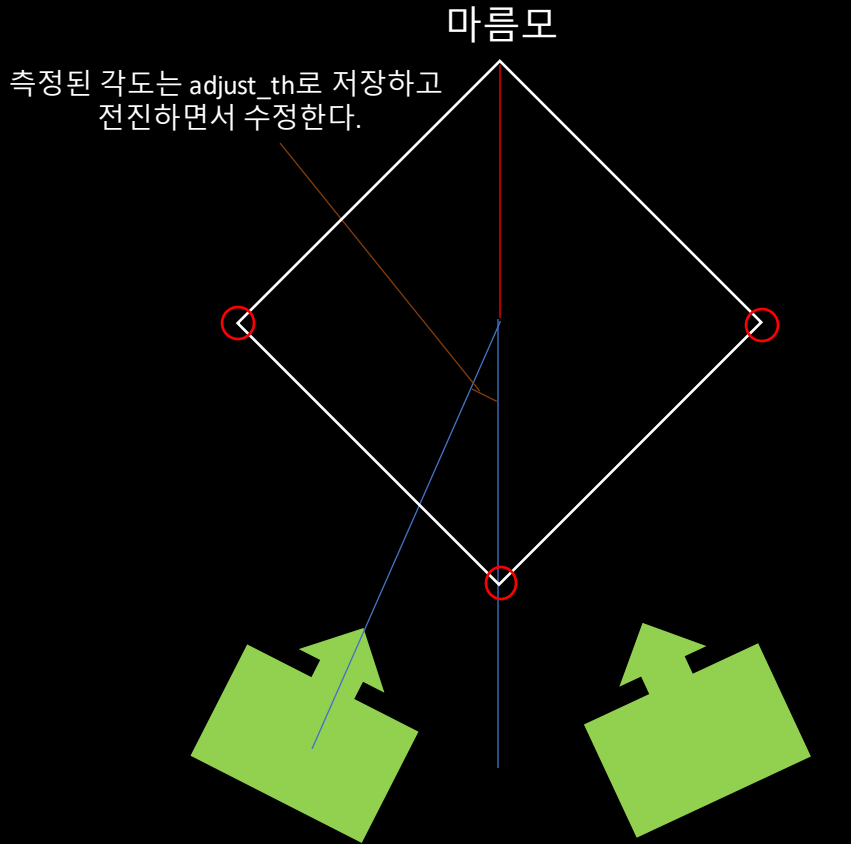
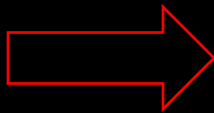
```
if (120>=H&&H>=98&&S>105&&V>125)
{
    frame.at<cv::Vec3b>(i,j)={0,0,255};
    marker_value++;
    y_pos+=double(i)/double(MARKER_pixel);
    x_pos+=double(j)/double(MARKER_pixel);
}
```



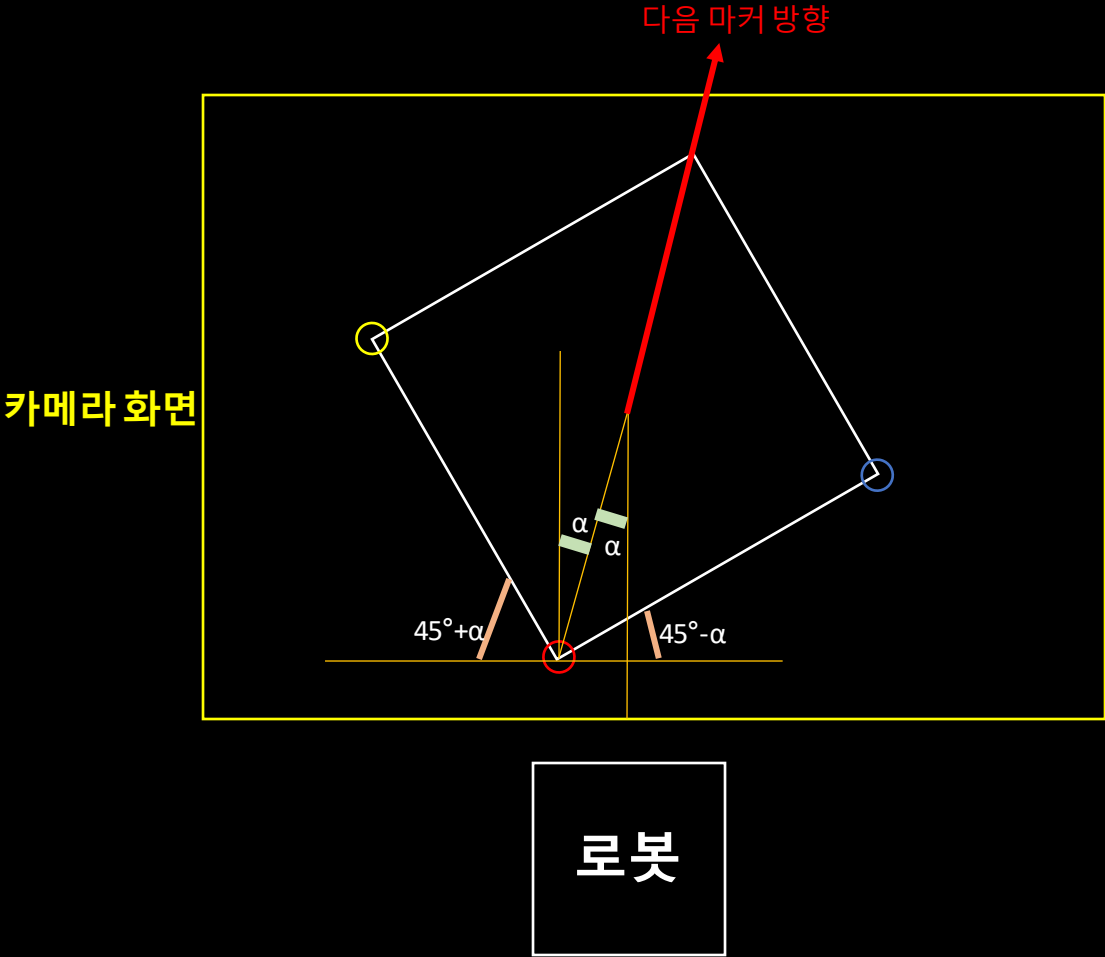
Run camera(2)





어느 쪽을 관측하여도 틀어진 각도를 알 수 없다.

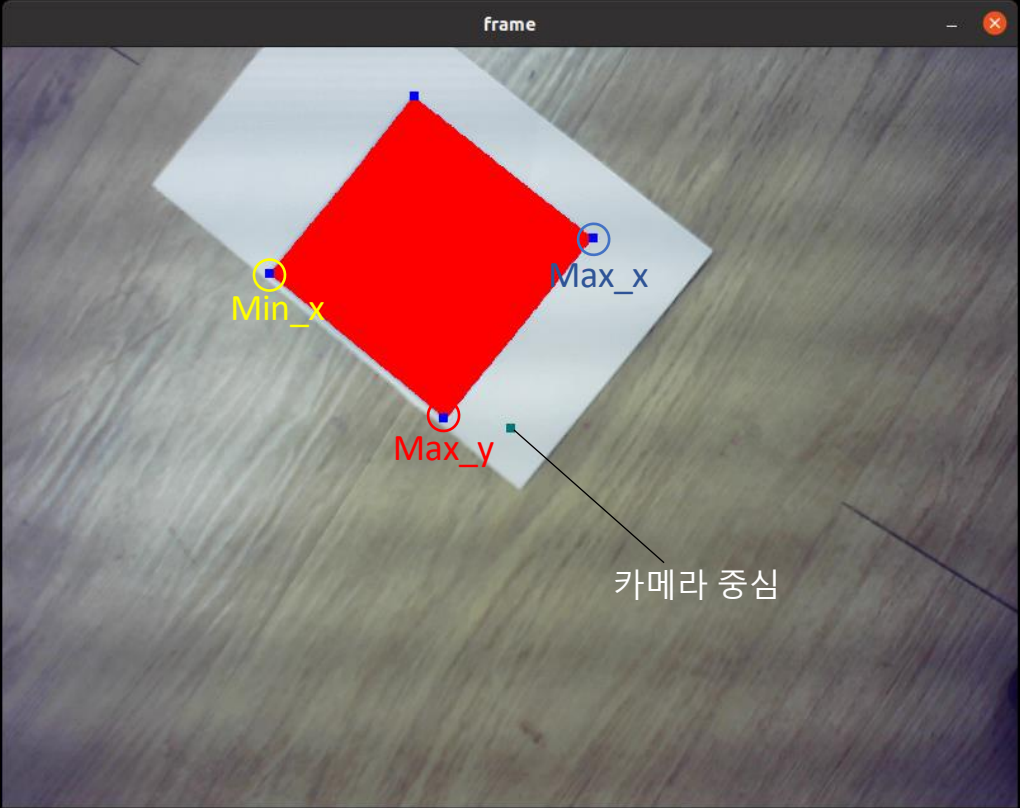


마름모의 꼭지점을 측정 시 각도가 측정 가능하다.

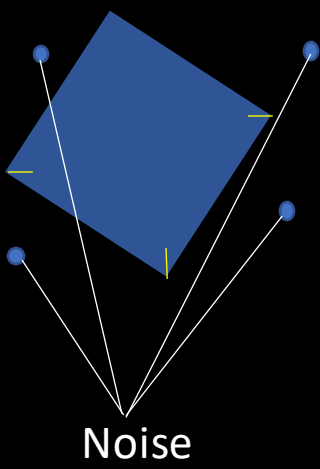


-   $\text{atan2}$ 를 이용해 왼쪽의 들어진 정도 오른쪽이 들어진 정도를 비교해 수정할 각도를 계산한다.  
[오른쪽 값 > 왼쪽 값] 로봇은 왼쪽으로 들어진 것이다. -> 왼쪽으로 보정
-  [왼쪽 값 > 오른쪽 값] 로봇은 오른쪽으로 들어진 것이다. -> 오른쪽으로 보정

Run camera(4)



파란색 마커를 인식하고 인식한 부분을 빨간색으로 칠한 모습  
노이즈를 고려해 연속된 픽셀이 존재 할 시에만 해당 점을 모서리로 인식하도록 설정

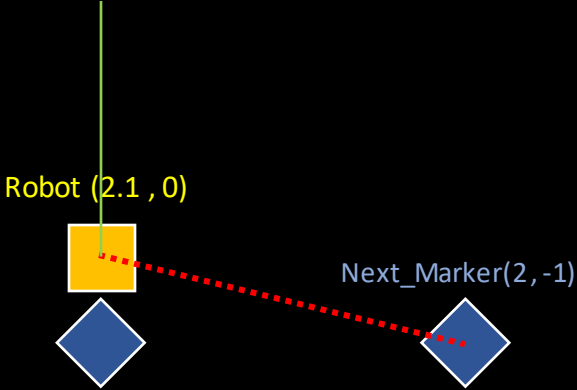


Is direction match(1)

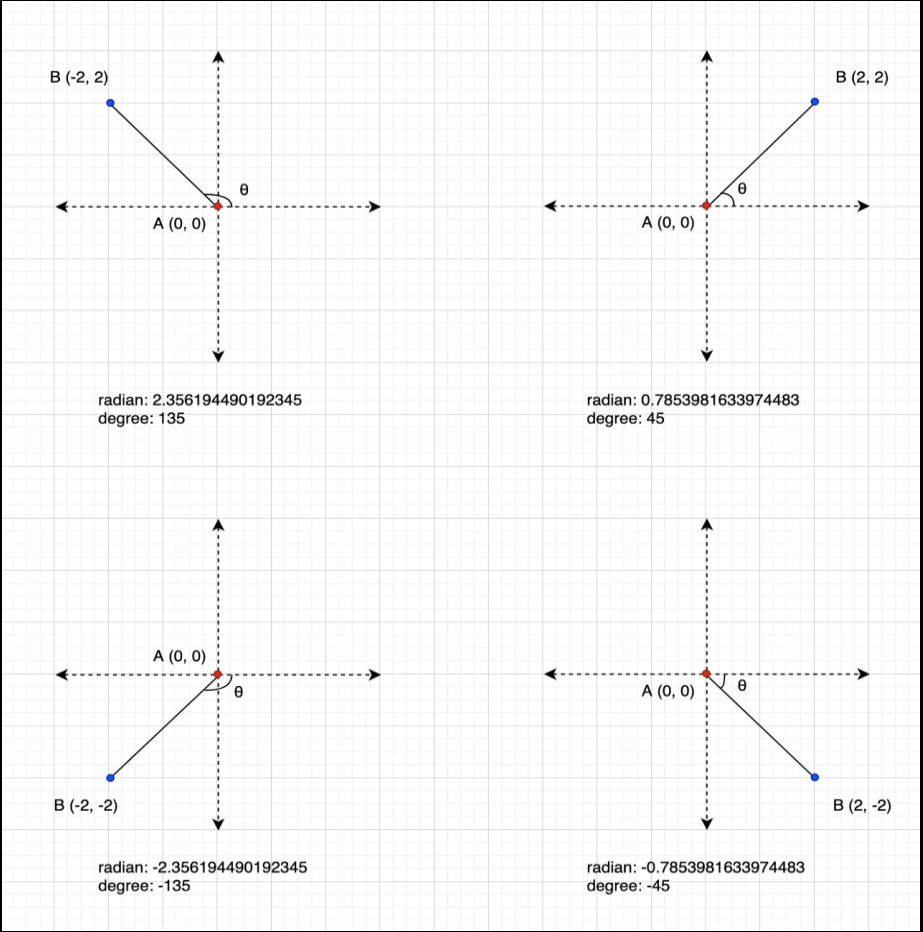
$\text{Marker\_th} = \text{Atan2}(-0.1, -1) = -1.67 = -90^\circ \sim -100^\circ$

$\text{각도} = \text{Marker\_th} - \text{Robot\_th} + \text{adjust\_th}$

각도가 0.1 이내 인 순간 앞으로 전진



atan2를 사용하는 이유



# Visualize(1)

```
void Pioneer::visualize()
{
    //Make blank map
    cv::Mat map(cv::Size(Map.map_row,Map.map_col),CV_8UC3,{0,0,0});
    //Make grid map

    count++;
    if(count==10)
    {
        count=0;

        PATH_index%=100;
        PATH_index++;
        if(PATH.size()<100)
            PATH.push_back(ROBOT);
        else
        {
            PATH[PATH_index]=ROBOT;
        }
    }

    for(int i=1;i<Map.cross;i++)
    {
        for(int k=0;k<Map.map_col;k++)
        {
            map.at<cv::Vec3b>(i*Map.row_block,k)={255,255,255};
        }
        for(int k=0;k<Map.map_row;k++)
        {
            map.at<cv::Vec3b>(k,i*Map.col_block)={255,255,255};
        }
    }

    for(int i=0;i<Pioneer::PATH.size();i++)
    {
        Pioneer::draw_path_at(convert_world_pos_y(PATH[i].x),convert_world_pos_x(-PATH[i].y),map,PATH_COLOR);
    }

    // Print Marker(which is found)
    for(int i=0;i<Pioneer::MARKER.size();i++)
    {
        Pioneer::draw_marker_at(convert_world_pos_x(-Pioneer::MARKER[i].y),convert_world_pos_y(Pioneer::MARKER[i].x),map,MARKER_COLOR);
    }

    // //Print Order Seq
    for(int i=0;i<Pioneer::Move_Order.size();i++)
    {
        Pioneer::draw_marker_at(convert_world_pos_x(-Pioneer::Move_Order[i].y),convert_world_pos_y(Pioneer::Move_Order[i].x),map,ORDER_COLOR);
    }

    // //Print Robot
    Pioneer::draw_robot_at(convert_world_pos_y(ROBOT.x),convert_world_pos_x(-ROBOT.y),ROBOT.th,&map);
    COLOR word;
    set_color(word,255,0,0,0);
    int gap=20;

    put_text(map,"X_val [" +to_string(ROBOT.x)+"]",cv::Point(10,30),4,0.6,cv::Scalar(word.B,word.G,word.R));
    put_text(map,"Y_val [" +to_string(ROBOT.y)+"]",cv::Point(10,30+gap*1),4,0.6,cv::Scalar(word.B,word.G,word.R));
    put_text(map,"Z_val [" +to_string(ROBOT.th)+"]",cv::Point(10,30+gap*2),4,0.6,cv::Scalar(word.B,word.G,word.R));
    put_text(map,"Next_Marker [" +to_string(Marker_index+1)+"]",cv::Point(10,30+gap*3),4,0.6,cv::Scalar(word.B,word.G,word.R));
    cv::namedWindow("map");
    cv::moveWindow("map",865,0);
    cv::imshow("map",map);
    cv::waitKey(10)==27;
}
```

검은색 빈 화면 매트릭스 생성

1초마다 로봇의 위치를 저장  
100개 이상 넘어갈 시 1번 부터 저장

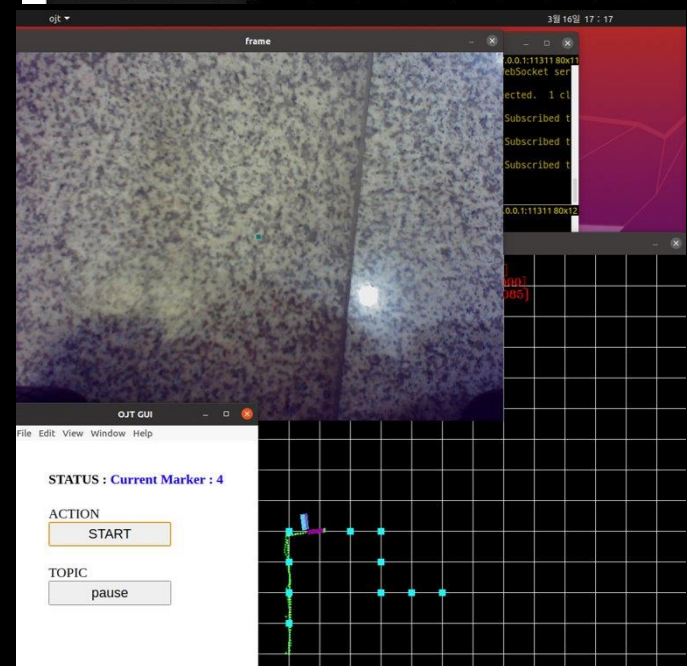
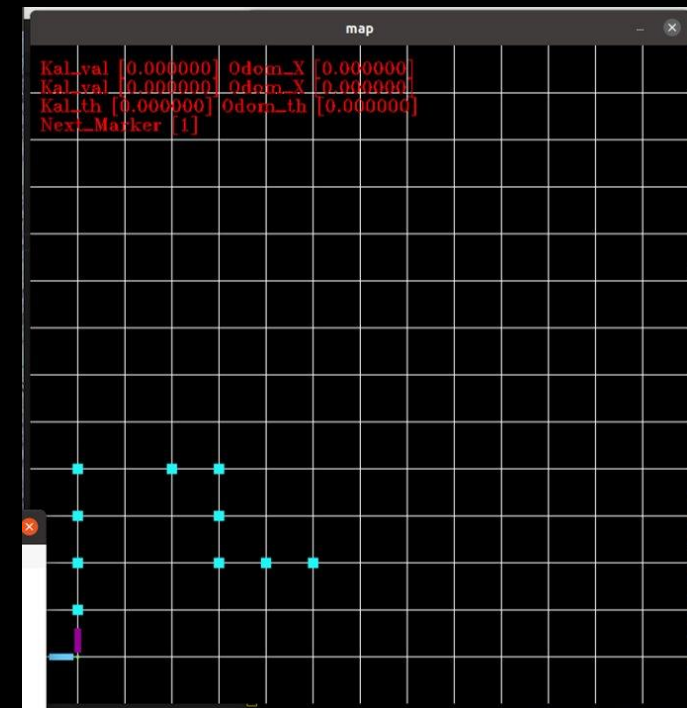
Map에 Grid를 그린다.

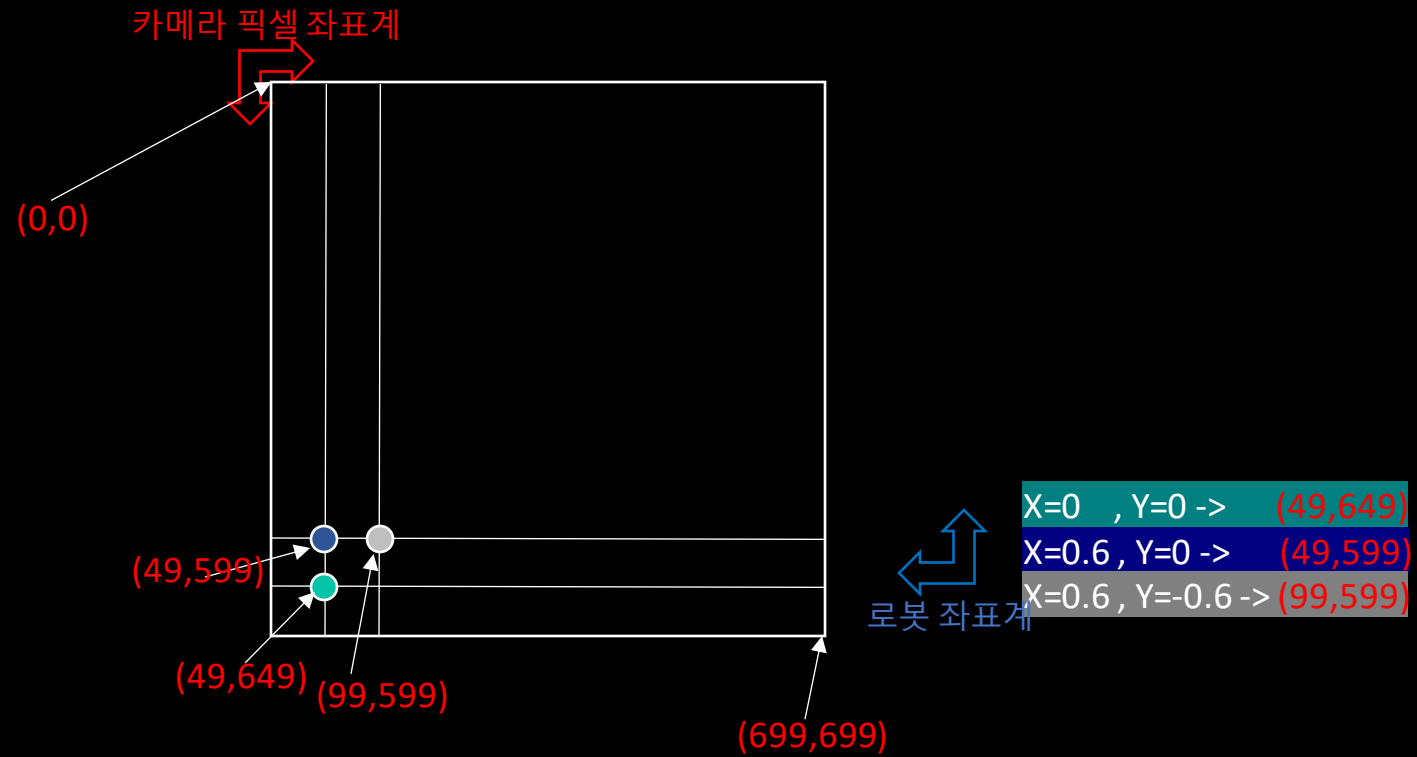
해당 위치에 경로를 그린다.

해당 위치에 마커를 그린다.

해당 위치에 로봇을 그린다.

Map에 글자를 새긴다.





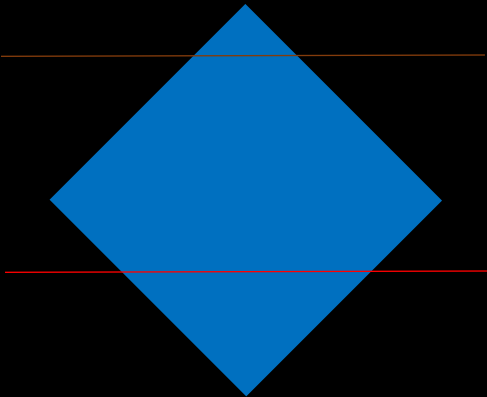
```
int Pioneer::convert_world_pos_x(double x)
{
    double world_x=Map.row_block*(x*(1.0/Marker_Gap)+1);
    return int(world_x);
}
int Pioneer::convert_world_pos_y(double y)
{
    double world_y=Map.col_block*(double(Map.cross-1)-y*(1.0/Marker_Gap));
    return int(world_y);
}
```



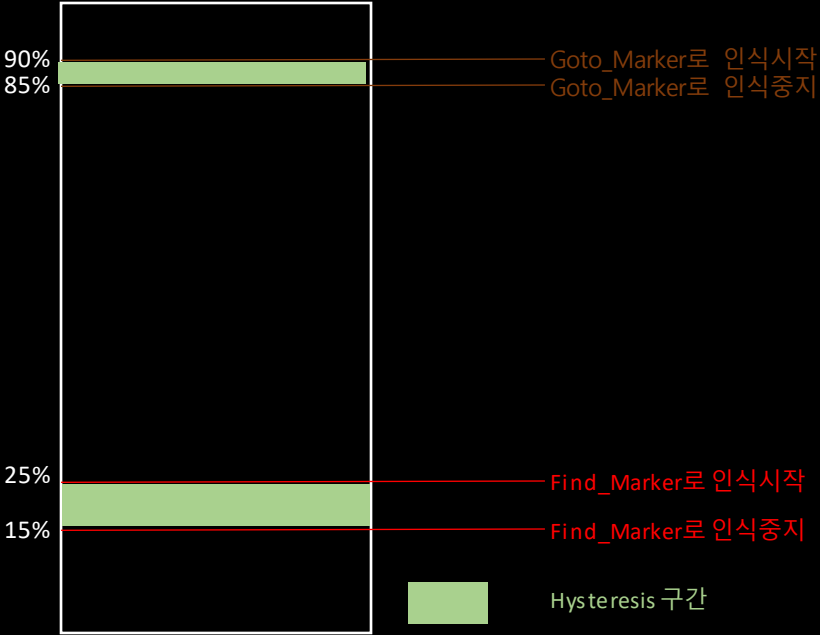
마커 관련 모드

```
void Pioneer::is_marker_on_sight()//90% of marker is shown
{
    if(Marker_mode==MARKER_MODE::Complete_Marker)
    {
        if(MARKER_pixel<MARKER_FULL_num*0.1)
        {
            Marker_mode=MARKER_MODE::No_Marker;
        }
    }
    else
    {
        if(MARKER_pixel>=MARKER_FULL_num*0.9)//90% of Marker is shown
        {
            Marker_mode=MARKER_MODE::Goto_Marker;
        }
        else if(MARKER_pixel>=MARKER_FULL_num*0.85&&Marker_mode==MARKER_MODE::Goto_Marker)//hysteresis - Goto
        {
            Marker_mode=MARKER_MODE::Goto_Marker;
        }
        else if(MARKER_pixel>=MARKER_FULL_num*0.25)//15% of Marker is shown
        {
            Marker_mode=MARKER_MODE::Find_Marker;
        }
        else if(MARKER_pixel>=MARKER_FULL_num*0.15&&Marker_mode==MARKER_MODE::Find_Marker)//hysteresis - Find
        {
            Marker_mode=MARKER_MODE::Find_Marker;
        }
        else
        {
            Marker_mode=MARKER_MODE::No_Marker;
            Find_Marker_once=false;
            Goto_Marker_once=false;
            Complete_Marker_once=false;
        }
    }
}
```

마커의 중앙이 카메라 중심과 일치하였다고 판단시 Complete\_Marker  
마커 픽셀이 다시 10% 수준으로 떨어져야 마커가 없다고 판단



카메라에 전부 보일 시 약 30000개의 픽셀 감지



# 마커 관련 모드

## 플리커 현상



카메라의 빠른 셔터 스피드 때문에 검은 줄이 물결처럼 생성

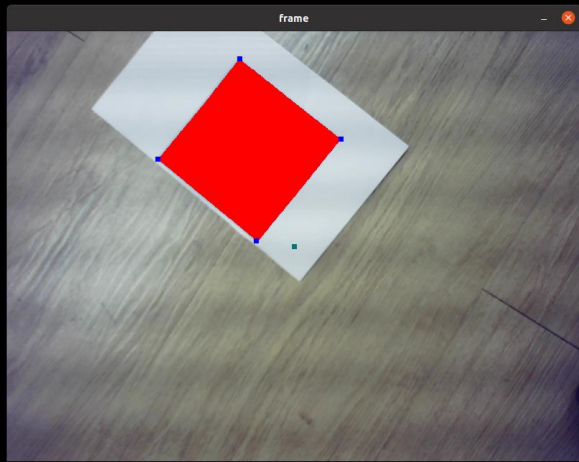
-> 형광등의 빛과 플리커 현상의 조합으로 색 감지에 노이즈가 낄

해결 방안 **Hysteresis 구간** 을 줌으로써 한번 인식한 후 어느정도 값이 떨어져도 상태를 유지

예) 90퍼 달성 -> Go to Marker 모드

노이즈로 인해 88퍼로 떨어짐 ->Go to Marker 상태 유지

마커가 멀어짐 84퍼 -> Find Marker 상태로 변경

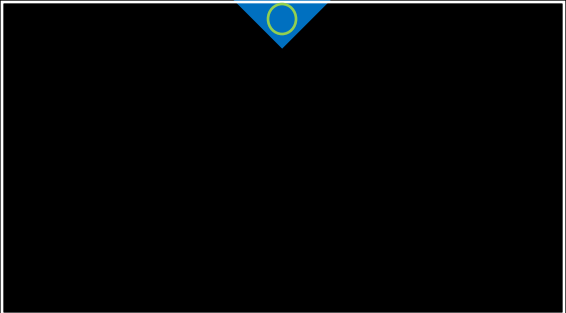


마커 관련 모드



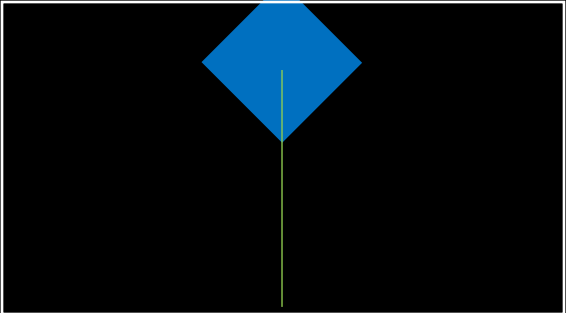
No Marker

마커 위치와 측정된 로봇 위치를 기반으로 방향을 정하고 움직임



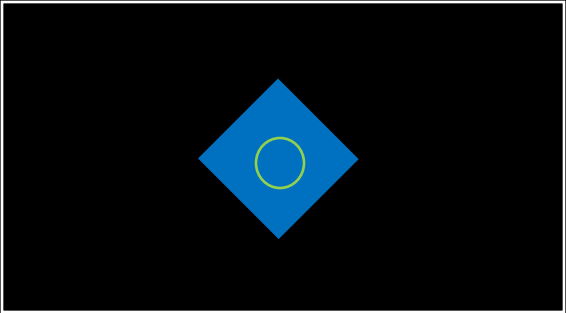
Find Marker  
25%

마커의 x 좌표와 y 좌표의 평균값을 구해 해당 위치로 이동



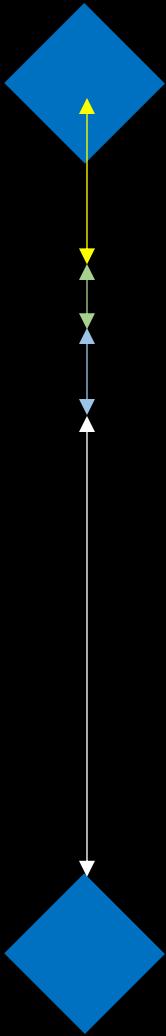
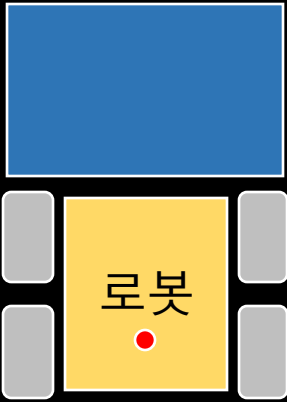
Goto Marker  
90%

최대한 중심으로 올 수 있도록 angle을 조정 후 전진



Complete Marker  
90% + center\_middle

원 내에 x y mean 값이 들어올 경우 마커의 인덱스를 늘린 후  
로봇의 베이스 링크와 일치시킨다.(카메라로 부터 약 42cm) ->42cm 전진  
마커가 10퍼센트도 보이지 않을 시 -> No Marker로 모드 변경



# 움직임 관련 모드

```
if(mode==MODE::Stop)
{
    Pioneer::stop();
}
else if(mode==MODE::Front)
{
    Pioneer::go_front();
}
else if(mode==MODE::Left)
{
    Pioneer::turn_left();
}
else if(mode==MODE::Right)
{
    Pioneer::turn_right();
}
else if(mode==MODE::Back)
{
    Pioneer::back();
}
```

```
void Pioneer::go_front()
{
    set_cmd_vel(speed,0);
}
void Pioneer::turn_left()
{
    set_cmd_vel(0,angle_speed);
}
void Pioneer::turn_right()
{
    set_cmd_vel(0,-angle_speed);
}
void Pioneer::stop()
{
    set_cmd_vel(0,0);
}
void Pioneer::back()
{
    set_cmd_vel(-speed,0);
}
```

정지 : linear = 0.0 angular = 0.0  
전진 : linear = 속도 angular = 0.0  
왼쪽 : linear = 0.0 angular = +각속도  
오른쪽 : linear = 0.0 angular = -각속도

반복 (10HZ)

마커 상태에 따른 움직임 상태 결정

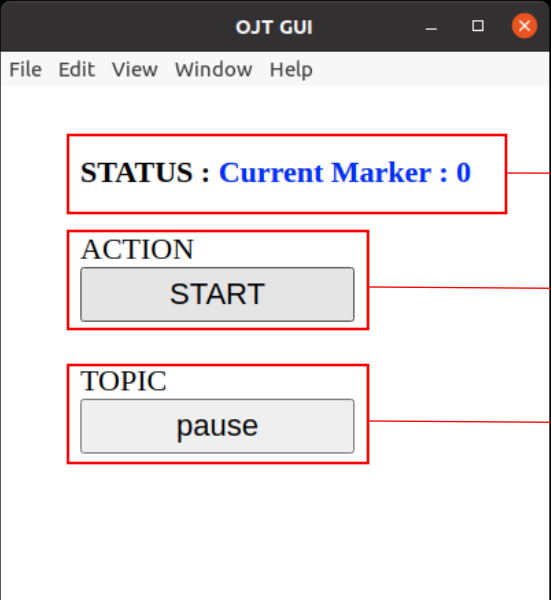
움직임 상태에 따른 cmd\_vel publish

cmd\_vel\_pub.publish(vel\_msg); 해당 cmd\_vel을 vel\_msg에 집어넣는다.

# Action Server

Nodejs를 이용해 Ros를 시작하기 전 선수과정

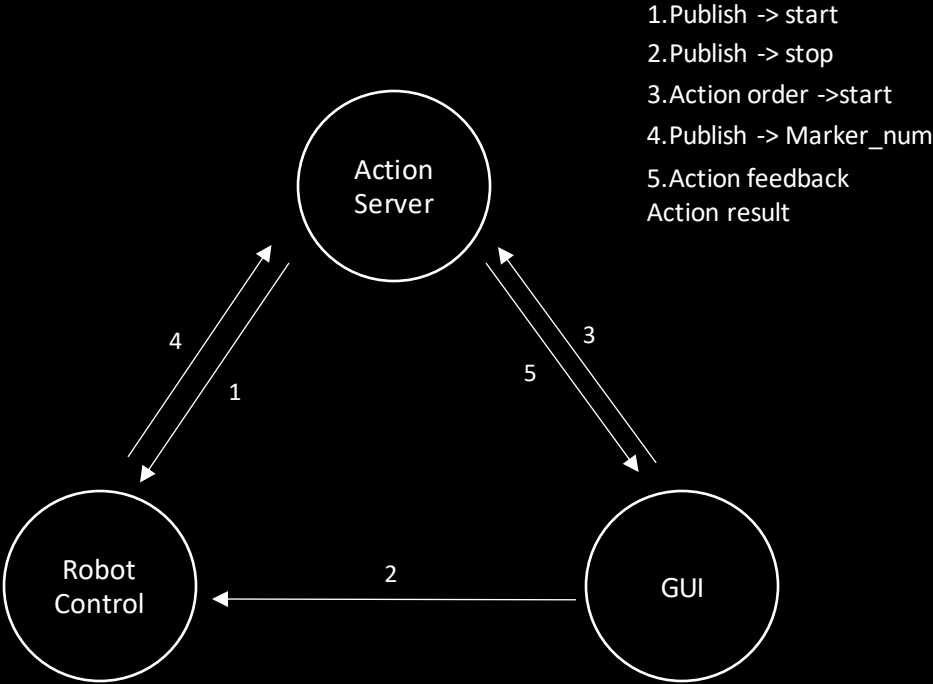
- Sudo apt-get install nodejs <-Node js 설치
- Sudo apt-get install npm <-npm 기능
- sudo apt-get install ros-noetic-rosbridge-suite <-웹소켓 설치
- roslaunch rosbridge\_server rosbridge\_websocket.launch <-웹소켓 실행
- 프로젝트 파일 내에서 nodejs 프로젝트 위치에 npm install 명령어 및 npm start를 눌러준다.



Action feedback || Action result

Action order

Pause publish





## 결과 사진

