moveit2

Version	V1.0
최종수정일	2025.01.28
작성자	김루진 강사

moveit2

Wifi
Rokey / rokey12345
방화벽 해지
sudo ufw disable

같은 ROS_DOMAIN_ID에 영향을 안받게 하기
export ROS_LOCALHOST_ONLY=1

환경변수 확인
echo $ROS_LOCALHOST_ONLY

주제 : moveit2 이해

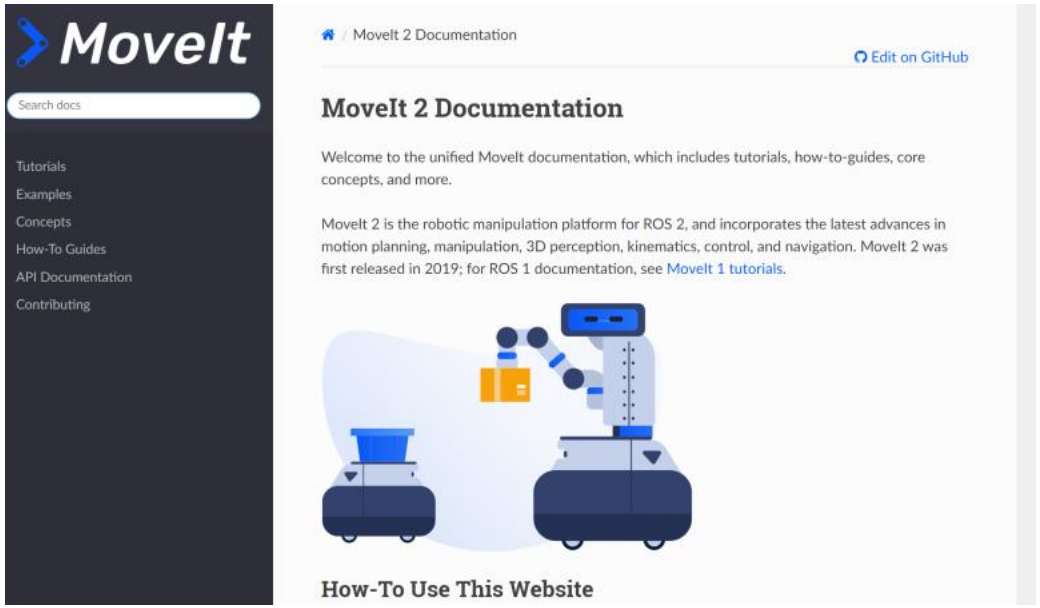
Moveit2란

1.MoveIt 2는 ROS 2 환경에서 로봇의 모션 플래닝, 조작, 제어, 시각화를 제공하는 강력한 라이브러리

2.로봇 매니퓰레이터(예: 로봇 팔)나 자율주행 로봇의 모션을 계획하고 제어하기 위한 필수 도구로 사용

MoveIt2는 로봇을 위한 오픈 소스 소프트웨어로, 로봇의 모션 플래닝, 조작, 3D 인식, 운동학, 제어 및 시뮬레이션을 포함하는 복잡한 로봇 애플리케이션을 개발하기 위한 통합 플랫폼

MoveIt2는 센서 데이터를 이용하여 로봇 주변 환경의 3D 모델을 생성하고 업데이트할 수 있으며, 이를 통해 로봇이 더 정확하게 환경을 인식하고 장애물을 피하는 데 도움을 줍니다. 로봇의 운동학적 및 동역학적 계산을 위한 도구도 제공하여, 로봇의 위치, 속도, 가속도를 정확하게 계산할 수 있습니다. 초보자에게 벽을 선사하는 역기구학 역시 대신 풀어주게 됩니다.



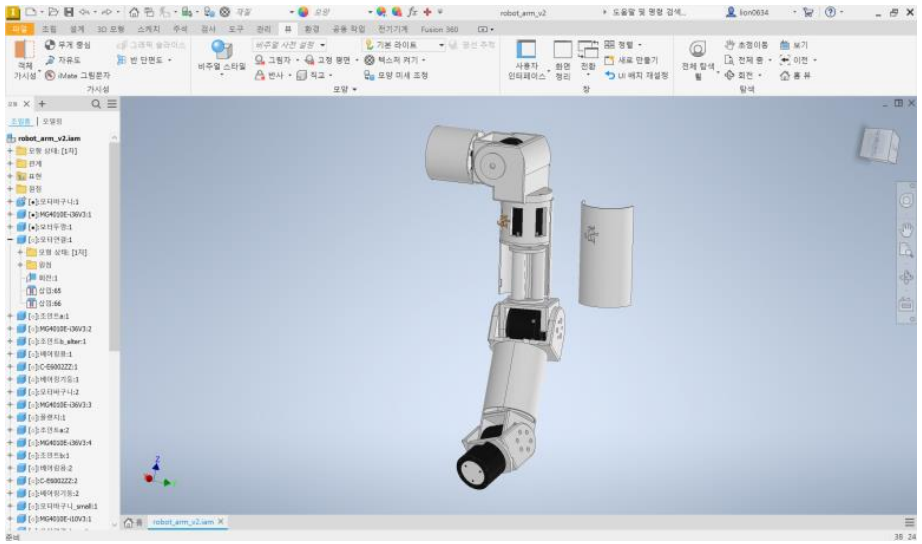
Moveit2 주요기능

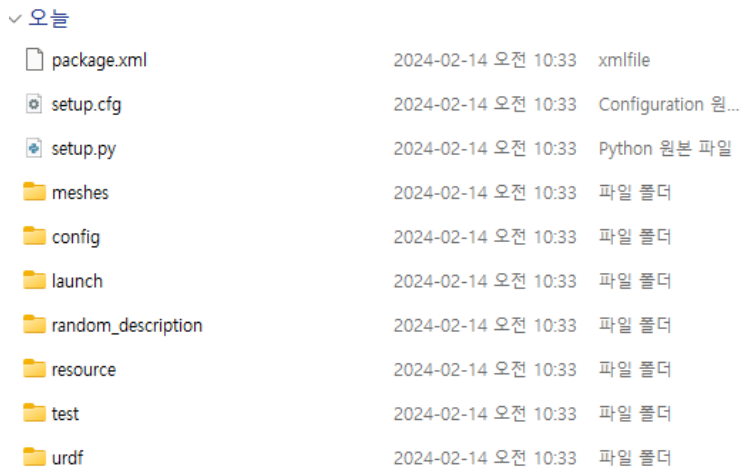
- **모션 플래닝(Motion Planning)**
  - 로봇의 경로를 계산하고 실행하기 위한 기능.
  - 샘플 기반 플래너(OMPL, STOMP, CHOMP 등) 지원.
  - 충돌 회피를 포함한 최적 경로를 계산.
- **충돌 검사(Collision Checking)**
  - 로봇의 링크 간 또는 환경과의 충돌을 실시간으로 검사.
- **Kinematics (역/순운동학)**
  - 로봇의 목표 위치를 계산하거나 로봇 관절의 상태를 조작.
  - IKFast, KDL, Trac-IK 같은 운동학 플러그인 지원.
- **로봇 시각화**
  - RViz2를 사용하여 로봇의 현재 상태, 목표, 경로를 시각화.
- **플래닝 장면 관리(Planning Scene Management)**
  - 로봇이 상호작용할 수 있는 가상 환경을 정의.
  - 장애물 추가/제거, 3D 객체 모델 관리 가능.
- **실시간 제어(Servoing)**
  - 카메라 입력이나 다른 센서를 기반으로 실시간으로 로봇 움직임 제어.
- **피드백 루프 통합**
  - 경로 계획 이후 로봇 하드웨어로 명령을 보내고, 센서 데이터를 기반으로 조정.

3D 모델링

1.모델링 툴

Solidworks나 Fusion360를 사용하는 것을 추천
urdf exporter가 가능함





export된 폴더를 보면 Rviz와 Gazebo launch파일까지 자동으로 생성

```
ros2 launch random_description display.launch.py
```

Inventor 프로그램을 통해 모델링을 했고, 추후 이를 Fusion360으로 옮겨와 변환

(Inventor과 Fusion360 모두 Autodesk 사의 제품이라 연동이 편리

Moveit2 적용하기 – Moveit Setup Assistant

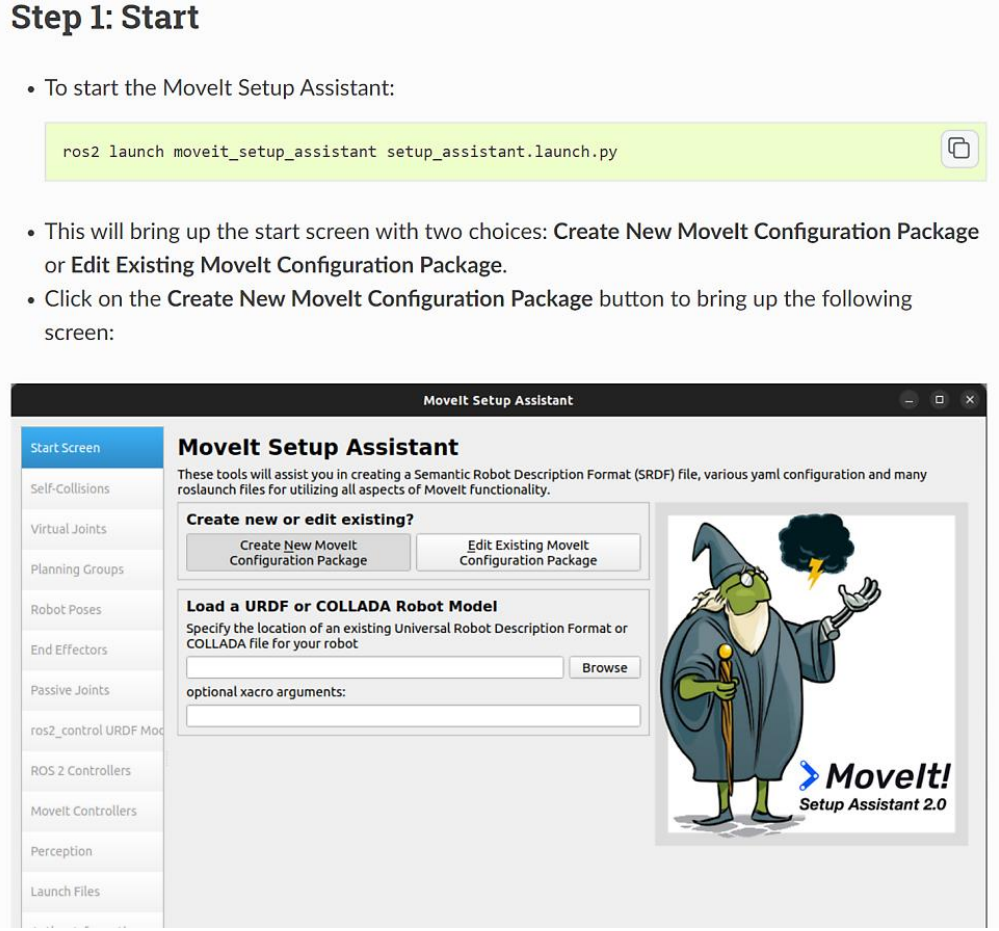
꼭 지켜야할 내용

1.시작 전 로봇을 ROS2에서 Build하고 Source 하기
2.URDF가 완벽해야 함!
3.완료 후 속도 & 가속도값 수정

모두 완벽하게 생성을 했더라도, 실제 실행을 하고 plan&execute 버튼으로 플래닝을 실행할 때, planning이 **FAILED** 라고 뜨는 경우가 있습니다.

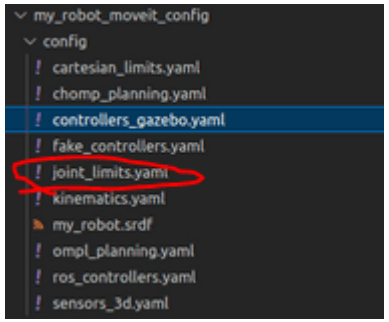
이 경우 생성된 setup assistant로 생성된 자신의 moveit description 파일의 config 에서 **joint_limits.yaml** 수정

로봇의 속도와 가속도 값을 제한할 수 있는데. 제한 사항에 false를 모두 true로 고쳐주고, 속도와 가속도 값을 모두 0이 아닌 다른 값으로 수정



Moveit2 적용하기 – Moveit Setup Assistant

올바른 joint_limits.yaml의 예시




```
# Joints limits
#
# Sources:
#
#  - Universal Robots e-Series, User Manual, UR10e, Version 5.8
#    https://s3-eu-west-1.amazonaws.com/ur-support-site/69139/99405_UR10e_User_Manual
#  - Support > Articles > UR articles > Max. joint torques
#    https://www.universal-robots.com/articles/ur-articles/max-joint-torques
#    retrieved: 2020-06-16, last modified: 2020-06-09
joint_limits:
  shoulder_pan_joint:
    # acceleration limits are not publicly available
    has_acceleration_limits: true
    has_effort_limits: true
    has_position_limits: true
    has_velocity_limits: true
    max_effort: 330.0
    max_position: !degrees  360.0
    max_velocity: !degrees  120.0
    max_acceleration: !degrees  120.0
    min_position: !degrees -360.0
  shoulder_lift_joint:
    # acceleration limits are not publicly available
    has_acceleration_limits: true
    has_effort_limits: true
    has_position_limits: true
    has_velocity_limits: true
    max_effort: 330.0
    max_position: !degrees  360.0
    max_velocity: !degrees  120.0
    max_acceleration: !degrees  120.0
    min_position: !degrees -360.0
  elbow_joint:
    # acceleration limits are not publicly available
    has_acceleration_limits: true
    has_effort_limits: true
    has_position_limits: true
```

OMPL(Open Motion Planning Libaray)

OMPL은 **경로 계획(Motion Planning)** 문제를 해결하기 위한 강력한 오픈소스 라이브러리입니다. 이 라이브러리는 다양한 샘플 기반 알고리즘(Sampling-Based Algorithms)을 제공하며, **MoveIt 2** 및 로봇 애플리케이션에서 경로를 계산하는 데 핵심적인 역할을 합니다.

- **샘플 기반 모션 플래닝(Sampling-Based Motion Planning) 지원**
- 대표적인 알고리즘:
  - RRT (Rapidly-exploring Random Tree)
  - RRT*
  - PRM (Probabilistic Roadmap)
  - KPIECE
  - BIT*
  - LazyPRM
  - FMT*
- **로봇의 자유도(Degree of Freedom, DoF)에 관계없이 확장 가능**
  - 단순 2D 로봇부터 복잡한 고자유도 매니퓰레이터까지 플래닝 가능.
- **플래너 독립성**
  - MoveIt 2와 통합되지만, 독립적으로 사용할 수도 있습니다.
- **C++ API 지원**
  - 로우 레벨에서 경로 계획을 직접 구성 및 제어할 수 있습니다.
- **확장성**
  - 사용자 정의 플래너와 상태 공간을 쉽게 추가할 수 있습니다.

OMPL의 주요 구성 요소

- **State Space (상태 공간)**
  - 로봇의 위치, 회전, 관절 각도 등의 상태를 정의.
  - 예: SE(2)(평면 상에서의 이동) 또는 SE(3)(3D 공간에서의 이동).
- **Planner (플래너)**
  - 상태 공간 내에서 경로를 생성.
  - 다양한 샘플 기반 알고리즘 제공.
- **Simple Setup**
  - OMPL에서 플래닝 문제를 간단히 설정하는 유틸리티.
  - 상태 공간, 플래너, 샘플링, 경로 유효성 검사를 쉽게 구성.
- **Collision Checker (충돌 검사)**
  - MoveIt 2와 통합될 때, 충돌 검사를 통해 생성된 경로가 유효한지 확인.
- **Path Simplification**
  - 초기 경로를 최적화하여 로봇이 실제로 실행할 수 있는 경로로 변환.

OMPL + Moveit2 통합

MoveIt 2는 OMPL을 플래닝 백엔드로 사용, 사용자는 **플래닝 파이프라인**을 통해 OMPL 플래너를 선택하고 구성.

플래너 설정

MoveIt 2에서 OMPL 플래너를 설정하려면 ompl_planning.yaml 파일을 수정합니다.

```yaml
planner_configs:
  RRTConnect:
    type: geometric::RRTConnect
    range: 0.5

  RRTStar:
    type: geometric::RRTstar
    range: 0.5
    goal_bias: 0.05

  PRM:
    type: geometric::PRM
    max_nearest_neighbors: 10

ompl:
  default_planner_config: RRTConnect
```

•**type**: 사용할 OMPL 플래너의 이름을 지정.
•**파라미터**: 플래너별로 제공되는 고유한 설정값(range, goal_bias 등)을 정의.

플래너 변경

RViz2에서 플래너를 변경하려면 MoveIt 플래닝 인터페이스에서 **플래너 설정** 수정.

OMPL의 주요 알고리즘

- **RRT (Rapidly-exploring Random Tree)**
- 샘플 기반 탐색으로 경로를 빠르게 계산.
- 장애물이 많은 환경에서 효과적.
- **RRT*** (RRT-Star)
- RRT를 기반으로 최적의 경로를 점진적으로 계산.
- 계산 시간이 더 오래 걸릴 수 있음.
- **PRM (Probabilistic Roadmap)**
- 상태 공간에 그래프를 생성하여 경로를 탐색.
- 다중 쿼리에 적합.
- **BIT*** (Batch Informed Trees)
- RRT*와 Dijkstra 알고리즘을 결합하여 효율적인 탐색 수행.
- **LazyPRM**
- 충돌 검사를 지연 수행하여 계산 효율성을 높임.

플래너 선택

- 빠른 계산: RRTConnect
- 최적 경로: RRT*
- 복잡한 환경: PRM

- **파라미터 튜닝**
  - 각 플래너의 파라미터를 조정해 성능을 최적화하세요.
- **충돌 검사기**
  - MoveIt 2의 충돌 검사기와 통합,더 신뢰할 수 있는 경로를 생성.

Plan and Execute using MoveGroupInterface

```
#include <memory>

#include <rclcpp/rclcpp.hpp>
#include
<moveit/move_group_interface/move_group_interface.h>
```

```
// Create the MoveIt MoveGroup Interface
using moveit::planning_interface::MoveGroupInterface;
auto move_group_interface = MoveGroupInterface(node, "panda_arm");

// Set a target Pose
auto const target_pose = []{
  geometry_msgs::msg::Pose msg;
  msg.orientation.w = 1.0;
  msg.position.x = 0.28;
  msg.position.y = -0.2;
  msg.position.z = 0.5;
  return msg;
}();
move_group_interface.setPoseTarget(target_pose);

// Create a plan to that target pose
auto const [success, plan] = [&move_group_interface]{
  moveit::planning_interface::MoveGroupInterface::Plan msg;
  auto const ok = static_cast<bool>(move_group_interface.plan(msg));
  return std::make_pair(ok, msg);
}();

// Execute the plan
if(success) {
  move_group_interface.execute(plan);
} else {
  RCLCPP_ERROR(logger, "Planing failed!");
}
```

Moveit2 - Motion Planning launch

ROS 2에서 MoveIt 2를 사용하여 모션 플래닝(Motion Planning)을 실행하려면, 여러 노드를 실행

1. 필수 노드

MoveIt 2에서 모션 플래닝을 수행하기 위해 실행해야 하는 핵심 노드들은 다음과 같습니다:

1.1 robot_state_publisher

•로봇의 URDF를 기반으로 TF(Transform Frames)를 퍼블리시하는 노드입니다.

```
ros2 launch robot_state_publisher robot_state_publisher.launch.py use_sim_time:=true
```

1.2 rviz2

•MoveIt 2에서 제공하는 RViz 플러그인을 사용하여 플래닝 장면을 시각화합니다.

```
ros2 launch moveit2_tutorials demo.launch.py rviz_config:=<your_rviz_config>
```

1.3 move_group

•MoveIt 2의 핵심 노드로, 플래닝, 실행 및 상호작용을 처리합니다.

```
ros2 launch moveit_ros_move_group move_group.launch.py
```

Moveit2 - Motion Planning launch

1.4 servo_node (선택 사항)

•MoveIt 2에서 실시간 조작(servoing)을 수행하려면 실행합니다.

ros2 launch moveit_servo servo_cpp_interface_demo.launch.py

1.5 플래너

•기본적으로 OMPL(Open Motion Planning Library)를 플래너로 사용합니다.
•실행 예: MoveIt 2의 내부에서 move_group 노드에 의해 자동으로 로드됩니다.

```python
from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():
    return LaunchDescription([
        # 로봇 상태 퍼블리셔
        Node(
            package='robot_state_publisher',
            executable='robot_state_publisher',
            output='screen',
            parameters=[{'use_sim_time': True}]
        ),

        # MoveIt 2의 move_group 실행
        Node(
            package='moveit_ros_move_group',
            executable='move_group',
            output='screen',
            parameters=[
                {'robot_description': '<path_to_your_urdf>'},
                {'robot_description_semantic': '<path_to_your_srdf>'},
                {'planning_pipeline': 'ompl'}
            ]
        ),

        # RViz2 실행
        Node(
            package='rviz2',
            executable='rviz2',
            output='screen',
            arguments=['-d', '<path_to_your_rviz_config>']
        )
    ])
```

Moveit2 - Motion Planning launch

3. Custom Launch 파일 구성

robot_description 및 robot_description_semantic

•URDF 및 SRDF 파일 경로를 지정해야 합니다.
•ROS 2 패키지 경로를 활용해 설정 예:
python

```
{'robot_description': Command(['xacro ', FindPackageShare('your_robot_description_package'), '/urdf/robot.urdf.xacro'])},
{'robot_description_semantic': PathJoinSubstitution([FindPackageShare('your_moveit_config_package'), 'config', 'robot.srdf'])}
```

4. 추가 노드 (옵션)

•**Gazebo**: 실제 시뮬레이션 환경에서 플래닝을 테스트하려면 Gazebo와의 통합이 필요합니다.
•**Controller Manager**: 로봇의 하드웨어 제어를 위해 ros2_control과 통합된 컨트롤러 노드를 실행해야 합니다.

Moveit2

IKFast kinematics solver

IKFast는 OpenRAVE에서 제공하는 강력한 역기구학(Inverse Kinematics) 솔버

주요 특징은:

1.해석적 해법을 사용하여 매우 빠른 계산 속도 제공
2.로봇 암의 모든 가능한 IK 해를 찾을 수 있음
3.URDF 파일로부터 자동으로 IK 솔버 생성
4.C++로 컴파일되어 실행 시 최적의 성능 제공

IKFast 솔버 생성이 필요합니다:

Ros2 run moveit_kinematics create_ikfast_moveit_plugin.py --robot your_robot --iktype Transform6D --search_mode OPTIMIZE_MAX_JOINT

•생성된 IKFast 플러그인을 빌드하고 설치
•ROS2 패키지에 의존성 추가:

주제 : moveit2 이해

Moveit2

```python
import rclpy
from rclpy.node import Node
from sensor_msgs.msg import JointState
from geometry_msgs.msg import Pose
from tf2_ros import TransformBroadcaster
import numpy as np
import ikfast_panda  # 가정: Panda 로봇을 위한 IKFast 모듈

class IKFastNode(Node):
    def __init__(self):
        super().__init__('ikfast_node')

        # Publishers
        self.joint_pub = self.create_publisher(
            JointState,
            'joint_states',
            10
        )

        # Subscribers
        self.pose_sub = self.create_subscription(
            Pose,
            'target_pose',
            self.pose_callback,
            10
        )

        # TF Broadcaster
        self.tf_broadcaster = TransformBroadcaster(self)

        # 로봇 설정
        self.joint_names = [
            'panda_joint1',
            'panda_joint2',
            'panda_joint3',
            'panda_joint4',
            'panda_joint5',
            'panda_joint6',
            'panda_joint7'
        ]

        self.get_logger().info('IKFast Node initialized')

    def pose_callback(self, msg: Pose):
        try:
            # Pose를 IKFast 입력 형식으로 변환
            transform = self.pose_to_transform(msg)

            # IKFast를 사용하여 역기구학 계산
            solutions = ikfast_panda.get_ik(transform)

            if not solutions:
                self.get_logger().warning('No IK solution found')
                return

            # 첫 번째 유효한 해 선택
            joint_values = solutions[0]

            # JointState 메시지 생성 및 발행
            joint_state = JointState()
            joint_state.header.stamp = self.get_clock().now().to_msg()
            joint_state.name = self.joint_names
            joint_state.position = joint_values

            self.joint_pub.publish(joint_state)

            self.get_logger().info(f'Published joint solution: {joint_values}')

        except Exception as e:
            self.get_logger().error(f'Error calculating IK: {str(e)}')

    def pose_to_transform(self, pose: Pose) -> list:
        """Pose 메시지를 IKFast 변환 행렬로 변환"""
        # 쿼터니언을 회전 행렬로 변환
        x, y, z, w = pose.orientation.x, pose.orientation.y, pose.orientation.z, pose.orientation.w

        # 회전 행렬 계산
        r00 = 1 - 2*y*y - 2*z*z
        r01 = 2*x*y - 2*z*w
        r02 = 2*x*z + 2*y*w
        r10 = 2*x*y + 2*z*w
        r11 = 1 - 2*x*x - 2*z*z
        r12 = 2*y*z - 2*x*w
        r20 = 2*x*z - 2*y*w
        r21 = 2*y*z + 2*x*w
        r22 = 1 - 2*x*x - 2*y*y

        # IKFast 형식의 변환 행렬 반환
        return [
            r00, r01, r02, pose.position.x,
            r10, r11, r12, pose.position.y,
            r20, r21, r22, pose.position.z,
            0.0, 0.0, 0.0, 1.0
        ]

def main(args=None):
    rclpy.init(args=args)
    node = IKFastNode()

    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        pass
    finally:
        node.destroy_node()
        rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Inverse kinematics

https://github.com/dbddqy/visual_kinematics
forward kinematic - 로봇의 각 관절의 각도로 로봇의 끝단의 위치와 방향을 찾는 것
다관절 로봇의 경우 inverse kinematic의 해가 유일하지 않다.

Visual Kinematics는 **운동학(Kinematics)**과 관련된 계산과 시각화를 쉽게 처리하기 위한 라이브러리 또는 소프트웨어입니다. 주로 로봇의 기구학적 계산(순운동학, 역운동학, 좌표계 변환 등)을 단순화하고, 결과를 시각적으로 표현할 수 있도록 설계되었습니다.

```
pip3 install visual-kinematics
```

```
import  numpy

from visual_kinematics.RobotSerial import *

from math import pi
```

Visual Kinematics

- **운동학 계산**
- 순운동학(FK) 및 역운동학(IK) 계산 지원.
- 로봇 모델의 DH 파라미터 또는 변환 행렬을 기반으로 동작.
- **시각화**
- 로봇의 각 링크, 관절, 엔드 이펙터(End-Effector)의 위치와 자세를 3D 그래픽으로 표현.
- **좌표계 변환**
- 여러 좌표계 간의 변환 계산.
- 변환 행렬을 사용하여 로봇의 링크 간 관계를 관리.
- **로봇 모델링**
- 로봇의 구조를 모델링(링크, 관절 등)하고, 이 모델을 기반으로 운동학 계산.
- **다양한 운동학 체계 지원**
- 직렬(Serial) 및 병렬(Parallel) 로봇 구조 모두 지원.

Visual Kinematics 순운동학 예제

Visual Kinematics는 로봇 운동학의 기본 개념을 학습하고 적용하기 위한 강력한 도구입니다.
시각화를 통해 복잡한 수학적 계산을 직관적으로 이해할 수 있고, 다양한 응용 분야에서 활용 가능합니다.

```python
from visual_kinematics import Robot

# 로봇 정의 (DH 파라미터 기반)
dh_params = [
    [0, 0, 0.5, 0],  # [theta, d, a, alpha]
    [0, 0, 0.3, 0],
    [0, 0, 0.2, 0]
]

robot = Robot(dh_params)

# 관절 값 (Joint Angles)
joint_values = [0.5, 1.0, -0.5]

# 순운동학 계산 (엔드 이펙터 위치)
end_effector_position = robot.forward_kinematics(joint_values)

print("End Effector Position:", end_effector_position)
```

Visual Kinematics 역운동학 예제

```
# 목표 위치 정의
goal_position = [0.5, 0.2, 0.3]

# 역운동학 계산
joint_values = robot.inverse_kinematics(goal_position)

print("Joint Angles:", joint_values)
```

시각화

robot.plot(joint_values)

Visual Kinematics 사용의 한계

1. **실제 로봇 하드웨어와의 통합 부족**:
    1. Visual Kinematics는 시뮬레이션과 계산 중심으로, 하드웨어 제어는 별도의 모듈로 처리해야 함.
2. **고급 물리 시뮬레이션 부족**:
    1. 단순 운동학 계산이 주 목표이므로 동역학(Dynamics)이나 충돌 검사는 지원하지 않음.

주제 : moveit2 이해

ROS2 + Moveit2 for RoArm-M2-S

Waveshare RoArm-M2-S 데스크탑 로봇 암 키트 하이 토크 직렬 버스 서보 ESP32 기반 확장 및 2 차 개발

roarm_ws_em0 Package Overview

4.roarm_moveit Kinematic Configuration:
Provides configurations for MoveIt, a motion planning framework, including setup files and parameters required for the kinematic control of the robotic arm.

5.roarm_moveit_ikfast_plugins IKFast Kinematics Solver:
Implements the IKFast kinematics solver, which is used for efficient and fast inverse kinematics calculations.

6.roarm_moveit_cmd Control Commands : ros2 service server
Includes scripts and nodes for sending control commands to the robotic arm, allowing for movement and task execution.

Roarm

ROS2 + Moveit2 for RoArm-M2-S

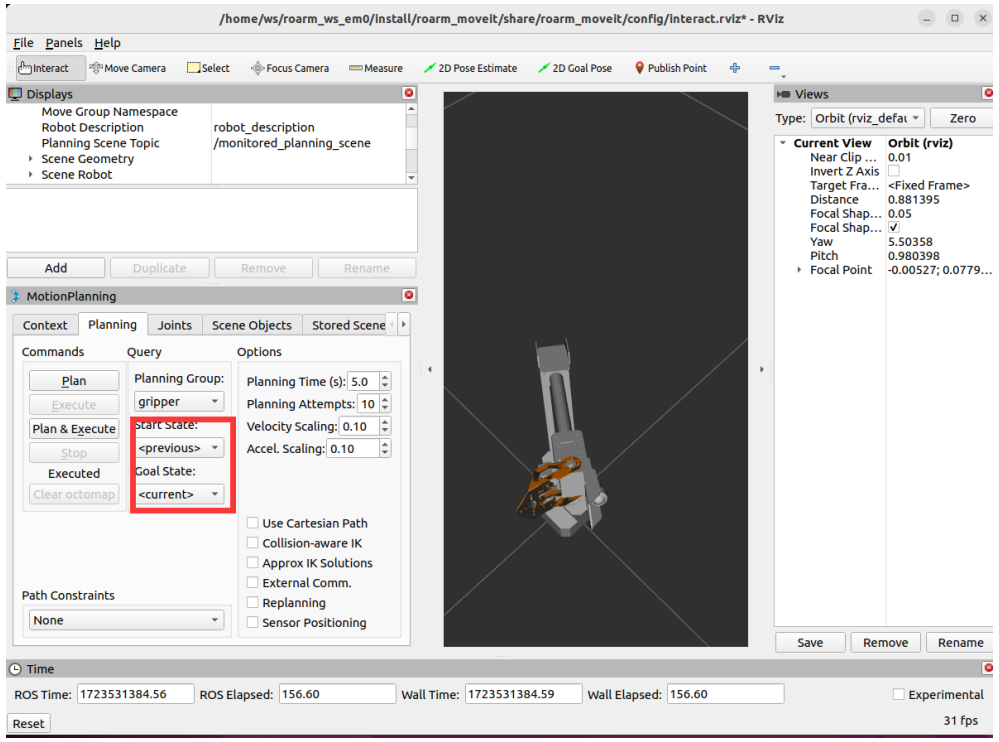
https://github.com/waveshareteam/roarm_ws_em0

```
sudo apt update
sudo apt upgrade

sudo apt install ros-humble-desktop

sudo apt install ros-dev-tools

sudo apt install net-tools
sudo apt install ros-humble-moveit-*
sudo apt install ros-humble-foxglove-bridge
sudo apt autoremove ros-humble-moveit-servo-*
```

참고: https://github.com/karlkwon/spark_x_F/blob/main/turtlebot3_manipulation_test.py

RoArm - Running the MoveIt2 Demo

ros2 launch roarm_moveit interact.launch.py



Controlling the Robotic Arm Using a Web Interface (Foxglove)

ROS2 + Moveit2 for RoArm-M2-S

Waveshare RoArm-M2-S 데스크탑 로봇 암 키트 하이 토크 직렬 버스 서보 ESP32 기반 확장 및 2 차 개발

Change the Serial Port Device

打开(O)

roarm_driver.py

~/roarm_ws_em0/src/roarm_main/roarm_driver/roarm_driver

```python
import rclpy
from rclpy.node import Node
import json
import serial
from serial import SerialException
from sensor_msgs.msg import JointState
from geometry_msgs.msg import Pose
from roarm_moveit.srv import GetPoseCmd
import queue
import threading
import logging
import time
import math

serial_port = "/dev/ttyUSB0"

class ReadLine:
    def __init__(self, s):
        self.buf = bytearray()
        self.s = s

    def readline(self):
        i = self.buf.find(b"\n")
        if i >= 0:
            r = self.buf[:i+1]
            self.buf = self.buf[i+1:]
            return r
        while True:
            i = max(1, min(512, self.s.in_waiting))
            data = self.s.read(i)
            i = data.find(b"\n")
            if i >= 0:
                r = self.buf + data[:i+1]
                self.buf[0:] = data[i+1:]
                return r
            else:
                self.buf.extend(data)

    def clear_buffer(self):
        self.s.reset_input_buffer()
```

•Your First C++ MoveIt Project

Move the End-Effector to a Specified Position

ros2 run roarm_moveit_cmd movepointcmd

ros2 service call /move_point_cmd roarm_moveit/srv/MovePointCmd "{x: 0.2, y: 0, z: 0}"

Control the gripper to the specified radian position

ros2 run roarm_moveit_cmd setgrippercmd

ros2 topic pub /gripper_cmd std_msgs/msg/Float32 "{data: 0.0}" -1

Control the gripper to the specified radian position

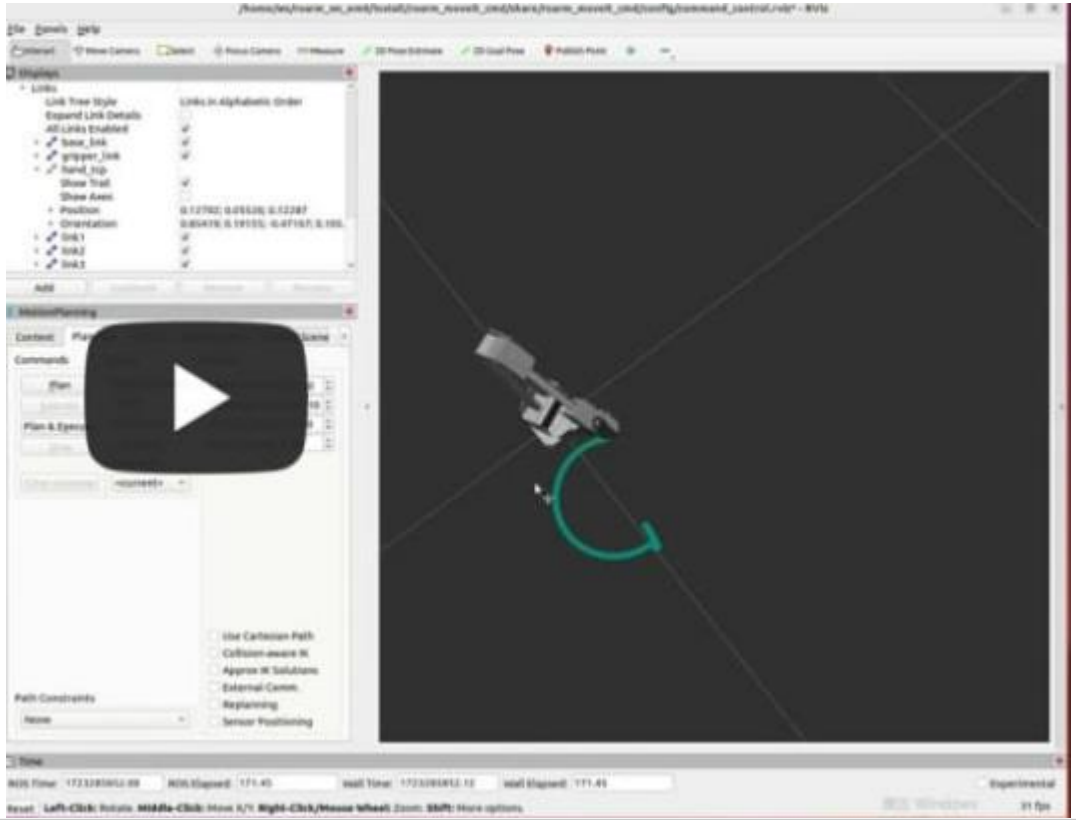
ros2 run roarm_moveit_cmd setgrippercmd

ros2 topic pub /gripper_cmd std_msgs/msg/Float32 "{data: 0.0}" -1

Moveit2

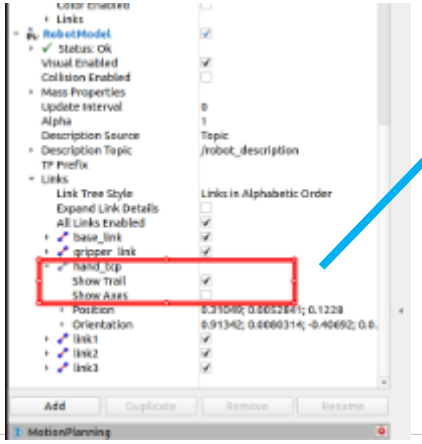
Draw a Circle at a Fixed Height

In Rviz2, click Add, add RobotModel, and in the RobotModel tab, find Description Topic to view the trajectory of the end-effector hand_tcp.



ros2 run roarm_moveit_cmd movecirclecmd

ros2 service call /move_circle_cmd roarm_moveit/srv/MoveCircleCmd "{x: 0.2, y: 0, z: 0, radius: 0.1}"



Hand_tcp -> show Trail

Roarm_moveit_cmd

https://github.com/waveshareteam/roarm_ws_em0/blob/ros2-humble/src/roarm_main/roarm_moveit_cmd/src/movepointcmd.cpp

```cpp
#include <memory>
#include <math.h>
#include <rclcpp/rclcpp.hpp>
#include <moveit/move_group_interface/move_group_interface.h>
#include <geometry_msgs/msg/pose.h>
#include <tf2_geometry_msgs/tf2_geometry_msgs.h>
#include "roarm_moveit/srv/move_point_cmd.hpp"
#include "roarm_moveit_cmd/ik.h"
void handle_service(const std::shared_ptr<roarm_moveit::srv::MovePointCmd::Request> request,
                    std::shared_ptr<roarm_moveit::srv::MovePointCmd::Response> response)
{
  // 在这里处理服务请求
  rclcpp::Node::SharedPtr node = std::make_shared<rclcpp::Node>("move_point_cmd_service_node");
  auto logger = node->get_logger();

  // 创建 MoveIt MoveGroup 接口
  moveit::planning_interface::MoveGroupInterface move_group(node, "hand");

  // 设置目标位置和姿态
  geometry_msgs::msg::Pose target_pose;
  target_pose.position.x = request->x;
  target_pose.position.y = -1*request->y;
  target_pose.position.z = request->z;

  cartesian_to_polar(1000*target_pose.position.x,1000*target_pose.position.y, &base_r, &BASE_point_RAD);
  simpleLinkageIkRad(l2, l3, base_r, 1000*target_pose.position.z);
```

여기에서 서비스 요청을 처리하세요.

MoveIt MoveGroup 인터페이스 생성

목표 위치 및 태도 설정

주제 : moveit2 이해

Roarm_moveit_cmd

```
  RCLCPP_INFO(logger, "BASE_point_RAD: %f, SHOULDER_point_RAD: %f, ELBOW_point_RAD: %f", BASE_point_RAD,
  RCLCPP_INFO(logger, "x: %f, y: %f, z: %f", request->x, request->y, request->z);
  // 定义目标关节值
  std::vector<double> target = {BASE_point_RAD, -SHOULDER_point_RAD, ELBOW_point_RAD};
  // 规划并执行轨迹
  moveit::planning_interface::MoveGroupInterface::Plan my_plan;
  bool success = (move_group.plan(my_plan) == moveit::planning_interface::MoveItErrorCode::SUCCESS);
  if(success) {
    move_group.execute(my_plan);
    response->success = true;
    response->message = "MovePointCmd executed successfully";
    RCLCPP_INFO(logger, "MovePointCmd service executed successfully");
  } else {
    response->success = false;
    response->message = "Planning failed!";
    RCLCPP_ERROR(logger, "Planning failed!");
  }
}

int main(int argc, char * argv[])
{
  rclcpp::init(argc, argv);
  auto node = std::make_shared<rclcpp::Node>("move_point_cmd_node");

  // 创建服务
  auto server = node->create_service<roarm_moveit::srv::MovePointCmd>("move_point_cmd", &handle_service)

  RCLCPP_INFO(node->get_logger(), "MovePointCmd service is ready.");
```

목표 관절 값 정의

궤적 계획 및 실행

Robotis OpenManipulator-x

OpenMANIPULATOR-X의 컨트롤러 액션 서버는 ROS(Robot Operating System)의 액션 서버 프레임워크

1.기본 구조

- •로봇 팔의 관절 위치와 그리퍼 제어를 위한 액션 서버를 제공
- •goal, feedback, result의 3가지 메시지 타입을 사용하여 통신
- •비동기적 실행을 지원하여 긴 작업 수행 중에도 다른 작업 가능

2.주요 액션 서버들

- •/joint_trajectory_controller: 관절 궤적 제어
- •/gripper_controller: 그리퍼 제어
- •/goal_task_space_path: 작업 공간에서의 경로 계획
- •/goal_joint_space_path: 조인트 공간에서의 경로 계획
- •/goal_drawing_trajectory: 그리기 작업을 위한 궤적 생성

Robotis OpenManipulator-x

3. 주요 기능:
    - 실시간 위치/속도 제어
    - 역기구학(Inverse Kinematics) 계산
    - 충돌 감지 및 회피
    - 경로 계획(Path Planning)
    - 그리퍼 제어

4. 에러 처리:
    - 관절 제한 검사
    - 속도 제한 검사
    - 토크 제한 검사
    - 실행 시간 초과 처리

Robotis OpenManipulator-x

```
import rclpy
from rclpy.node import Node
from rclpy.action import ActionClient
from control_msgs.action import FollowJointTrajectory
from trajectory_msgs.msg import JointTrajectory, JointTrajectoryPoint
from rclpy.duration import Duration

class OpenManipulatorControl(Node):
    def __init__(self):
        super().__init__('open_manipulator_control')

        # Define joint names for OpenMANIPULATOR-X
        self.joint_names = [
            'joint1',
            'joint2',
            'joint3',
            'joint4',
            'arm'
        ]

        # Create action client
        self.trajectory_client = ActionClient(
            self,
            FollowJointTrajectory,
            '/arm_controller/follow_joint_trajectory'
        )

        self.get_logger().info('Waiting for action server...')
        self.trajectory_client.wait_for_server()
        self.get_logger().info('Action server connected!')
```

노드 초기화 (OpenManipulatorControl 클래스)

관절 이름 정의

OpenMANIPULATOR-X의 관절 이름을 설정합니다.

액션 클라이언트 생성

/arm_controller/follow_joint_trajectory 액션 서버에 연결하는 클라이언트를 생성합니다.

주제 : moveit2 이해

Robotis OpenManipulator-x

```
def move_to_joint_positions(self, positions, duration=5.0):
    """Send joint positions to the robot"""
    trajectory_msg = JointTrajectory()
    trajectory_msg.joint_names = self.joint_names[:-1]  # Exclude gripper

    # Create trajectory point
    point = JointTrajectoryPoint()
    point.positions = positions
    point.velocities = [0.0] * len(positions)
    point.accelerations = [0.0] * len(positions)
    point.time_from_start = Duration(seconds=duration).to_msg()

    trajectory_msg.points.append(point)

    # Create goal
    goal_msg = FollowJointTrajectory.Goal()
    goal_msg.trajectory = trajectory_msg

    # Send goal
    self.get_logger().info(f'Sending goal: {positions}')
    self._send_goal(goal_msg)
```

- 특정 **관절 위치(positions)** 로 이동하는 기능을 수행합니다.
- duration=5.0은 목표 위치로 이동하는 데 걸리는 시간을 설정합니다

JointTrajectory 메시지를 생성하고, 사용할 관절 이름을 설정합니다.

JointTrajectoryPoint 객체를 생성하여 관절 목표 위치를 설정합니다.

FollowJointTrajectory.Goal 메시지를 생성하여, 경로(trajectory_msg)를 포함합니다.

_send_goal() 메서드를 호출하여 목표를 전송합니다.

Robotis OpenManipulator-x

```
def _send_goal(self, goal_msg):
    """Send goal and handle result"""
    future = self.trajectory_client.send_goal_async(goal_msg)
    rclpy.spin_until_future_complete(self, future)

    goal_handle = future.result()
    if not goal_handle.accepted:
        self.get_logger().error('Goal rejected')
        return

    self.get_logger().info('Goal accepted')

    # Wait for result
    result_future = goal_handle.get_result_async()
    rclpy.spin_until_future_complete(self, result_future)

    result = result_future.result().result
    if result.error_code == 0:
        self.get_logger().info('Goal succeeded!')
    else:
        self.get_logger().error(f'Goal failed with error code: {result.error_code}')
```

Robotis OpenManipulator-x

```
def main():
    rclpy.init()
    node = OpenManipulatorControl()

    try:
        # Example positions (in radians)
        # Home position
        node.move_to_joint_positions([0.0, 0.0, 0.0, 0.0])

        # Forward position
        node.move_to_joint_positions([0.0, -1.0, 0.3, 0.7])

        # Side position
        node.move_to_joint_positions([1.57, -0.8, 0.5, 0.8])

        # Return to home
        node.move_to_joint_positions([0.0, 0.0, 0.0, 0.0])

    except KeyboardInterrupt:
        pass
    finally:
        node.destroy_node()
        rclpy.shutdown()

if __name__ == '__main__':
    main()
```

주제 : moveit2 이해

Robotis OpenManipulator-x

감사합니다.