**실행 방법 설명**

p\_n\_p.cpp 파일 수정 후

cd ~/catkin\_ws

catkin\_make후 roslaunch gen3\_lite\_examples p\_n\_p.launch 수행합니다.

(이때 roslaunch gen3\_lite\_examples gen3\_khu\_gazebo\_middle\_term.launch를 실행시키고 파일 초기화 되었을 때 실행시켜야 합니다.)

**코드 설명**

**# list\_to\_pose method**

geometry\_msgs::Pose list\_to\_pose(**double** x,**double** y,**double** z,**double** roll,**double** pitch,**double** yaw)

{

geometry\_msgs::Pose target\_pose;

tf2::Quaternion orientation;

orientation.setRPY(roll,pitch, yaw);

target\_pose.orientation= tf2::toMsg(orientation);

target\_pose.position.x=x;

target\_pose.position.y=y;

target\_pose.position.z=z;

**return** target\_pose;

}

list\_to\_pose함수는 x,y,z축의 값과 회전값인 roll,pitch,yaw를 입력으로 받아서 geometry\_msgs의 Pose 타입으로 반환해주는 함수입니다. 이 함수는 target으로 move하기 위해서 사용되는 pose데이터 입니다.

**# go\_to\_pose\_goal method**

**void** go\_to\_pose\_goal(moveit::planning\_interface::MoveGroupInterface &move\_group\_interface,

geometry\_msgs::Pose &target\_pose) {

move\_group\_interface.setPoseTarget(target\_pose);

moveit::planning\_interface::MoveGroupInterface::Plan my\_plan;

**bool** success = (move\_group\_interface.plan(my\_plan) == moveit::planning\_interface::MoveItErrorCode::SUCCESS);

move\_group\_interface.execute(my\_plan);

}

해당 함수는 입력으로 받은 pose 값에 따라 gen3가 이동하도록 하는 method입니다.

이때 움직일 MoveGroupInterface 또한 입력으로 같이 받아야합니다. 해당 함수는 arm의 position을 이동시키기 위해서 사용하였습니다.

**# gripper\_move method**

**void** gripper\_move(moveit::planning\_interface::MoveGroupInterface &move\_group\_interface,**double** value)

{

moveit::planning\_interface::MoveGroupInterface::Plan my\_plan;

std::vector<**double**> joint\_group\_positions;

joint\_group\_positions=move\_group\_interface.getCurrentJointValues();

joint\_group\_positions[0]=0;

joint\_group\_positions[1]=0;

joint\_group\_positions[2]=value;

joint\_group\_positions[3]=0;

move\_group\_interface.setJointValueTarget(joint\_group\_positions);

**bool** success = (move\_group\_interface.plan(my\_plan) == moveit::planning\_interface::MoveItErrorCode::SUCCESS);

move\_group\_interface.execute(my\_plan);

}

해당 함수는 value만큼 joint position을 움직이기 위해서 사용하였습니다. 입력값으로 MoveGroupInterface를 받는데 이거를 gripper의 interface를 넘겨주고 value로는 gripper를 열거나 닫을때 어느 값을 줄지를 입력으로 넘겨줬습니다.

**# p\_n\_p method**

**void** p\_n\_p(moveit::planning\_interface::MoveGroupInterface& arm\_interface, moveit::planning\_interface::MoveGroupInterface& gripper\_interface)

{

std::vector<std::vector<**double**>> block\_pos\_list = {

{0.2,0.1,0.06},

{0.2,0.0,0.06},

{0.2,-0.1,0.06},

{0.3,0.1,0.08},

{0.3,0.0,0.06},

{0.3,-0.1,0.07},

{0.4,0.1,0.06},

{0.4,0.0,0.09}

};

std::vector<std::vector<**double**>> target\_pos\_list = {

{-0.15,0.35,0.06},

{-0.05,0.35,0.06},

{0.15,0.25,0.06},

{0.05,0.35,0.08},

{-0.05,0.25,0.06},

{0.15,0.35,0.07},

{0.05,0.25,0.06},

{-0.15,0.25,0.09},

};

**for** (**int** i=0;i<8;i++) {

//grab

geometry\_msgs::Pose block\_pose;

block\_pose=list\_to\_pose(block\_pos\_list[i][0], block\_pos\_list[i][1], block\_pos\_list[i][2]+0.05, -M\_PI,0,0);

ROS\_INFO("block pos move");

go\_to\_pose\_goal(arm\_interface,block\_pose);

ros::WallDuration(2.0).sleep();

ROS\_INFO("gripper move");

gripper\_move(gripper\_interface,0.5);

ros::WallDuration(2.0).sleep();

block\_pose=list\_to\_pose(block\_pos\_list[i][0], block\_pos\_list[i][1], block\_pos\_list[i][2]-0.01, -M\_PI,0,0);

ROS\_INFO("block pos down move");

go\_to\_pose\_goal(arm\_interface,block\_pose);

ros::WallDuration(2.0).sleep();

ROS\_INFO("gripper move");

**if** (i == 6) {

gripper\_move(gripper\_interface,0.275);

}

**else** {

gripper\_move(gripper\_interface,0.377);

}

ros::WallDuration(1.0).sleep();

block\_pose=list\_to\_pose(block\_pos\_list[i][0], block\_pos\_list[i][1], block\_pos\_list[i][2]\*3, -M\_PI,0,0);

ROS\_INFO("block pos up move");

go\_to\_pose\_goal(arm\_interface,block\_pose);

ros::WallDuration(2.0).sleep();

//out

geometry\_msgs::Pose target\_pose;

target\_pose=list\_to\_pose(target\_pos\_list[i][0], target\_pos\_list[i][1], target\_pos\_list[i][2]\*3,-M\_PI,0,0);

ROS\_INFO("target move");

go\_to\_pose\_goal(arm\_interface,target\_pose);

ros::WallDuration(2.0).sleep();

**if** (i ==2) {

target\_pose=list\_to\_pose(target\_pos\_list[i][0], target\_pos\_list[i][1], target\_pos\_list[i][2]+0.05,-M\_PI,0,0);

ROS\_INFO("target down move");

go\_to\_pose\_goal(arm\_interface,target\_pose);

ros::WallDuration(4.0).sleep();

}

**else** **if** (i == 6) {

target\_pose=list\_to\_pose(target\_pos\_list[i][0], target\_pos\_list[i][1], target\_pos\_list[i][2]+0.05,-M\_PI,0,0);

ROS\_INFO("target down move");

go\_to\_pose\_goal(arm\_interface,target\_pose);

ros::WallDuration(4.0).sleep();

}

**else** {

target\_pose=list\_to\_pose(target\_pos\_list[i][0], target\_pos\_list[i][1], target\_pos\_list[i][2]+0.05,-M\_PI,0,0);

ROS\_INFO("target down move");

go\_to\_pose\_goal(arm\_interface,target\_pose);

ros::WallDuration(.0).sleep();

}

ROS\_INFO("gripper move");

gripper\_move(gripper\_interface,0.5);

ros::WallDuration(2.0).sleep();

target\_pose=list\_to\_pose(target\_pos\_list[i][0], target\_pos\_list[i][1], target\_pos\_list[i][2]\*3,-M\_PI,0,0);

ROS\_INFO("target up move");

go\_to\_pose\_goal(arm\_interface,target\_pose);

ros::WallDuration(2.0).sleep();

}

}

**else** {

gripper\_move(gripper\_interface,0.377);

}

ros::WallDuration(1.0).sleep();

block\_pose=list\_to\_pose(block\_pos\_list[i][0], block\_pos\_list[i][1], block\_pos\_list[i][2]\*3, -M\_PI,0,0);

ROS\_INFO("block pos up move");

go\_to\_pose\_goal(arm\_interface,block\_pose);

ros::WallDuration(2.0).sleep();

//out

geometry\_msgs::Pose target\_pose;

target\_pose=list\_to\_pose(target\_pos\_list[i][0], target\_pos\_list[i][1], target\_pos\_list[i][2]\*3,-M\_PI,0,0);

ROS\_INFO("target move");

go\_to\_pose\_goal(arm\_interface,target\_pose);

ros::WallDuration(2.0).sleep();

target\_pose=list\_to\_pose(target\_pos\_list[i][0], target\_pos\_list[i][1], target\_pos\_list[i][2]+0.05,-M\_PI,0,0);

ROS\_INFO("target down move");

go\_to\_pose\_goal(arm\_interface,target\_pose);

ros::WallDuration(.0).sleep();

ROS\_INFO("gripper move");

gripper\_move(gripper\_interface,0.5);

ros::WallDuration(2.0).sleep();

target\_pose=list\_to\_pose(target\_pos\_list[i][0], target\_pos\_list[i][1], target\_pos\_list[i][2]\*3,-M\_PI,0,0);

ROS\_INFO("target up move");

go\_to\_pose\_goal(arm\_interface,target\_pose);

ros::WallDuration(2.0).sleep();

}

}

해당 함수는 실제로 도형을 옮기기 위한 함수로 첫번째 인자로 arm\_interface를 입력을 받고 두번째로 gripper\_interface를 입력으로 받게 됩니다. Block의 위치와 target의 위치를 2차원 벡터로 표현하여 각각의 변수에 저장을 하였고(이때 세번째 값으로 도형의 높이를 넣어주었습니다.) 블록은 총 8개가 있으므로 8번의 똑같은 과정을 반복하였습니다. 각각의 과정들은 잡는 과정과 놓는 과정을 포함하고 있습니다.

먼저 잡는 과정입니다. 처음에 block 위치로 “block pos move”라는 info message를 출력하고 이동합니다. 이 때 도형 높이보다 좀 떨어진 높이에서 gripper를 벌리기 위해 z축에 0.05를 더했고 -M\_PI는 도형을 집기 편하기 위해 각도를 조정하려고 넣었습니다. 그 다음 gripper를 집기 전에 “gripper move”라는 메시지를 출력한 후 gripper에 0.5의 값을 넣어서 도형을 집기 위한 크기 정도로 벌릴 수 있도록 하였습니다. 그 다음 “block pos down move”라는 메시지를 출력하고 해당 블록을 잡기 위해 팔이 내려갈 수 있도록 하였고 “gripper move”라는 메시지를 출력하고 도형의 크기만큼 gripper가 줄어들어서 도형을 집을 수 있도록 하였습니다. 이때 삼각형의 경우 다른 도형들보다 작기 때문에 0.275라는 값을 주었고 나머지 도형의 경우 크기가 비슷하기 때문에 0.377의 값을 주어서 gripper가 잡을 수 있도록 하였습니다. 도형을 잡은 다음 “block pos up move”라는 메시지를 출력하고 도형을 도형의 높이의 3배만큼 들어올리도록(이동하면서 다른 도형이 부딪히지 않도록 큰값을 주었습니다.) 하였습니다.

다음 도형을 놓는 과정입니다. 잡고 위로 올린 다음 “target move”라는 메시지를 출력하고 도형을 놓을 위치인 target\_pos\_list의 한 값으로 이동하게 하였습니다. 그 다음 도형을 놓기 위해서 도형의 높이에서 +0.05한 값으로 z축을 이동하도록 하였습니다(이때 “target down move”라는 메시지를 출력하도록 했습니다). 이후 “gripper move”라는 메시지를 출력하고 0.5만큼 gripper를 벌려서 도형을 놓을 수 있도록 하였습니다. 도형을 놓은 후에는 “target up move”라는 메시지를 출력하고 다시 그 도형의 세배만큼의 높이로 올라가도록 하였습니다.

이렇게 하나의 루프를 돈 다음 다시 처음으로 돌아가서 도형을 잡기 위해서 방금과 같은 과정을 반복하고 총 블록의 개수인 8번을 반복해서 옮기면 프로그램이 종료됩니다.